Detail Lesson Plan

Introduction to App Development : Advanced

Day 1

a. Revision to APP Code: Basics

i. What we learnt in basic course: revision to applab

ii. Conditional statements: To know if a condition is *True* of *False*, we need a new type of data: the booleans. They allow logical operations. A logic statement or operation can be evaluated to be *True* or *False*. Our conditional statement can then be understood like this:

if *(a condition evaluates to True)*:

then do these things only for 'True'

else:

otherwise do these things only for 'False'.

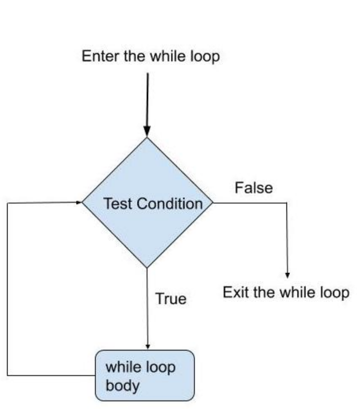
The condition can be anything that evaluates as *True* or *False*. Comparisons always return *True* or *False*, for example == (equal to), > (greater than), < (less than.) The else part is optional. If you leave it off, nothing will happen if the conditional evaluates to 'False'.

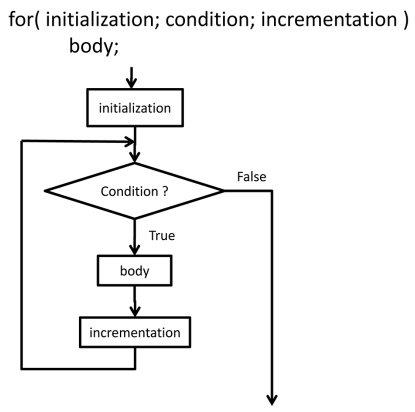
iii. Loops: In computer science, a loop is a programming structure that repeats a sequence of instructions until a specific condition is met. Programmers use loops to cycle through values, add sums of numbers, repeat functions, and many other things.

Loops are supported by all modern programming languages, though their implementations and syntax may differ. Two of the most common types of loops are the **while loop** and the **for loop**.

While Loop: A while loop is the simplest form of a programming loop. It states that while a condition is valid, keep looping. In the

PHP example below, the while loop will continue until i is equal to num.





For Loop: A for loop is similar to a while loop, but streamlines the source code. The for loop statement defines the start and end point as well as the increment for each iteration. Below is the same loop above defined as a while loop.

Database and storage: A database is a collection of information that is organized so that it can be easily accessed, managed and updated. Computer databases typically contain aggregations of data records or files, containing information about sales transactions or interactions with specific customers.

Types of Database

Relational database: A relational database, is a tabular database in which data is defined so that it can be reorganized and accessed in a number of different ways.Relational databases are made up of a set of tables with data that fits into a predefined category. Each table has at least one data category in a column, and each row has a certain data instance for the categories which are defined in the columns.The Structured Query Language (SQL) is the standard user and application program interface for a relational database. Relational databases are easy to extend, and a new data category can be added after the original database creation without requiring that you modify all the existing applications.

Distributed database: A distributed database is a database in which portions of the database are stored in multiple physical locations, and in which processing is dispersed or replicated among different points in a network.Distributed databases can be homogeneous or heterogeneous. All the physical locations in a homogeneous distributed database system have the same underlying hardware and run the same operating systems and database applications. The hardware, operating systems or database applications in a heterogeneous distributed database may be different at each of the locations.

Cloud database: A cloud database is a database that has been optimized or built for a virtualized environment, either in a hybrid cloud, public cloud or private cloud. Cloud databases provide benefits such as the ability to pay for storage capacity and bandwidth on a per-use basis, and they provide scalability on demand, along with high availability.A cloud database also gives enterprises the opportunity to support business applications in a software-as-a-service deployment.

NoSQL database: NoSQL databases are useful for large sets of distributed data.NoSQL databases are effective for big data performance issues that relational databases aren't built to solve. They are most effective when an organization must analyze large chunks of unstructured data or data that's stored across multiple virtual servers in the cloud.

Object-oriented database: Items created using object-oriented programming languages are often stored in relational databases, but object-oriented databases are well-suited for those items. An object-oriented database is organized around objects rather than actions, and data rather than logic. For example, a multimedia record in a relational database can be a definable data object, as opposed to an alphanumeric value.

Graph database: A graph-oriented database, or graph database, is a type of NoSQL database that uses graph theory to store, map and query relationships. Graph databases are basically collections of nodes and edges, where each node represents an entity, and each edge represents a connection between nodes.Graph databases are growing in popularity for analyzing interconnections. For example, companies might use a graph database to mine data about customers from social media.Graph databases often employ SPARQL, a declarative programming language and protocol for graph database analytics. SPARQL has the capability to perform all the analytics that SQL can perform, plus it can be used for semantic analysis, the examination of relationships. This makes it useful for performing analytics on data sets that have both structured and unstructured data. SPARQL allows users to perform analytics on information stored in a relational database, as well as friend-of-a-friend (FOAF) relationships, PageRank and shortest path.

b. Introduction to app inventor

   i. Apply principles from basic course to app inventor

   ii. How to build your app

   iii. How to run app in real time in your mobile phone

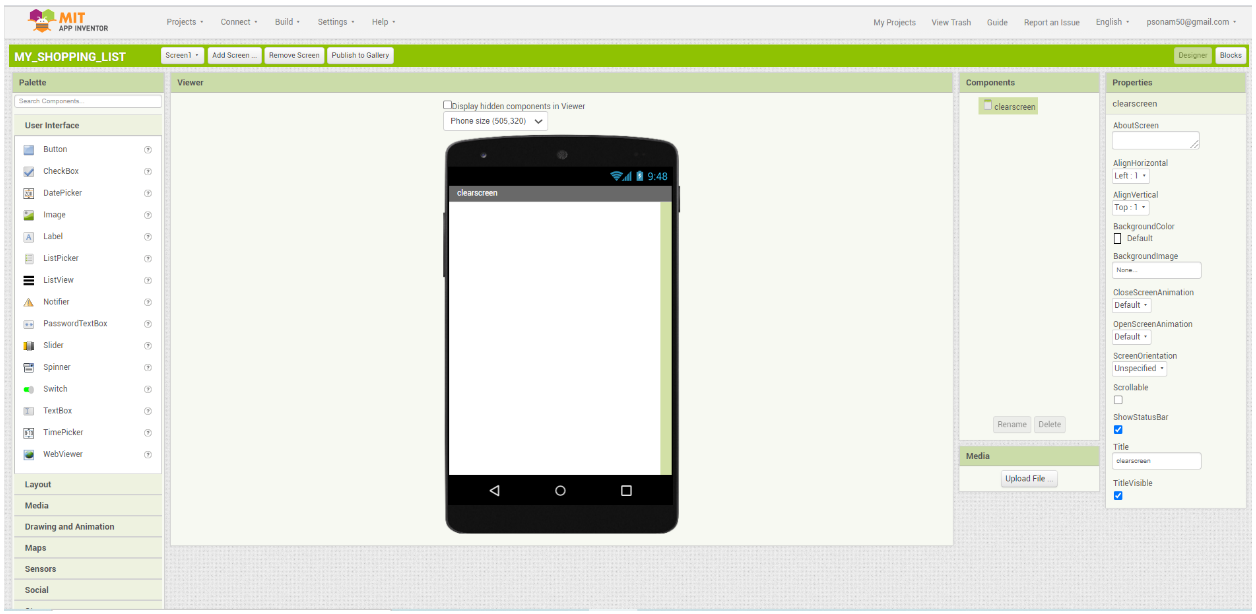
   iv. Build your apk and install

   v. How to change icon of app in mobile

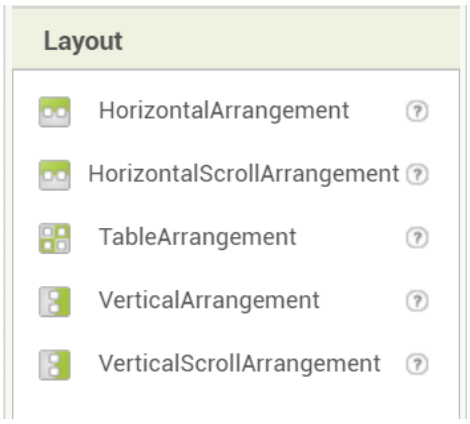
Day 2

c. Layouts in App Inventor

i. User interface of MIT APP INVENTOR



ii. Different Layouts in app inventor



iii. How app works in our mobile:

iv. Arranging and designing screen in app

v. Database and storage:

CloudDB: Non-visible component allowing you to store data on a Internet connected database server (using Redis software). This allows the users of your App to share data with each other. By default data will be stored in a server maintained by MIT, however you can setup and run your own server. Set the "RedisServer" property and "RedisPort" Property to access your own server.

File: Non-visible component for storing and retrieving files. Use this component to write or read files on your device. The default behaviour is to write files to the private data directory associated with your App. The Companion is special cased to write files to /sdcard/AppInventor/data to facilitate debugging. If the file path starts with a slash (/), then the file is created relative to /sdcard. For example writing a file to /myFile.txt will write the file in /sdcard/myFile.txt.

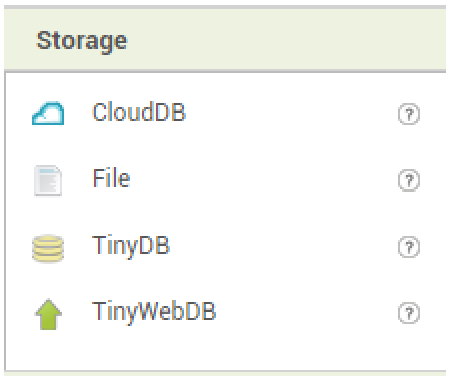
TinyDB: TinyDB is a non-visible component that stores data for an app.Apps created with App Inventor are initialized each time they run: If an app sets the value of a variable and the user then quits the app, the value of that variable will not be remembered the next time the app is run. In contrast, TinyDB is a *persistent* data store for the app, that is, the data stored there will be available each time the app is run. An example might be a game that saves the high score and retrieves it each time the game is played.

Data items are strings stored under *tags* . To store a data item, you specify the tag it should be stored under. Subsequently, you can retrieve the data that was stored under a given tag.

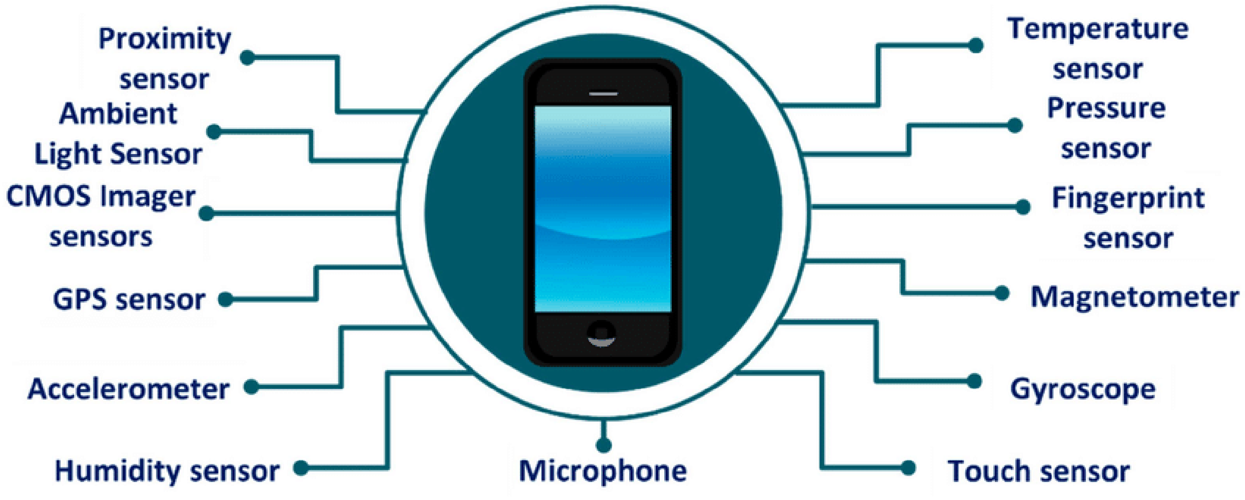
There is only one data store per app. Even if you have multiple TinyDB components, they will use the same data store. To get the effect of separate stores, use different keys. Also each app has its own data store. You cannot use TinyDB to pass data between two different apps on the phone, although you *can* use TinyDb to shares data between the different screens of a multi-screen app.

When you are developing apps using the AI Companion, all the apps using that companion will share the same TinyDb. That sharing will disappear once the apps are packaged. But, during development, you should be careful to clear the TinyDb each time you start working on a new app.

TinyWebDB: Non-visible component that communicates with a Web service to store and retrieve information.



d. Sensors in app: **Sensors** can be used to monitor the three-dimensional device movement or change in the environment of the device. Android provides sensor api to work with different types of sensors.



Types of Sensors: Android supports three types of sensors:

1) Motion Sensors

These are used to measure acceleration forces and rotational forces along with three axes.

2) Position Sensors

These are used to measure the physical position of device.

3) Environmental Sensors

These are used to measure the environmental changes such as temperature, humidity etc.

i. Accelerometer sensor: Non-visible component that can detect shaking and measure acceleration approximately in three dimensions using SI units (m/s^2). The components are:

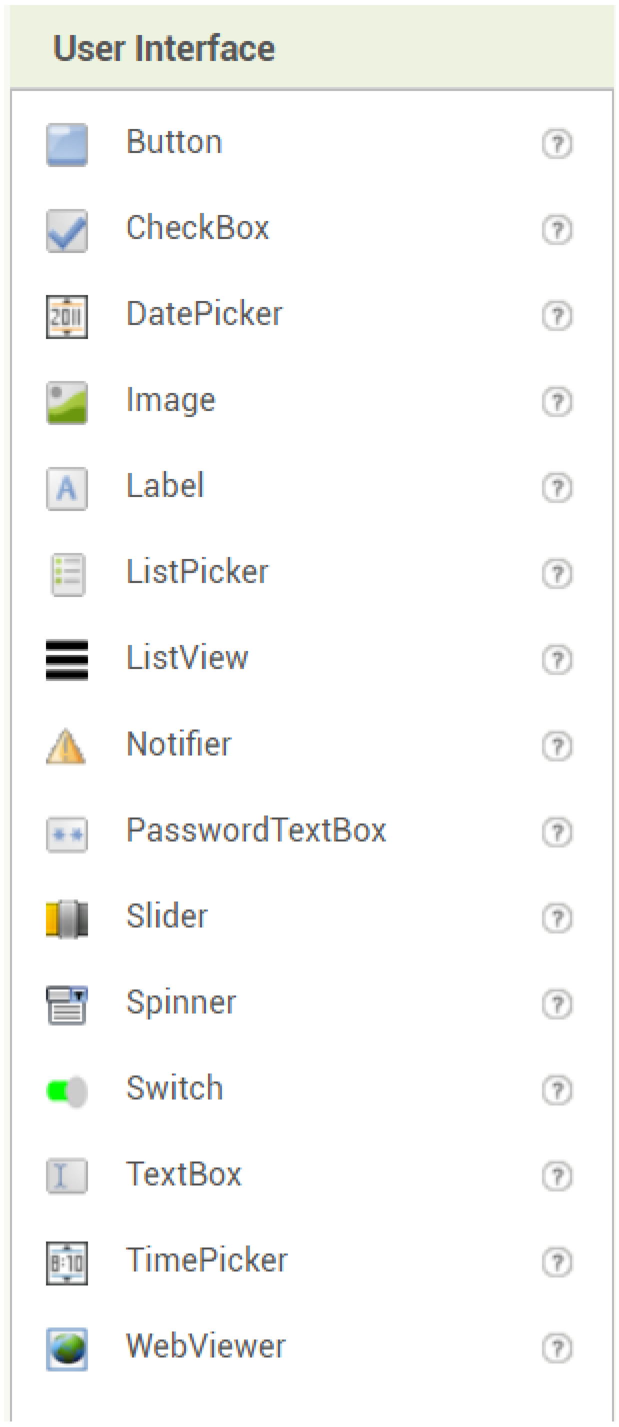
- **xAccel**: 0 when the phone is at rest on a flat surface, positive when the phone is tilted to the right (i.e., its left side is raised), and negative when the phone is tilted to the left (i.e., its right size is raised).
- **yAccel**: 0 when the phone is at rest on a flat surface, positive when its bottom is raised, and negative when its top is raised.
- **zAccel**: Equal to -9.8 (earth's gravity in meters per second per second when the device is at rest parallel to the ground with the display facing up, 0 when perpendicular to the ground, and +9.8 when facing down. The value can also be affected by accelerating it with or against gravity.

ii. Pedometer: A Component that acts like a Pedometer. It senses motion via the Accelerometer and attempts to determine if a step has been taken. Using a configurable stride length, it can estimate the distance traveled as well

Day 3

e. Screen Layout

i. Components in user interface



User Interface

- Button
- CheckBox
- DatePicker
- Image
- Label
- ListPicker
- ListView
- Notifier
- PasswordTextBox
- Slider
- Spinner
- Switch
- TextBox
- TimePicker
- WebViewer

ii. Storage

CloudDB

The CloudDB component is a Non-visible component that allows you to store data on a Internet connected database server (using Redis software). This allows the users of your App to share data with each other. By default data will be stored in a server maintained by MIT, however you can setup and run your own server. Set the RedisServer property and RedisPort property to access your own server.

Properties

ProjectID
Gets the ProjectID for this CloudDB project.

RedisPort
The Redis Server port to use. Defaults to 6381

RedisServer
The Redis Server to use to store data. A setting of “DEFAULT” means that the MIT server will be used.

Token
This field contains the authentication token used to login to the backed Redis server. For the “DEFAULT” server, do not edit this value, the system will fill it in for you. A system administrator may also provide a special value to you which can be used to share data between multiple projects from multiple people. If using your own Redis server, set a password in the server’s config and enter it here.

UseSSL
Set to true to use SSL to talk to CloudDB/Redis server. This should be set to True for the “DEFAULT” server.

Events

CloudDBError(*message*)
Indicates that an error occurred while communicating with the CloudDB Redis server.

DataChanged(*tag*,*value*)

Indicates that the data in the CloudDB project has changed. Launches an event with the tag that has been updated and the value it now has.

FirstRemoved(*value*)
Event triggered by the RemoveFirstFromList function. The argument value is the object that was the first in the list, and which is now removed.

GotValue(*tag*,*value*)
Indicates that a GetValue request has succeeded.

TagList(*value*)
Event triggered when we have received the list of known tags. Run in response to a call to the GetTagList function.

Methods

AppendValueToList(*tag*,*itemToAdd*)
Append a value to the end of a list atomically. If two devices use this function simultaneously, both will be appended and no data lost.

ClearTag(*tag*)
Remove the tag from CloudDB.

CloudConnected()
Returns true if we are on the network and will likely be able to connect to the CloudDB server.

