ISY Developer's Manual

Insteon JSDK (Java SDK)

TABLE OF CONTENTS

1.0 Introduction

ISY is a sophisticated events based network platform which affords its clients unprecedented levels of integration and functionality. ISY, at its core, is based on UPnP standards and, as such, most of its capabilities are also immediately available to UPnP as well as other Web Services clients such as WSD/DPWS (Web Services on Devices and Device Profiles for Web Services).

At a high level, ISY operates and may be communicated with in the following order:

1. **Upon power up, ISY sends out broadcasts messages of its location to all the UPnP and WSD clients on the network**

2. **Interested clients may choose to:**

    a. **Search for a specific ISY (based on the device type it supports such as Insteon)**
    b. **Listen in for ISY generated announcements on the network**
    c. **Immediately start communicating with ISY using a predefined IP (static) address and port, if one is already known**

3. **Upon discovery of an ISY – regardless of the method chosen – the clients would then have to do perform the following in the exact order:**

    a. **Get Supported Algorithms and Protocols (in case ISY is security enabled) and choose an algorithm which best suits them**
    b. **Get a Life Time Sequence Base number to reduce play-back attacks (in case ISY is security enabled)**
    c. **Create a session which, in case of security enabled ISY, might include negotiating public keys, bulk keys, and signature digests**
    d. **Authenticate themselves to ISY using UPnP Security 1.0**
    e. **Subscribe to the ISY events from which time ISY continuously notifies the clients of the changes in its state. Upon successful subscription, ISY publishes all its current states to the client so that the client and ISY are in synch at the moment of subscription. In this respect, then, the clients are started with the current state of ISY and are notified of all the changes as they occur and thus will never have to poll ISY**
    f. **Issuing Web Services to effect a change in ISY. In case of security enabled ISY, all Web Services requests have to be at the minimum signed (using ISY's signature key) and fully encrypted (using ISY's bulk key) in case of sensitive services such as authentication and reboot**

4. **During application exit, the client must notify ISY that it wishes to terminate its session. This is achieved by issuing:**

    a. **Expiring the session (in case ISY is security enabled)**
    b. **Unsubscribing from ISY**

5. **During normal operations, the client *must* always respond back (immediately) with an Ack to ISY's Heartbeat events otherwise ISY assumes a client malfunction (the client didn't exit gracefully as outlined in step 4) and terminates the associated session**

Since adhering to and implementing a Web Services client is sometimes a daunting task, ISY's JSDK wraps all the complexity into a developer friendly library which enables developers to implement ISY clients in a matter of hours. JSDK implements and extends all the functionalities contained within ISY and, as such, developers have immediate access to *all* the same features as currently available on ISY's default Java applet.

As mentioned before, ISY is event driven and thus every change in ISY is notified/published to all the ISY subscribed clients in real-time and almost immediately. In this respect, then, one could use the default ISY User Interface (a signed Java applet) to effect a change while using one's own client to view all the changes that are taking place (and vice versa).

2.0 ISY Building Blocks

ISY's stack, like any other good software platform, is layered such that each layer only knows about the interfaces of the next layer in the stack and thus changes in one layer do not impact any other layer. ISY has the following layers

1. ***Network*** **– HTTP, UPnP, and Web Services**
2. ***Model*** **– The logical representation of ISY**
3. ***Device*** **– The abstraction of a physical device which relieves the higher layers from having to know the specifics of any device's communications protocol (such as the Insteon Protcol)**
4. ***Protocol*** **– Takes care of the actual details of communicating on a physical (or wireless) medium (such as RS232/422/485/IR/ZWave/etc.)**

There is also an *Administrative* layer which can access all the other layers in the stack and control their functions and behaviors while providing other functionalities such as Logging, Time Management/Scheduling, D2D (Triggers), etc.

Since this manual is geared towards ISY Java client developers, the remainder of the contents herein shall focus on Layer 2 – the Model. All of Layer 1's functionality and specifics are handled within the JSDK, and therefore, shall not be discussed unless absolutely necessary. Similarly, Layers 3, 4, and the Administrative layer are handled within ISY's firmware and shall not be discussed.

2.1 Layer 2 – ISY Model

ISY Model comprises just a few but very important building blocks the understanding of which makes it easier to extend JSDK's functionality for even more powerful applications and integration scenarios. Herein under is a list of the most important building blocks:

2.1.1 Control (com.universaldevices.device.model.UDControl)

A *Control* is the logical representation of either a state or a function that may be performed on a physical device (or a scene) linked to ISY. For example, "DON" is the name of the *Control* which instructs ISY to turn a "Deivce On" while "ST" is the name of the *Control* which holds that *state* of a device.

