CyberLink for Java

Programming Guide

Document Version 1.3

Table of Contents

1 Introduction

UPnP™*[1] architecture is based on open networking to enable discovery and control of networked devices and services, such as media servers and players at home.

UPnP™ architecture is based on many standard protocols, such as GENA, SSDP, SOAP, HTTPU and HTTP. Therefore you have to understand and implement these protocols to create your devices of UPnP™.

CyberLink for Java is a development package for UPnP™ developers. CyberLink controls these protocols automatically, and supports to create your devices and control points quickly.

Please see the following site and documents to know about UPnP™ in more detail.

Document	URL
UPnP™ Forum	http://www.upnp.org/
Universal Plug and Play Device Architecture	http://www.upnp.org/download/UPnPDA10_20000613.htm
Universal Plug and Play Vendor's Implementation Guide	http://www.upnp.org/download/UPnP_Vendor_Implementation_Guide_Jan2001.htm

[1] UPnP™ is a certification mark of the UPnP™ Implementers Corporation.

2 Setup

To use the package, copy the package into your JDK and JRE library directory. For example,

```
cp clink???.jar C:¥jdk1.x¥jre¥lib¥ext¥
cp clink???.jar C:¥Program Files¥JavaSoft¥JRE¥1.x¥lib¥ext¥
```

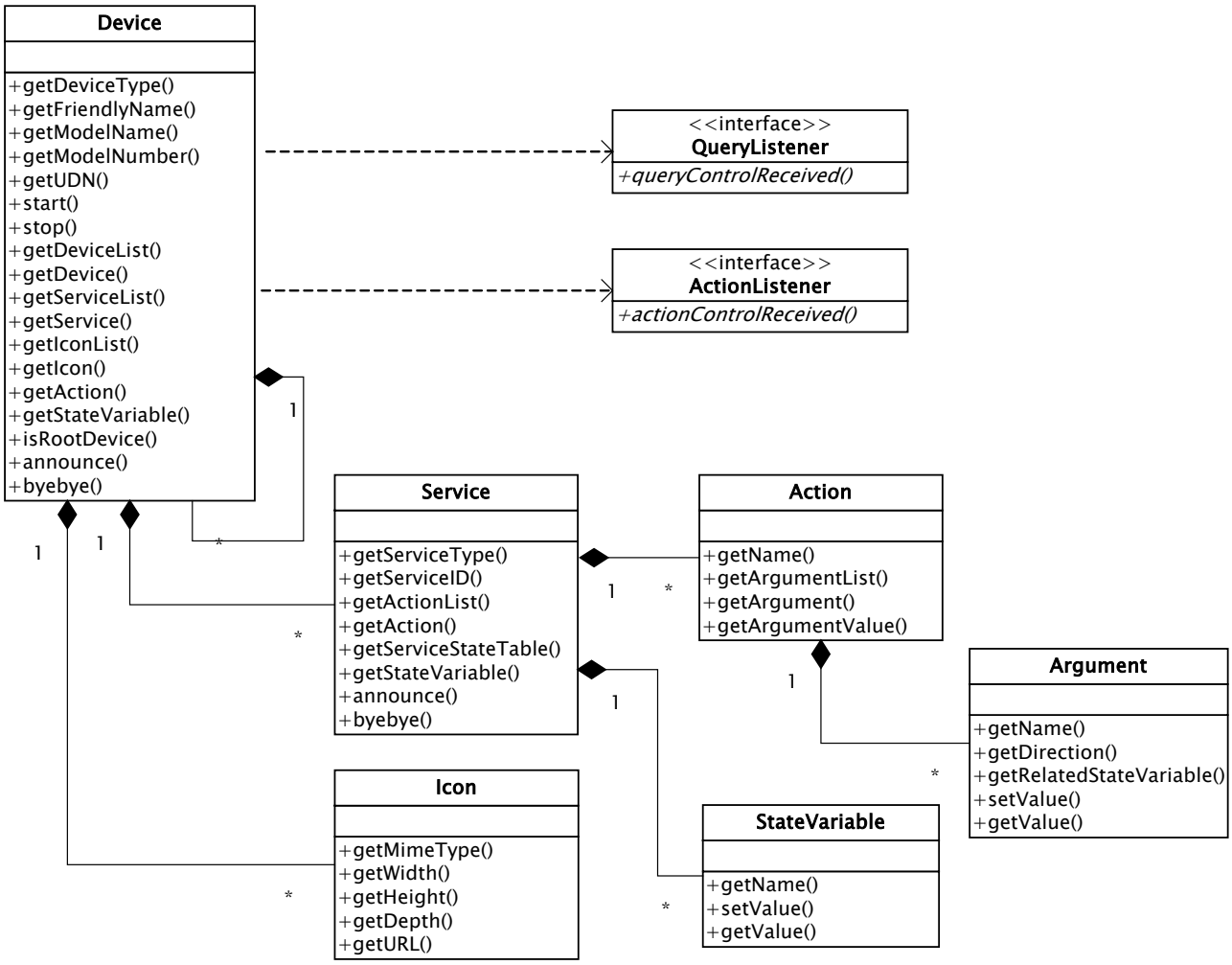
The current package needs the following package to parse the XML and SOAP requests. CyberLink uses the Xerers package as the default parser. Therefore you don't have to install the KXML2 package if you want to use it.

