# App Inventor extensions

Initial Release 1.0 (Introduced as part of MIT App Inventor Release nb149)
DRAFT: September 27, 2015
Updated: June 9, 2017
Updated: May 17, 2021
sample extensions are at: MIT App Inventor Extension
source code for sample extensions at: http://appinventor.mit.edu/extensions
Appinventor help: https://community.appinventor.mit.edu/

---

**Note:** App Inventor extensions are supported only on Android devices running API Level 8 (Android system 2.2 Froyo) and above. This applies to creating extensions, building projects that import extensions, and running packaged APKs of projects that use extensions. Extensions are not (yet) supported in iOS.

---

# 1. Overview of App Inventor extension components

App Inventor apps are built using components. Components let the apps use the built-in features of the mobile device (like Camera or LocationSensor) or services on the Web (like Twitter or FusionTables). App Inventor includes a large collection of components, and the App Inventor development team adds new capabilities to the system by implementing new components.

There have been many requests to include additional features in App Inventor. Some of these are special-purpose features that would have only a few users, where it would be undesirable to include these in the core system for everyone. Other features might be good additions to the core system, but the App Inventor development team's effort has gone to other priorities. Anyone can use the App Inventor free and open source code to implement their own components, but until now the only way to make these available to others has been to include them in private versions of App Inventor that are hosted and managed individually.

*App Inventor Extensions* let anyone create *Extension Components.* Extension components can be used in building projects, just like other components. The difference is that extension components can be distributed on the Web and loaded into App Inventor dynamically: they do not have to be built into the App Inventor system, and they can be imported into projects as needed. With extensions, the range of App Inventor apps can be virtually unlimited.

One use of extensions, for example, is for educators and educational software developers to provide extension components tailored to specific lessons and activities that students can use in building apps. Examples might be simulations or data analysis tools. Those apps might be unfeasible to create directly with the built-in App Inventor blocks, either because of processing speed or programming complexity. But those same apps might be readily programmable in the App Inventor framework, using extension components that perform the necessary processing.

Anyone can create extension components. This requires gaining familiarity with the App Inventor source code (located on Github) and programming the extension in Java. Extension components are packaged as *aix files*. Once you create an extension component, anyone can use it in their App Inventor projects. Extension component aix files can be housed anywhere on the Web. The aix files need not be stored at MIT or any other particular place, although MIT hosts a repository where people can make aix files available for sharing and public use.

*Note:* One limitation of the current extension component implementation is that it creates non-visible components only (i.e., components that do not show on the designer screen). MIT plans to remove this restriction in future versions of the extension component system.

# 2. How to use extension components

You use an App Inventor extension component just as you would use an ordinary built-in component, except that you'll first need to import the extension into your project.

## 2.1 Importing extension components

Before you can import an extension component, you must have a project open. Start a new project, or open an existing one. To import an extension, look in the components palette at the left of the screen under *Extensions* and click the "Import extension" link as in the figure below. This will bring up a window that lets you specify an extension to import. Extensions are defined by *aix* files. You can import an extension from an aix file on your computer or you can specify a URL to import from the Web.