In essence, then, Control is what "captures" and "controls" changes in the states of physically linked devices or groups/scenes. Since Controls may be associated with states, thus, all ISY publications (publish) to all clients contain a Control parameter which identifies "what changed".

For example, a CLISP (Climate SetPoint) Control not only allows the client to effect a SetPoint change on a linked Thermostat but also, as soon as the change takes effect (or the state changes), ISY notifies all the clients of the change in "CLISP" and the current value thereto (see section 2.1.2: Action) if any.

The most important attributes of a Control are:

***A Name* – this is the Control's only meaningful unit of communications with ISY such as "CLISP", "DON", "DIM", etc.**

***A Label* – this is an optional label that the developer/manufacturer may ascribe to a Control such as "SetPoint", "On", "Dim", "Fast On", etc.**

***Action*s – this is a list of optional while permissible actions which may be performed on a Control such as "50" which, when applied to "DON", means turn the "Device On to 50%". Or, when "HEAT" is applied to "CLIMD" (Climate Mode) it means change the thermostat "Mode to Heat". For more details, see section 2.1.2: Action.**

2.1.2 Action (com.universaldevices.device.model.UDAction)

An *Action* is the permissible "value" which may be applied to a *Control*. A Control may have a set of permissible Actions which are captured by a list.

When communicating a state change request to ISY, Action may be null. This said, however, when ISY publishes (to its subscribers) changes to a Control – and if the Control is associated with some state – then this attribute holds the "current value" of the state. For example, when issuing a "DON" to ISY the "ST" Control (which is associated with a state) is updated and, as such, ISY shall notify all the subscribed clients of a change in "ST" with Action being the current value of "ST" such as "50"%.

The most important attributes of an Action are:

***A Name* – this is the Action's only meaningful unit of communications with ISY. Depending on the Control, this attribute may take the form of a free text/object field the value of which is filled in by ISY upon publications of events.**

***A Label* – this is an optional label that the developer/manufacturer may ascribe to an Action such as "Heat".**

2.1.3 Node (com.universaldevices.device.model.UDNode)

A Node is a logical representation of a physical device linked to ISY. So, for instance, KeypadLinc's button A is a node and so are its buttons B, C, D through H.

In essence – and when put in the context of a Control and Action – the Node is the only missing piece which, when all put together, enables effecting the desired change on a physical device linked to ISY.

The most important attributes of Node are:

An Address **– this is the address which ISY uses to communicate with the actual physical device such as 4 E 52 1**

A Name **– this is the user friendly name which can be changed by any ISY client**

States (device Variables) **– this is the list of all the Controls for a Node and their current associated Actions (values)**

A note on Insteon addresses:
Since, as mentioned before, every button is also considered a device within ISY, thus, each button shall have its own address conforming to the following syntax:
X X X B – where X is the actual Insteon address for the device in hex and B is the button group number.

For instance, a 6 button KeypadLinc with address 04 E8 52 will have the following nodes within ISY:
4 E8 52 1 – the main [loaded] button
4 E8 52 A – Button A
4 E8 52 B – Button B
4 E8 52 C – Button C
4 E8 52 D – Button D

2.1.4 Group/Scene (com.universaldevices.device.model.UDGroup)

A Group is a specialization of Node with the added capability of aggregating associated/linked Nodes. Just like a Node, a Group may also be used to effect a change in ISY. The only difference is that issuing a state change on a Group results in ISY sending notifications on the states of all the Nodes within that Group/Scene (if there were any changes).

2.1.5 Putting it Together

By having a triplet {control, action, [node or group/scene]} it's quite easy to effect change on the physical devices which are linked/attached to ISY. For instance:

1. To turn on the light at address 7 B0 B2 to 60%, a simple method call of the form *changeNodeState*("DON", "60", "7 B0 B2 1"), is all it takes.

2. To turn off the scene at address 52626, a simple method call of the form *changeGroupState*("DOF", null, "52626"), is all it takes.

ISY-JSDK takes care of all the protocol handling/hand shaking and Web Services calls behind the scenes and, as mentioned before, any changes in ISY are published to all the clients.

2.1.6 ISY Messages and Web Services

UDML™ (Universal Devices Markup Language) and DIML™ (Device Intelligence Markup Language) collectively are XML representations of messages and models which are communicated between ISY and its clients.