Package	URL
Apache Xerces	http://xml.apache.org/xerces2-j/index.html
kXML2	http://www.kxml.org/

3 Device

3.1 Class Overview

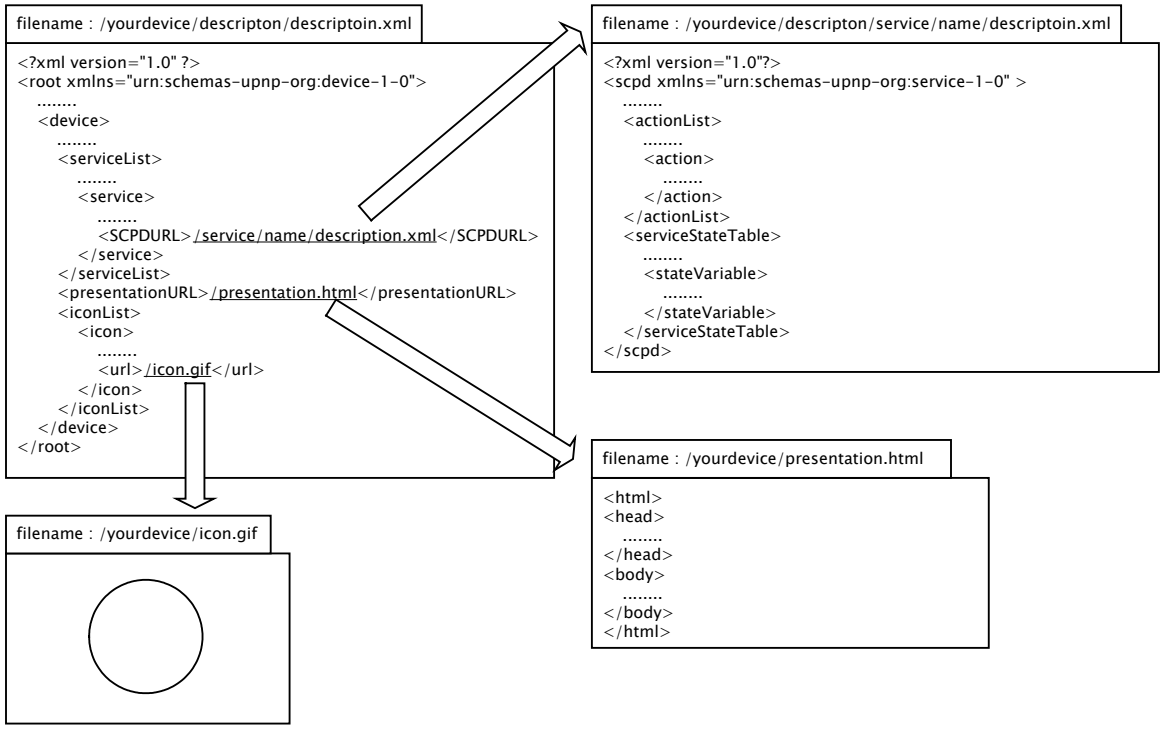
The following static structure diagram is related classes of CyberLink to create your device of UPnP™. The device has some embedded devices and services, and the services have some actions and state variables.