GetTagList()
Asks CloudDB to retrieve all the tags belonging to this project. The resulting list is returned in the event TagList.

GetValue(*tag*,*valueIfTagNotThere*)
GetValue asks CloudDB to get the value stored under the given tag. It will pass the result to the GotValue will be given.

RemoveFirstFromList(*tag*)
Obtain the first element of a list and atomically remove it. If two devices use this function simultaneously, one will get the first element and the the other will get the second element, or an error if there is no available element. When the element is available, the FirstRemoved event will be triggered.

StoreValue(*tag*,*valueToStore*)

Asks CloudDB to store the given value under the given tag.

File

Non-visible component for storing and retrieving files. Use this component to write or read files on the device. The default behavior is to write files to the private data directory associated with the app. The Companion writes files to /sdcard/AppInventor/data for easy debugging. If the file path starts with a slash (/), then the file is created relative to /sdcard. For example, writing a file to /myFile.txt will write the file in /sdcard/myFile.txt.

Properties

LegacyMode
Allows app to access files from the root of the external storage directory (legacy mode). Starting with Android 11, this will no longer be allowed and the behavior is strongly discouraged on Android 10. Starting with Android 10, App Inventor by default will attempt to store files relative to the app-specific private directory on external storage in accordance with this security change.

Note: Apps that enable this property will likely stop working after upgrading to Android 11, which strongly enforces that apps only write to app-private directories.

Events

AfterFileSaved(*fileName*)
Event indicating that the contents of the file have been written.

GotText(*text*)
Event indicating that the contents from the file have been read.

Methods

AppendToFile(*text*,*fileName*)
Appends text to the end of a file. Creates the file if it does not already exist. See the help text under SaveFile for information about where files are written. On success, the AfterFileSaved event will run.

Delete(*fileName*)
Deletes a file from storage. Prefix the fileName with / to delete a specific file in the SD card (for example, /myFile.txt will delete the file /sdcard/myFile.txt). If the fileName does

not begin with a /, then the file located in the program's private storage will be deleted. Starting the fileName with // is an error because asset files cannot be deleted.

ReadFrom(*fileName*)

Reads text from a file in storage. Prefix the fileName with / to read from a specific file on the SD card (for example, /myFile.txt will read the file /sdcard/myFile.txt). To read assets packaged with an application (also works for the Companion) start the fileName with // (two slashes). If a fileName does not start with a slash, it will be read from the application's private storage (for packaged apps) and from /sdcard/AppInventor/data for the Companion.

SaveFile(*text*,*fileName*)

Saves text to a file. If the fileName begins with a slash (/) the file is written to the sdcard (for example, writing to /myFile.txt will write the file to /sdcard/myFile.txt). If the fileName does not start with a slash, it will be written in the program's private data directory where it will not be accessible to other programs on the phone. There is a special exception for the AI Companion where these files are written to /sdcard/AppInventor/data to facilitate debugging.

Note that this block will overwrite a file if it already exists. If you want to add content to an existing file use the AppendToFile method.

TinyDB

TinyDB is a non-visible component that stores data for an app.

Apps created with App Inventor are initialized each time they run. This means that if an app sets the value of a variable and the user then quits the app, the value of that variable will not be remembered the next time the app is run. In contrast, TinyDB is a persistent data store for the app. The data stored in a TinyDB will be available each time the app is run. An example might be a game that saves the high score and retrieves it each time the game is played.

Data items consist of tags and values. To store a data item, you specify the tag it should be stored under. The tag must be a text block, giving the data a name. Subsequently, you can retrieve the data that was stored under a given tag.

You cannot use the TinyDB to pass data between two different apps on the phone, although you can use the TinyDB to share data between the different screens of a multi-screen app.

When you are developing apps using the AI Companion, all the apps using that Companion will share the same TinyDB. That sharing will disappear once the apps are packaged and installed on the phone. During development you should be careful to clear the Companion app’s data each time you start working on a new app.

Properties

Namespace
Namespace for storing data.

Events

None

Methods

ClearAll()
Clear the entire data store.

ClearTag(*tag*)
Clear the entry with the given tag.

GetTags()
Return a list of all the tags in the data store.

GetValue(*tag*,*valueIfTagNotThere*)
Retrieve the value stored under the given tag. If there’s no such tag, then return valueIfTagNotThere.

StoreValue(*tag*,*valueToStore*)
Store the given valueToStore under the given tag. The storage persists on the phone when the app is restarted.

TinyWebDB

The TinyWebDB component communicates with a Web service to store and retrieve information. Although this component is usable, it is very limited and meant primarily as a demonstration for people who would like to create their own components that talk to the Web. The accompanying Web service is at (http://tinywebdb.appinventor.mit.edu). The component has methods to store a value under a tag and to retrieve the value associated with the tag. The interpretation of what “store” and “retrieve” means is up to the Web service. In this implementation, all tags and values are strings (text). This restriction may be relaxed in future versions.

Properties

ServiceURL
Specifies the URL of the Web service. The default value is the demo service running on App Engine.

Events

GotValue(*tagFromWebDB*,*valueFromWebDB*)
Indicates that a GetValue server request has succeeded.

ValueStored()
Event indicating that a StoreValue server request has succeeded.

WebServiceError(*message*)
Indicates that the communication with the Web service signaled an error.

Methods

GetValue(*tag*)
GetValue asks the Web service to get the value stored under the given tag. It is up to the Web service what to return if there is no value stored under the tag. This component just accepts whatever is returned. The GotValue event will be run on completion.

StoreValue(*tag*,*valueToStore*)
Sends a request to the Web service to store the given valueToStore under the given tag. The ValueStored event will be run on completion.

iii. Using emulator

If you do not have an Android phone or tablet, you can still build apps with App Inventor. App Inventor provides an Android emulator, which works just like an Android but appears on your computer screen. So you can test your apps on an emulator and still distribute the app to others, even through the Play Store. Some schools and after-school programs develop primarily on emulators and provide a few Androids for final testing.

Build your project on your computer
Test it in real-time on your computer with the onscreen emulator

To use the emulator, you will first need to first install some software on your computer (this is not required for the wifi solution). Follow the instructions below for your operating system, then come back to this page to move on to starting the emulator

Important: If you are updating a previous installation of the App Inventor software, see How to update the App Inventor Software. You can check whether your computer is running the latest version of the software by visiting the page App Inventor 2 Connection Test.

Step 1. Install the App Inventor Setup Software

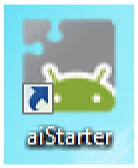
- Instructions for Mac OS X
- Instructions for Windows

- Instructions for GNU/Linux

Step 2. Launch aiStarter (Windows & GNU/Linux only)

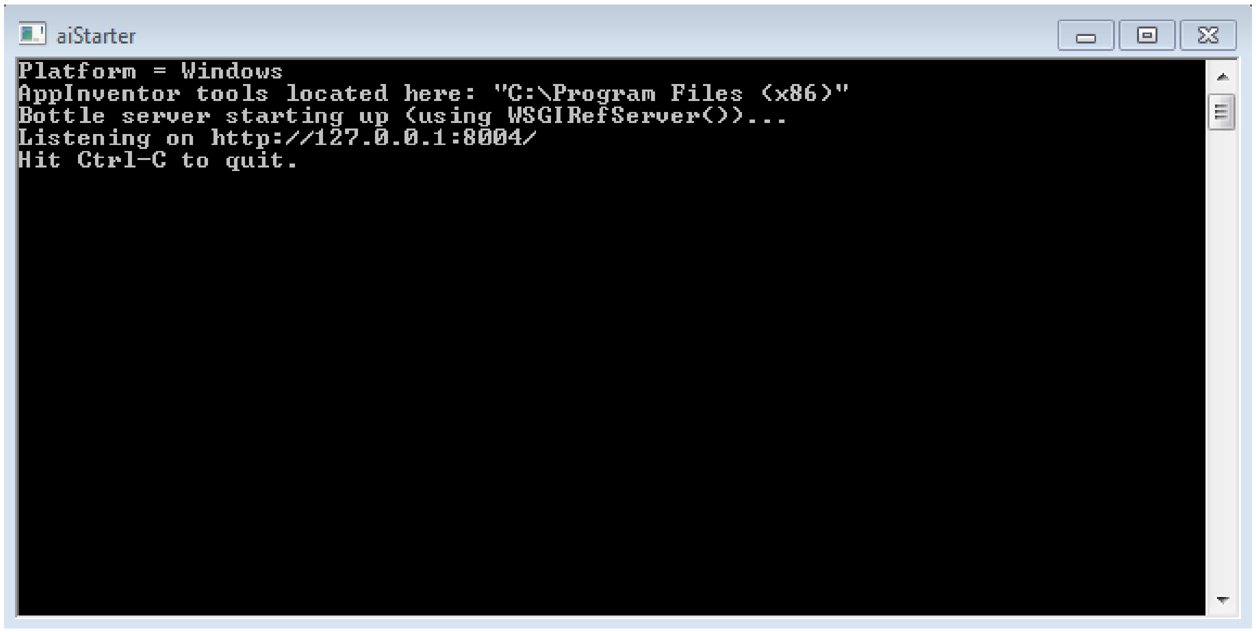
Using the emulator or the USB cable requires the use of a program named *aiStarter*. This program is the helper that permits the browser to communicate with the emulator or USB cable. The aiStarter program was installed when you installed the App Inventor Setup package. You do not need aiStarter if you are using only the wireless companion.

- On a Mac, aiStarter will start automatically when you log in to your account and it will run invisibly in the background.
- On Windows, there will be shortcuts to aiStarter from your Desktop, from the Start menu, from All Programs and from Startup Folder. If you want to use the emulator with App Inventor, you will need to manually launch aiStarter on your computer when you log in. You can start aiStarter this by clicking the icon on your desktop or using the entry in your start menu.

  The aiStarter Icon on Windows

  To launch aiStarter on Windows, double click on the icon (shown above). You'll know that you've successfully launched aiStarter when you see a window like the following:



```
Platform = Windows
AppInventor tools located here: "C:\Program Files (x86)"
Bottle server starting up (using WSGIRefServer())...
Listening on http://127.0.0.1:8004/
Hit Ctrl-C to quit.
```

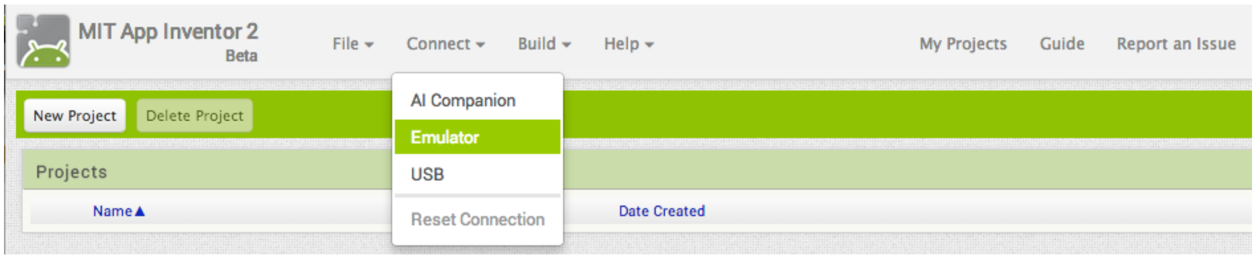
- On GNU/Linux, aiStarter will be in the folder /usr/google/appinventor/commands-for-Appinventor and you'll need to launch it manually. You can launch it from the command line with /usr/google/appinventor/commands-for-appinventor/aiStarter &

For Help with aiStarter, see Connection Help.

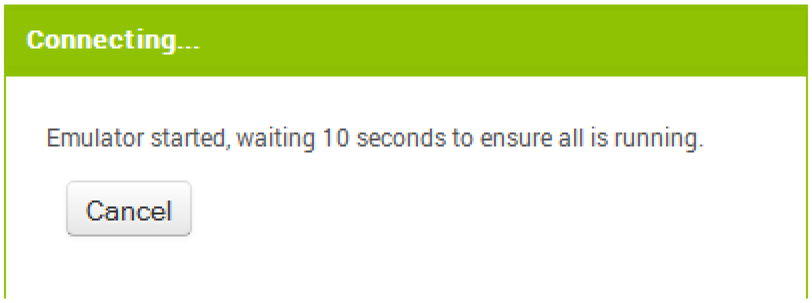
Step 3. Open an App Inventor project and connect it to the emulator

First, go to App Inventor and open a project (or create a new one -- use *Project > Start New Project* and give your project a name).

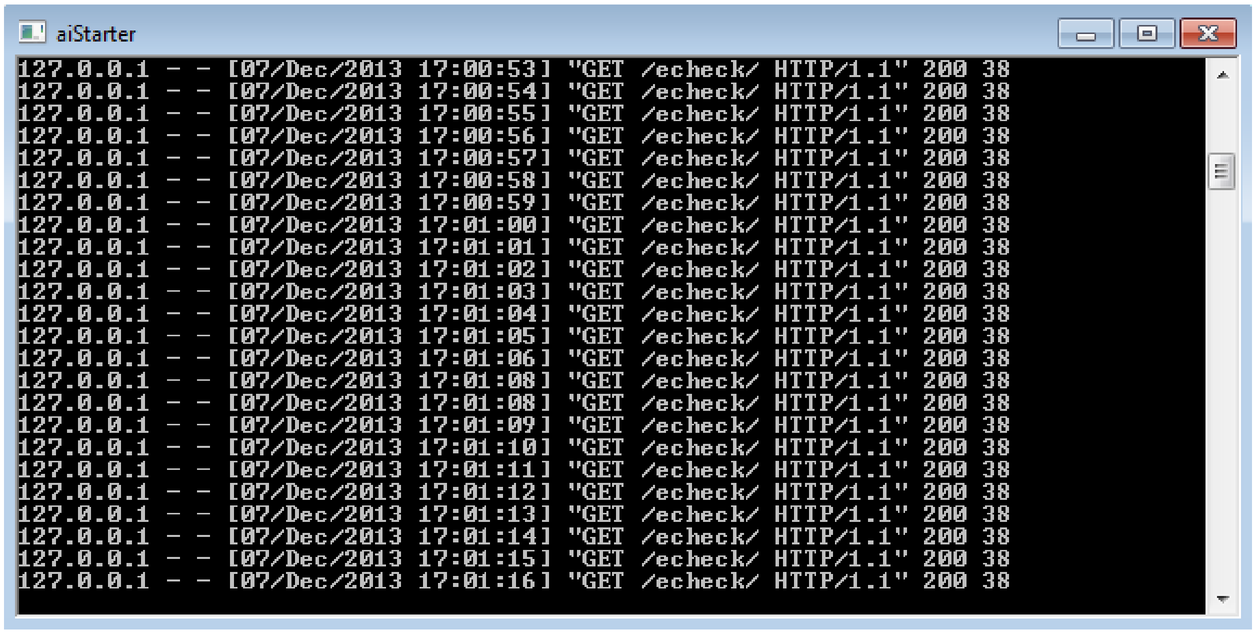
Then, from App Inventor's menu (on the App Inventor cloud-based software at ai2.appinventor.mit.edu), go to the Connect Menu and click the Emulator option.



You'll get a notice saying that the emulator is connecting. Starting the emulator can take a couple of minutes. You may see update screens like the following as the emulator starts up:



Emulator started, waiting 10 seconds to ensure all is running.

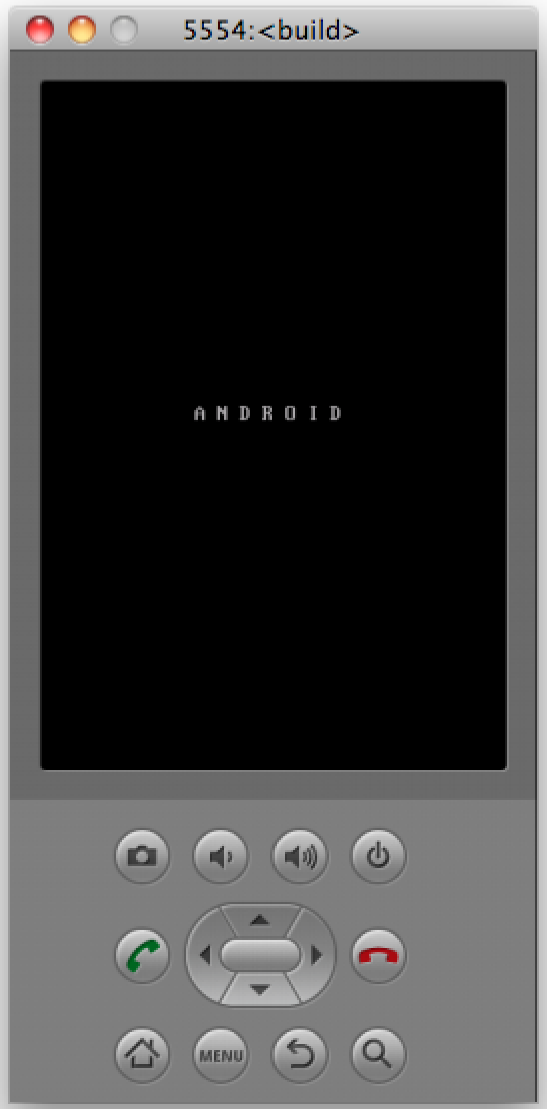


```
127.0.0.1 - - [07/Dec/2013 17:00:53] "GET /echeck/ HTTP/1.1" 200 38
127.0.0.1 - - [07/Dec/2013 17:00:54] "GET /echeck/ HTTP/1.1" 200 38
127.0.0.1 - - [07/Dec/2013 17:00:55] "GET /echeck/ HTTP/1.1" 200 38
127.0.0.1 - - [07/Dec/2013 17:00:56] "GET /echeck/ HTTP/1.1" 200 38
127.0.0.1 - - [07/Dec/2013 17:00:57] "GET /echeck/ HTTP/1.1" 200 38
127.0.0.1 - - [07/Dec/2013 17:00:58] "GET /echeck/ HTTP/1.1" 200 38
127.0.0.1 - - [07/Dec/2013 17:00:59] "GET /echeck/ HTTP/1.1" 200 38
127.0.0.1 - - [07/Dec/2013 17:01:00] "GET /echeck/ HTTP/1.1" 200 38
127.0.0.1 - - [07/Dec/2013 17:01:01] "GET /echeck/ HTTP/1.1" 200 38
127.0.0.1 - - [07/Dec/2013 17:01:02] "GET /echeck/ HTTP/1.1" 200 38
127.0.0.1 - - [07/Dec/2013 17:01:03] "GET /echeck/ HTTP/1.1" 200 38
127.0.0.1 - - [07/Dec/2013 17:01:04] "GET /echeck/ HTTP/1.1" 200 38
127.0.0.1 - - [07/Dec/2013 17:01:05] "GET /echeck/ HTTP/1.1" 200 38
127.0.0.1 - - [07/Dec/2013 17:01:06] "GET /echeck/ HTTP/1.1" 200 38
127.0.0.1 - - [07/Dec/2013 17:01:08] "GET /echeck/ HTTP/1.1" 200 38
127.0.0.1 - - [07/Dec/2013 17:01:08] "GET /echeck/ HTTP/1.1" 200 38
127.0.0.1 - - [07/Dec/2013 17:01:09] "GET /echeck/ HTTP/1.1" 200 38
127.0.0.1 - - [07/Dec/2013 17:01:10] "GET /echeck/ HTTP/1.1" 200 38
127.0.0.1 - - [07/Dec/2013 17:01:11] "GET /echeck/ HTTP/1.1" 200 38
127.0.0.1 - - [07/Dec/2013 17:01:12] "GET /echeck/ HTTP/1.1" 200 38
127.0.0.1 - - [07/Dec/2013 17:01:13] "GET /echeck/ HTTP/1.1" 200 38
127.0.0.1 - - [07/Dec/2013 17:01:14] "GET /echeck/ HTTP/1.1" 200 38
127.0.0.1 - - [07/Dec/2013 17:01:15] "GET /echeck/ HTTP/1.1" 200 38
127.0.0.1 - - [07/Dec/2013 17:01:16] "GET /echeck/ HTTP/1.1" 200 38
```

The emulator will initially appear with an empty black screen (#1). Wait until the emulator is ready, with a colored screen background (#2). Even after the background appears, you should wait until the emulated phone has finished preparing its SD card: there will be a notice at the top of the phone screen while the card is being prepared. When connected, the emulator will launch and show the app you have open in App Inventor.

