ICL Low Level Context Server Demo Harness

This document describes about the concepts used inside the ICL server, which are designed for extendibility and plug-and-play recognizer.

# **DeviceType**

A DeviceType refers to multiple types of data, which are sent from one device. For example, smartphone may send accelerometer, gyroscope, magnetic sensor data, geolocation, user voice and facial image data. Currently in the ICL all these raw data are included in one DeviceType, ***Smartphone***, since they are sent from one device, smartphone.

All the DeviceType must be registered in org**.**uclab**.**mm**.**icl**.**llc**.**config.DeviceType enum, and there must be actual data class which is used in the internal process of the ICL. Each of the actual data class, for a registered DeviceType, must inherit parent class org**.**uclab**.**mm**.**icl**.**data**.**SensorData

## **DeviceType in current ICL**

Currently ICL has two DeviceTypes.

### ***Smartphone***

***Smartphone*** DeviceTypes is composed of the raw sensor data from Smartphone. It is important to note that, since smartphone and smartwatch communicates via Bluetooth communication, and smartwatch cannot send the data itself to the server, the data from smartwatch are also included in this DeviceType.

The list of raw data is as follows. If there is no description where the data is coming from, it is coming from smartphone.

* Accelerometer sensor data, 3 axes, 150 items for 3 seconds, from smartphone and smartwatch
* Gyroscope sensor data, 3 axes, 150 items for 3 seconds, from smartphone and smartwatch
* Magnetic sensor data, 3 axes, 150 items for 3 seconds, from smartphone and smartwatch
* ECG sensor data, 3 axes, 150 items for 3 seconds, from smartphone and smartwatch
* User voice data, collected during the call, for 3 second.
* User facial image data, collected from front camera.
* Food photo data, which is taken by user manually
* Food tag data, which is input by user, in the list of predefined food categories

### ***Kinect***

***Kinect***, DeviceType is composed of the skeleton point data, sent from Kinect. A single timeframe consists of 25 skeleton points of human boday, and each point is composed of 3 floating point values, for x, y and z axis.

## **DeviceType enum**

Registering a DeviceType is done by adding a new enum on org**.**uclab**.**mm**.**icl**.**llc**.**config.DeviceType. Current enum status is as follows

***SmartPhone***(1), ***Kinect***(2)

Note that, the order and value for each enum type is crucial. The number (and corresponding order) corresponds to the number assigned to sensors in Data Curation Layer, so if you are going to modify it, you should do it carefully.

The methods you should implement when adding a new enum is as follows

1. **public** Data parseSensorData(Object obj)

This function designates the way how data sent from DCL should be converted into the data class used in ICL. It is sent basically from the sensor device, so you should parse them based on the serialization method used in the sensor device.

**@param** obj : Data contents, sent from DCL

**@return** data : Data class defined for each enum, which inherits the abstract class Data

## **Abstract SensorData Class**

For each registered DeviceType in org**.**uclab**.**mm**.**icl**.**llc**.**config.DeviceType, there must be a instantiable data class, which inherits org**.**uclab**.**mm**.**icl**.**llc**.**data.SensorData class.

The abstract methods which you should implement when inheriting from org**.**uclab**.**mm**.**icl**.**llc**.**data.SensorData interface is as follows.

1. **public** **abstract** Long getUserID()

Getter function for the userID attribute

**@return** User id, which is stored in DeviceType instance

1. **public** **abstract** void setUserID(long userID)

Setter function for the userID attribute

**@param** userID: User id to set on the DeviceType instance

1. **public** **abstract** Long getTimeStamp()

Getter function for the timestamp attribute.

**@return** timestamp, which is stored in DeviceType instance

1. **public** **abstract** void setTimeStamp(String timestamp)

setter function for the timestamp attribute.

**@param** TimeStamp: timestamp to set on the DeviceType instance

# **ContextType**

A ContextType refers to a context of a user. One example in current ICL is ***Activity***, which refers to the physical activity of a user, such as Walking, Running, and Sitting.