The above static structure diagram is modified simplify to explain. See the JavaDoc documentation to know other methods and variables in more detail.

3.2 Description

At first, you have to make some description files of your devices and the services when you want to create your UPnP™ device. The URLs in the device description should be relative locations from the directory of the device description file.



The description of the root device should not have URLBase element because the element is added automatically when the device is created using the description.

The service descriptions is required to create a device, but the presentationURL and the iconList are recommended option. Please see UPnP™ specifications about the description format in more detail.

3.3 Initiating

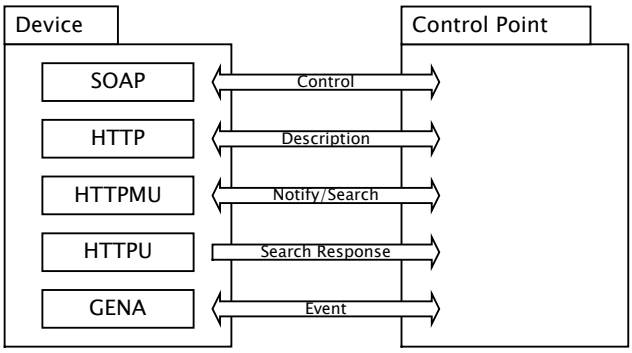
To create a UPnP™ device, create a instance of Device class with the root description file. The created device is a root device, and only root device can be active using Device::start(). The device is announced to the UPnP™ network when the device is started. The following shows an example of the initiating device.

```
import org.cybergarage.upnp.*;
import org.cybergarage.upnp.device.*;
.....
String descriptionFileName = "description/description.xml";
Try {
  Device upnpDev = new Device(descriptionFileName);
  .....
```

```
    upnpDev.start();
}
catch (InvalidDescriptionException e){
    String errMsg = e.getMessage();
    System.out.println("InvalidDescriptionException = " + errMsg);
}
```

The InvalidDescriptionException is occurred when the description is invalid. Use the getMessage() to know the exception reason in more detail.

The active root device has some server processes, and returns the responses automatically when a control points sends a request to the device. For example, the device has a HTTP server to return the description files when a control point gets the description file. The device searches an available port for the HTTP server automatically on the machine when the device is started.

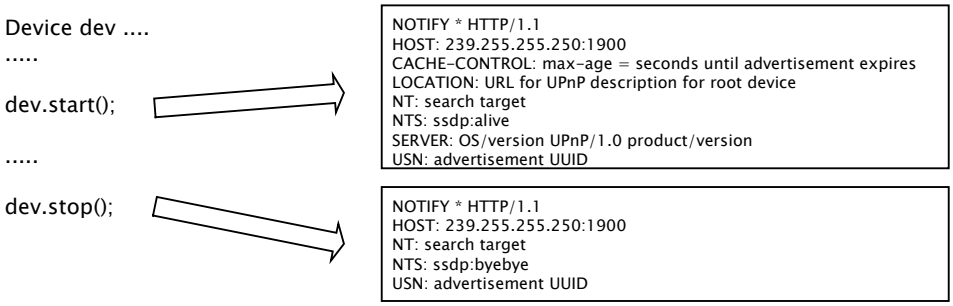


The root device is created with the following default parameters, you can change the parameters using the following methods before the root device is started.

	Parameter	Default	Method	Detail
1	HTTP port	4004	setHTTPPort()	The http server uses the port in the root device.
2	Description URI	/description.xml	setDescriptionURI()	The description URI of the root device.
3	Lease time	1800	setLeaseTime()	The lease time of the root device.

3.4 Notify

Your device is announced using Device::start() to the UPnP™ network using a notify message with ssdp::alive automatically when the device is started. When device is stopped using Device::stop(), a notify message is posted with ssdp::byebye. You can announce the notify messages using Device::announce() and Device::byebye().