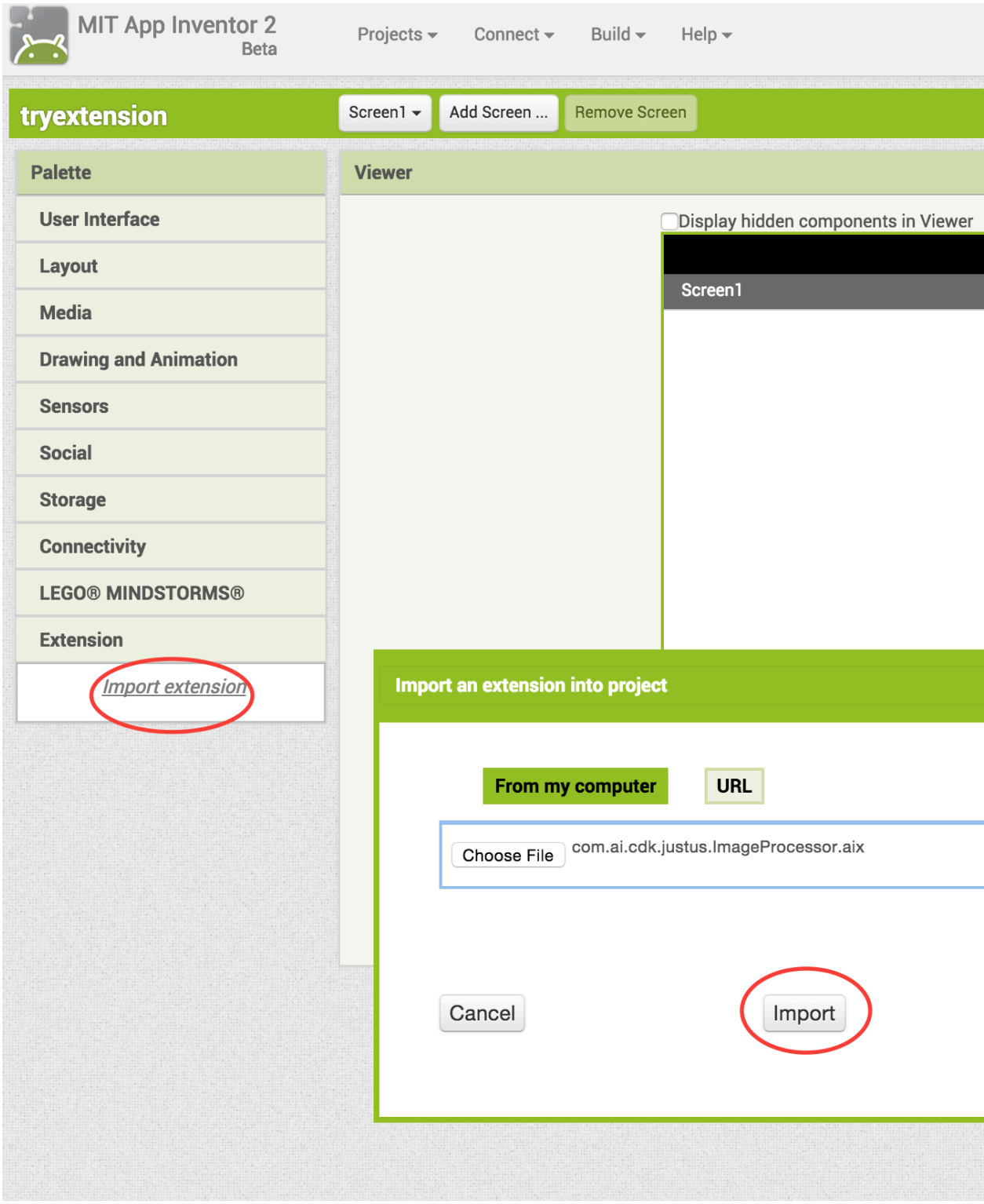



Figure 1: Importing an extension

When you import an extension, App Inventor will give you the opportunity to rename it, as shown in the figure.