All of the ContextTypes must be registered in org**.**uclab**.**mm**.**icl**.**llc**.**config.ContextType enum. Every ContextType is instantiated by org**.**uclab**.**mm**.**icl**.**llc.LLCManager.ContextLabel Class, and each instance of the org**.**uclab**.**mm**.**icl**.**llc.LLCManager.ContextLabel class has a local ContextType variable, indicating which ContextType it belongs to. Output of every recognizer is one of the ContextType, as an instance of ContextLabel class.

## **ContextType in current ICL**

Currently ICL has 3 context types.

### ***Activity***

***Activity*** ContextType refers to the physical activity of the user, such as walking, Running, and so on. Currently they are recognized from recognizers ***ARSang*** and ***VARThien.*** The list of labels are as follows: **Eating, Running, Sitting, Standing, Walking, Stretching, Sweeping, LyingDown, NoActivity**.

### ***Emotion***

***Emotion*** ContextType refers to the emotional context of the user, such as being angry or sad. Currently they are recognized from recognizers ***EmotionJH*** and ***EmotionVUI.*** The list of labels are as follows: **Angry, Hapiness, Neutral, Sadness, NoEmotion.**

### ***Location***

***Location*** ContextType refers to the physical activity of the user, such as walking, Running, and so on. Currently they are recognized from recognizer ***LocSan.*** The list of labels are as follows: **Home, Office, Mall, GYM, NoLocation.**

## **ContextType enum**

All of the ContextType active on the ICL is registered in org**.**uclab**.**mm**.**icl**.**llc**.**config.ContextType. Current enum status is as follows

***Activity***(0), ***Emotion***(1), ***Location***(2)

The order and the value of the above enum is not as much crucial as in the case of DeviceType. However, as the name of the enum is used as an identifier in the ICL, for example in in configuration file, you should define the name carefully and should modify it with consideration.

The methods you should implement when adding a new enum is as follows

1. **public** **int** getDefaultWindowSize()

Returns the default window size of the context, which is a period for the unification of the recognized context from recognizers. If the window size is not designated in the iclconfig.json, ICL will use this windowsize as a default.

**@return** int: default window size of the context.

1. **public** **String[]** getLabels()

Returns the label of the current context. The label of the context should be defined in the enum class, and the last label should be “No(ContextName)”, which represents the situation when there is no corresponding context recognized from the all active recognizers in the ICL.

**@return** string[]: label list of the corresponding context, which haing last elements as No(ContextName).

## **Instantiable ContextLabel Class**

The org**.**uclab**.**mm**.**icl**.**llc**.**LLCManager.ContextLabel class is the only instantiable class for the ContextType. On the constructor, it receives ContextType enum as a parameter, and with this local variable we are able to distinguish which context the corresponding ContextLabel class belongs to. All of the ContextLabel should be initialized with userID, Label, TimeStamp and ContextType, and it provides getter and setter methods for all of the local data it containts.

# RecognizerType

A RecognizerType refers to a recognizer of ICL which takes DeviceType as an input, and produces one instance of ContextType. One example in current ICL is ***IAR***, which stans for Inertial Activity Recognizer. It takes DeviceType ***SmartPhone*** as an input, consumes the inertial sensor data (Accelerometer, Gyroscope sensor data) inside the ***SmartPhone*** data instance, and produces ContextType ***Activity***.

All of the RecognizerType must be registered in org**.**uclab**.**mm**.**icl**.**llc**.**config.RecognizerType enum. Every RecognizerType is instantiated by its own class, which must inherit org**.**uclab**.**mm**.**icl**.**llc.LLCRecognizer.LLCRecognizer superclass. In the org**.**uclab**.**mm**.**icl**.**llc.LLCRecognizer.LLCRecognizer class, there is various abstract methods which must be implemented, and with those abstract methods ICL recognizes the ContextType user context with DeviceType input data, by iterating through all recognizers registered in org**.**uclab**.**mm**.**icl**.**llc**.**config.RecognizerType.

## **RecognizerType in current ICL**

Currently ICL has 5 different recognizers.

### ***SER***

***SER*** stands for Segmented Emotion Recognizer. It takes DeviceType ***SmartPhone*** as an input, consumes audio sensory data of user voice, and produces ContextType ***Emotion*** as an output, using support vector machine, Fourier Transformation and other techniques. Notable thing is that, this recognizer can determine whether the audio data in ***SmartPhone*** instance is human voice or silence, and it only stores the human voice to get better accuracy in classification. This is the reason why the term “Segmented” is attached.