When a control point sends a search request with M-SEARCH to the UPnP™ network, the active device send the search response to the control point automatically. The device repeats the announce in the lease time automatically.

3.5 Embedded Devices

The devices may have some embedded devices. Use Device::getDeviceList() to get the embedded device list. The following example outputs friendly names of all embedded devices in the device.

```
public void printDevice(Device dev)
{
    String devName = dev.getFriendlyName();
    System.out.println(devName);

    DeviceList childDevList = dev.getDeviceList();
    int nChildDevs = childDevList.size();
    for (int i=0; i<nChildDevs; i++) {
      Device childDev = rootDevList.getDevice(n);
      printDevice(childDev);
    }
}
 ......
Dev rootDev = ....;
......
DeviceList childDevList = rootDev.getDeviceList();
int childDevs = childDevList.size();
for (int n=0; n< childDevs; n++) {
```

```
        Device childDev = rootDevList.getDevice(n);
        printDevice(childDev);
    }
```

You can find a embedded device by the friendly name or UDN using Device::getDevice(). The following example gets a embedded device by the friendly name.

```
Device homeServerDev ....
Device musicDev = homeServerDev.getDevice("music");
```

3.6 Service

Use Device::getServiceList() to access embedded services of the device. The service may has some actions and state variables. Use Service::getActionList() to get the actions, and use Service::getServiceStateTable() to the state variables. The following example outputs the all actions and state variables in a device.

```
Device dev ....
ServiceList serviceList = dev.getServiceList();
int serviceCnt = serviceList.size();
for (int n=0; n<serviceCnt; n++) {
    Service service = serviceList.getService(n);
    ActionList actionList = service.getActionList();
    int actionCnt = actionList.size();
    for (int i=0; i<actionCnt; i++) {
      Action actoin = actionList.getActoin(i);
      System.out.println("action [" + i + "] = " + action.getName());
    }
    ServiceStateTable stateTable = service.getStateTable();

    int varCnt = stateTable.size();
    for (int i=0; i<actionCnt; i++) {
      StateVariable stateVar = stateTable.getServiceStateVariable(n);
      System.out.println("stateVar [" + i + "] = " + stateVar.getName());
    }
}
```

You can find a service in the device by the service ID using Device::getService(), and you can find an action or state variable in the service by name too. Use Device::getAction() or Service::getAction() to find the action, and use Device::getStatearible() or Service::getStateVariable() to find the state variable. The following example gets a service, a action and a state variable in a device by name.

```
Device clockDev ....
Service timerSev = clockDev.getService("timer");
Action getTimeAct = clockDev.getAction("GetTime");
StateVariable timeStat = clockDev.getStateVariable("time");
```

3.7 Control

To receive action control events from control points, the device needs to implement the ActionListener interface. The listener have to implement a actionControlReceived() that has the action and argument list parameter. The input arguments has the passed values from the control point, set the response values in the output arguments and return a true when the request is valid. Otherwise return a false when the request is invalid. UPnPError response is returned to the control point automatically when the returned value is false or the device has no the interface. The UPnPError is INVALID_ACTION as default, but use Action::setSetStatus() to return other UPnP errors.

To receive query control events from control points, the device needs to implement the QueryListener interface. The listener have to implement a queryControlReceived() that has the service variable parameter and return a true when the request is valid. Otherwise return a false when the request is invalid. UPnPError response is returned to the control point automatically when the returned value is false or the device has no the interface. The UPnPError is INVALID_ACTION as default, but use ServiceVariable::setSetStatus() to return other UPnP errors.