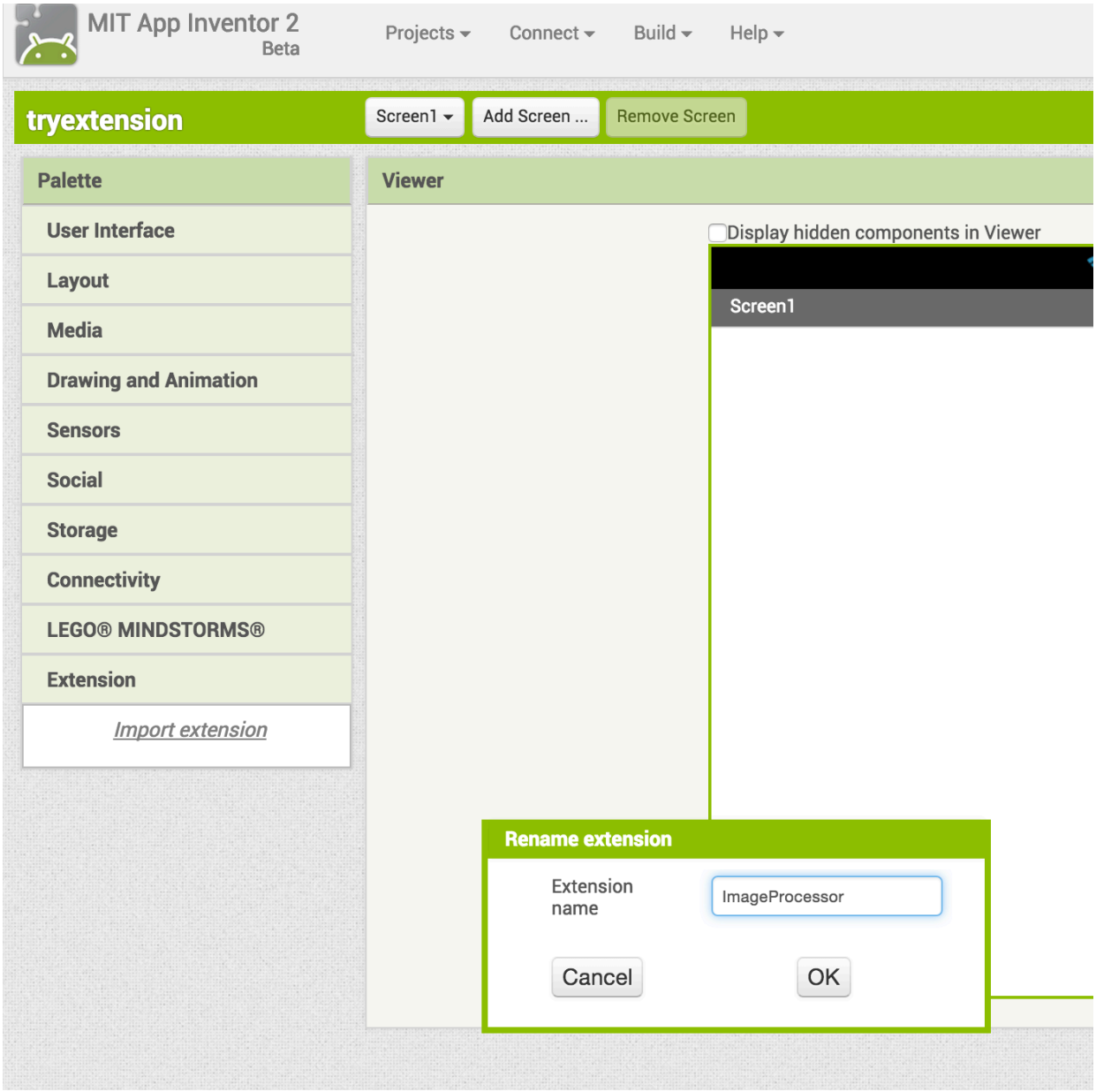



Figure 2: Renaming an extension when you import it.

You'll generally want to keep the default name. See the section below on naming for cases where you might want to choose a different name.

Once you import an extension, it will appear in the components palette under the extensions category as shown in the figure below.

After importing an extension, make sure to restart the companion app, else an error may occur.

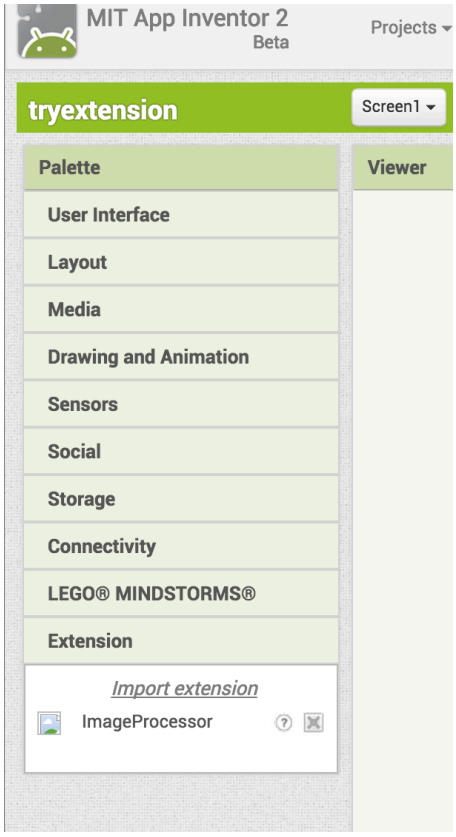



Figure 3: An imported extension shown in components palette

## 2.2 Some sample extensions to try

Here are some sample extension components to experiment with. You can find these in the MIT App Inventor Extensions repository at MIT App Inventor Extensions. MIT will be adding more examples to this collection, and would like to evolve it into a site for people to share extensions. You can import the extensions using the URL indicated, or you can download the aix files to your local computer and import the files from there.