### ***ER***

***ER*** stands for Emotion Recognizer. It takes DeviceType ***SmartPhone*** as an input, consumes audio sensory data of user voice, and produces ContextType ***Emotion*** as an output, using support vector machine and other techniques.

### ***IAR***

***IAR*** stands for Inertial Activity Recognizer. It takes DeviceType ***SmartPhone*** as an input, consumes Inertial data of smartphone and smartwatch including 3-axis accelerometer and gyroscope data, and produces ContextType ***Activity*** as an output, using decision fusion technique, support vector machine and other techniques. Note that this recognizer uses Weka Library for the implementation of SVM.

### ***VAR***

***IAR*** stands for Video Activity Recognizer. It takes DeviceType ***Kinect*** as an input, consumes 25 joint coordinate data in 3-dimension which is recognized in Kinect via video, and produces ContextType ***Activity*** as an output, using Sequential Minimal Optimization (SMO) trained SVM. Note that this recognizer uses Weka Library for the implementation of SMO.

### ***LR***

***LR*** stands for Location Recognizer. It takes DeviceType ***Smartphone*** as an input, consumes geolocation longitude and latitude coordinate which is recognized in smartphone using GPS, and produces ContextType ***Location*** as an output,

## **RecognizerType enum**

All of the ContextType active on the ICL is registered in org**.**uclab**.**mm**.**icl**.**llc**.**config.RecognizerType. Current enum status is as follows

***SER***(0), ***ER***(1) , ***IAR***(2) , ***VAR***(3) , ***LR***(4)

The order and the value of the above enum is not as much crucial as in the case of DeviceType. However, as the name of the enum is used as an identifier in the ICL, for example in configuration file. You should define the name carefully and should modify it with consideration.

The methods you should implement when adding a new enum is as follows

1. **public** **int** getValue()

This function returns the number assigned to current enum. For example, in ***SER***(0) instance it returns 0.

**@return** integer value assigned to current enum

1. **public** LLCRecognizer getRecognizer(**long** userID)

This function Returns the instance of recognizer corresponding to the current enum value, as a instance of subclass inheriting LLCRecognizer, with proper initialization step.

**@param** userID: user ID to assign on the recognizer.

**@return** RecognizerType: recognizer instance, corresponding to current enum.

1. **public** ContextType getContext()

Every recognizer produces one ContextType as an output. This function tells us which ContextType that recognizer produces, which is corresponding to current enum value.

**@return** ContextType: ContextType that the recognizer corresponding to current enum value produces,

1. **public** **Boolean** isOn()

Based on the configuration in the external configuration file iclconfig.json, each recognizer could be turned on or off. This functions tells that the recognizer corresponding to current enum value is turned on or not.

**@return** Boolean: Designate whether the recognizer corresponding to current enum value is turned on or not

## **LLCRecognizer Interface**

The org**.**uclab**.**mm**.**icl**.**llc**.**LLCRecognizer.LLCRecognizer interface is the superclass of every recognizer running on the ICL. It includes abstract methods, which are called inside the ICL to recognize a ContextType user context from the DeviceType data. Therefore, if anyone want to add a new recognizer, one must register the recognizer in the RecognizerType enum, and inherit the interface and properly implement the abstract methods in the interface.

The abstract methods which you should implement when inheriting from org**.**uclab**.**mm**.**icl**.**llc**.**LLCRecognizer.LLCRecognizer interface is as follows.

1. **public** **abstract** **void** ConsumeData(DeviceType it, SensorData obj)

For each registered recognizer in the RecognizerType enum, This function is called every time new DeviceType data arrives. Therefore, if you want to use some data in specific DeviceType, you should write a code to store the sensor data in your recognizer in this method.