The Schemas and WSDLs for these languages shall be published in the ISY Web Services Developers Guide (WS-SDK) in the near future. This said, however, ISY itself makes these documents accessible through the following URLs where http://a.b.c.d:port is ISY's base URL:

3. **http://a.b.c.d:port/WEB/UPNPSVC.XML - is ISY's Web Services interface**
4. **http://a.b.c.d:port/0/h - is ISY's configuration document which describes the characteristics of the ISY as well as those of Controls (see section 2.1.1) and Actions (see section 2.1.2) and any relationship thereto**
5. **http://a.b.c.d:port/WEB/NODESCNF.XML - is ISY's Node/Group (see section 2.1.3 and section 2.1.4) configuration document**

Apart from the aforementioned URLs, the developer is encouraged to "walk" through ISY's client library and inspect these structures by simply invoking the "toDIML" method on the relevant objects.

3.0 ISY-JSDK for Insteon

ISY-JSDK contains all the necessary code to implement fully functional ISY-Insteon clients. In order to make this possible, ISY implements an abstract class, named *ISYInsteonClient* (com.udi.isy.jsdk.insteon), which is a façade to all methods and attributes one needs to develop applications that communicate, monitor, and even administer ISY for Insteon devices. Starting with this class, alongside a fully functional example (example.*MyISYInsteonClient*) and a complete set of Javadocs for all the ISY-JSDK classes have proven sufficient for designing and developing ISY clients in a matter of hours.

3.1 ISY-JSDK – The Building Blocks

Although utilizing ISYInsteonClient is, in and of itself, sufficient to develop quite sophisticated ISY client applications/applets for Insteon, however – and for the sake of completeness - herein under is a list of the most important building blocks/packages in ISY-JSDK and a brief description thereto.

For a complete description and information on all the packages and classes described herein under, please refer to and peruse the accompanying Javadocs.

3.1.1 Package com.universaldevices.common

This package includes common definitions used across the library.

Constants – system wide constants
DeviceTypes – system wide device types (category/subcategory)

3.1.2 Package com.universaldevices.resources.errormessages

This package includes error messages and error handling routines.

Errors –	**implements Error handling routines and messages. Clients interested in receiving library generated errors must implement the ErrorEventListener interface (in the same package) and use Errors.addErrorListener() to register their interest in receiving error notifications**

3.1.3 Package com.universaldevices.device.model

This package has implementations for the ISY Model as described in section 2.1 .

UDControl –	**implements a Control as described in section 2.1.1**
UDAction –	**implements an Action as described in section 2.1.2**
UDNode –	**implements a Node as described in section 2.1.3**
UDGroup –	**implements a Group as described in section 2.1.4 Groups are used as representations of a Scene in Insteon**

IModelChangeListener – this interface, when implemented by subclasses, has all the call-back methods which are invoked by the underlying library to notify the client of any change in ISY

3.1.4 Package com.universaldevices.client

This package includes the core client class.

UDClient – implements the "core" and very basic client functionality. This class is the super-class of all ISY clients

3.1.5 Package com.universaldevices.upnp

This package includes UPnP implementations alongside a logical (proxy) representation of an ISY.

UDProxyDevice – is the proxy to ISY. All direct communications to/from ISY are handled within this class
UDControlPoint – is an implementation of UPnP Control-Point

3.1.6 Package com.universaldevices.scheduler

The classes within this package take care of administering/configuring ISY's sophisticated scheduler.

ScheduleItem – is the representation of one schedule which can be submitted to
Schedules – is a group of schedules

3.1.7 Package com.universaldevices.d2d (Triggers)

The classes within this package take care of administering/configuring ISY's Trigger engine.

D2DElements	**– a Condition or a Response**
D2DSense	**– a Trigger: a group of Condition/Response scenarios**
D2D	**– aggregation of D2DSense (triggers)**

3.1.8 Package com.udi.isy.jsdk

ISYClient – extends UDClient functionality (see section 3.1.2) and wraps (façade pattern) all the pertinent methods in UDProxyDevice and UDControlPoint to give the developer an easier way of developing ISY clients.

$$\begin{tikzcd} \textbf{UDClient} \text{ – core client functionality} \\ \textbf{ISYClient} \text{ – adds ISY communications functionality} \arrow[u] \end{tikzcd}$$

3.1.9 Package com.udi.isy.jsdk.insteon

ISYInsteonClient – extends ISYClient functionality (see section 3.1.6) to encompass Insteon behavior such as scene management.

$$\begin{tikzcd} \textbf{UDClient} \text{ – core client functionality} \\ \textbf{ISYClient} \text{ – adds ISY communications functionality} \arrow[u] \\ \textbf{ISYInsteonClient} \text{ – adds Insteon Behavior} \arrow[u] \end{tikzcd}$$

All ISY Insteon Clients must extend this class.