The following sample is a clock device. The device executes two action control requests and a query control request.

```
public class ClockDevice extends Device implements ActionListener, QueryListener
{
  public ClockDevice()
  {
    super ("/clock/www/description.xml");
    Action setTimeAction = getAction("SetTime");
    setTimeAction.setActionListener(this);
    Action getTimeAction = getAction("GetTime");
    getTimeAction.setActionListener(this);
    StateVariable stateVar = getStateVariable("Timer");
    stateVar.setQueryListener(this);
  }

  public boolean actionControlRecieved(Action action)
  {
    ArgumentList argList = action.getArgumentList();
```

```
        String actionName = action.getName();
        if (actionName.equals("SetTime") == true) {
          Argument inTime = argList.getArgument("time");
          String timeValue = inTime.getValue();
          If (timeValue == null || timeValue.length() <= 0)
            return false;
          ........
          Argument outResult = argList.getArgument("result");
          arg.setValue("TRUE");
          return true;
        }
        else if (actionName.equals("GetTime") == true) {
          String currTimeStr = .....
          Argument currTimeArg = argList.getArgument("currTime");
          currTimeArg.setValue(currTimeStrs);
          return true;
        }
        action.setStatus(UPnP::INVALID_ACTION, ".....");
        return false;
      }

      public bool queryControlReceived(StateVariable stateVar)
      {
        if (varName.equals("Time") == true) {
          String currTimeStr = ....;
          stateVar.setValue(currTimeStr);
          return true;
        }
        stateVar.setStatus(UPnP::INVALID_VAR, ".....");
        return false;
      }
    }
```

Use Device::setActionListner() or Service::setActionListnerer() to add a listener into all control actions in a device or a service. Use Device::setQueryListner() or Service::setQueryListner() to add a listener into all query actions in a device or a service. The following sample sets a listener into all actions in a device.

```
class ClockDevice : public Device, public ActionListener, public QueryListener
{
  public:
```

```
    ClockDevice() : Device("/clock/www/description.xml")
    {
        setActionListner(this);
        setQueryListener (this);
    }
    bool actionControlRecieved(Action *action) { ..... }
    bool queryControlReceived(StateVariable *stateVar) { ..... }
}
```

3.8 Event

The control point may subscribe some events of the device. You don't need manage the subscription messages from control points because the device manages the subscription messages automatically. For example, the device adds a control point into the subscriber list when the control point sends a subscription message to the device, or the device removes the control point from the subscriber list when the control point sends a unsubscription message.

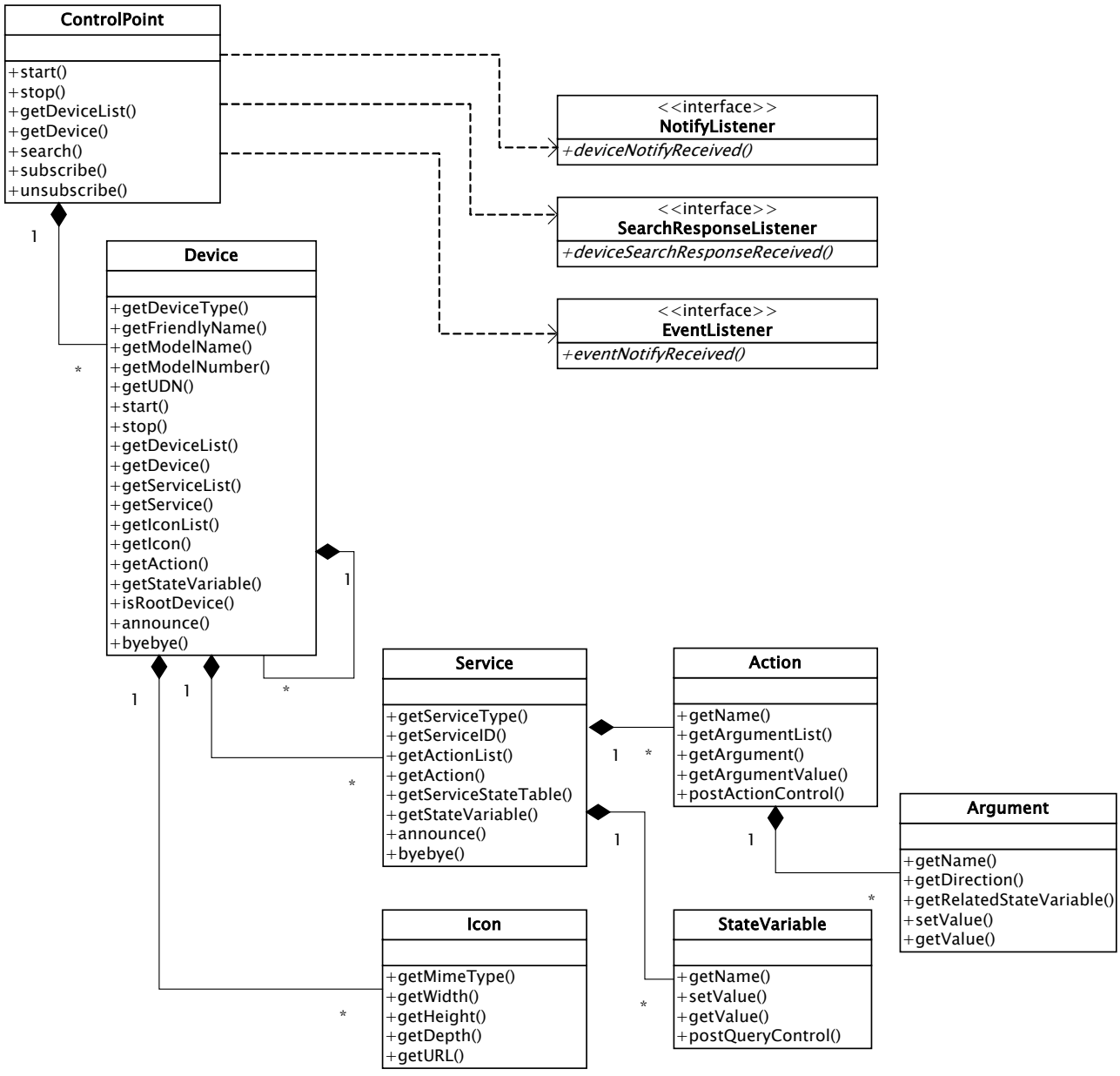
Use ServiceStateVariable::setValue() when you want to send the state to the subscribers. The event is sent to the subscribers automatically when the state variable is updated using ServiceStateVariable::setValue(). The following example updates a state variable, and the updated state is distributed to the subscribers automatically.

```
Device clockDevice = ....
StateVariable timeVar = clockDevice.getStateVariable("Time");
String timeStr = .....
timeVar.setValue(timeStr);
```