**@param** it: DeviceType of the current data

**@param** obj: Object (inheriting SensorData) of DeviceType instance

1. **public** **abstract** RecognizerType getType()

Every recognizer must be registered in RecognizerType. This method tells which RecognizerType enum that the current instance is registered

**@return** RecognizerType: enum instance that current instance is registered

1. **public** **abstract** ContextLabel recognize(**final** InputDataType input, String timeStamp)

This function recognizes the ContextType context, from the data which is gathered from ConsumeData function.

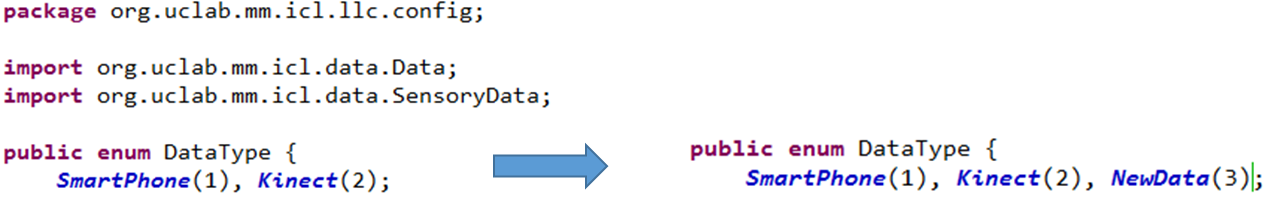
**@param** input: input raw data to recognize context

**@param** timeStamp: timestamp to assign in the context

**@return** ContextLabel: recognized context from input

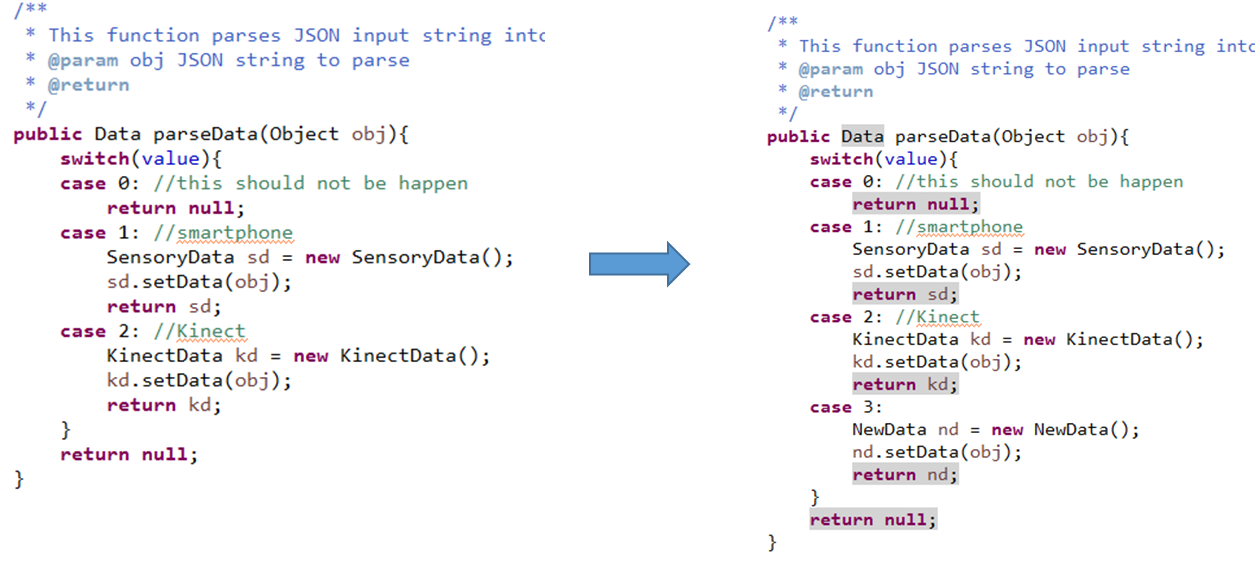
# Tutorial 1. Adding new DeviceType

First of all, you should register your new DeviceType in org**.**uclab**.**mm**.**icl**.**llc**.**config.DeviceType enum.



And modify the methods defined in the enum. The list of methods are specified not only in the enum class, but also in [here](#_DataType_enum)

The example of modifying parseSensorData function is as follows



You can see that case 3 for the NewData enum is added. Note that, NewData class should inherit org**.**uclab**.**mm**.**icl**.**llc**.**data.Data class, and the process is described below.