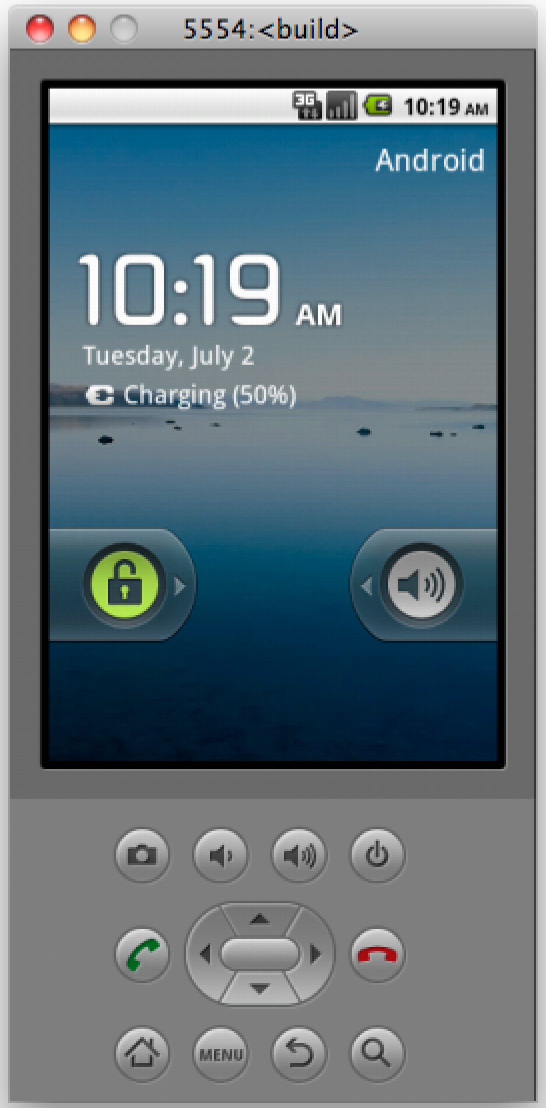
If this is the first time you are using the emulator after installing the App Inventor Setup software, you will see a message asking you to update the emulator. Follow the directions on the screen to perform the update and reconnect the emulator. You will need to do this kind of update whenever there is a new version of the App Inventor software.

For problems with aiStarter, or if the Emulator does not connect, go to Connection Help for information about what might be wrong.

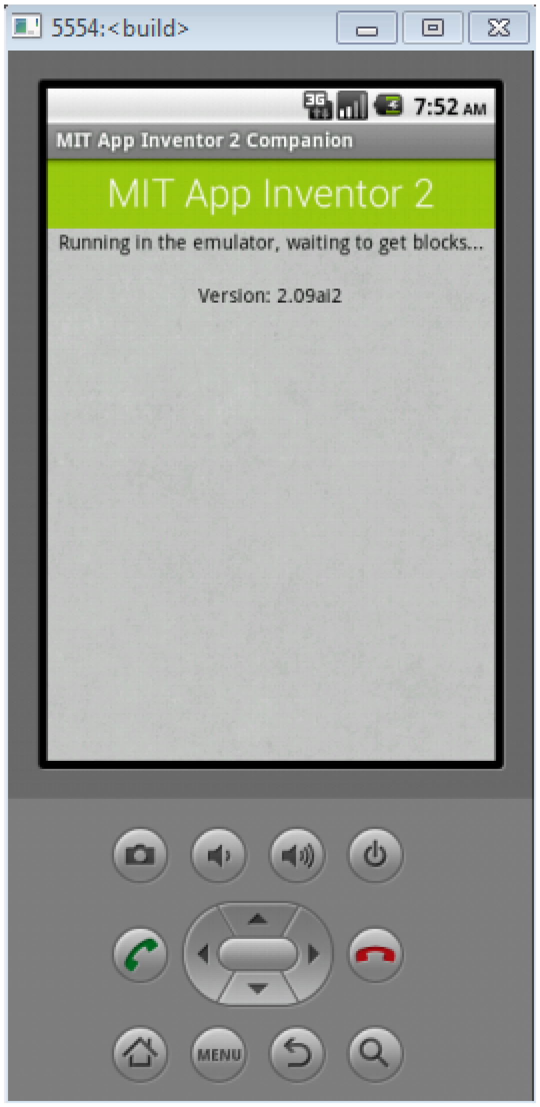


#1

#2



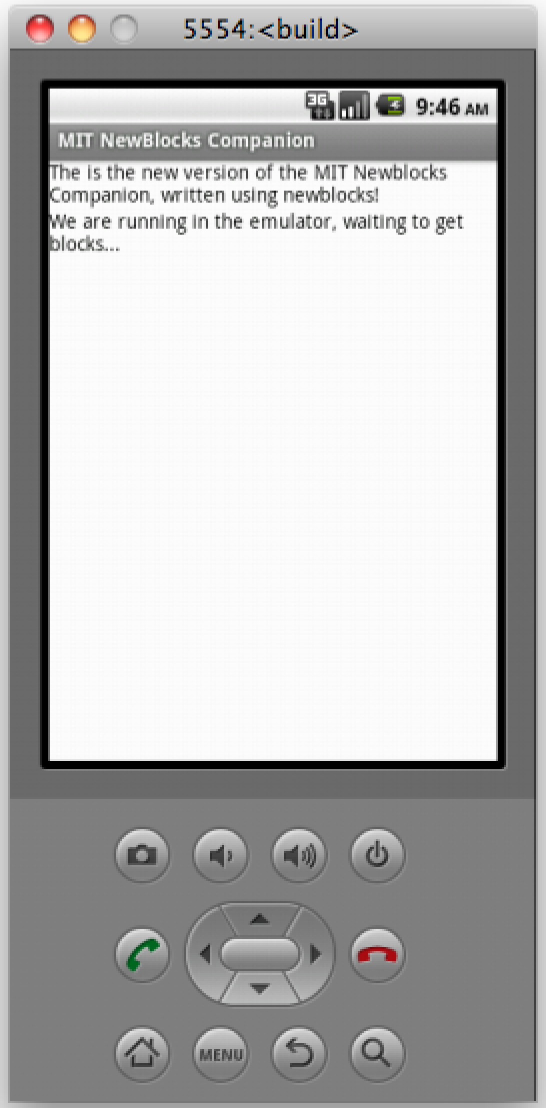
#3



Running in the emulator, waiting to get blocks...

Version: 2.09ai2

#4



The is the new version of the MIT Newblocks Companion, written using newblocks!
We are running in the emulator, waiting to get blocks...

Setup complete! You are now ready to build your first app!

f. Sensor

i. Orientation sensor

Use an orientation sensor component to determine the phone's spatial orientation.

An orientation sensor is a non-visible component that reports the following three values, in degrees:

- Roll : 0 degree when the device is level, increasing to 90 degrees as the device is tilted up onto its left side, and decreasing to −90 degrees when the device is tilted up onto its right side.
- Pitch : 0 degree when the device is level, increasing to 90 degrees as the device is tilted so its top is pointing down, then decreasing to 0 degree as it gets turned over. Similarly, as the device is tilted so its bottom points down, pitch decreases to −90 degrees, then increases to 0 degree as it gets turned all the way over.
- Azimuth : 0 degree when the top of the device is pointing north, 90 degrees when it is pointing east, 180 degrees when it is pointing south, 270 degrees when it is pointing west, etc.

These measurements assume that the device itself is not moving.

Properties

Angle
Returns an angle that tells the direction in which the device is tiled. That is, it tells the direction of the force that would be felt by a ball rolling on the surface of the device.

Available
Indicates whether the orientation sensor is present on the device.

Azimuth
Returns the azimuth angle of the device. To return meaningful values the sensor must be enabled.

Enabled
Specifies whether the orientation sensor is enabled.

Magnitude
Returns a number between 0 and 1 indicating how much the device is tilted. It gives the magnitude of the force that would be felt by a ball rolling on the surface of the device. For the angle of tilt, use Angle.

Pitch
Returns the pitch angle of the device. To return meaningful values the sensor must be enabled.

Roll
Returns the roll angle of the device. To return meaningful values the sensor must be enabled.

Events

OrientationChanged(*azimuth*,*pitch*,*roll*)
The OrientationChanged event handler is run when the orientation has changed.

ii. Location sensor

Non-visible component providing location information, including Latitude, Longitude, Altitude (if supported by the device), speed (if supported by the device), and address. This can also perform “geocoding”, converting a given address (not necessarily the current one) to a latitude (with the LatitudeFromAddress method) and a longitude (with the LongitudeFromAddress method).

In order to function, the component must have its Enabled property set to true, and the device must have location sensing enabled through wireless networks or GPS satellites (if outdoors).

Location information might not be immediately available when an app starts. You’ll have to wait a short time for a location provider to be found and used, or wait for the LocationChanged event.

The emulator does not emulate sensors on all devices. Code should be tested on a physical device.

Properties

Accuracy
The LocationSensor will be able to locate the device with a varying degree of confidence, based on the quality of satellite, cell towers, and other data used to estimate location. The Accuracy value is the radius in meters around the sensor’s detected location. The device has a 68% chance to be located within this radius. More precise location detection will result in a smaller accuracy number, which allows the app to have more confidence where the device is actually located.

If the accuracy is not known, the return value is 0.0

Altitude
Altitude of the device measured in meters, if available.

Altitude is measured from the World Geodetic System 84 reference ellipsoid, not sea level.

Note that it is difficult for devices to accurately sense altitude. Altitude reported on a phone/tablet can easily be off by 30 meters or more.

AvailableProviders
List of available service providers, such as gps or network. This information is provided as a list and in text form.

CurrentAddress
Physical street address of the device from Google’s map database.

The address might not always be available from the provider, and the address reported may not always be of the building where the device is located.

If Google has no address information available for a particular location, this will return No address available.

DistanceInterval
Determines the minimum distance interval, in meters, that the sensor will try to use for sending out location updates. For example, if this is set to 50, then the sensor will fire a LocationChanged event only after 50 meters have been traversed. However, the sensor does not guarantee that an update will be received at exactly the distance interval. It may take more than 5 meters to fire an event, for instance.

It is also useful to check against Accuracy when using this property. When your device is moving, the accuracy of the detected location is constantly changing.

Enabled
If true, the LocationSensor will attempt to read location information from GPS, WiFi location, or other means available on the device. This setting does not control whether location information is actually available. Device location must be enabled or disabled in the device settings.

HasAccuracy
If true, the device can report its accuracy level.

HasAltitude
If true, the device can report its altitude.

HasLongitudeLatitude
If true, the device can report longitude and latitude. It is always the case that either both or neither are.

Latitude
The most recently available latitude value in degrees reported to 5 decimal places. If no value is available, 0 will be returned. Latitude is a value between 90 (north) and -90 (south), where 0 marks the Equator.

Longitude
The most recent available longitude value in degrees reported to 5 decimal places. If no value is available, 0 will be returned. Longitude is a value between 180 (east) and -180 (west), where 0 marks the Prime Meridian.

ProviderLocked
The device will not change the service provider.

It is possible for a device to switch service providers when the current provider is unable to provide adequate location information. ProviderLocked is a Boolean value: true/false. Set to true to prevent providers from changing. Set to false to allow for automatic switching when necessary.

ProviderName
The current service provider. The provider will most likely be either GPS or network.

TimeInterval
Determines the minimum time interval, in milliseconds, that the sensor will try to use for sending out location updates. However, location updates will only be received when the location of the phone actually changes, and use of the specified time interval is not guaranteed. For example, if 30000 is used as the time interval, location updates will never be fired sooner than 30000ms, but they may be fired anytime after.

Values smaller than 30000ms (30 seconds) are not practical for most devices. Small values may drain battery and overwork the GPS.

Events

LocationChanged(*latitude*,*longitude*,*altitude*,*speed*)
Indicates that a new location has been detected. Speed is reported in meters/second Other values match their properties.

StatusChanged(*provider*,*status*)
Indicates that the status of the location provider service has changed, such as when a provider is lost or a new provider starts being used.

Methods

LatitudeFromAddress(*locationName*)
Derives latitude from the given locationName.

LongitudeFromAddress(*locationName*)
Derives longitude from the given locationName.

Day 4

g. Screen Layout

HorizontalArrangement

Use a horizontal arrangement component to display a group of components laid out from left to right.

This component is a formatting element in which you place components that should be displayed from left to right. If you want to have components displayed one over another, use VerticalArrangement instead.

In a HorizontalArrangement, components are arranged along the horizontal axis, vertically center-aligned.

If a HorizontalArrangement's Height property is set to Automatic, the actual height of the arrangement is determined by the tallest component in the arrangement whose Height property is not set to Fill Parent. If a HorizontalArrangment's Height property is set to Automatic and it contains only components whose Height properties are set to Fill Parent, the actual height of the arrangement is calculated using the automatic heights of the components. If a HorizontalArrangement's Height property is set to Automatic and it is empty, the Height will be 100.

If a HorizontalArrangement's Width property is set to Automatic, the actual width of the arrangement is determined by the sum of the widths of the components. If a HorizontalArrangement's Width property is set to Automatic, any components whose Width properties are set to Fill Parent will behave as if they were set to Automatic.

If a HorizontalArrangement's Width properties are set to Fill Parent will equally take up the width not occupied by other components.

Properties

AlignHorizontal

A number that encodes how contents of the HorizontalArrangement are aligned horizontally. The choices are: 1 = left aligned, 2 = right aligned, 3 = horizontally centered. Alignment has no effect if the HorizontalArrangement's Width is Automatic.

AlignVertical

A number that encodes how the contents of the HorizontalArrangement are aligned vertically. The choices are: 1 = aligned at the top, 2 = aligned at the bottom, 3 = vertically centered. Alignment has no effect if the HorizontalArrangement's Height is Automatic.

BackgroundColor

Specifies the background color of the HorizontalArrangement as an alpha-red-green-blue integer. If an Image has been set, the color change will not be visible until the Image is removed.

Height

Specifies the HorizontalArrangement's vertical height, measured in pixels.

HeightPercent

Specifies the HorizontalArrangement's vertical height as a percentage of the Screen's Height.

Image

Specifies the path of the background image of the HorizontalArrangement.

Visible

Specifies whether the HorizontalArrangement should be visible on the screen. Value is true if the HorizontalArrangement is showing and false if hidden.

Width

Specifies the horizontal width of the HorizontalArrangement, measured in pixels.

WidthPercent

Specifies the horizontal width of the HorizontalArrangement as a percentage of the Screen's Width.

Events

None

Methods

None

HorizontalScrollArrangement

A formatting element in which to place components that should be displayed from left to right. If you wish to have components displayed one over another, use VerticalScrollArrangement instead.

This version is scrollable.

Properties

AlignHorizontal

A number that encodes how contents of the HorizontalScrollArrangement are aligned horizontally. The choices are: 1 = left aligned, 2 = right aligned, 3 = horizontally centered. Alignment has no effect if the HorizontalScrollArrangement's Width is Automatic.

AlignVertical

A number that encodes how the contents of the HorizontalScrollArrangement are aligned vertically. The choices are: 1 = aligned at the top, 2 = aligned at the bottom, 3 = vertically centered. Alignment has no effect if the HorizontalScrollArrangement's Height is Automatic.

BackgroundColor

Specifies the background color of the HorizontalScrollArrangement as an alpha-red-green-blue integer. If an Image has been set, the color change will not be visible until the Image is removed.

Height

Specifies the HorizontalScrollArrangement's vertical height, measured in pixels.

HeightPercent

Specifies the HorizontalScrollArrangement's vertical height as a percentage of the Screen's Height.

Image

Specifies the path of the background image of the HorizontalScrollArrangement.

Visible

Specifies whether the HorizontalScrollArrangement should be visible on the screen. Value is true if the HorizontalScrollArrangement is showing and false if hidden.

Width

Specifies the horizontal width of the HorizontalScrollArrangement, measured in pixels.

WidthPercent

Specifies the horizontal width of the HorizontalScrollArrangement as a percentage of the Screen's Width.

Events

None

Methods

None

TableArrangement

Use a table arrangement component to display a group of components in a tabular fashion.

This component is a formatting element in which you place components that should be displayed in tabular form.

In a TableArrangement, components are arranged in a grid of rows and columns, with not more than one component visible in each cell. If multiple components occupy the same cell, only the last one will be visible.

Within each row, components are vertically center-aligned.

The width of a column is determined by the widest component in that column. When calculating column width, the automatic width is used for components whose Width property is set to Fill Parent. However, each component will always fill the full width of the column that it occupies.

The height of a row is determined by the tallest component in that row whose Height property is not set to Fill Parent. If a row contains only components whose Height properties are set to Fill Parent, the height of the row is calculated using the automatic heights of the components.

Properties

Columns

Determines the number of columns in the table.

Height

Specifies the TableArrangement’s vertical height, measured in pixels.

HeightPercent

Specifies the TableArrangement’s vertical height as a percentage of the Screen’s Height.

Rows

Determines the number of rows in the table.

Visible

Specifies whether the TableArrangement should be visible on the screen. Value is true if the TableArrangement is showing and false if hidden.

Width

Specifies the horizontal width of the TableArrangement, measured in pixels.

WidthPercent

Specifies the horizontal width of the TableArrangement as a percentage of the Screen’s Width.

Events

None

Methods

None

VerticalArrangement

Use a VerticalArrangement component to display a group of components laid out from top to bottom, left-aligned.

This component is a formatting element in which you place components that should be displayed one below another. The first child component is stored on top, the second beneath it, and so on. If you want to have components displayed next to one another, use HorizontalArrangement instead.

In a VerticalArrangement, components are arranged along the vertical axis, left-aligned.