- VectorArithmetic created by Ethan Hon
  Takes in two vectors and can add them to return a result vector.
  https://drive.google.com/uc?export=download&id=0B22N7pfciq0CSHlFQVBCRk5OYlB3U3FDWFNlNDhvOVp5bDM4

- ImageProcessor created by Justus Raju
  ImageProcessor can do a weighted combine of two images, return the grayscale of an image.
  http://appinventor.mit.edu/extensions/data/extensions/ai.cdk.justus.ImageProcessor.aix
  Here is a clean project that demos the use of the component. You can import this aia directly too. http://appinventor.mit.edu/extensions/data/demo/ImageProcessorDemo.aia

- SoundAnalysis created by Mouhamadou Oumar Sall
  Multimedia component that analyzes the pitch of a sound through the microphone and returns it. It can be used as an input to different situations, for instance to control some components with a specific sound whistle(pitch > 500Hz) or clap(pitch > 2000Hz).
  https://drive.google.com/uc?export=download&id=0B22N7pfciq0CUTZONWpKZUw2YXJlWHBMcERuaUliZWREdU5R

- ScaleDetector created by Hal Abelson
  This component adds a pinch detector capability to a Canvas. For an explanation, see Using App Inventor extensions to implement multitouch.

### 2.3 Building projects with extension components

Once you have imported an extension component into a project, it will appear in the component palette under the *Extensions* section. You use the extension component just like any other component by dragging it from the palette to the designer screen, where it will appear under the designer screen as an non-visible component. (Currently, all extension components are non-visible components.) You'll also see it in the Blocks Editor, together with blocks for its methods, properties and events. The figures below illustrate how an imported ImageProcessor extension would appear in the Blocks Editor.


Figure 4: Blocks for an imported ImageProcessor component

## 2.4 Deleting extension components

To delete an extension from the project, click the X beside the extension. This will remove all uses of the extension and its blocks from the project and you will not see it in *Extensions* palette panel anymore.

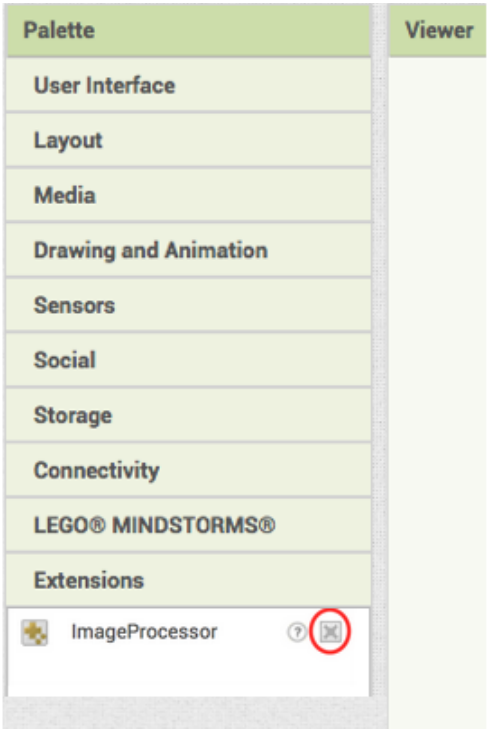



Figure 5: Click the X to delete an imported extension.

## 2.5 Sharing projects that use extension components

You export and import projects that use extension components just like other App Inventor projects, as *aia* files. You need not do anything different in publishing the project than in publishing other App Inventor projects. If someone imports a project (*aia* file) that uses an extension, they do not have to import the extension (*aix* file) separately: When they open the project, the extension will appear in the components palette (under Extensions) together with the project's other components.

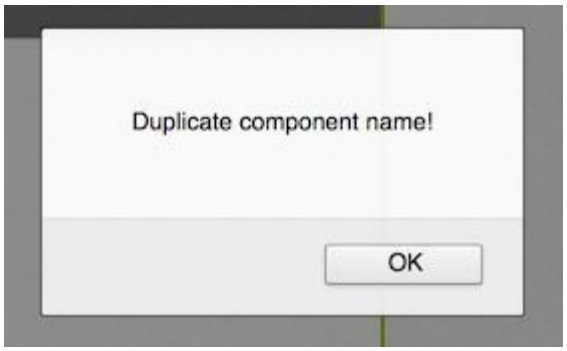
## 2.6 Extension component repositories

Extensions can be downloaded from any URL (provided you have access to it). Anyone who creates extensions can make the aix files available however they choose. It might also be convenient to create collections of extensions, called *extension component repositories* for private or shared use. For instance, an educator might create a repository of extension components for use in a course.

The MIT AppInventor project has created a public repository for general use, to encourage people to create and share extensions. The repository is located at http://appinventor.mit.edu/extensions.