3.1.10 Package com.udi.insteon.client

This package include constants and utilities which make it easier to handle Insteon devices.

InsteonConstants	**– Insteon specific constants such as the Control and Action names**
InsteonOps	**– utilities for Insteon device management**
SceneProfileAttributes	**– encompasses scene attributes (on-level/ramp-rate) for a node within a scene**

3.2 Building a Simple ISY Client Application

As mentioned before, ISY is based on an events based framework which publishes all the changes within to all the subscribed clients. In order to make it easier to capture all these publications while having a simple interface to communicate with ISY, all ISY Inseton clients must extend the *ISYInsteonClient* (see section 3.1.9) abstract class which:

a. **Has all the necessary methods to communicate with ISY**
b. **Provides simple and intuitive abstract call-back methods which are to be implemented by all the clients**

As it is with every engineering effort, it's of utmost import to design an application based on requirements while applying system constraints based on the tools, software, and system patterns to be utilized. In this respect, then, it's always desirable to have documentation and example code while investigating a technology for its merits and constraints. As such, the following sections describe the step necessary to develop, run, and test an ISY Insteon Client based on the fully functional example which can be found in example.**MyISYInsteonClient.**

3.2.1 Step 1 – Extend *ISYInsteonClient*

The first step in developing an ISY Insteon client is to extend the *ISYInsteonClient* which immediately gives the subclass all that is necessary to effect any change in ISY while being notified of all ISY events.

The code snippet below is taken directly out of the fully functional example which may be found at example.**MyISYInsteonClient which can serve as the tool to investigate and appreciate the ease with which an ISY Insteon client can be developed.**

```
public class MyISYInsteonClient extends ISYInsteonClient{

/**
 * Constructor
 * Registers this class as IModelChangeListener
 *
 * @see IModelChangeListener
 *
 */
public MyISYInsteonClient(){
       super();
}

/**
 * The following methods are all call-back which should be implemented if
 * and only if there's a need to be notified of that specific event
 *
 */

public void onNewDeviceAnnounced(UDProxyDevice device) {
       if (device.securityLevel>UPnPSecurity.NO_SECURITY){
              if (device.isAuthenticated && device.isOnline)
                     return;
              try{
                     //authenticate does all the UPnP Security hand-
                     //shaking, key exchange, etc. and when all is
                     //successful, then it subscribes to ISY events by
                     //calling device.subscribeToEvent()
                     super.authenticate("admin", "admin");
              }catch(NoDeviceException e){
                     System.err.println("This should never happen!");
              }
       }else{
              //just subscribe to events
              device.subscribeToEvents(true);
       }

}


public void onDiscoveringNodes() {
       System.out.println("I am in Linking Mode ...");
}

public void onNodeDiscoveryStopped() {
       System.out.println("I am no longer in Linking mode ...");

}

public void onGroupRemoved(String groupAddress) {
       System.out.println("Scene: "+groupAddress+
       " was removed by someone or something!");

}
public void onGroupRenamed( UDGroup group) {
       System.out.println("Scene: "+group.address+"
                            was renamed to "+group.name);

}
```

```
//This is the method which is called every time there's a change
//in a node, control, and action triplet

public void onModelChanged(UDControl control, Object value, UDNode node)
{
        System.out.println("Someone or something changed "+
        control.label+ " to "+value+" at "+node.name);

}

public void onNetworkRenamed(String newName) {
        System.out.println("Ah, the network was renamed to " + newName);
}

public void onNewGroup(UDGroup newGroup) {
        System.out.println("Yummy: we now have a new scene with address
        "+newGroup.address+" and name "+newGroup.name);
}

public void onNewNode(UDNode newNode) {
        System.out.println("Yummy: we now have a new Insteon device with
        address "+newNode.address+" and name "+newNode.name);

}

public void onNodeError(UDNode node) {
        System.out.println("What's going on? The Insteon device at address
        "+node.address + " and name "+ node.name+" is no longer responding
        to my communication attempts!");

}

public void onNodeRemoved(String nodeAddress) {
        System.out.println("Whooah ... node with address "+nodeAddress+"
        was permanently removed from ISY");

}

public void onNodeRemovedFromGroup(UDNode node, UDGroup group) {
        System.out.println("Insteon device with address "+node.address+"
        and name "+node.name+" is no longer part of the "+group.name+ "
        scene!");

}

public void onNodeRenamed(UDNode node) {
        System.out.println("Insteon device with address "+node.address+"
        was renamed to "+node.name);

}

public void onNodeMovedAsMaster(UDNode node, UDGroup group) {
        System.out.println("Insteon device "+node.name+" is now part of
        the "+group.name+ " scene as a master/controller");

}

public void onNodeMovedAsSlave(UDNode node, UDGroup group) {
        System.out.println("Insteon device "+node.name+" is now part of
        the "+group.name+ " scene as a slave/responder");

}
```

```
public void onDeviceOffLine() {
       System.out.println("oo; ISY is offLine. Did you unplug it?");

}

public void onDeviceOnLine() {
       System.out.println("Hooray: ISY is on line ...");

}

public void onSchedulesUpdated() {
       System.out.println("Some of the schedules have been updated;
       get the lastest schedules");

}

public void onSystemStatus( boolean busy) {
       if (busy)
              System.out.println("I am busy now; please give me some
              reprieve and don't ask me for more!");
       else
              System.out.println("I am ready and at your service");
}

public void onInternetAccessDisabled() {
       System.out.println("You can no longer reach me through the
       internet");

}

public void onInternetAccessEnabled(String url) {
       System.out.println("I can now be accessed on the internet at
       "+url);

}

}
```