1. At the same time, you should define an instantiable data class for newly introduced DeviceType. New class should inherit org**.**uclab**.**mm**.**icl**.**llc**.**data.SensorData class, and the list of abstract methods in the class are specified not only in the class itself, but also [here](#_Abstract_Data_Class). There is no limitation of the location of new data class, but it is strongly recommended to place them on the same package of parent data class.



1. Now you have successfully added a new DeviceType into ICL server. Now whenever your datatype is sent from Data Curation Layer, (which means that, sensor ID which matches to your enum value, 3 for example for the NewData, is sent from DCL) the data will be parse by the parseData() function you have implemented, and ultimately consumed by each recognizers.

# Tutorial 2. Adding a new ContextType, and a RecognizerType which recognizes the ContextType.

1. Firstly, we are going to make a new context PhoneDirection which tells us that whether the phone is placed upwards or backwards.

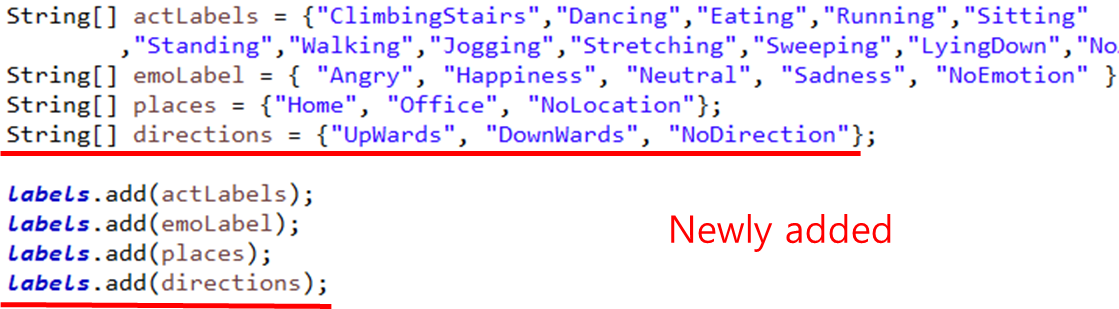
 

Smartphone placed upwards Smartphone placed backwards

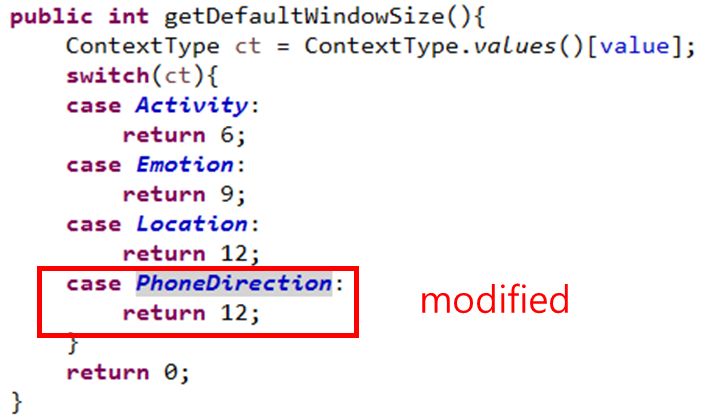
1. You should add new ContextType in org**.**uclab**.**mm**.**icl**.**llc**.**config.ContextType enum,



1. add labels,

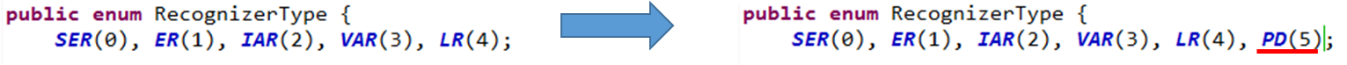


1. and finaly, modify methods in ContextType enum.



