Control Interface Tutorial

The ControlInterface class provides developers with the fundamentals methods to create an interfacing scheme. By extending this class developers have access to the following methods:

- navNext() – navigates to the next content node on the screen
- navPrev() - navigates to the previous content node on the screen
- selectCurrent() – clicks on the current focused node
- clickNode(description) – clicks the node with the correspondent description
- focusIndex(index) – focus the node
- highlighIndex(index) – highlights the content index
- clearHighlights() – clears all highlights
- home() – performs the global action go home
- back() – performs the global action back

Through these methods developers can easily create and manipulate navigation system wide.

To illustrate how to use it, in this tutorial we will develop a simple control interface that will take advantage of this class and the IO Receiver mentioned in another tutorial

TouchController

We will create a simple control interface that will select a node when we tap the screen and will go to the next node if we slide.

To do so we create a class TouchController that extends ControlInterface and implements IOReceiver.

```
public class TouchController extends ControlInterface implements IOReceiver {
```

We want this the control interface to be initialised when the SWAT ends calibration procedure and we want to ensure only one control interface is initialise. To ensure that this is the selected control interface we first must add it to the system preference. To do so:

1. In the res/values/arrays.xml we add an item to the “mode” and “modeValues” array. We will add the description of the control interface “Simple Navigation” and the value “touchController”.

```
<string-array name="mode">
    <item name="1">None</item>
    <item name="2">Simple Navigation</item>
</string-array>
<string-array name="modeValues">
    <item name="1">null</item>
    <item name="2">touchController</item>
</string-array>
```

2. Now we added the simple navigation to the configuration of SWAT. As such we can go to the configuration of the service and set the active control interface.

Now we have a control interface that can be initialised when SWAT finish calibrating, to do so we simply implement the onReceive from the BroadCastReceiver and ensure that our interface is the select one by checking the intent extra.

```
@Override
public void onReceive(Context context, Intent intent) {

    // Triggered when the service starts
    if (intent.getAction().equals("mswat_init")
            && intent.getExtras().get("controller")
                    .equals("touchController")) {
```

After this we are ready to create our control interface.

1. We start by registering the IO Receiver;
2. Since we are going to react to touch we need to monitor and recognize the touches. Has such we start monitoring and get the active Touch Pattern Recognizer (TPR – this can be select in the service configuration according to your touch screen driver);
3. We also need to block the system normal reactions to touches;
4. We will activate the auto highlight to get visual feedback of the focus;

```
// initialise touch pattern Recogniser
tpr = CoreController.getActiveTPR();

// register ioReceiver
registerIOReceiver();

// starts monitoring touchscreen
int deviceIndex = CoreController.monitorTouch();

// blocks the touch screen
CoreController.commandIO(CoreController.SET_BLOCK, deviceIndex,
        true);

// sets automatic highlighting
 CoreController.setAutoHighlight(true);
```

All that is left is to recognize the touches and react accordingly. In the onUpdateIO we call the TPR and wait until one type of touch is recognized. Since we only want to recognize slides and taps we will use the method identifyOnRelease.

```
@Override
public void onUpdateIO(int device, int type, int code, int value,
        int timestamp) {
    if(tpr==null){
        tpr=CoreController.getActiveTPR();
    }
    int touchType;
    if ((touchType = tpr.identifyOnRelease(type, code, value, timestamp)) != -1) {
        Log.d(LT, "Touch Type" + touchType );
        handleTouch(touchType);
    }
}
```

When a type of touch is recognize we will send it to handleTouch(type). This method uses the most basic functions of the Control Interface (i.e. navNext(), selectCurrent()).

```
private void handleTouch(int touchType){
    switch (touchType) {
    case TouchRecognizer.SLIDE:
        navNext();
        break;
    case TouchRecognizer.TOUCHED:
        selectCurrent();
        break;
    case TouchRecognizer.LONGPRESS: // Stops service
        CoreController.stopService();
    }
}
```

As a final touch in order to support the use of this control interface through the WifiControl we will also implement the onTouchReceived(type) by simply calling the handleTouch(type).

```
@Override
public void onTouchReceived(int type) {
    handleTouch(type);

}
```