3.2.2 Step 2 – Start the Client

There are two ways to start the client (where myISY refers to an instance of MyISYInsteonClient)**:**

a. **If the computer running the client is UPnP enabled, the simple invocation of the start() method on MyISYInsteonClient is all that's necessary:**
   myISY.start();

b. **If UPnP is not available or if there are firewall issues, the combination of ISY's IP address/Port and its UUID are required to start communicating with ISY:**
   myISY.start("uuid:00:03:f4:02:66:e0","http://192.168.0.131:16872/");

During this step it's also advised to implement and register an *ErrorEventListener* (see section 2.2.1) which is going to be notified of all the errors as they occur either within the library itself or generated by ISY. All implementers of the ErrorEventListener interface which are registered with the Errors class are notified of the error's code and, as such, they could conditionally decide what course of action should be taken next. The following is an implementation of ErrorEventListener which simply prints out the error code and any text messages associated with it. This code can be found at example.**MyISYErrorHandler**

```
public class MyISYErrorHandler implements ErrorEventListener{

         */
        public boolean errorOccured(int status, String msg, Object object)
        {
               System.err.println("Ooops, what went wrong? "+status + " "
               +msg==null?" ":msg);
               return false; //o, I don't have a GUIErrorHandler
        installed, so don't try to display it
        }

}
```

Now all that needs to be done is to make an application, instantiate MyISYInsteonClient, register our error handler, and start the client.

The following code snippet is taken directly from example.**MyISYInsteonClientApp and it depicts a functional ISY Insteon client which, upon start and authentication, notifies the application of all the changes as they take place in ISY:**

```
public class MyISYInsteonClientApp  {

        private static MyISYInsteonClient myISY = null;

        public static void main(String[] args) {
               myISY=new MyISYInsteonClient();
               Errors.addErrorListener(new MyISYErrorHandler());
               myISY.start(); //start the client

        /**
         * If you have firewall issues, you can directly access
         * ISY using it's uuid ( see admin/config guide) and
         * its base url (http://x.y.z.w:port) as follows
               try{
myISY.start("uuid:00:03:f4:02:66:e0","http://192.168.0.131:16872/");
               }catch(Exception e){
                      System.err.println("BAD ISY URL\n");
               }
        */

}
```

The above example simply notifies the application of changes in ISY as they happen. In order to effect a change in ISY (such as turning devices on/off, creating scenes, adding schedules/triggers), all that's necessary is to invoke the relevant methods on the one and only instance of MyISYInsteonClient.

MyISYInsteonClientApp, as packaged in the library, implements a command prompt sample set of behaviors which can be performed on ISY. As such, walking through the code as well as the Javadocs is the best possible way for the user to get the complete picture of how things are tied together.

In short, there are only two steps involved in order to create a functional ISY client: a) extend ISYInsteonClient b) start the client inside an application.

4.0 Insteon Specific Packages

As it may have already been deciphered, ISY is based on quite a generic framework which affords it to be extended for almost any type of device, protocol, and communications medium without major development effort. In this respect, thus, this section is dedicated to Insteon specific functionality and behavior as exhibited through ISY and its clients for Insteon. There are only two Insteon specific packages in the whole library, namely:

1. **Package com.udi.isy.jsdk.insteon**
2. **Package com.udi.insteon.client**

Please peruse the accompanying Javadocs for more information as they have complete information about these packages, all the classes contained therein, and all the methods and attribute for each class.