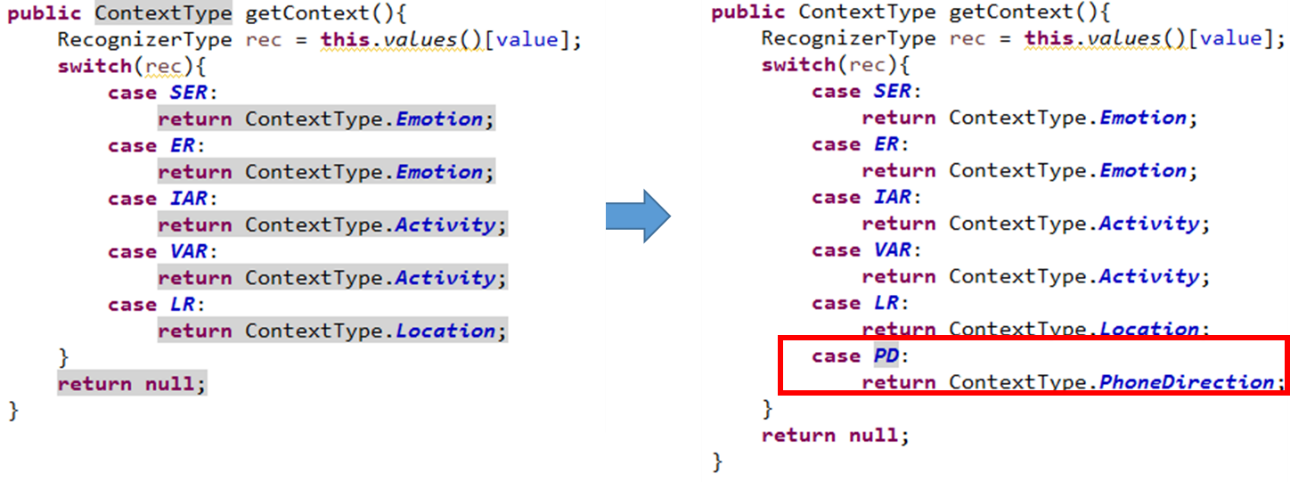
Now you have successfully added a new ContextType. When we add a new recognizer which recognizes the phone direction, ICL now will gather every recognized PhoneDirection Label in every 12 seconds, unify the labels and produce the representative label for that 12seconds based on the unification method (currently it is using weighted majority voting). On the RecognizerType tutorial, we will add a new recognizer PhoneDirectionRecognizer, which consumes the DeviceType SmartPhone and produces PhoneDirection context.

1. Now, we are going to add PhoneDirectionRecognizer which will run on the ICL to decide whether the phone is lied upwards or backwards, with the SmartPhone data sent from smartphone device.
2. Firstly we need to register our PhoneDirectionRecognizer into org**.**uclab**.**mm**.**icl**.**llc**.**config.RecognizerType .

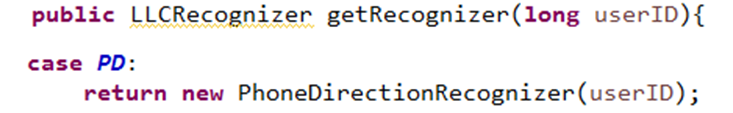
  
where PD stands for Phone Direction.

Also, we need to modify the methods in org**.**uclab**.**mm**.**icl**.**llc**.**config.RecognizerType.

As we said, our PhoneDirectionRecognizer will produce ContextType. PhoneDirection, so we will specify it on the code.



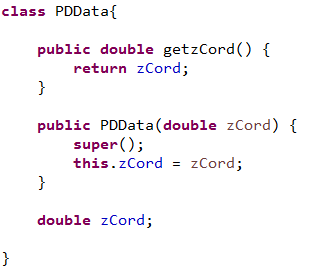
and we need to add PhoneDirectionRecognizer on the getRecognizer function.



Now, we need to implement PhoneDirectionRecognizer, extending the org**.**uclab**.**mm**.**icl**.**llc.LLCRecognizer.LLCRecognizer interface.

1. PhoneDirectionRecognizer class

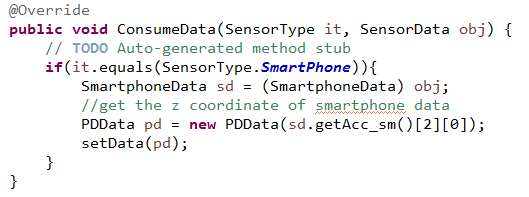
Firstly, we need to implement data class for the PhoneDirectionRecognizer



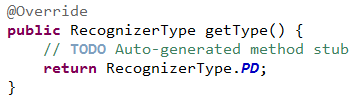
And extend the LLCRecognizer class.