4 Control Point

4.1 Class Overview

The following static structure diagram is related classes of CyberLink to create your control point of UPnP™. The control point has some root devices in the UPnP™ network.



The above static structure diagram is modified simplify to explain. See the JavaDoc documentation to know other methods and variables in more detail.

4.2 Initiating

To create a UPnP™ control point, create a instance of ControlPoint class. Use ControlPoint::start() to active the contol point. The control point multicasts a discovery message searching for all devices to the UPnP™ network automatically when the control point is active.

```
import org.cybergarage.upnp.*;
import org.cybergarage.upnp.device.*;
......
ControlPoint ctrlPoint = new ControlPoint();
......
ctrlPoint.start();
```

The active control point has some server processes, and returns the responses automatically when other UPnP™ devices send the messages to the control point. For example, the control point has a SSDP server to get M-SEARCH responses, and the control point searches a available port for the SSDP server automatically on the machine when the control point is started.

The control point is created with the following default parameters. You can change the parameters using the following methods before the control point is started.

	Parameter	Default	Method	Detail
1	HTTP port	8058	setHTTPPort()	The port is used to receive subscription events.
2	SSDP port	8008	setSSDPPort()	The port is used to receive search responses.
3	Subscription URI	/eventSub	setEventSubURI()	The URI is used to receive subscription events.
4	Search Response	3	setSerchMx()	Maximum wait for device searching

4.3 Notify

The control point receives notify events from devices in the UPnP™ network, and the devices are added or removed form the control point automatically. The expired device is removed from the device list of the control point automatically too.You don't manage the notify events, but you can receive the event to implement the NotifyListener interface. The following sample receives the notify messages.

```
public class MyCtrlPoint extends ControlPoint implements NotifyListener
{
  public MyCtrlPoint()
  {
    ........
    addNotifyListener(this);
    start();
  }
```

```
  public void deviceNotifyReceived(SSDPPacket ssdpPacket)
  {
    String uuid = ssdpPacket.getUSN();
    String target = ssdpPacket.getNT();
    String subType = ssdpPacket.getNTS();
    String location = ssdpPacket.getLocation();
    .......
  }
```