### 2.7 Naming extension components

Extensions are created by different people working independently. So it's likely that different extensions might have the same name. For example, there might be two different extensions called ImageProcessor, and you'd like to import them both. There's no problem using the two extensions if you use them in different projects (other than that you might get confused). But you can't import two extension with the same name into a single project. When you import and extension to a project, App Inventor will give you the opportunity to rename it, as shown in Figure 2. This will let you keep the extension names unique and also (should you prefer) choose names for the extensions that are more convenient for you than the original names.




(warning when you give the extension a name that already exists in the project)

If you are developing extensions (not just using them) you can choose extension *package names to help* minimize name conflicts. See the section on packages below.

### 2.8 Updating projects that use extensions

We have not addressed the issue of what happens when extensions are updated and how people using the extension can be notified about this. There are two scenarios to consider:

1. The extension developer creates a new version of the extension, but the old version still works.

2. The App Inventor system is updated so that the extension no longer works.

In either case, people importing the extension will need to look to extension developer to provide information about updating projects that import the extension

**Warning:** The extensions system is still experimental and the internal format of extensions is changing. If you create a project (aia file) that imports an extension, it is possible that the extension (and the project) will stop working when the extension system is upgraded. You'll have to rely on the extension developer to provide an updated extension.

## 3. How to create extension components

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.
You can create extension components for private use, or you can share them by sending people the aix files or making the files available on the Web.

The process of creating an extension component is essentially the same as what people go through in creating ordinary App Inventor components, both for MIT App Inventor or for local versions built using the App Inventor free and open source code. The difference is that the only way to make an ordinary component available to other people is to host your own App Inventor instance, whereas with an extension component, you can publish the aix files so that anyone can import it into any version of App Inventor.

Building an extension component is no more difficult than building an ordinary component -- but then again, it's no easier. If you've never created a component before, you should start by reading these guides:

- How to Build App Inventor from the MIT sources
  This shows how to get the MIT App Inventor sources and how to use them to create a local App Inventor instance on your own computer for personal use

- How to Add a Component
  This gives an overview of component programming, including how to implement the Java code for the properties, methods and events, and how to define the blocks for the Blocks editor.

It would also be a good idea to look at the resources at the MIT App Inventor Open Source Web page. You might also want to participate in the MIT App Inventor Open Source Development group which is a good place to find advice on implementing components.

Here are the steps we recommend in building an extension component. These are detailed in the sections below:

1. Set up a local version of App Inventor for your own use, building a simple extension component for it, as practice.
2. Build your desired component for your local instance and test it carefully.

3. Generate an aix file, so that the extension can be imported.
4. Test the aix file by importing the extension into a project you build on the public App Inventor server.
5. Publish your extension.

### 3.1 Practice creating a sample component

You can skip this step if you've already built App Inventor components. But if you haven't created components, you should first get some practice by creating a simple example. Even if the required Java coding is minimal, there are a lot of details involved in setting up the software for creating and running a local instance of App Inventor.

The detailed instructions for building components are in How to Build App Inventor from the MIT sources. The instructions begin by explaining how to set up your development environment and download the App Inventor source code from Github.

Once your environment is set up, a simple way to practice building components is to create an ordinary extension component that's a copy for an existing component and give it a different name. This section shows how to create a component called AltCamera that behaves exactly like the Camera component, except for its name. This section also shows how to convert the AltCamera component to an extension.

To build AltCamera, install the App Inventor source code in a git repository on your local machine, following the instructions in How to Build App Inventor from the MIT sources. Go to the directory

```
appinventor/components/src/com/google/appinventor/components/runtime
```

and copy the file Camera.java to a new file AltCamera.java. Near the top of the file, you'll see

```
@DesignerComponent(version = YaVersion.CAMERA_COMPONENT_VERSION,
  description = "A component to take a picture using the device's camera. " +
        "After the picture is taken, the name of the file on the phone " +
        "containing the picture is available as an argument to the " +
        "AfterPicture event. The file name can be used, for example, to set " +
        "the Picture property of an Image component.",
  category = ComponentCategory.MEDIA,
  nonVisible = true,
  iconName = "images/camera.png")
```

Change the description as follows:

```
description = “This is an alternate version of the Camera component.”,
```

The description is the text that will appear when the user presses the question mark next to the component name in the designer pallet.