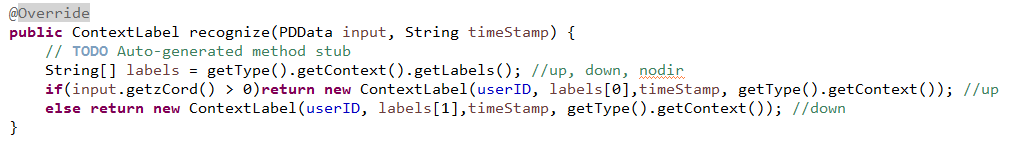
1. we need to implement abstract method ConsumeData, to get the data from the smartphone



1. we also need to implement another abstract method, getType. Since our PhoneDirectionRecognizer is registered in the RecognizerType as PD, we write the following code.

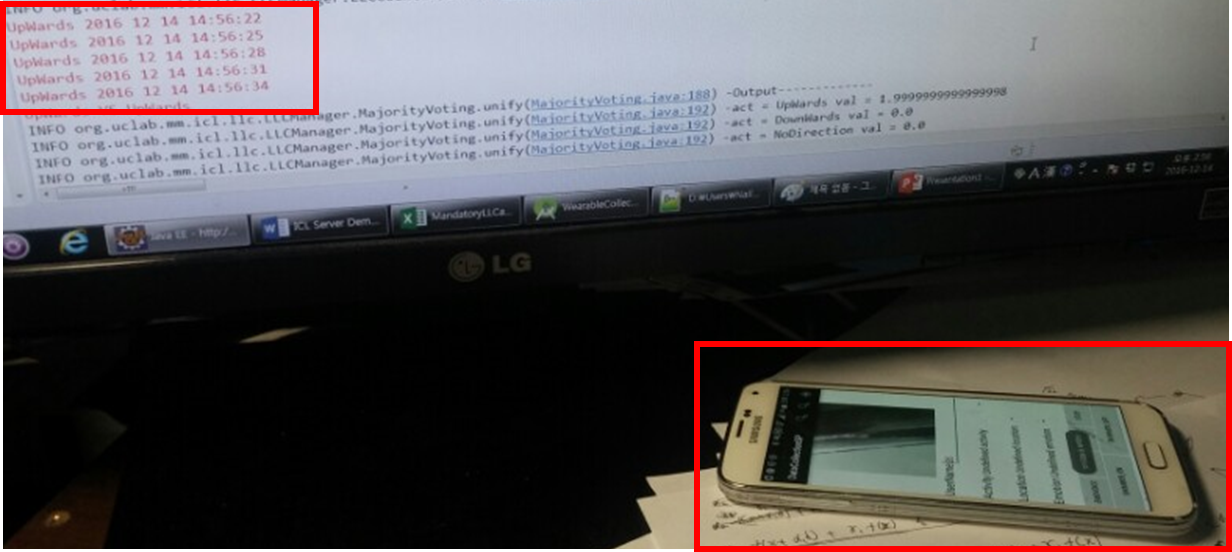


1. Finally, we implement abstract method recognize, to specify the way we recognize the direction of lying Phone from the PDDate data class we defined above.