If a VerticalArrangement's Width property is set to Automatic, the actual width of the arrangement is determined by the widest component in the arrangement whose Width property is not set to Fill Parent. If a VerticalArrangement's Width property is set to Automatic and it contains only components whose Width properties are set to Fill Parent, the actual width of the arrangement is calculated using the automatic widths of the components. If a VerticalArrangement's Width property is set to Automatic and it is empty, the width will be 100.

If a VerticalArrangement's Height property is set to Automatic, the actual height of the arrangement is determined by the sum of the heights of the components. If a VerticalArrangement's Height property is set to Automatic, any components whose Height properties are set to Fill Parent will behave as if they were set to Automatic.

If a VerticalArrangement's Height property is set to Fill Parent or specified in pixels, any components whose Height properties are set to Fill Parent will equally take up the height not occupied by other components.

Properties

AlignHorizontal

A number that encodes how contents of the VerticalArrangement are aligned horizontally. The choices are: 1 = left aligned, 2 = right aligned, 3 = horizontally centered. Alignment has no effect if the VerticalArrangement's Width is Automatic.

AlignVertical

A number that encodes how the contents of the VerticalArrangement are aligned vertically. The choices are: 1 = aligned at the top, 2 = aligned at the bottom, 3 = vertically centered. Alignment has no effect if the VerticalArrangement's Height is Automatic.

BackgroundColor

Specifies the background color of the VerticalArrangement as an alpha-red-green-blue integer. If an Image has been set, the color change will not be visible until the Image is removed.

Height

Specifies the VerticalArrangement's vertical height, measured in pixels.

HeightPercent

Specifies the VerticalArrangement's vertical height as a percentage of the Screen's Height.

Image

Specifies the path of the background image of the VerticalArrangement.

Visible

Specifies whether the VerticalArrangement should be visible on the screen. Value is true if the VerticalArrangement is showing and false if hidden.

Width

Specifies the horizontal width of the VerticalArrangement, measured in pixels.

WidthPercent

Specifies the horizontal width of the VerticalArrangement as a percentage of the Screen's Width.

Events

None

Methods

None

VerticalScrollArrangement

A formatting element in which to place components that should be displayed one below another. (The first child component is stored on top, the second beneath it, etc.) If you wish to have components displayed next to one another, use HorizontalScrollArrangement instead.

This version is scrollable.

Properties

AlignHorizontal

A number that encodes how contents of the VerticalScrollArrangement are aligned horizontally. The choices are: 1 = left aligned, 2 = right aligned, 3 = horizontally centered. Alignment has no effect if the VerticalScrollArrangement's Width is Automatic.

AlignVertical

A number that encodes how the contents of the VerticalScrollArrangement are aligned vertically. The choices are: 1 = aligned at the top, 2 = aligned at the bottom, 3 = vertically centered. Alignment has no effect if the VerticalScrollArrangement's Height is Automatic.

BackgroundColor

Specifies the background color of the VerticalScrollArrangement as an alpha-red-green-blue integer. If an Image has been set, the color change will not be visible until the Image is removed.

Height

Specifies the VerticalScrollArrangement's vertical height, measured in pixels.

HeightPercent

Specifies the VerticalScrollArrangement's vertical height as a percentage of the Screen's Height.

Image

Specifies the path of the background image of the VerticalScrollArrangement.

Visible

Specifies whether the VerticalScrollArrangement should be visible on the screen. Value is true if the VerticalScrollArrangement is showing and false if hidden.

Width

Specifies the horizontal width of the VerticalScrollArrangement, measured in pixels.

WidthPercent

Specifies the horizontal width of the VerticalScrollArrangement as a percentage of the Screen's Width.

Events

None

Methods

None

i. List view

This is a visible component that allows to place a list of text elements in your Screen to display. The list can be set using the ElementsFromString property or using the Elements block in the blocks editor.

Warning: This component will not work correctly on Screens that are scrollable if its Height is set to Fill Parent.

Properties

BackgroundColor
The color of the ListView background.

Elements
Specifies the list of choices to display.

ElementsFromString
Set the list of choices from a string of comma-separated values.

Height
Specifies the ListView's vertical height, measured in pixels.

HeightPercent
Specifies the ListView's vertical height as a percentage of the Screen's Height.

Selection
Returns the text in the ListView at the position of SelectionIndex.

SelectionColor
The color of the item when it is selected.

SelectionIndex
The index of the currently selected item, starting at 1. If no item is selected, the value will be 0. If an attempt is made to set this to a number less than 1 or greater than the number of items in the ListView, SelectionIndex will be set to 0, and Selection will be set to the empty text.

ShowFilterBar
Sets visibility of the filter bar. true will show the bar, false will hide it.

TextColor
The text color of the ListView items.

TextSize

Specifies the ListView item's text font size

Visible
Specifies whether the ListView should be visible on the screen. Value is true if the ListView is showing and false if hidden.

Width
Specifies the horizontal width of the ListView, measured in pixels.

WidthPercent
Specifies the horizontal width of the ListView as a percentage of the Screen's Width.

Events

AfterPicking()
Simple event to be raised after the an element has been chosen in the list. The selected element is available in the Selection property.

ii. Lists and dictionaries : Data collection

Dictionaries, called in other languages terms such as maps, associative arrays or lists, are data structures that associate one value, often called the key, with another value. A common way of displaying dictionaries is using the JavaScript Object Notation (JSON), for example:

```
{
  "id":  1,
  "name":  "Tim the Beaver",
  "school": {
    "name": "Massachusetts Institute of Technology"
  },
  "enrolled": true,
  "classes": ["6.001", "18.01", "8.01"]
}
```

The above example shows that in JSON the keys (quoted text before the :) can map to different types of values. The allowed types are number, text, other dictionaries, booleans, and lists. In the blocks language, you can bulid this dictionary as follows:

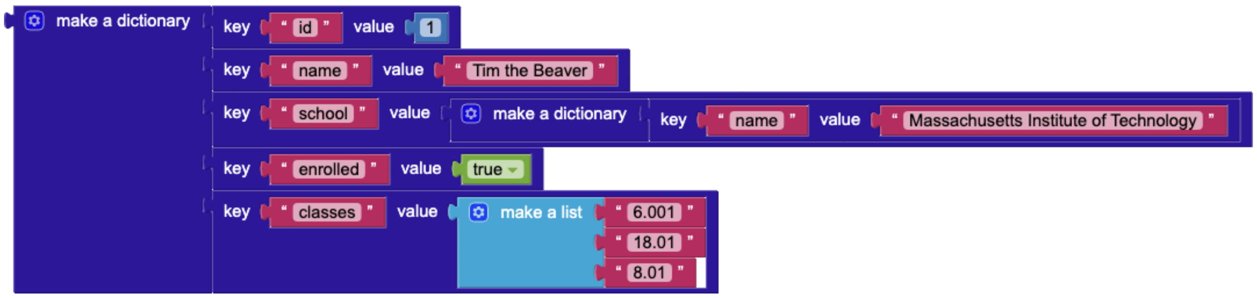

make a dictionary
key "id" value 1
key "name" value "Tim the Beaver"
key "school" value make a dictionary key "name" value "Massachusetts Institute of Technology"
key "enrolled" value true
key "classes" value make a list "6.001" "18.01" "8.01"

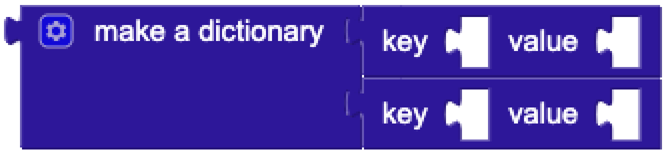
Figure 1: A blocks representation of the JSON code snippet shown above.

create empty dictionary

create empty dictionary

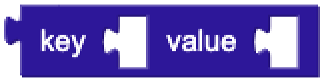
The create empty dictionary block creates a dictionary without any key-value pairs. Entries can be added to the empty dictionary using the set value for key block. The create empty dictionary block can also be turned into a make a dictionary block by using the blue mutator button to add pair entries.

make a dictionary



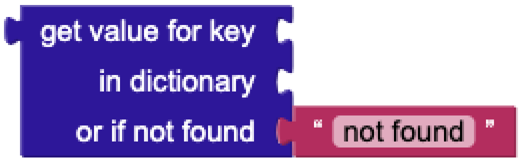
The make a dictionary is used to create a dictionary with a set of pairs known in advance. Additional entries can be added using set value for key.

pair



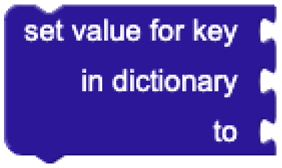
The pair block is a special purpose block used for constructing dictionaries.

get value for key



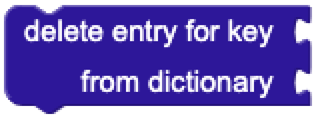
The get value for key block checks to see if the dictionary contains a corresponding value for the given key. If it does, the value is returned. Otherwise, the value of the not found parameter is returned. This behavior is similar to the behavior of the lookup in pairs block.

set value for key



The set value for key block sets the corresponding value for the given key in the dictionary to value. If no mapping exists for key, a new one will be created. Otherwise, the existing value is replaced by the new value.

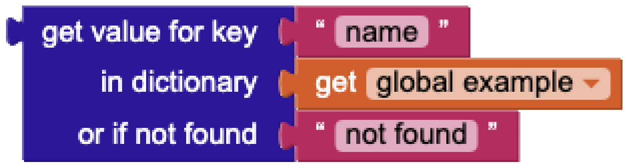
delete entry for key

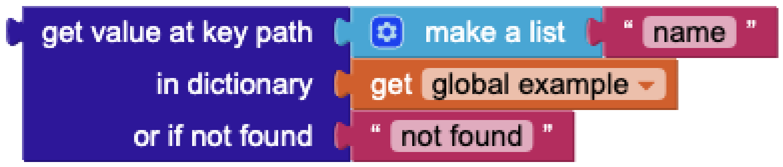


The delete entry for key block removes the key-value mapping in the dictionary for the given key. If no entry for the key exists in the dictionary, the dictionary is not modified.

get value at key path

The get value at key path block is a more advanced version of the get value for key block. Rather than getting the value of a specific key, it instead takes a list of valid keys and numbers representing a path through a data structure. The get value for key block is equivalent to using this block with a key path of length 1 containing the key. For example, the following two blocks would return "Tim the Beaver":



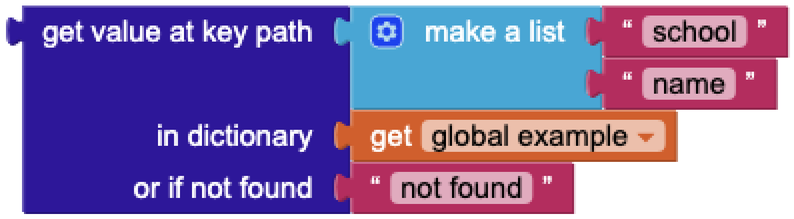


It walks the data structure, starting from the initial dictionary, using the path provided in order to retrieve values nested deeply in complex data structures. It is best used for processing JSON data from web services. Starting from the initial input, it takes the first element in the key path and checks to see if a key (if the input is a dictionary) or index (if the input is a list) exists at that point. If so, it selects that item as the input and proceeds to check the next element in the key path, continuing until either the whole path has been followed, at which point it returns what is at that location, or the "not found" parameter.

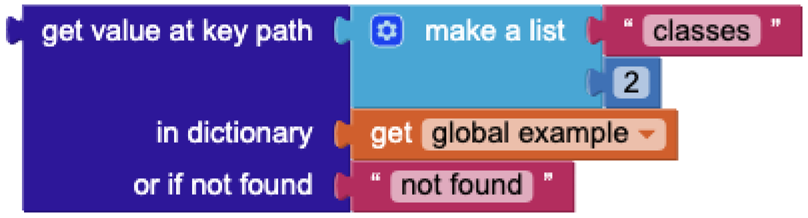
Examples

```
{
 "id":  1,
 "name":  "Tim the Beaver",
 "school": {
   "name": "Massachusetts Institute of Technology"
 },
 "enrolled": true,
 "classes": ["6.001", "18.01", "8.01"]
}
```

For example, given the JSON dictionary above, the following use of get value at key path will yield the result "Massachusetts Institute of Technology".

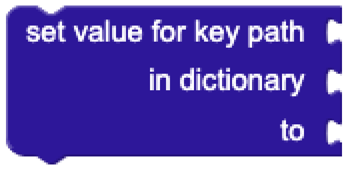


The get value at key path< allows for the path to include numbers representing the index of elements to traverse when dictionaries and lists are mixed. For example, if we wanted to know the second class that Tim was taking, we could do the following:



which returns the value "18.01".

set value for key path



The set value for key path block updates the value at a specific key path in a data structure. It is the mirror of get value for key path, which retrieves a value at a specific key path. The path must be valid, except for the last key, which if a mapping does not exist will create a mapping to the new value. Otherwise, the existing value is replaced with the new value.

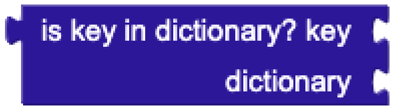
get keys

The get keys returns a list of keys in the dictionary.

get values

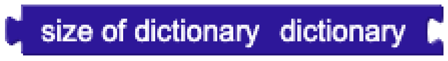
The get values returns a list containing the values in the dictionary. Modifying the contents of a value in the list will also modify it in the dictionary.

is key in dictionary?



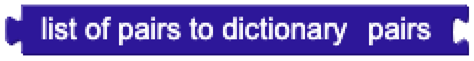
The is key in dictionary? tests whether the key exists in the dictionary and returns true if it does, otherwise it returns false.

size of dictionary



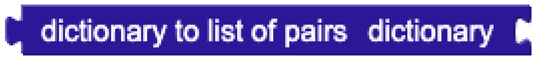
The size of dictionary block returns the number of key-value pairs present in the dictionary.

list of pairs to dictionary

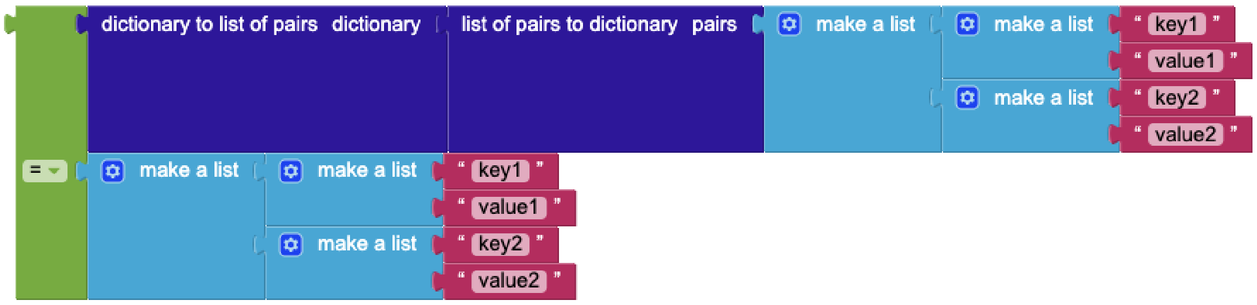


The list of pairs to dictionary block converts an associative list of the form ((key1 value1) (key2 value2) ...) into a dictionary mapping the keys to their values. Because dictionaries provide better lookup performance than associative lists, if you want to perform many operations on a data structure it is advisable to use this block to convert the associative list into a dictionary first.

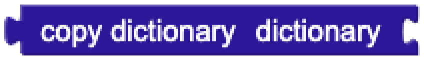
dictionary to list of pairs



The dictionary to list of pairs converts a dictionary into an associative list. This block reverses the conversion performed by the list of pairs to dictionary block.

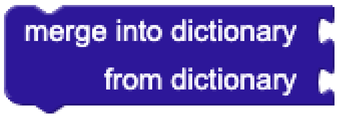


copy dictionary



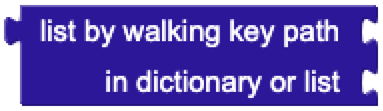
The copy dictionary makes a deep copy of the given dictionary. This means that all of the values are copied recursively and that changing a value in the copy will not change it in the original.

merge into dictionary



The merge into dictionary from dictionary block ccopies the key-value pairs from one dictionary into another, overwriting any keys in the target dictionary.

list by walking key path

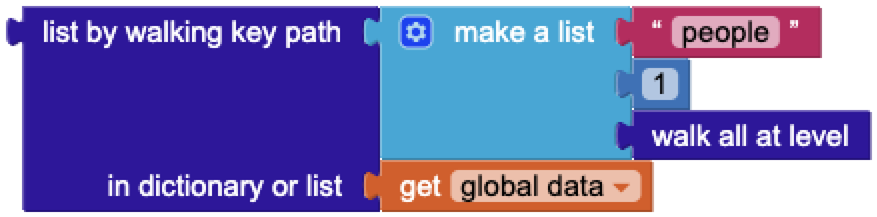


The list by walking key path block works similarly to the get value at key path, but creates a list of values rather than returning a single value. It works by starting at the given dictionary and walking down the tree of objects following the given path. Unlike the get value at key path though, its path can be composed of three major types: dictionary keys, list indices, and the walk all at level block. If a key or index is provided, the specific path is taken at that point in the tree. If the walk all at level is specified, every value at that point is followed in succession (breadth-first), at which point the walk continues from the next element in the path. Any element that matches the whole path is added to the output list.

Examples

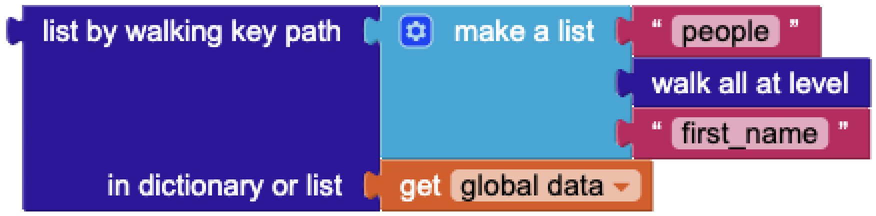
Consider the following JSON and blocks:

```
{
  "people": [{
    "first_name": "Tim",
    "last_name": "Beaver"
  },{
    "first_name": "John",
    "last_name": "Smith",
  },{
    "first_name": "Jane",
    "last_name": "Doe"
  }]
}
```



If global data contains a dictionary represented by the JSON, then the list by walking key path block will produce the list ["Tim", "Beaver"]. First, the value of the "people" tag, that is the list of people, is chosen. Next, the first element in the list is chosen. Lastly, the walk all at level block selects the values in the object at that point, that is, the values "Tim" and "Beaver".