> **Note:** For App Inventor’s built-in components, the description text is internationalized so it can appear in the local language that App Inventor is set to. The extensions implementation does not support internationalization and the description text will remain as you’ve entered it, regardless of how the local language is set.

Find the class constructor

```
public Camera(ComponentContainer container) { ...
```

and change “Camera” to “AltCamera”.

In the Take Picture method, find the line

```
final Camera me = this;
```

and change “Camera” to “AltCamera”.

Finish building your modified App Inventor code, as described in How to Build App Inventor from the MIT sources, launch it on your local machine, and connect a browser to localhost port 8888. Log in and create a project that uses AltCamera. If all has gone well, you should see two camera components in the designer palette Media Section: the original Camera and a new AltCamera. Test this by using AltCamera in a project, opening the Blocks Editor, and observing that AltCamera has all the correct blocks -- the same blocks as Camera.

## 3.2 Convert your sample component to an extension

### 3.2.1 Build the aix file for your extension

Converting your component to an extension is done with the aid of the App Inventor Extension T emplate. The template is in the github repository

https://github.com/mit-cml/extension-template

Clone this repository to your local machine following the instructions at the bottom of the github page.

In your cloned repository, find the folder my-extension/src and copy your AltCamera.java file there. In that file, find the line

```
  category = ComponentCategory.MEDIA,
```

and change it to

```
  category = ComponentCategory.EXTENSION,
```

Now go to the folder myextension and run the command

```
ant
```

This should build your extension. It should be in the my-extension/out folder with the name

```
com.google.appinventor.components.runtime.aix
```

### 3.2.2 Test your extension

Now that you've built an extension, check that you can load it into App Inventor and use it in an app. Open your browser to an App Inventor server. You could use ai2.appinventor.mit.edu, or you could use a server you've built on your local machine.

Start a new project. In the designer palette open the Extension drawer and click *import extension.* Choose the aix file you built just above and click *import.* AltCamera should now appear as a component in the Extension drawer.

To do more testing, check that you can use AltCamera in your new project you, just as your would use Camera. You can see in the Blocks Editor that it has the same blocks, which behave just as the blocks for Camera. (They have the same implementation code.) Try building a simple app and running it.

## 3.3 Details on creating extensions

Now that you've verified that you can create an extension component, you try experimenting with new components of your own. As with AltCamera, it might be simpler to begin by implementing an ordinary component, verifying that it works, and then converting the component to an extension using the App Inventor Extension Template.

This selection list some issues to keep in mind.

### 3.3.1 When you start to build

You can follow the method described above and begin by creating an ordinary component on a local instance of App Inventor, together with an app that uses the new component. For testing on a phone or tablet, you'll need a special App Inventor Companion that includes the new component. You can use "*ant installplay*" to perform the companion installation on the device, as explained in How to Build App Inventor from the MIT sources. Later, when you transform you component to an extension, you do **not** need a special companion, but can use the ordinary companion, e.g., from the Play Store. (Part of the extension "magic" is that is that the required code for the extension will be installed dynamically in the companion when App Inventor loads the extension.

Your component can be anything you can implement, with the following restrictions:

- It must be a non-visible component. Keep in mind that "non-visible" here refers to visibility in the App Inventor designer, not visibility on the phone screen. You can still create components that use anything in the Android SDK and produce visible results in the app on the phone; they just won't appear in the designer screen.

- The property editors (i.e., the things that determine how users specify property values in the designer), must be ones that are already included in core appinventor (e.g. those that are implemented here). One exception to this introduced with the App Inventor for IoT release (nb158; June 2017) is that PROPERTY_TYPE_COMPONENT can be restricted to specific classes to allow extensions to interact with one another by appending a colon (:) followed by the fully qualified class name of the extension. This is used, for example, to limit inputs to the BluetoothLE extension by appending ":edu.mit.appinventor.ble.BluetoothLE".

- The category specified in @DesignerComponent must be

  category = ComponentCategory.EXTENSION

- The icon name file specified in @DesignerComponent should be

  iconName = "images/extension.png"

  Starting with nb158, you may also include your icon in a subpackage to your extension called aiwebres. This special package name is used by App Inventor to identify resources required to be servable by the online editor and is currently restricted to extension icons. If you use this functionality, specify your icon as:

```
iconName = "aiwebres/iconname.png"
```

You could include a link to your icon on the internet, for example

```
iconName = "http://somewhere.com/images/yourIcon.png"
```

The icon should be a 16×16 png file. *Caution:* If you supply a URL for the icon, the icon will not be packaged with your extension. Rather, App Inventor will read it from the Web each time the extension is shown in the designer, which may lead to strange behavior if the link is slow, or the target of the link changes.