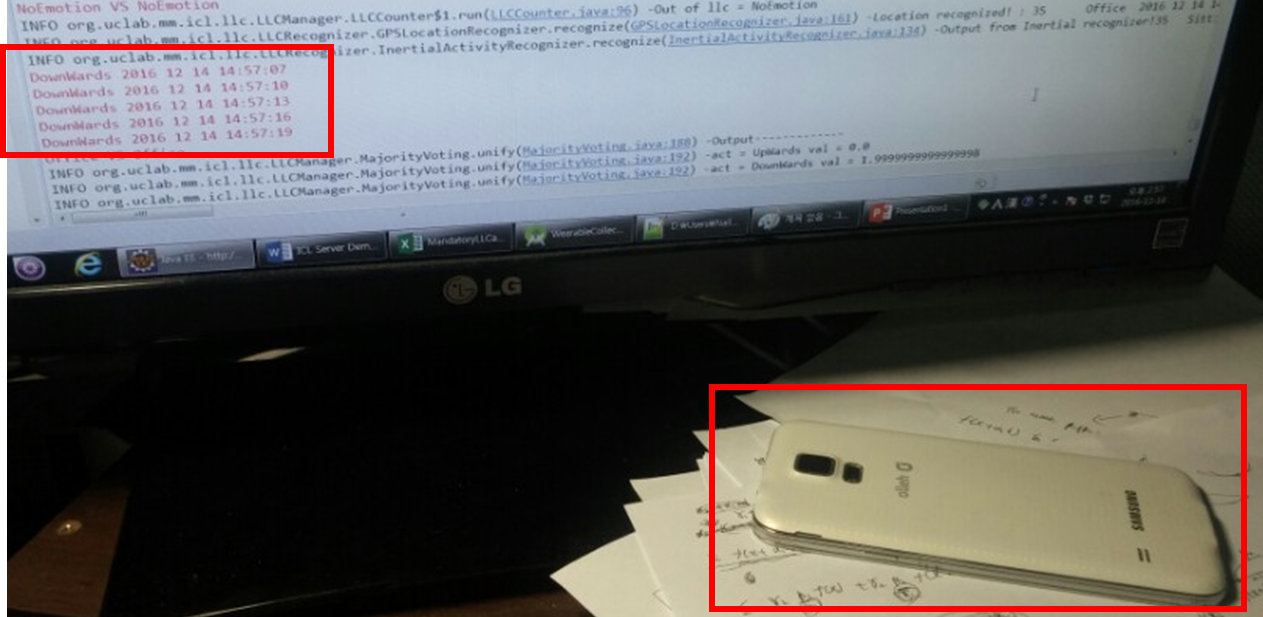


We have successfully added the PhoneDirectionRecognizer. To see that it is really working, let us print the labels we have before the unification.

1. Upward Case



1. Downward case



# External Configuration File

Below is the description of the attributes in iclconfig.json, which is located in the root folder of War distribution file, or /FILES folder in eclipse IDE. With this configuration file the user can modify various parameters for ICL. Note that the changes take effect after restarting the server. The generator of iclconfig.json file is in the org**.**uclab**.**mm**.**icl.ConfigJsonGenerator.

* DCLAddress: URI of Data Curation Layer.
* LLCAddress: URI of Low Level Context Recognition module. Currently Low Level Context module and High Level context modules are integrated for debugging convenience, but they can be modularized at any time.
* HLCLAddress: URI of High Level Context Recognition module. Currently Low Level Context module and High Level context modules are integrated for debugging convenience, but they can be modularized at any time.
* WindowConfig: contains configuration related to the unifying window size of ContextType
* Activity: window size of the ContextType Activity. Final activity context is drawn from ICL with this period. Note that, the label is equivalent to the name of the enum ContextType.
* EmoWindow: window size of the ContextType Emotion. Final emotion context is drawn from ICL with this period. Note that, the label is equivalent to the name of the enum ContextType.
* LocWindow: window size of the ContextType Location. Final location context is drawn from ICL with this period. Note that, the label is equivalent to the name of the enum ContextType.
* ActiveRecognizers: contains configuration related to the unifying window size of ContextType
* ARSang: Determines the activeness of RecognizerType ARSang. If the value is false, the recognizer will not active on the ICL. Note that, the label is equivalent to the name of the enum RecognizerType. ]
* VARThien: Determines the activeness of RecognizerType VARThien. If the value is false, the recognizer will not active on the ICL. Note that, the label is equivalent to the name of the enum RecognizerType.
* EmotionJH: Determines the activeness of RecognizerType EmotionJH. If the value is false, the recognizer will not active on the ICL. Note that, the label is equivalent to the name of the enum RecognizerType.
* EmotionVUI: Determines the activeness of RecognizerType EmotionVUI. If the value is false, the recognizer will not active on the ICL. Note that, the label is equivalent to the name of the enum RecognizerType.
* HLCDelay: Determines the initial delay of the HLC server.