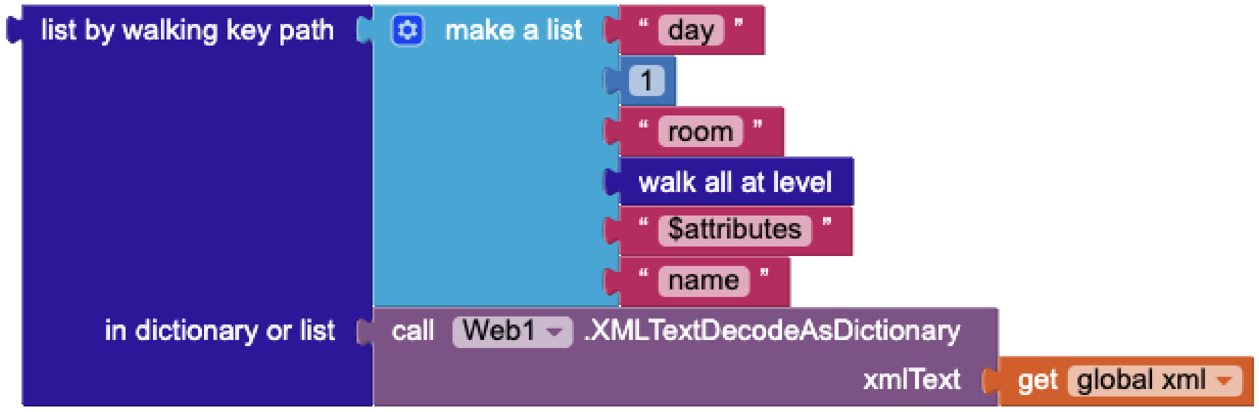
You can also use walk all at level at a level containing a list. For example, the following block selects the first names of all of the people in the structure, i.e., ["Tim", "John", "Jane"].



This block can also be used with XML parsed using the Web.XMLTextDecodeAsDictionary block. Consider the following XML document:

```
<schedule>

  <day>

    <room name="Hewlitt" />

    <room name="Bleil" />

  </day>

  <day>

    <room name="Kiva" />

    <room name="Star" />

  </day>

</schedule>
```

You can use the following blocks to get a list of the names of the rooms on the first day, i.e. ["Hewlitt", "Bleil"].



walk all at level

The walk all at level block is a specialized block that can be used in the key path of a list by walking key path. When encountered during a walk, it causes every item at that level to be explored. For dictionaries, this means that every value is visited. For lists, each item in the list is visited. This can be used to aggregate information from a list of items in a dictionary, such as the first name of every person in a database represented by JSON objects.

is a dictionary?

The is a dictionary? block tests to see whether the thing given to it is a dictionary or not. It will return true if the thing is a dictionary and false otherwise.

iii. Storage
    1. Tinydb and its use to retrieve values
iv. Multiple screen layout

h. Sensor
    i. Light sensor

Physical world component that can measure the light level.

Properties

Available
Specifies whether or not the device has the hardware to support the LightSensor component.

AverageLux
Returns the brightness in lux by averaging the previous 10 measured values. The sensor must be enabled and available to return meaningful values.

Enabled
Specifies whether the sensor should generate events. If true, the sensor will generate events. Otherwise, no events are generated.

Lux
Returns the last measured brightness in lux. The sensor must be enabled and available to return meaningful values.

RefreshTime
The requested minimum time in milliseconds between changes in readings being reported. Android is not guaranteed to honor the request. Setting this property has no effect on pre-Gingerbread devices.

Events

LightChanged(*lux*)
Indicates the light level changed.

ii. Other rarely available sensor
    1. How to check if its available on device
    2. How to integrate any sensor based on requirement

Day 5

i. Storage
   i. Firebase db

The Firebase component communicates with a Web service to store and retrieve information. The component has methods to store a value under a tag and to retrieve the value associated with the tag. It also possesses a listener to fire events when stored values are changed.

Additional Information

Properties

DeveloperBucket
Getter for the DeveloperBucket.

FirebaseToken
Getter for the FirebaseToken.

FirebaseURL
Specifies the URL for the Firebase.

The default value is currently my private Firebase URL, but this will eventually changed once the App Inventor Candle plan is activated.

Persist
If true, variables will retain their values when off-line and the App exits. Values will be uploaded to Firebase the next time the App is run while connected to the network. This is useful for applications which will gather data while not connected to the network. Note: AppendValue and RemoveFirst will not work correctly when off-line, they require a network connection.

Note: If you set Persist on any Firebase component, on any screen, it makes all Firebase components on all screens persistent. This is a limitation of the low level Firebase library. Also be aware that if you want to set persist to true, you should do so before connecting the Companion for incremental development.

ProjectBucket
Getter for the ProjectBucket.

Events

DataChanged(*tag*,*value*)
Indicates that the data in the Firebase has changed. Launches an event with the tag and value that have been updated.

FirebaseError(*message*)
Indicates that the communication with the Firebase signaled an error.

FirstRemoved(*value*)
Event triggered by the “RemoveFirst” function. The argument “value” is the object that was the first in the list, and which is now removed.

GotValue(*tag*,*value*)
Indicates that a GetValue request has succeeded.

TagList(*value*)
Event triggered when we have received the list of known tags. Used with the “GetTagList” Function.

Methods

AppendValue(*tag*,*valueToAdd*)
Append a value to the end of a list atomically. If two devices use this function simultaneously, both will be appended and no data lost.

ClearTag(*tag*)
Asks Firebase to forget (delete or set to “null”) a given tag.

GetTagList()
Get the list of tags for this application. When complete a “TagList” event will be triggered with the list of known tags.

GetValue(*tag*,*valueIfTagNotThere*)
GetValue asks Firebase to get the value stored under the given tag. It will pass valueIfTagNotThere to GotValue if there is no value stored under the tag.

RemoveFirst(*tag*)
Return the first element of a list and atomically remove it. If two devices use this function simultaneously, one will get the first element and the the other will get the

second element, or an error if there is no available element. When the element is available, the “FirstRemoved” event will be triggered.

StoreValue(*tag*,*valueToStore*)
Asks Firebase to store the given value under the given tag.

Unauthenticate()
Unauthenticate from Firebase.

Firebase keeps track of credentials in a cache in shared_prefs It will re-use these credentials as long as they are valid. Given That we retrieve a FirebaseToken with a version long life, this will effectively be forever. Shared_prefs survive an application update and depending on how backup is configured on a device, it might survive an application removal and reinstallation.

Normally this is not a problem, however if we change the credentials used, for example the App author is switching from one Firebase account to another, or invalided their firebase.secret, this cached credential is invalid, but will continue to be used, which results in errors.

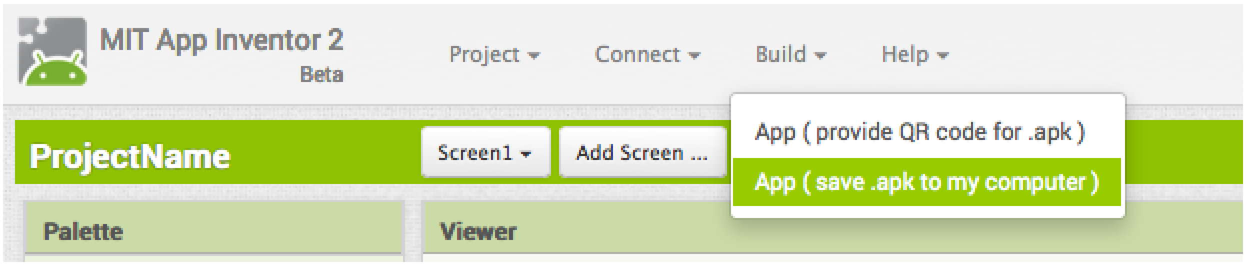
This function permits us to unauthenticate, which tosses the cached credentials. The next time authentication is needed we will use our current FirebaseToken and get fresh credentials.

ii. Creating apk and share

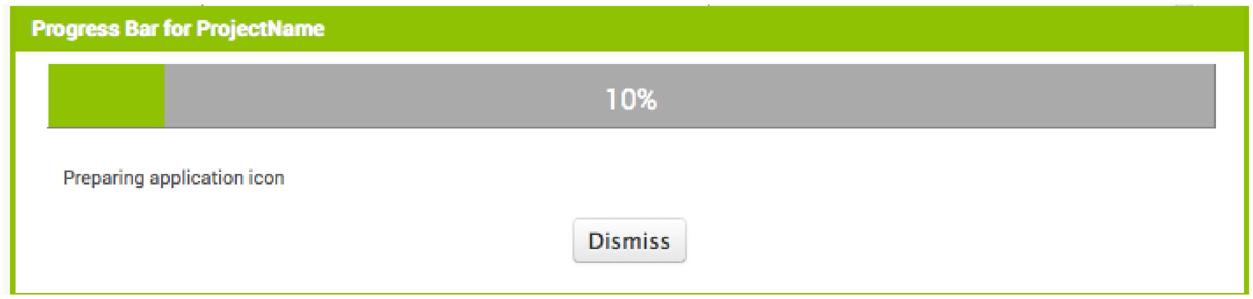
You can share your app in an executable form (.apk) that can be installed on a device, or in source code form (.aia) that can be loaded into App Inventor and remixed.

Sharing your app for use

Package the app (.apk file) by going to the "Build" menu on the App Inventor toolbar.



Select "Application (Save to my Computer)." A pop-up box should alert you that your download has begun.



Once it completes, you can email the app to your friends who can install it by opening the email from their phone. If you want to distribute it more widely, you can upload it to a website that both you and your friend can access. Note that people installing your app may need to change the settings of their phone to allow installation of non-market applications.

Sharing your app for remix

Choose File | Export Project to export the source code (blocks) for your project. The source code is downloaded in a .aia file. If you send it to a friend, they can open it with File | Import Project.

Day 6

j. Sensor
   i. Clock
      1. How to use clock in app

Non-visible component that provides the instant in time using the internal clock on the phone. It can fire a timer at regularly set intervals and perform time calculations, manipulations, and conversions.

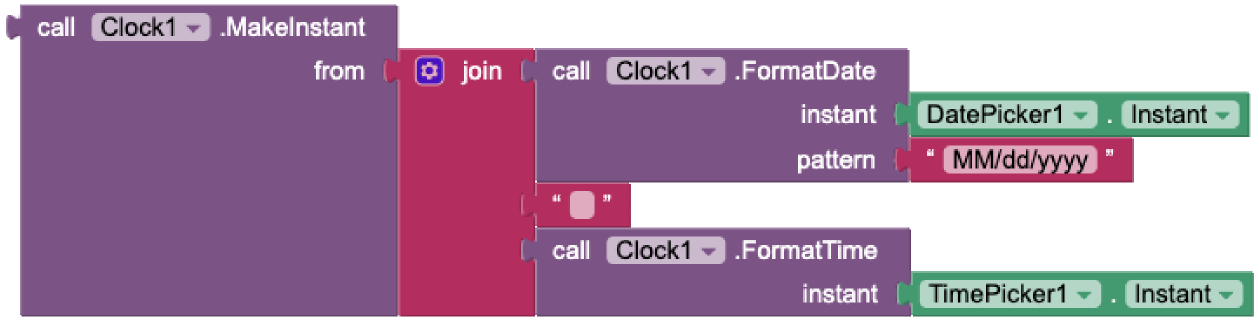
Operations on dates and times, such as from DatePicker and TimePicker, are accomplished through methods in Clock. Date and Time are represented as InstantInTime and Duration.

- Instant: consists of Year, Month, DayOfMnoth, Hour, Minute, and SEcond. An instant can be created using the MakeInstant, MakeInstantFromMillis and MakeInstantFromParts methods.
- Duration: time in milliseconds elapsed between instants. Duration can be obtained by the Duration method.

Instants are assumed to be in the device's local time zone. When they are converted to or from milliseconds, the milliseconds for a given Instance are calculated from January 1, 1970 in UTC (Greenwich Mean Time).

Methods to convert an Instant to text are also available. Acceptable patterns are empty string, MM/dd/YYYY HH:mm:ss a, or MMM d, yyyy HH:mm. The empty string will provide the default format, which is "MMM d, yyyy HH:mm:ss a" for FormatDateTime, "MMM d, yyyy" for FormatDate. To see all possible formats, please see here.

A note on combining date and time: In order to combine the date from one Instant and the time from another, for example from a DatePicker and TimePicker, extract the parts as text and use the text to create a new Instant. For example:



Properties

TimerAlwaysFires
Will fire even when application is not showing on the screen if true

TimerEnabled
Specifies whether the Timer event should run.

TimerInterval
Specifies the interval between subsequent Timer events.

Note: Drift may occur over time and that the system may not honor the timing specified here if the app or another process on the phone is busy.

Events

Timer()
The Timer event runs when the timer has gone off.

Methods

AddDays(*instant,quantity*)
Returns an instant in time some days after the given instant.

AddDuration(*instant,quantity*)
Returns an instant in time some duration after the argument

AddHours(*instant,quantity*)
Returns an instant in time some hours after the given instant.

AddMinutes(*instant,quantity*)
Returns an instant in time some minutes after the given instant.

AddMonths(*instant,quantity*)

Returns an instant in time some months after the given instant.

AddSeconds(*instant*,*quantity*)
Returns an instant in time some seconds after the given instant.

AddWeeks(*instant*,*quantity*)
Returns An instant in time some weeks after the given instant.

AddYears(*instant*,*quantity*)
Returns an instant in time some years after the given instant.

DayOfMonth(*instant*)
Returns the day of the month.

Duration(*start*,*end*)
Returns the milliseconds by which end follows start (+ or -)

DurationToDays(*duration*)
Returns the duration converted from milliseconds to days.

DurationToHours(*duration*)
Returns the duration converted from milliseconds to hours.

DurationToMinutes(*duration*)
Returns the duration converted from milliseconds to minutes.

DurationToSeconds(*duration*)
Returns the duration converted from milliseconds to seconds.

DurationToWeeks(*duration*)
Returns the duration converted from milliseconds to weeks.

FormatDate(*instant*,*pattern*)
Converts and formats an instant into a string of date with the specified pattern. To learn more about valid patterns, please see SimpleDateFormat.

FormatDateTime(*instant*,*pattern*)
Converts and formats an instant into a string of date and time with the specified pattern. To learn more about valid patterns, please see SimpleDateFormat.

FormatTime(*instant*)
Converts and formats the given instant into a string with the specified pattern. To learn more about valid patterns, please see SimpleDateFormat.

GetMillis(*instant*)

Returns the instant in time measured as milliseconds since 1970.

Hour(*instant*)
Returns the hours for the given date.

MakeDate(*year*,*month*,*day*)
Returns an instant in time specified by year, month, date in UTC. Valid values for the month field are 1-12 and 1-31 for the day field.

MakeInstant(*from*)
Returns an instant in time specified by MM/dd/YYYY hh:mm:ss or MM/dd/YYYY or hh:mm.

MakeInstantFromMillis(*millis*)
Returns an instant in time specified by the milliseconds since 1970 in UTC.

MakeInstantFromParts(*year*,*month*,*day*,*hour*,*minute*,*second*)
Returns an instant in time specified by year, month, date, hour, minute, second in UTC.

MakeTime(*hour*,*minute*,*second*)
Returns an instant in time specified by hour, minute, second in UTC.

Minute(*instant*)
Returns the minutes for the given date.

Month(*instant*)
Returns the number of the month for the given instant.

MonthName(*instant*)
Returns the name of the month for the given instant.

Now()
Returns the current instant in time read from phone’s clock.

Second(*instant*)
Returns the seconds for the given instant.

SystemTime()
Returns the phone’s internal time.

Weekday(*instant*)
Returns the weekday for the given instant.

WeekdayName(*instant*)

Returns the name of the weekday for the given instant.

Year(*instant*)
Returns the year of the given instant.

ii. Integrate sensor data and store in firebase db
iii. Barcode scanner

Component for scanning a QR code and getting back the resulting string.

Properties

Result
Gets the text result of the previous scan.

UseExternalScanner
Set whether or not you wish to use an External Scanning program such as Bar Code Scanner. If false a version of ZXing integrated into App Inventor will be used.

Events

AfterScan(*result*)
Indicates that the scanner has read a (text) result and provides the result

Methods

DoScan()
Begins a barcode scan, using the camera. When the scan is complete, the AfterScan event will be raised.

k. Learn how to open webpages and different websites in app

Component for viewing Web pages.

The HomeUrl can be specified in the Designer or in the Blocks Editor. The view can be set to follow links when they are tapped, and users can fill in Web forms.

Warning: This is not a full browser. For example, pressing the phone's hardware Back key will exit the app, rather than move back in the browser history.

You can use the WebViewString property to communicate between your app and Javascript code running in the WebViewer page. In the app, you get and set WebViewString. In the WebViewer, you include Javascript that references the window.AppInventor object, using the methods getWebViewString() and setWebViewString(text).

For example, if the WebViewer opens to a page that contains the Javascript command

document.write("The answer is" + window.AppInventor.getWebViewString());

and if you set WebViewString to “hello”, then the web page will show

The answer is hello.

And if the Web page contains Javascript that executes the command

windowAppInventor.setWebViewString("hello from Javascript"),

then the value of the WebViewString property will be

hello from Javascript.

Calling setWebViewString from JavaScript will also run the WebViewStringChange event so that the blocks can handle when the WebViewString property changes.

Beginning with release nb184a, you can specify a HomeUrl beginning with http://localhost/ to reference assets both in the Companion and in compiled apps. Previously, apps needed to use file:///android_asset/ in compiled apps and /sdcard/AppInventor/assets/ in the Companion. Both of these options will continue to work but the http://localhost/ approach will work in both scenarios. You may also use “file:///appinventor_asset/” which provides more security by preventing the use of asynchronous requests from JavaScript in your assets from going out to the web.

Properties

CurrentPageTitle
Returns the title of the page currently being viewed

CurrentUrl

Returns the URL currently being viewed. This could be different from the HomeUrl if new pages were visited by following links.

FollowLinks

Determines whether to follow links when they are tapped in the WebViewer. If you follow links, you can use GoBack and GoForward to navigate the browser history.

Height

Specifies the WebViewer's vertical height, measured in pixels.

HeightPercent

Specifies the WebViewer's vertical height as a percentage of the Screen's Height.

HomeUrl

Specifies the URL of the page the WebViewer should initially open to. Setting this will load the page.

IgnoreSslErrors

Determine whether or not to ignore SSL errors. Set to true to ignore errors. Use this to accept self signed certificates from websites.

PromptforPermission

Determine if the user should be prompted for permission to use the geolocation API while in the WebViewer. If true, prompt the user of the WebViewer to give permission to access the geolocation API. If false, assume permission is granted.

UsesLocation

Specifies whether or not this WebViewer can access the JavaScript geolocation API.

Visible

Specifies whether the WebViewer should be visible on the screen. Value is true if the WebViewer is showing and false if hidden.

WebViewString

Gets the WebView's String, which is viewable through Javascript in the WebView as the window.AppInventor object.

Width

Specifies the horizontal width of the WebViewer, measured in pixels.

WidthPercent

Specifies the horizontal width of the WebViewer as a percentage of the Screen's Width.

Events

BeforePageLoad(*url*)
When a page is about to load this event is run.

ErrorOccurred(*errorCode*,*description*,*failingUrl*)
When an error occurs this event is run.

PageLoaded(*url*)
When a page is finished loading this event is run.

WebViewStringChange(*value*)
Event that runs when the AppInventor.setWebViewString method is called from JavaScript. The new WebViewString is given by the value parameter.