### 3.3.2 Requesting permissions for the extensions you define

Your component might need to request permissions, just as ordinary components do, with the @UsePermissions declaration. That will be sufficient for using the extension in projects that you build and package.

But there is a complication with live development. Importing the extension, even if it specifies @UsePermission, will not guarantee that the permission will be available to the App Inventor companion for live development. In most cases, this will not be a problem because the Companion comes with a wide range of permissions used by any of the App Inventor built-in components. But if your extension uses a permission that is not one of these, it might not be possible to build a project with it using live development.

### 3.2.3 Using external libraries

In addition to your own Java code, you can also include external libraries, by following the steps in How to Add a Component. The library jar files will be included in the aix file when you package your extension

One complication here is that the external libraries (jar files) must be unique across projects. If a project imports several extensions, then two different extensions cannot import the same jar file.\

### 3.2.4 Choosing a package name for your extension

The first line of your extension Java file should be a package specification, for example,

```
package com.google.appinventor.components.runtime;
```

The runtime package is the standard package used for App Inventor components.

If you are creating an extension solely for personal use, you needn't be concerned about the package name. You can make the package name anything you like. There is one proviso: If your Java code needs to reference other App Inventor classes defined in some package, you should put your extension in that package. For example, the AltCamera class extends Android onVisibleComponent:

```
public class AltCamera extends AndroidNonvisibleComponent
```

which is defined in the runtime package, so AltCamera.java is placed in the runtime package. If you don't want to place it in that package, you can import the AndroidNonVisibleComponent class explicitly.

If you are publishing extensions for other people to use, you should pick names for your extensions in a way that will help guarantee that your names will be unique. The method here is similar to how websites are named. For example, if the MIT App Inventor project was going to create a Web site named OurNewSite. we'd be following good practice by putting the Web site at the URL *OurNewSite.appinventor.mit.edu.* In a similar way, if we were publishing an extension named *ImageProcessor,* we would do well to use an analogous extension name that specifies the extension as coming from *appinventor* and *mit* and *edu*. The convention here (adopted from the way people name Java libraries) is to list the path elements in reverse order, so the name would be *edu.mit.appinventor.ImageProcessor*.

In this naming scheme, ImageProcessor is the *extension name* and edu.mit.appinventor is the *package name*. The two combined, *edu.mit.appinventor.ImageProcessor*, is called the *fully qualified name.* When people import your extension, they see the extension name, not the fully qualified name. The extension name is the name they can change when they import the extension, as described above and shown in figure 2. But the fully qualified name is fixed: Appinventor will not import two different components with the same fully qualified name. If you try to do that, you'll get an error as shown in figure 6.

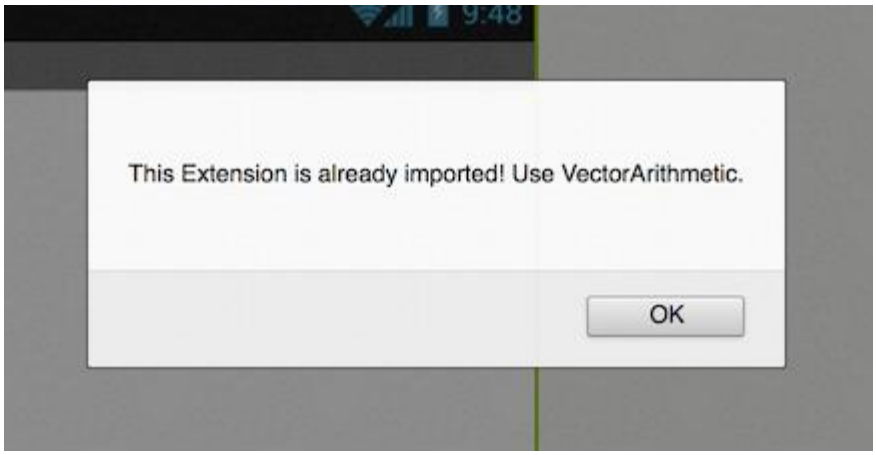


Figure 6: Error signalled when trying to import a duplicate extension (i.e. with same fully qualified name as an existing extension)

