# Developer Guide for Creating New Components in Facilino App Inventor 2 Extension

## About App Inventor2 and extensions

App Inventor 2 is an online web platform to create Android Apps using block programming tools. It is based on Blockly (developed by Google) and is widely used my lot of people to create simplistic apps.

One of the most interesting features about App Inventor 2 is that is able to include extensions (this was not possible in the beginning), but now it has become very popular to generate App Inventor 2 extensions.

They allow non-developer uses to include extra functionalities fully integrated within the App Inventor 2 block programming.

Here is a list with some of the extensions:

<https://mit-cml.github.io/extensions/>

<https://puravidaapps.com/extensions.php>

App Inventor 2 has two types of components: visible components and non-visible components. Extensions are always non-visible components.

## Add an extension to your project

In order to add an extension to your project, you need to get the “aix” file, a binary file as a result of compiling an extension for App Inventor 2. You just simply need to click on the “Import extension” link within the “Extension” category in the Designer View and search for the file in your computer. Once you click “Import”, you all new components included in the extension will be added to your project.

## Build App Inventor 2 sources

These steps are needed in order to compile the extension. It is documented here:

<https://docs.google.com/document/d/1Xc9yt02x3BRoq5m1PJHBr81OOv69rEBy8LVG_84j9jc/pub>

This is a summary of required steps to build an extension:

1. Get App Inventor 2 source code:

git clone <https://github.com/mit-cml/appinventor-sources.git>

1. Install App Inventor dependencies: Java Development Kit (JDK) 1.8 and Apache Ant 1.10+. Make sure that the appropriate directories are in your PATH environment variable, otherwise commands in the remainder of this document likely will fail.

For instance, you can use:

<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

<http://ant.apache.org/bindownload.cgi>

The environment variables you need to set are:

\_JAVA\_OPTIONS set it to -Xmx1024m

ANT\_HOME set it to “apache-ant-xxx-bin” folder.

ANT\_OPTS set it to -Xmx256M

JAVA\_HOME set it to “C:\Program Files\Java\jdkxxxx”

CLASSPATH add the values %ANT\_HOME%\lib;%JAVA\_HOME%\lib

PATH add “;%ANT\_HOME%\bin;%JAVA\_HOME%\bin”

1. Download the Facilino Extension for App Inventor 2 source code in:

<https://github.com/roboticafacil/facilino>

Inside the “ai2/src” folder you will find the code of the extension. You need to copy the contents of the “es/roboticafacil/Facilino/runtime” code inside the “appinventor2/components/src” folder of App Inventor 2.

1. Build App Inventor 2:

ant clean

ant MakeAuthKey

ant

ant extensions

*Building should take several minutes and end with the message BUILD SUCCESSFUL. If something goes wrong, it’s most likely a problem with ant or Java or the App Inventor source.*

## Facilino App Inventor 2 Extension

Extension components are created by programming in Java. The Java code can be original code that you write, and it can also include Java libraries (jar files) from other sources. Here, we describe the structure of Facilino App Inventor 2 extension code.

Facilino extension is based on a BluetoothClient component, but it will be extended to TCP UDP connections in the future too. The important files are:

### FacilinoBase.java

Here we define all constants for building telegrams. These number MUST be the same in your Arduino code when you decode them. Please, take a look to them and if you need to add new telegrams, define a proper command for your sensor/actuator. It also handles the list of sensors and actuators that have been attached to the Facilino Component. Whenever a new telegram is received, this class decode the basic telegram structure and passes the data to the corresponding component (the FacilinoBase class does not know what to do with the data, it only knows about the telegram structure). It also has an abstract method “SendBytes”, which must be implemented by each of the individual components (sensors or actuators) to request something.

### FacilinoBluetooth.java

This class inherits from “FacilinoBase” class to implement the Bluetooth functionalities. It needs an instance of an App Inventor 2 bluetooth client. This is accessible to the final user as a property to set and this is indeed the component that you must add to your project you handle a Bluetooth connection.

### FacilinoSensorBase.java

Each sensor component must inherit from this class. It includes methods for attaching and detaching the sensor.

### FacilinoActuatorBase.java

Each actuator component must inherit from this class. It includes methods for attaching and detaching the actuator.

## Creating your own component

In this case, the easiest way to start with is to copy an existing example (the one with the most similar features) and adapt the code. Please, follow these steps:

1. Import additional libraries that your code might require (if necessary).
2. In “@DesignerComponent” tag, modify your component description and the static URL with the 16x16 icon file (if you want to be uploaded to roboticafacil server, please send an e-mail to [soporte@roboticafacil.es](mailto:soporte@roboticafacil.es) to request that).
3. Change the class name.
4. Define internal variables depending on your needs at the beginning of the class file and initialize the correct values in the class constructor. Particularly, you might need to change the component type. If it is one of the listed types within the FacilinoBase class use it, if not, create a new type inside the FacilinoBase class (this is useful to dispatch to the correct component once a telegram is received).
5. If you need to add a new property (it will be only accessible on the block page by default, but not in the designer page), you need to add the following tag before your method.

@SimpleProperty(category = PropertyCategory.BEHAVIOR)

If you want to be added as an editable property at the design stage, then you need to and the following tag too:

@DesignerProperty(editorType = PropertyTypeConstants.PROPERTY\_TYPE\_NON\_NEGATIVE\_INTEGER,

defaultValue = "0")

Please, find other possible propery types available in App Inventor 2 within the package “com.google.appinventor.components.common.PropertyTypeConstants”.

Adding a descripton on the @SimpleProperty tag will help to know what is your property about.

1. If you need to add a function (or method), you need to add the following tag:

@SimpleFunction(description = "xxxx")

Change the function description to your own description.

1. If you need to create a method, then xxx

@SimpleEvent(description = "xxx event.")

Change the event description to your own description.

This is an example of an Event method. The name included in the Event dispatcher must be the same as the name of your method:

public void BlackDetected(){

EventDispatcher.dispatchEvent(this, "BlackDetected");

}

If your event need to inform the user of an specific argument, you can add that too:

public void Changed(boolean value){

EventDispatcher.dispatchEvent(this, "changed",value);

}

1. In order to send a telegram, you can modify the following method:

private YailList readTelegram()

This method is returning a YailList with all data to be send over Bluetooth (or TCP/UDP). It must meet the telegram structure defined by the Facilino App Inventor 2 extension (see Telegram Structure section).

Make your that the number of reserved bytes is the correct one for your telegram!

1. Override the method “dispatchData” in “FacilinoBase” class to process data you receive. Data is the field inside the telegram, so that data[0] will be the first byte of this field. You also have access to the received command, just in case is not for this component. At the end, you should set “\_dataDispatched” to true. If based on the received data, you need to throw an event, the place to call the methods is inside this method.

public void dispatchData(byte cmd, byte[] data)

### Telegram Structure

Any telegram has the following structure:

@|CMD|LEN|DATA0|DATA1|…|DATAN-1|\*

Being | a byte delimiter.

Each telegram must start with the character symbol ‘@’, followed by a single byte command. See the class FacilinoBase for the actual list of commands (either for requesting or as a response). Then, you will find the data length in bytes, it is one single byte indicating the amount of data expected to receive or to send in the DATA field. The DATA field is flexible, but it is recommended you include the pin number of the sensor and actuator you want to control, because there might be several of them in a general application. Finally, finish your telegram with the symbol ‘\*’.