Methods

CanGoBack()
Returns true if the WebViewer can go back in the history list.

CanGoForward()
Returns true if the WebViewer can go forward in the history list.

ClearCaches()
Clear the internal webview cache, both ram and disk. This is useful when using the WebViewer to poll a page that may not be sending appropriate cache control headers.

ClearCookies()
Clear the webview's cookies. This is useful if you want to sign the user out of a website that uses them to store logins.

ClearLocations()
Clear Stored Location permissions. When the geolocation API is used in the WebViewer, the end user is prompted on a per URL basis for whether or not permission should be granted to access their location. This function clears this information for all locations.

As the permissions interface is not available on phones older then Eclair, this function is a no-op on older phones.

GoBack()

Go back to the previous page in the history list. Does nothing if there is no previous page.

GoForward()

Go forward to the next page in the history list. Does nothing if there is no next page.

GoHome()

Loads the page from the home URL. This happens automatically when home URL is changed.

GoToUrl(*url*)

Load the page at the given URL.

Reload()

Reload the current page.

RunJavaScript(*js*)

Run JavaScript in the current page.

StopLoading()

Stop loading a page.

Day 7

I. API : Application programming interface
   i. Introduction

Using HTTP Methods for RESTful Services

- Quick-Tips
- Resource Naming

The HTTP verbs comprise a major portion of our “uniform interface” constraint and provide us the action counterpart to the noun-based resource. The primary or most-commonly-used HTTP verbs (or methods, as they are properly called) are POST, GET, PUT, PATCH, and DELETE. These correspond to create, read, update, and delete (or CRUD) operations, respectively. There are a number of other verbs, too, but are utilized less frequently. Of those less-frequent methods, OPTIONS and HEAD are used more often than others.

Below is a table summarizing recommended return values of the primary HTTP methods in combination with the resource URIs:

HTTP Verb	**CRUD**	**Entire Collection (e.g. /customers)**	**Specific Item (e.g. /customers/{id})**
POST	Create	201 (Created), 'Location' header with link to /customers/{id} containing new ID.	404 (Not Found), 409 (Conflict) if resource already exists..
GET	Read	200 (OK), list of customers. Use pagination, sorting and filtering to navigate big lists.	200 (OK), single customer. 404 (Not Found), if ID not found or invalid.

PUT	Update/Replace	405 (Method Not Allowed), unless you want to update/replace every resource in the entire collection.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
PATCH	Update/Modify	405 (Method Not Allowed), unless you want to modify the collection itself.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
DELETE	Delete	405 (Method Not Allowed), unless you want to delete the whole collection—not often desirable.	200 (OK). 404 (Not Found), if ID not found or invalid.

ii. A general look to different types of API
iii. Exploring weather API
   1. Create account in weatherapi.com
   2. Get API Keys
   3. Using API to get weather and other related information
iv. Various type of request
v. Understanding the different types of web component methods
   1. GET
   2. POST
   3. PUT
   4. DELETE

Day 8

m. API : Application programming interface
    i. Exploring weather API
    ii. Various type of request
    iii. Understanding the different types of web component methods : continued
        1. GET
        2. POST
        3. PUT
        4. DELETE

n. JSON
    i. What is JSON?
    ii. How to parse JSON?
    iii. Learn by example : weather API

JSON is a text-based data format following JavaScript object syntax, which was popularized by Douglas Crockford. Even though it closely resembles JavaScript object literal syntax, it can be used independently from JavaScript, and many programming environments feature the ability to read (parse) and generate JSON.

JSON exists as a string — useful when you want to transmit data across a network. It needs to be converted to a native JavaScript object when you want to access the data. This is not a big issue — JavaScript provides a global JSON object that has methods available for converting between the two.

Note: Converting a string to a native object is called *deserialization*, while converting a native object to a string so it can be transmitted across the network is called *serialization*.

A JSON string can be stored in its own file, which is basically just a text file with an extension of `.json`, and a MIME type of `application/json`.

JSON structure

As described above, JSON is a string whose format very much resembles JavaScript object literal format. You can include the same basic data types inside JSON as you can in a standard JavaScript object — strings, numbers, arrays, booleans, and other object literals. This allows you to construct a data hierarchy, like so:

```
{
  "squadName": "Super hero squad",
  "homeTown": "Metro City",
  "formed": 2016,
  "secretBase": "Super tower",
  "active": true,
  "members": [
    {
      "name": "Molecule Man",
      "age": 29,
      "secretIdentity": "Dan Jukes",
      "powers": [
        "Radiation resistance",
        "Turning tiny",
        "Radiation blast"
      ]
    },
    {
      "name": "Madame Uppercut",
      "age": 39,
      "secretIdentity": "Jane Wilson",
      "powers": [
        "Million tonne punch",
        "Damage resistance",
        "Superhuman reflexes"
      ]
    },
    {
      "name": "Eternal Flame",
      "age": 1000000,
      "secretIdentity": "Unknown",
      "powers": [
        "Immortality",
        "Heat Immunity",
        "Inferno",
        "Teleportation",
        "Interdimensional travel"
      ]
    }
  ]
}
```

Copy to Clipboard

If we loaded this string into a JavaScript program, parsed it into a variable called `superHeroes` for example, we could then access the data inside it using the same dot/bracket notation we looked at in the JavaScript object basics article. For example:

```
superHeroes.homeTown
```

```
superHeroes['active']
```

Copy to Clipboard

To access data further down the hierarchy, you have to chain the required property names and array indexes together. For example, to access the third superpower of the second hero listed in the members list, you'd do this:

```
superHeroes['members'][1]['powers'][2]
```

1. First we have the variable name — `superHeroes`.
2. Inside that we want to access the `members` property, so we use `["members"]`.
3. `members` contain an array populated by objects. We want to access the second object inside the array, so we use `[1]`.
4. Inside this object, we want to access the `powers` property, so we use `["powers"]`.
5. Inside the `powers` property is an array containing the selected hero's superpowers. We want the third one, so we use `[2]`.

Note: We've made the JSON seen above available inside a variable in our JSONTest.html example (see the source code). Try loading this up and then accessing data inside the variable via your browser's JavaScript console.

Arrays as JSON

Above we mentioned that JSON text basically looks like a JavaScript object inside a string. We can also convert arrays to/from JSON. Below is also valid JSON, for example:

```
[
  {
    "name": "Molecule Man",
    "age": 29,
    "secretIdentity": "Dan Jukes",
    "powers": [
      "Radiation resistance",
      "Turning tiny",
      "Radiation blast"
    ]
  },
  {
    "name": "Madame Uppercut",
    "age": 39,
    "secretIdentity": "Jane Wilson",
    "powers": [
      "Million tonne punch",
      "Damage resistance",
      "Superhuman reflexes"
    ]
  }
]
```

Copy to Clipboard

The above is perfectly valid JSON. You'd just have to access array items (in its parsed version) by starting with an array index, for example [0]["powers"][0].

Other notes

JSON is purely a string with a specified data format — it contains only properties, no methods.

JSON requires double quotes to be used around strings and property names. Single quotes are not valid other than surrounding the entire JSON string.

Even a single misplaced comma or colon can cause a JSON file to go wrong, and not work. You should be careful to validate any data you are attempting to use (although computer-generated JSON is less likely to include errors, as long as the generator program is working correctly). You can validate JSON using an application like JSONLint.

JSON can actually take the form of any data type that is valid for inclusion inside JSON, not just arrays or objects. So for example, a single string or number would be valid JSON.

Unlike in JavaScript code in which object properties may be unquoted, in JSON only quoted strings may be used as properties.

- o. Some more components
    - i. Slider

This class is used to display a Slider.

A Slider is a progress bar that adds a draggable thumb. You can touch the thumb and drag left or right to set the slider thumb position. As the Slider thumb is dragged, it will trigger the PositionChanged event, reporting the position of the Slider thumb. The reported position of the thumb can be used to dynamically update another component attribute, such as the TextBox's FontSize of a TextBox or the Radius of a Ball.

The Slider uses the following default values. However these values can be changed through the Designer or Blocks editor:

- MinValue = 10
- MaxValue = 50
- ThumbPosition = 30

Properties

ColorLeft
Specifies the color of the slider bar to the left of the thumb as an alpha-red-green-blue integer, i.e., 0xAARRGGBB. An alpha of 00 indicates fully transparent and FF means opaque.

ColorRight
Specifies the color of the slider bar to the right of the thumb as an alpha-red-green-blue integer, i.e., 0xAARRGGBB. An alpha of 00 indicates fully transparent and FF means opaque.

HeightPercent
Specifies the Slider's vertical height as a percentage of the Screen's Height.

MaxValue
Sets the maximum value of slider. If the new maximum is less than the current minimum, then minimum and maximum will both be set to this value. Setting MaxValue resets the thumb position to halfway between MinValue and MaxValue and signals the PositionChanged` event.

MinValue
Sets the minimum value of slider. If the new minimum is greater than the current maximum, then minimum and maximum will both be set to this value. Setting MinValue resets the thumb position to halfway between MinValue and MaxValue and signals the PositionChanged` event.

ThumbEnabled
Whether or not the slider thumb is being be shown.

ThumbPosition

Sets the position of the slider thumb. If this value is greater than MaxValue, then it will be set to same value as MaxValue. If this value is less than MinValue, then it will be set to same value as MinValue.

Visible
Specifies whether the Slider should be visible on the screen. Value is true if the Slider is showing and false if hidden.

Width
Specifies the horizontal width of the Slider, measured in pixels.

WidthPercent
Specifies the horizontal width of the Slider as a percentage of the Screen's Width.

Events

PositionChanged(*thumbPosition*)
Indicates that the position of the slider thumb has changed.

Methods

None

ii. Switch

Switch components can detect user taps and can change their boolean state in response. They are identical to CheckBoxes except in appearance.

Switches have an on (true) state and an off (false) state. A Switch component raises an event when the user taps it to toggle between states.

Properties

BackgroundColor
Specifies the background color of the Switch as an alpha-red-green-blue integer.

Enabled
Specifies whether the Switch should be active and clickable.

FontBold
Specifies whether the text of the Switch should be bold. Some fonts do not support bold.

FontItalic

Specifies whether the text of the Switch should be italic. Some fonts do not support italic.

FontSize

Specifies the text font size of the Switch, measured in sp(scale-independent pixels).

FontTypeface

Specifies the text font face of the Switch as default, serif, sans serif, or monospace.

Height

Specifies the Switch's vertical height, measured in pixels.

HeightPercent

Specifies the Switch's vertical height as a percentage of the Screen's Height.

On

True if the switch is in the On state, false otherwise.

Text

Specifies the text displayed by the Switch.

TextColor

Specifies the text color of the Switch as an alpha-red-green-blue integer.

ThumbColorActive

Specifies the Switch's thumb color when switch is in the On state.

ThumbColorInactive

Specifies the Switch's thumb color when switch is in the Off state.

TrackColorActive

Specifies the Switch's track color when in the On state.

TrackColorInactive

Specifies the Switch's track color when in the Off state.

Visible

Specifies whether the Switch should be visible on the screen. Value is true if the Switch is showing and false if hidden.

Width

Specifies the horizontal width of the Switch, measured in pixels.

WidthPercent

Specifies the horizontal width of the Switch as a percentage of the Screen’s Width.

Events

Changed()
User change the state of the Switch from On to Off or back.

GotFocus()
Switch became the focused component.

LostFocus()
Switch stopped being the focused component.

iii. Learn to implement by shopping app

Day 9

p. Some more components

   i. Notifier different methods

The Notifier component displays alert messages and creates Android log entries through an assortment of methods.

Properties

BackgroundColor
Specifies the background color for alerts (not dialogs).

NotifierLength
Specifies the length of time that the alert is shown – either “short” or “long”.

TextColor
Specifies the text color for alerts (not dialogs).

Events

AfterChoosing(*choice*)
Event after the user has made a selection for ShowChooseDialog.

AfterTextInput(*response*)
Event raised after the user has responded to ShowTextDialog.

ChoosingCanceled()
Event raised when the user cancels choosing an option. ShowChooseDialog.

TextInputCanceled()
Event raised when the user cancels ShowChooseDialog, ShowPasswordDialog, or ShowTextDialog.

Methods

DismissProgressDialog()
Dismisses the alert created by the ShowProgressDialog block

LogError(*message*)
Writes an error message to the Android system log. See the Google Android documentation for how to access the log.

LogInfo(*message*)
Writes an information message to the Android log.

LogWarning(*message*)
Writes a warning message to the Android log. See the Google Android documentation for how to access the log.

ShowAlert(*notice*)
Display a temporary notification.

ShowChooseDialog(*message,title,button1Text,button2Text,cancelable*)
Shows a dialog box with two buttons, from which the user can choose. If cancelable is true there will be an additional CANCEL button. Pressing a button will raise the AfterChoosing event. The “choice” parameter to AfterChoosing will be the text on the button that was pressed, or “Cancel” if the CANCEL button was pressed. If canceled, the TextInputCanceled event will also run.

ShowMessageDialog(*message,title,buttonText*)
Display an alert dialog with a single button that dismisses the alert.

ShowPasswordDialog(*message,title,cancelable*)
Shows a dialog box where the user can enter password (input is masked), after which the AfterTextInput event will be raised. If cancelable is true there will be an additional CANCEL button. The AfterTextInput and TextInputCanceled events behave the same way as described in ShowTextDialog.

ShowProgressDialog(*message,title*)
Shows a dialog box with an optional title and message (use empty strings if they are not wanted). This dialog box contains a spinning artifact to indicate that the program is working. It cannot be canceled by the user but must be dismissed by the App Inventor Program by using the DismissProgressDialog method.

ShowTextDialog(*message,title,cancelable*)
Shows a dialog box where the user can enter text, after which the AfterTextInput event will be raised. If cancelable is true there will be an additional CANCEL button. Entering text will raise the AfterTextInput event. The “response” parameter to AfterTextInput will be the text that was entered, or “Cancel” if the CANCEL button was pressed. If canceled, the TextInputCanceled event will also run.

ii. How to use maps with location sensor?

A two-dimensional container that renders map tiles in the background and allows for multiple Marker elements to identify points on the map. Map tiles are supplied by OpenStreetMap contributors and the the United States Geological Survey.

The Map component provides three utilities for manipulating its boundaries with App Inventor. First, a locking mechanism is provided to allow the map to be moved relative to other components on the Screen. Second, when unlocked, the user can pan the Map to any location. At this new location, the "Set Initial Boundary" button can be pressed to save the current Map coordinates to its properties. Lastly, if the Map is moved to a different location, for example to add Markers off-screen, then the "Reset Map to Initial Bounds" button can be used to re-center the Map at the starting location.

Properties

BoundingBox
Sets or gets the current boundary for the map's drawn view. The value is a list of lists containing the northwest and southeast coordinates of the current view in the form ((North West) (South East)).

CenterFromString
Set the initial center coordinate of the map. The value is specified as a comma-separated pair of decimal latitude and longitude coordinates, for example, 42.359144, -71.093612.

In blocks code, it is recommended for performance reasons to use PanTo with numerical latitude and longitude rather than convert to the string representation for use with this property.

EnablePan
Enables or disables the ability of the user to move the Map.

EnableRotation
Enables or disables the two-finger rotation gesture to rotate the Map.

EnableZoom
Enables or disables the two-finger pinch gesture to zoom the Map.

Features

Gets the list of features attached to the Map (without regard to the value of the feature's Visible property). This list also includes any features created on the Map by calls to FeatureFromDescription.

Height
Specifies the Map's vertical height, measured in pixels.

HeightPercent
Specifies the Map's vertical height as a percentage of the Screen's Height.

Latitude
Gets the latitude of the center of the Map. To change the latitude, use the PanTo method.

LocationSensor
Uses the provided LocationSensor for user location data rather than the built-in location provider.

Longitude
Gets the longitude of the center of the Map. To change the longitude, use the PanTo method.

MapType
Sets or gets the tile layer used to draw the Map background. Defaults to Roads. Valid values are:

1. Roads
2. Aerial
3. Terrain

Note: Road layers are provided by OpenStreetMap and aerial and terrain layers are provided by the U.S. Geological Survey.

Rotation
Specifies the rotation of the map in decimal degrees, if any.

ScaleUnits
Specifies the units used for the scale overlay. 1 (the default) will give metric units (km, m) whereas 2 will give imperial units (mi, ft).

ShowCompass

Specifies whether to a compass overlay on the Map. The compass will be rotated based on the device's orientation if a digital compass is present in hardware.

ShowScale
Shows a scale reference on the map.

ShowUser
Shows or hides an icon indicating the user's current location on the Map. The availability and accuracy of this feature will depend on whether the user has location services enabled and which location providers are available.

ShowZoom
Shows or hides an icon indicating the user's current location on the Map. The availability and accuracy of this feature will depend on whether the user has location services enabled and which location providers are available.

UserLatitude
Returns the user's latitude if ShowUser is enabled.

UserLongitude
Returns the user's longitude if ShowUser is enabled.

Visible
Specifies whether the Map should be visible on the screen. Value is true if the Map is showing and false if hidden.

Width
Specifies the horizontal width of the Map, measured in pixels.

WidthPercent
Specifies the horizontal width of the Map as a percentage of the Screen's Width.

ZoomLevel
Specifies the zoom level of the map. Valid values of ZoomLevel are dependent on the tile provider and the latitude and longitude of the map. For example, zoom levels are more constrained over oceans than dense city centers to conserve space for storing tiles, so valid values may be 1-7 over ocean and 1-20 over cities. Tile providers may send warning or error tiles if the zoom level is too great for the server to support.

Events

BoundsChange()

The BoundsChange event runs when the user changes the map bounds, either by zooming, panning, or rotating the view.

DoubleTapAtPoint(*latitude*,*longitude*)
The DoubleTapAtPoint runs when the user double taps at a point on the map. The tapped location will be reported in map coordinates via the latitude and longitude parameters.

FeatureClick(*feature*)
When a feature is clicked, the parent Map will also receive a FeatureClick event. The feature parameter indicates which child feature was clicked. This event is run *after* the Click event on the corresponding feature and after the when any ... Click event if one is provided.

FeatureDrag(*feature*)
When the user drags a feature, the parent Map will also receive a FeatureDrag event. The feature parameter indicates which child feature was dragged. This event is run *after* the Drag event on the corresponding feature and after the when any ... Drag event if one is provided.

FeatureLongClick(*feature*)
When a feature is long-clicked, the parent Map will also receive a FeatureLongClick event. The feature parameter indicates which child feature was long-clicked. This event is run *after* the LongClick event on the corresponding feature and after the when any ... LongClick event if one is provided.

FeatureStartDrag(*feature*)
When the user starts dragging a feature, the parent Map will also receive a FeatureStartDrag event. The feature parameter indicates which child feature was dragged. This event is run *after* the StartDrag event on the corresponding feature and after the when any ... StartDrag event if one is provided.

FeatureStopDrag(*feature*)
When the user stops dragging a feature, the parent Map will also receive a FeatureStopDrag event. The feature parameter indicates which child feature was dragged. This event is run *after* the StopDrag event on the corresponding feature and after the when any ... StopDrag event if one is provided.