### 3.3.1 Change your extension to have the desired package name

To make your extension have the desired package name, you need put its Java code in a directory tree whose subdirectories mirror the package name. This tree should sit under *appinventor/components/src.* For example, to make the *Imageprocessor* extension be in the package *edu.mit.appinventor*, you would go to the appinventor/components/src directory and in there create the tree *edu/mit/appinvento*r. In more detail

- go to the folder /appinventor/components/src, and in that folder
- create a folder edu, and in that folder
- create a folder mit, and in that folder
- create a folder appinventor, and in that folder
- put the file ImageProcessor.java

Create the corresponding tree for your desired package name, and move the Java file that defines your component to the tree. (If you copied the Java file instead of moving it, don't forget to delete it from its old location in ../components/runtime.)

Now go to the Java file you just moved. At the top, you should see the line

```
package com.google.appinventor.components.runtime;
```

Change the package name from com.google.appinventor.components.runtime to your actual package name. For example if your package name is edu.mit.appinventor, as our example above, change the line to

```
package edu.mit.appinventor;
```

Finally, at the top of your component definition Java file, right under the package name, you'll see a list of imports. Add a new import at the top of the list:

```
import com.google.appinventor.components.runtime.*;
```

The reason for adding this is that your component Java definition code might have used some modules in the original package, and these might not be available when you move to a new package.

*Note:* You don't need to import all of runtime.*, just the actual files you use. You can identify these by attempting to build the instance and seeing where you get undefined symbol errors.

Now that you've made these changes, run

ant clean

and build your local App Inventor instance again. Run your instance and connect a browser. The result should look the same as before you changed the package. Test everything again to verify that you can create a new working companion, and you can build projects with the new component.

3.4 Create the aix file for your new component

When you've tested enough (*you can never test enough!*) it's time to create the aix file that lets people import your extension.

Make sure you have changed the @SimpleObject line just over the top of your extension Java class to @SimpleObject(external = true).

Use the cd command to change to the directory appinventor-sources/appinventor directory:

cd <*path to appinventor-sources*>/appinventor

and run the command

ant extensions

Your aix file should appear in the directory

appinventor-sources/appinventor/components/build/extensions

The name of the aix file should be the package name (prior to May 2017 it was the fully qualified class name) of the component you created.

To test that your extension works, copy the aix file to a convenient location on your computer. Now go to a public instance of App Inventor that does not have your extension. Make sure to switch your companion back to the version for that instance.
Create a new project and import the extension, using the *aix* file from the your computer and test once more that the imported extension works in a project.

Test your project using companion in live development mode, and also test that you can build the project to produce a working  apk file.  Make sure to test both modes.

### 3.4 Sharing your extension

You can share your extensions by making the aix files available in any way that you choose. People just need to access the files and import them into their projects.  The MIT App Inventor group has a small set of sample extensions at appinventor.mit.edu/extensions.  We plan to open this site as a repository for people to share extensions, after we get a bit more experience.

Remember that when you share your extensions, you are publishing software for others to use in building their projects, and they will be relying on you to keep your extensions up to date and bug-free.

> ***Note:*** The internal representation of extensions is still undergoing change.  Please use the current testing system to experiment with creating extensions, but don't be surprised if these will need to be regenerated as the extensions implementation evolves.

Project that import extensions are not permitted on the MIT App Inventor Gallery, since the aix file contains only the binary code of the extension, and apps on the Gallery must be fully available in source code form.  We are considering ways to handle this issue, but this is not yet ready.


### Acknowledgements

App Inventor extensions are a major addition to App Inventor.  They were created as part of the Google 2015 Summer of Code project by Ethan Hon, Justus Raju and Mouhamadou Sall.  They did a remarkable piece of work, and they deserve the thanks of the entire App Inventor community.


Appendix A: Temporary instructions for obtaining the source code

To work with extensions before the merge you should replace the instructions on getting source code at the beginning of section 2.1 of How to Build App Inventor from the MIT sources by the following.

> The App Inventor source code is available as Free and Open Source Software that you can download.  Once you've installed the other software listed in section 3, you can clone a git repository of the source code by running the following git command from a shell:

This will create a folder named “appinventor-sources” where the sources (and a copy of the repository) will reside. If you have problems with this command, please visit the MIT Center for Mobile Learning  github site at https://github.com/mit-cml.

Now fetch all the branches of the source repository and check out a new branch based on the master branch for your own work:

git fetch origin

git checkout -b myNewBranch

As you create your extension in your repository work on this new branch.