4.4 Search

You can update the device lists using ControlPoint::search(). The discovered root devices are added to the control point automatically, and you can receive the response to implement the SearchResponseListener interface. The following sample receives the notify messages.

```
public class MyCtrlPoint extends ControlPoint implements SearchResponseListener
{
  public MyCtrlPoint()
  {
    ........
    addSearchResponseListener(this);
    start();
    ........
    search("upnp:rootdevice");
  }
  public void deviceSearchResponseReceived(SSDPPacket ssdpPacket)
  {
    String uuid = ssdpPacket.getUSN();
    String target = ssdpPacket.getST();
    String location = ssdpPacket.getLocation();
   ........
  }
```

4.5 Root Devices

Use ControlPoint:: getDeviceList() that returns only root devices to get the discovered device list. The following example outputs friendly names of the root devices.

```
ControlPoint ctrlPoint = new ControlPoint();
......
ctrlPoint.start();
......
```

```
DeviceList rootDevList = ctrlPoint.getDeviceList();
int nRootDevs = rootDevList.size();
for (int n=0; n<nRootDevs; n++) {
    Device dev = rootDevList.getDevice(n);
    String devName = dev.getFriendlyName();
    System.out.println("[" + n + "] = " + devName);
}
```

You can find a root device by the friendly name, the device type, or the UDN using ControlPoint::getDevice(). The following example gets a root device by the friendly name.

```
ControlPoint ctrlPoint = new ControlPoint();
......
ctrlPoint.start();
......
Device homeServerDev = ctrlPoint.getDevice("xxxx-home-server");
```

4.6 Control

The control point can send action or query control messages to the discovered devices. To send the action control message, use Action::setArgumentValue() and Action::postControlAction (). You should set the action values to the all input arguments, and the output argument values is ignored if you set. The following sample posts a action control request that sets a new time, and output the response result.

```
Device clockDev = ....
Action setTimeAct = clockDev.getAction("SetTime");
String newTime = ....
setTimeAct.setArgumentValue("time", newTime); // setTimeAct.getArgument("time").setValue(newTime);
if (setTimeAct.postControlAction() == true) {
  ArgumentList outArgList = setTimeAct.getOutputArgumentList();
  int nOutArgs = outArgList.size();
  for (int n=0; n<nOutArgs; n++) {
    Argument outArg = outArgList.getArgument(n);
    String name = outArg.getName();
    String value = outArg.getValue();
    ......
  }
}
else {
  UPnPError err = setTimeAct.getUPnPError();
  System.out.println("Error Code = " + err.getCode());
```

```
    System.out.println("Error Desc = " + err.getDescription());
}
```

To send the query control message, use StateVariable::postQueryControl(). The following sample posts a query control request, and output the return value.

```
Device clockDev = ....
StateVariable timeStateVar = clockDev.getStateVariable("time");
if (timeStateVar.postQueryControl() == true) {
    String value = timeStateVar.getValue();
    ......
}
else {
    UPnPError err = timeStateVar.getUPnPError();
    System.out.println("Error Code = " + err.getCode());
    System.out.println("Error Desc = " + err.getDescription());
}
```

4.7 Event

The control point can subscribe events of the discovered devices, get the state changes of the services Use ControlPoint::subscribe() and implement the EventListener interface. The listener have to implement a eventNotifyReceived().

```
public MyControlPoint extends ControlPoint implements EventListener
{
    public MyControlPoint()
    {
        .....
        addEventListener(this);
    }
    .....
    public void eventNotifyReceived(String uuid, long seq, String name, String value)
    {
          ....
    }
}
```

The ControlPoint::subscribe() returns true when the subscription is accepted from the service, and you can get the subscription id and timeout.

```
ControlPoint ctrlPoint = .....
Device clockDev = ctrlPoint.getDevice("xxxx-clock");
Service timeService = clockDev.getService("time:1");
boolean subRet = ctrlPoint.subscribe(timeService);
if (subRet == true) {
   String sid = timeService.getSID();
   long timeout = timeService.getTimeout();
}
```

5 IPv6

CyberLink binds all interfaces in the platform when the devices or control points are created, and the IPv6 sockets are created automatically if the interfaces have IPv6 addresses.