GotFeatures(*url*,*features*)

The GotFeatures event is run when when a feature collection is successfully read from the given url. The features parameter will be a list of feature descriptions that can be converted into components using the FeatureFromDescription method.

InvalidPoint(*message*)
The InvalidPoint event runs when the program encounters an invalid point while processing geographical data. Points are considered invalid when the latitude or longitude for the point is outside the acceptable range ([-90, 90] and [-180, 180], respectively). The message parameter will contain an explanation for the error.

LoadError(*url,responseCode,errorMessage*)
The LoadError event is run when an error occurs while processing a feature collection document at the given url. The responseCode parameter will contain an HTTP status code and the errorMessage parameter will contain a detailed error message.

LongPressAtPoint(*latitude,longitude*)
The LongPressAtPoint runs when the user long-presses at a point on the map without moving their finger (which would trigger a drag). The location of the long-press will be reported in map coordinates via the latitude and longitude parameters.

Ready()
The Ready event runs once the Map has been initialized and is ready for user interaction.

TapAtPoint(*latitude,longitude*)
The TapAtPoint runs when the user taps at a point on the map. The tapped location will be reported in map coordinates via the latitude and longitude parameters.

ZoomChange()
The ZoomChange event runs when the user has changed the zoom level of the map, such as by pinching or double-tapping to zoom.

Methods

CreateMarker(*latitude,longitude*)
Creates a new Marker on the Map at the specified latitude and longitude.

FeatureFromDescription(*description*)
Converts a feature description into an App Inventor map feature. Points are converted into Marker components, LineStrings are converted into LineString components, and Polygons (and MultiPolygons) are converted into Polygon components. If the feature

has properties, they will be mapped into App Inventor properties using the following mapping:

- description becomes Description
- draggable becomes Draggable
- infobox becomes EnableInfobox
- fill becomes FillColor
- fill-opacity becomes FillOpacity
- image becomes ImageAsset
- stroke becomes StrokeColor
- stroke-opacity becomes StrokeOpacity
- stroke-width becomes StrokeWidth
- title becomes Title
- visible becomes Visible

LoadFromURL(*url*)
Loads a feature collection in GeoJSON format from the given url. On success, the event GotFeatures will be raised with the given url and a list of features parsed from the GeoJSON as a list of (key, value) pairs. On failure, the LoadError event will be raised with any applicable HTTP response code and error message.

PanTo(*latitude*,*longitude*,*zoom*)
Pans the map center to the given latitude and longitude and adjust the zoom level to the specified zoom.

Save(*path*)
Saves the features on the Map as a GeoJSON file at the specified path.

q. Google API : Introduction
    1. Create account
    2. Get API Keys
    3. Using API to get distance and location etc.

The Google Drive API allows you to create apps that leverage Google Drive cloud storage. You can develop applications that integrate with Google Drive, and create robust functionality in your application using Google Drive API.

This diagram shows the relationship between your Google Drive app, Google Drive, and Google Drive API:

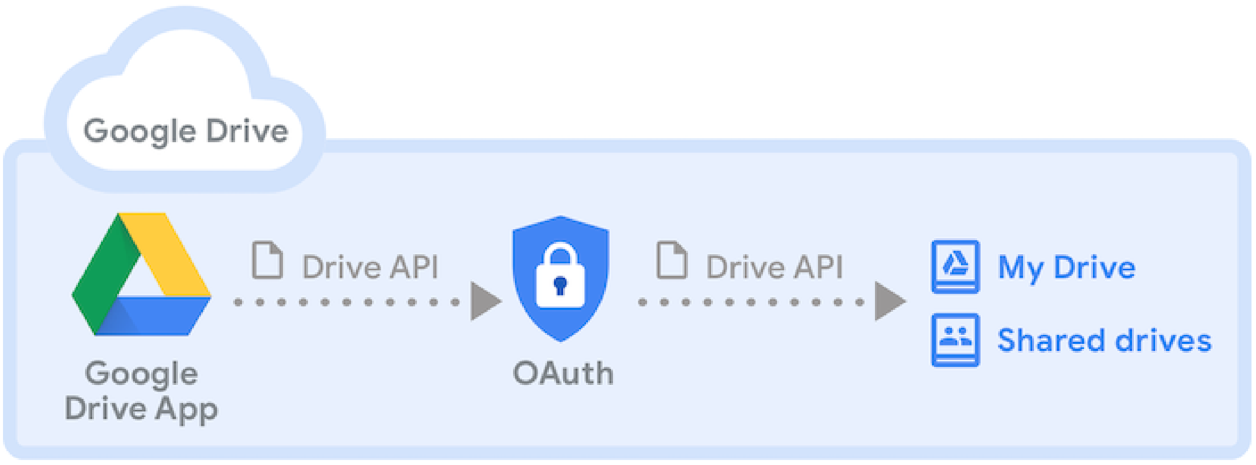


Figure 1. Google Drive API Relationship Diagram

These terms define the key components shown in Figure 1:

Google Drive

Google's cloud file storage service that provides users with a personal storage space, called *My Drive*, and the option to access collaborative shared folders, called *shared drives*.

Google Drive API

The REST API that allows you to leverage Google Drive storage from within your app.

Google Drive app

An app that leverages Google Drive as its storage solution.

Google Drive UI

Google's user interface that manages files stored on Google Drive. If your app is an editor-type app, such as a spreadsheet or word processor app, you can integrate with the Drive UI to create and open files within your app.

My Drive

A Google Drive storage location that a specific user owns. Files stored on My Drive can be shared with other users, but ownership of the content remains specific to an individual user.

OAuth 2.0

The authorization protocol that Google Drive API requires to authenticate your app users. If your application uses Google Sign-in, it handles the OAuth 2.0 flow and application access tokens.

Shared drive

A Google Drive storage location that owns files that multiple users share to collaborate. Any user with access to a shared drive has access to all files it contains. Users can also be granted access to individual files inside the shared drive.

What can you do with Google Drive API?

You can use Google Drive API to:

Download files from Google Drive and Upload files to Google Drive.

Search for files and folders stored in Google Drive. Create complex search queries that return any of the file metadata fields in the Files resource.

Let users share files, folders and drives to collaborate on content.

Combine with the Google Picker API to search all files in Google Drive, then return the file name, URL, last modified date, and user.

Create *third-party shortcuts* that are external links to data stored outside of Drive, in a different data store or cloud storage system.

Create a dedicated Drive folder to store your application’s data so that the app cannot access all the user's content stored in Google Drive. See Store application-specific data.

Integrate with the *Google Drive UI*, which is Google's standard web UI you can use to interact with Drive files. To learn all that you can do with a Drive app that you integrate with the Google Drive UI, see Drive UI integration overview

r. Discussion on project
    i. Idea review
    ii. Implementation discussion

Day 10

s. Project discussion
    i. QA session
    ii. Problem solving
t. Cloud
    i. Different cloud providers
    ii. Things to consider while choosing provider

Day 11

u. Raspberry pi integration
    i. What is Raspberry Pi?

Raspberry Pi (/paɪ/) is a series of small single-board computers (SBCs) developed in the United Kingdom by the Raspberry Pi Foundation in association with Broadcom. The Raspberry Pi project originally leaned towards the promotion of teaching basic computer science in schools and in developing countries.The original model became more popular than anticipated, selling outside its target market for uses such as robotics. It is widely used in many areas, such as for weather monitoring, because of its low cost, modularity, and open design. It is typically used by computer and electronic hobbyists, due to its adoption of HDMI and USB devices.

v. Project discussion
    i. QA session
    ii. Problem solving
w. Introduction to Google Play Store

Day 12

1. Test Your Application

It is needless to mention how crucial testing your application is. No matter how many incredible features you have packed into your app, if it does not perform upto the expectations of the user, it will be ditched like a hot potato. In which case, it is imperative for you to painstakingly test your app as many times as possible and be 100 percent certain that it is going to perform remarkably, before you look ahead to upload apps on Play store.

You can always use emulators for this purpose, however, using an Android-powered device will make the testing process more effective. It will give the experience of using your app on a real device, as would users, and enable you to analyze any bugs or discrepancies.

2. Concise App Size

In terms of applications, the size of the app matters a lot. Users do not feel inclined to download an app that takes too much space in their device storage. Moreover, Google only permits the app size to up to 50MB.

Although, if your app exceeds this limit, then you can use Android APK's Expansion file, to successfully upload the app to Play Store. This will break your app into parts and each can be up to 2GB, giving an additional 4GB space to your app. This added data is saved in Google Cloud and is retrieved whenever the app is installed.

3. Get App Licensed

Though this is an optional choice, it wouldn’t hurt you to get your app licensed before you upload your app to Google Play Store. Licensing your application will prove most beneficial for you if it is paid in nature. By adding the End User License Agreement, you will gain full control over your application which may help in the future, should any discrepancies arise.

> Create your APK file with Bundle ID and Version Number
>
> You need to prepare an APK file in which you can assign a version number to your application which will help you in the future when you need to upload a new update for your app. This number is mentioned in the code and would increase as the updates for the application are introduced.
>
> Bundle ID, also called App ID, is used to make an application unique, making it a crucial part of the prerequisites for submitting your application. This is applicable for all the applications for Android 5.0 and above.

4. Sign App With Security Certificate

Here, you need to create a private key using Release Keystore. This is a security certificate signed as an APK which you will need every time you publish an app to the Play Store. This is also known as JSK file including credentials such as Keystore password.

5. Prepare App Store Listing

App listing is a powerful element that helps your application in gaining downloads. Not every one devotes their time on app listing but if you do this before android app submission, you will definitely see some mind-blowing results.

In the app listing, you provide some information to users about what type of application it is and what are its features. One of the best practices of app listing is using high-quality screenshots. Play Store requirements allow developers to use a maximum of 8 images and a minimum of 2.

Here are the best practices and what you should include in your app listing-

Title
Short Description
Full Description
Screenshots of your app (JPEG or 24-bit PNG)(Min-320px,Max-3840px)
Hi-resolution icon (512 x 512)((with alpha) 32-bit PNG)
Feature Graphic (1024 w x 500 h)(JPG or 24-bit PNG (no alpha))
Type of application
Category of your app
Rating of the content
Email of Developer or Company
Url for Privacy Policy

6. Go Through Guidelines

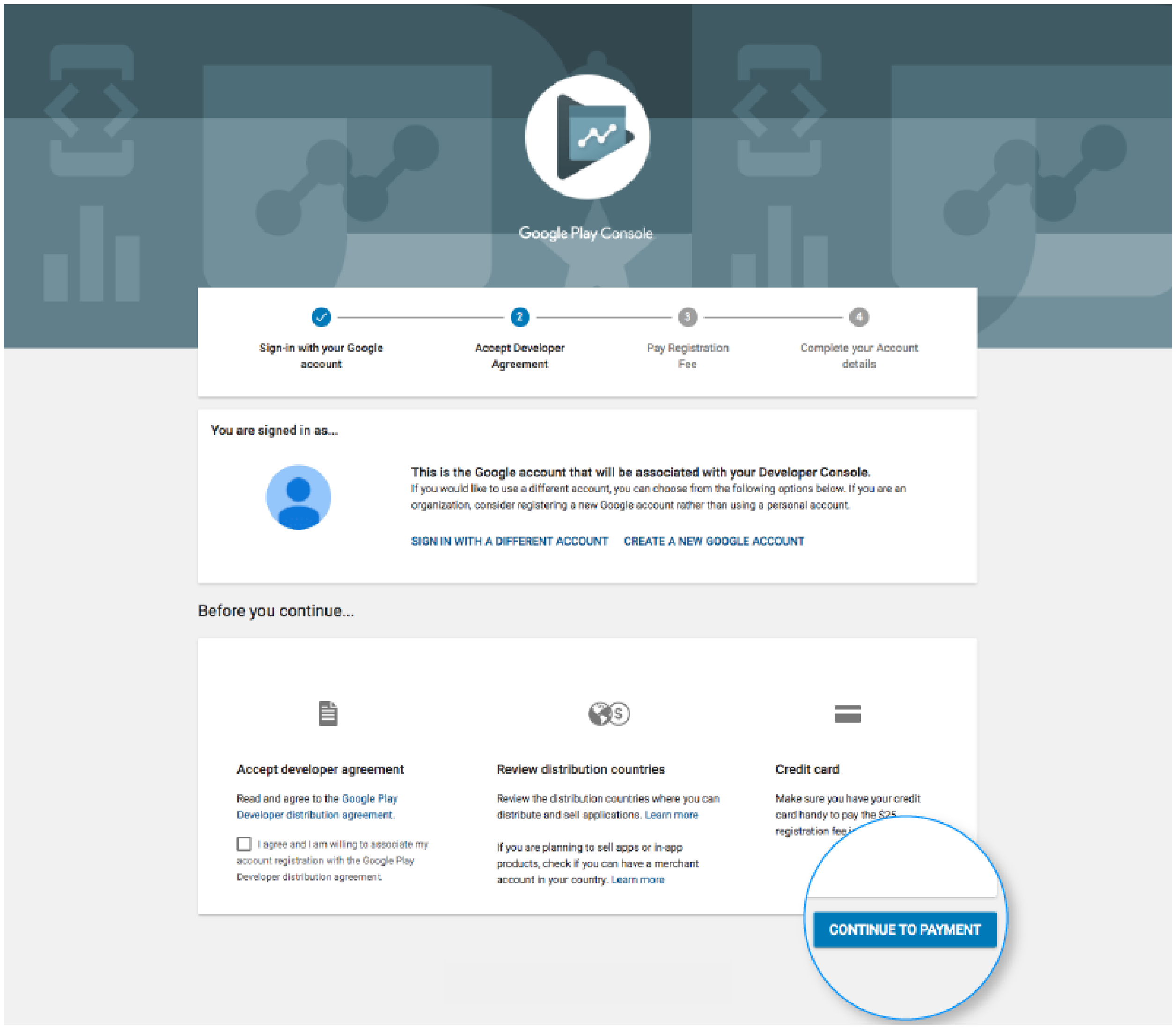
When it comes to App Store vs Play Store guidelines, it is safe to say that Google's guidelines are more flexible, which works in favor of developers.

However, you have to be careful when you add an app to the Play Store and you would have to make sure you follow all the guidelines given by Google, lest your app will be kicked out of the platform. Now, this is something you would want to avoid at all costs, right?

Step-by-Step Process to Upload App To Google Play Store

Now that the obvious is out of the way, let’s move on to the steps regarding how to upload an app on Play Store. Make sure you follow each in the exact chronological order to avoid any mistakes in the process.

1. Google Play Developer Console


Google Play Console

Sign-in with your Google account
Accept Developer Agreement
Pay Registration Fee
Complete your Account details

You are signed in as...

This is the Google account that will be associated with your Developer Console.
If you would like to use a different account, you can choose from the following options below. If you are an organization, consider registering a new Google account rather than using a personal account.

SIGN IN WITH A DIFFERENT ACCOUNT CREATE A NEW GOOGLE ACCOUNT

Before you continue...

Accept developer agreement

Read and agree to the Google Play Developer distribution agreement.

☐ I agree and I am willing to associate my account registration with the Google Play Developer distribution agreement.

Review distribution countries

Review the distribution countries where you can distribute and sell applications. Learn more

If you are planning to sell apps or in-app products, check if you can have a merchant account in your country. Learn more

Credit card

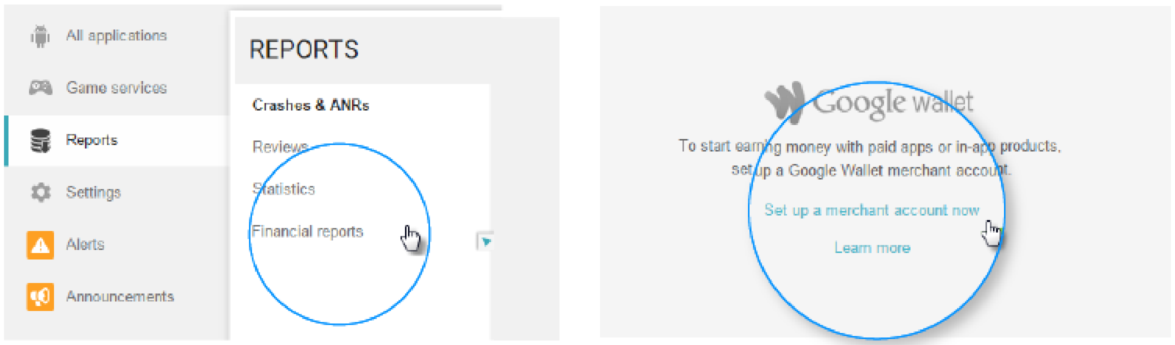
Make sure you have your credit card handy to pay the $25 registration fee i

CONTINUE TO PAYMENT

In order to upload a mobile app to Google Play Store, a developer dashboard is imperative. Developer console is kind of a backend controlling center, from where developers submit an app to Play Store. There is a one-time fee of $25 by which a developer can open an account, loaded with functions and control features. After paying this one-time fee, you can upload apps to Google Play Store for free.

You need to fill out all the credentials asked while creating the account, such as your name, country and more. Once you submit your account it will take upto 48 hours to get approved.

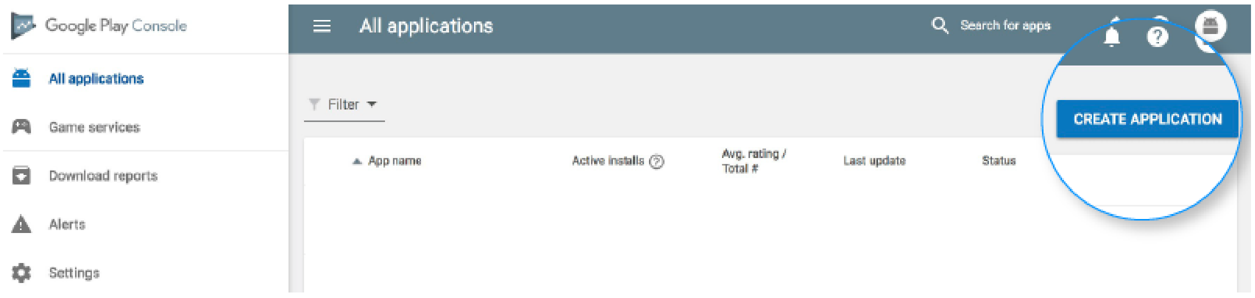
2. Link Developer Account with Google Wallet Merchant Account



If the app getting uploaded to Play Store supports in-app purchases, then you will need a merchant account. To create one you can sign in to your Google Console account and click on ‘Reports’ followed by ‘Financial Reports’ option. After this, you may select ‘Set up a merchant account now’ option and simply fill out your details.

The merchant account will automatically get linked to your Google Console account and will allow you to manage and examine app sales.

3. Create Application



This is yet another step towards how to publish an app to the play store.