CyberLink supports IPv4 and IPv6 both as default. If you want to use only IPv6 interfaces, call the following method before the devices or control points are created.

```
UPnP.setEnable(UPnP. USE_ONLY_IPV6_ADDR)
```

Link local is the default scope for multicasting of CyberLink. Use UPnP.setEnable() to change the scope. The following example changes the scope to the site local.

```
UPnP.setEnable(UPnP. USE_IPV6_SITE_LOCAL_SCOPE)
```

6 Inside CyberLink

6.1 Overriding HTTP Service

The Device class of CyberLink implements a HTTPRequestListner interface of org.cybergarage.http package to handle some HTTP requests from the control points. The HTTPRequestListener interface is bellow.

```
public interface HTTPRequestListener
{
    public void httpRequestRecieved(HTTPRequest httpReq);
}
```

To overide the interface, import the org.cybergarage.http and override the httpRequestRecieved method in your device that is a sub class of the Device class. The following example is a clock device using CyberLink, and adds the override method to return the presentation page.

```
import org.cybergarage.http.*;
.....
public class ClockDevice extends Device implements ActionListener, QueryListener
{
    .....
    private final static String PRESENTATION_URI = "/presentation";
    public void httpRequestRecieved(HTTPRequest httpReq)
    {
            String uri = httpReq.getURI();
            if (uri.startsWith(PRESENTATION_URI) == false) {
                    super.httpRequestRecieved(httpReq);
                    return;
            }
            Clock clock = Clock.getInstance();
            String contents = "<HTML><BODY><H1>" + clock.toString() + "</H1></BODY></HTML>";
            HTTPResponse httpRes = new HTTPResponse();
            httpRes.setStatusCode(HTTPStatus.OK);
            httpRes.setContent(contents);
            httpReq.post(httpRes);
    }
}
```

6.2 Using your XML Parser

CyberLink can support various XML parsers to read device discriptions or execute actions of UPnP because it has an abstract class for XML parsering, org.cybergarage.xml.Parser, to change various parsers easiliy. The abstract class is bellow.

```
package org.cybergarage.xml;
public abstract class Parser
{
    public abstract Node parse(InputStream inStream) throws ParserException;
}
```

Using the interface, CyberLink supports the following pasers, Apache Xerces and kXML.

Parser	Class
Apache Xerces	org.cybergarage.xml.parser.XercesParser
kXML2	org.cybergarage.xml.parser.kXML2Parser

To change the parsers, use UPnP::setXMLParser(). The following sample shows to use kXML as the XML parser. CyberLink set Apache Xerces as the default parser in UPnP::static().

```
UPnP::setXMLParser(new org.cygergarage.xml.parser.kXML2Parser());
```

7 Transitioning From Version 1.2

7.1 QueryListner

CyberLink v1.3 has changed the QueryListner interface to return user error infomation and set the listener to the StateVariable instance instead of the Service instance. The difference is bellow.

Version	Interface
1.2	String queryControlReceived(Service service, String varName)
1.3	bool queryControlReceived(StateVariable stateVar)

Use StateVariable::getName() to know the variable name, and use StateVariable::setValue() to return the result value. The following sample shows the difference between v1.2 and v1.3.

```
v1.2:
String queryControlReceived(Service service, String varName ) {
        return Clock.GetCurrentTimeString();
}

v1.3:
bool queryControlReceived(StateVariable stateVar) {
        const char varName = stateVar.getName();
        stateVar.setValue(Clock.GetCurrentTimeString());
        return true;
}
```

To set the query listener, use StateVariable:: setQueryListener() instead of Service::setQueryListner(). However, Service::setQueryListner() that sets the specifed listener to all state variables in the service is not deprecated to ensure the compatibility. The implematation is bellow.

```
void Service::setQueryListener(QueryListener listener)
{
        ServiceStateTable stateTable = getServiceStateTable();
        for (int n=0; n< stateTable.size(); n++)
                StateVariable var = stateTable.getStateVariable(n);
                var.setQueryListener(listener);
        }
}
```

8 License