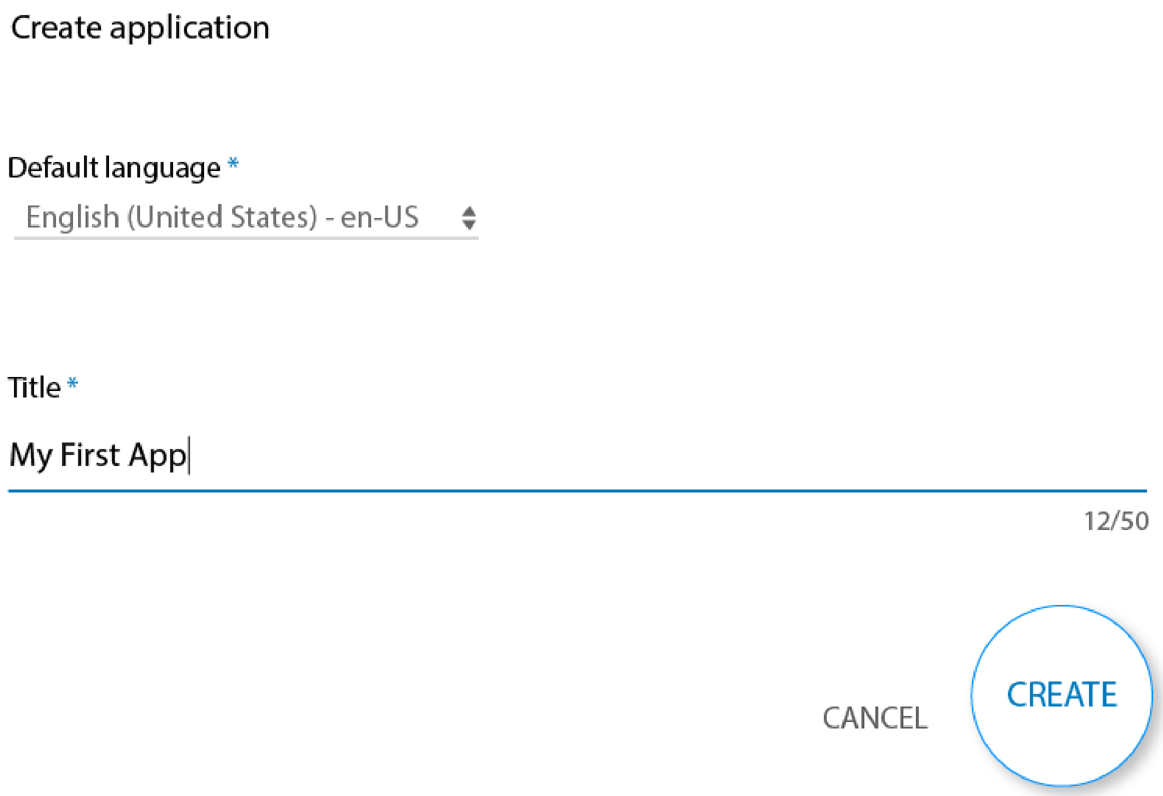
Once you are logged into your developer or publisher account, here are a few steps you need to take:

In the menu, go to the 'All applications' tab

You will see an option 'Create Application' – select it

From the drop-down menu, choose the application's default language

Enter your application's title (it can be changed later)


Create application

Default language *

English (United States) - en-US

Title *

My First App

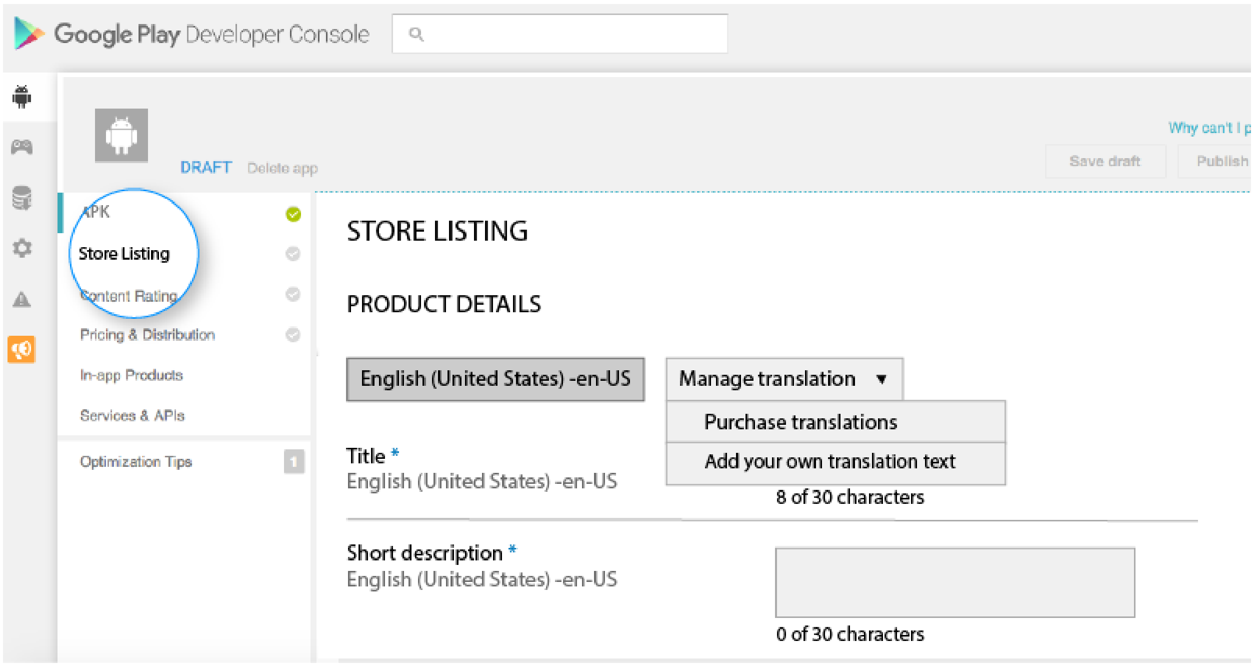
12/50

Now, click on “Create”

4. App Store Listing

It is at this point, your preparations will come handy.

In this step around how to upload an app to the play store, you are required to fill out all the information and details you have already prepared with caution before. The table below shows what information you need to fill in the app listing-



STORE LISTING

PRODUCT DETAILS

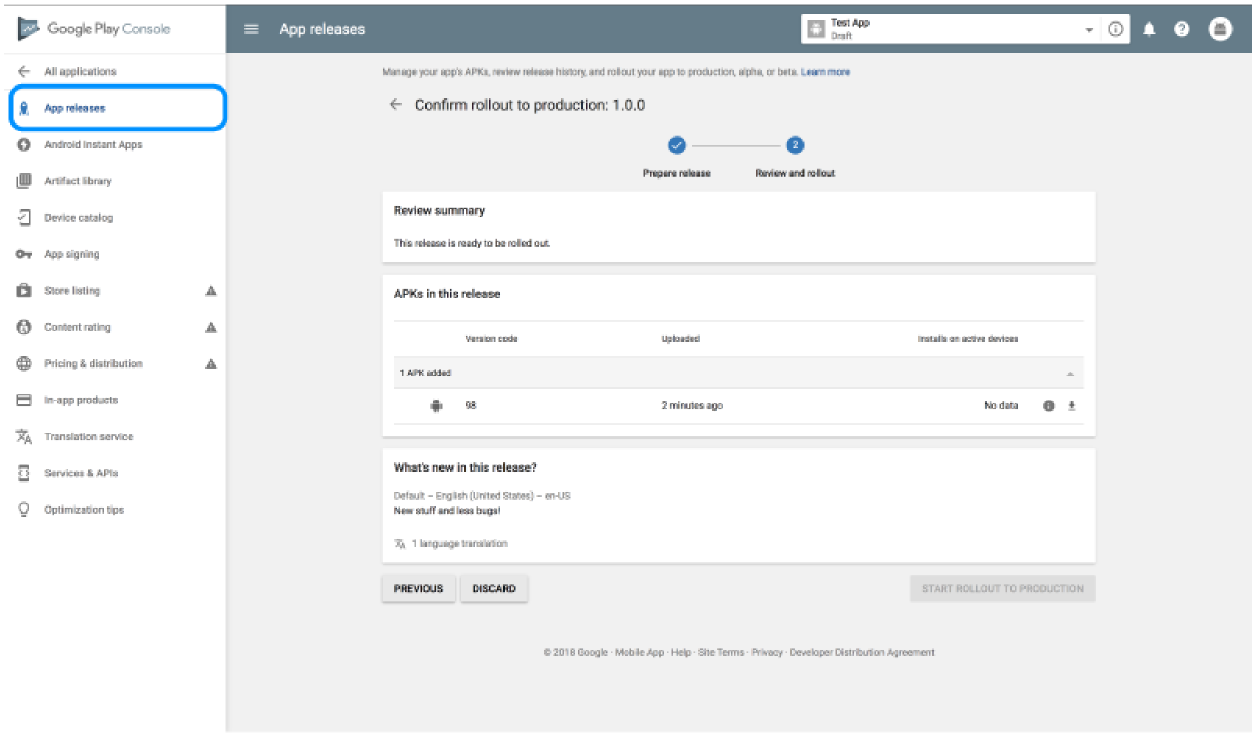
English (United States) -en-US | Manage translation ▼
- Purchase translations
- Add your own translation text

Title *
English (United States) -en-US
8 of 30 characters

Short description *
English (United States) -en-US
0 of 30 characters

Make sure to use appropriate keywords in your app description to increase the chances of your app showing up in searches. Along with this, make sure to use all the data we have talked about in the prerequisite section for app listing.

5. Upload App Bundles or APK To Google Play



Manage your app's APKs, review release history, and rollout your app to production, alpha, or beta. Learn more

Confirm rollout to production: 1.0.0

Prepare release — Review and rollout

Review summary

This release is ready to be rolled out.

APKs in this release

	Version code	Uploaded	Installs on active devices
1 APK added			
	98	2 minutes ago	No data

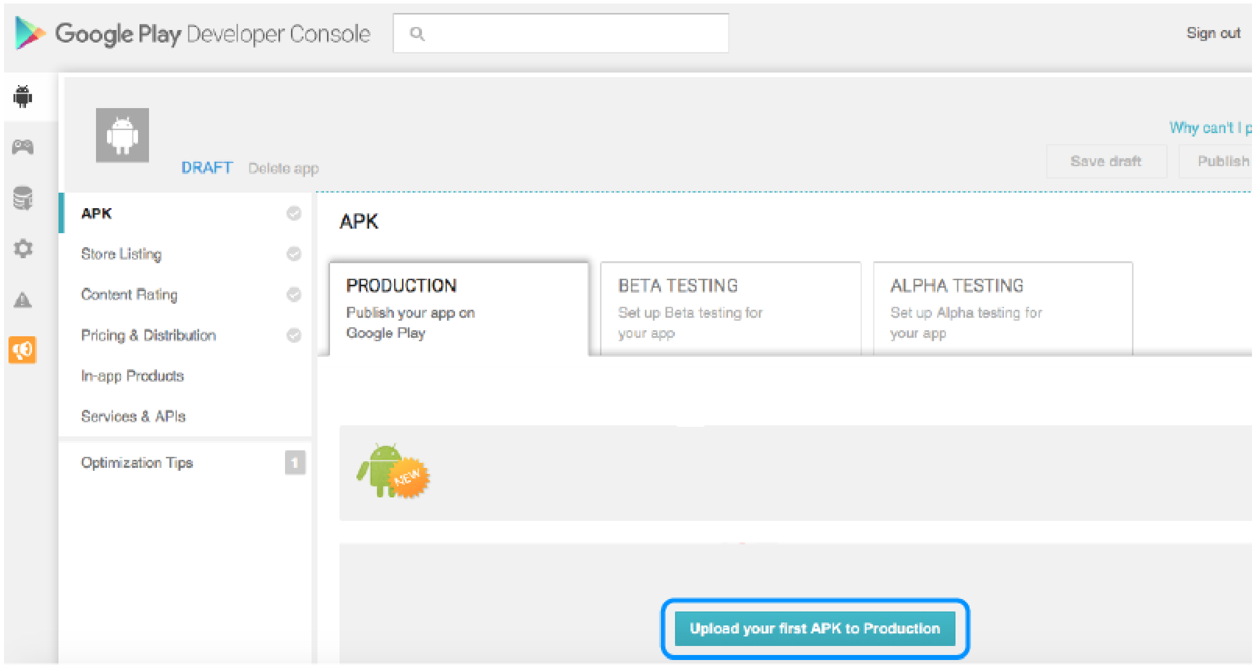
What's new in this release?

Default – English (United States) – en-US
New stuff and less bugs!

1 language translation

PREVIOUS DISCARD START ROLLOUT TO PRODUCTION

© 2018 Google · Mobile App · Help · Site Terms · Privacy · Developer Distribution Agreement



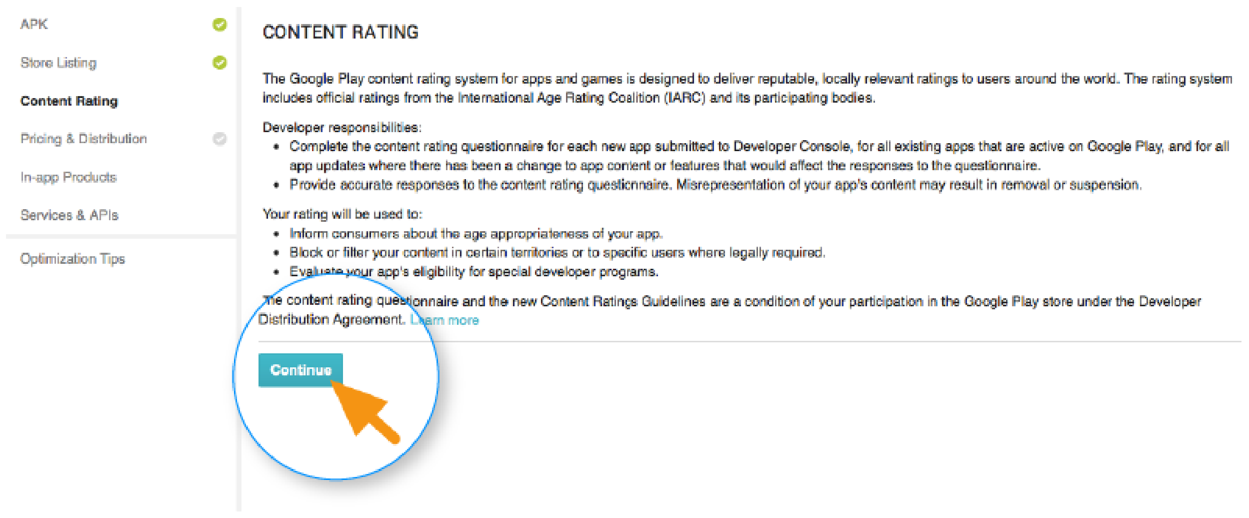
Now, you are required to use the files such as App bundle or APK and signed app release and upload them into your application. This is how you do it: Navigate to the 'Release Management' and then 'App Release' tab in the menu. After this, you will be asked to choose any one type of release from four options- internal test, close test, production release, and an open test.

Once, you have made a decision regarding which type of release you want, you may select ‘Create Release’.

At this point, you will be redirected to the New release to the production page. Here, you are again required to make another decision- to opt for Google Play app signing on the app or not. If you choose the latter, then simply click on the ‘OPT-OUT’ option.

Now, select ‘Browse files’ and then look into how to upload apk to google play store while naming and describing your release through on-screen instructions. You can also click on ‘Review’ to confirm the information. When everything is taken care of, press ‘Save’.

6. Time For Content Rating



CONTENT RATING

The Google Play content rating system for apps and games is designed to deliver reputable, locally relevant ratings to users around the world. The rating system includes official ratings from the International Age Rating Coalition (IARC) and its participating bodies.

Developer responsibilities:

- Complete the content rating questionnaire for each new app submitted to Developer Console, for all existing apps that are active on Google Play, and for all app updates where there has been a change to app content or features that would affect the responses to the questionnaire.
- Provide accurate responses to the content rating questionnaire. Misrepresentation of your app's content may result in removal or suspension.

Your rating will be used to:

- Inform consumers about the age appropriateness of your app.
- Block or filter your content in certain territories or to specific users where legally required.
- Evaluate your app's eligibility for special developer programs.

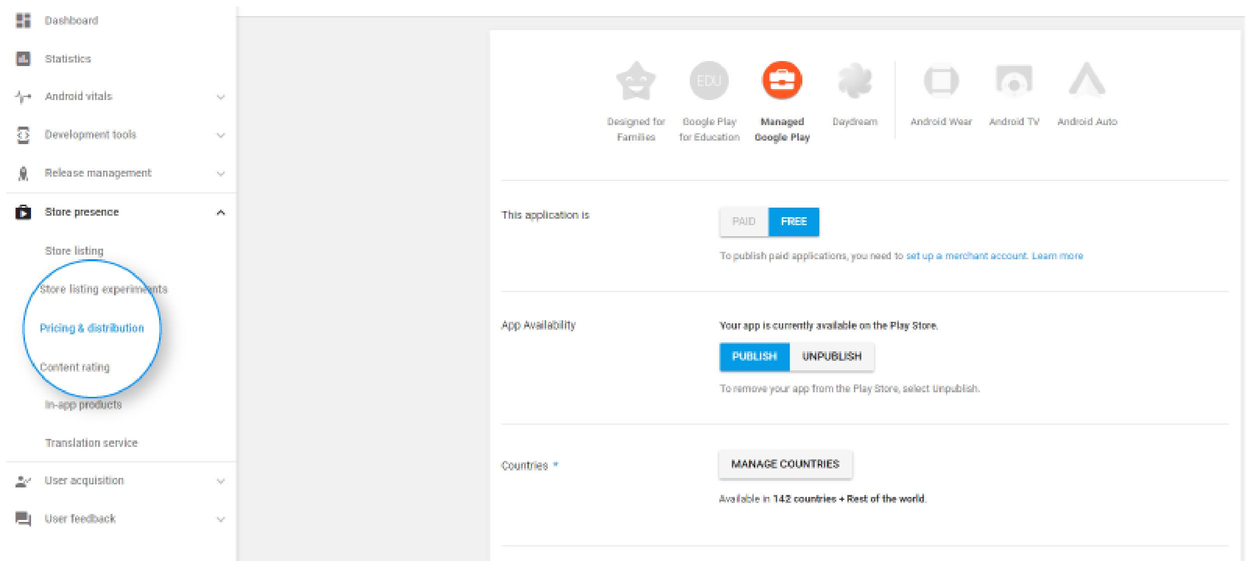
The content rating questionnaire and the new Content Ratings Guidelines are a condition of your participation in the Google Play store under the Developer Distribution Agreement. Learn more

The next step regarding how to publish apps on the Play Store is to rate your app. This is crucial for it is listed as ‘Unrated’, it might get removed altogether from the store, so it is imperative to rate the application.

For Content Rating, you must again navigate to the menu on the left side of the screen and then select the same. By clicking on ‘Continue’ you can move forward and then type your email address in the respective field and then ‘Confirm’ it.

Now, you may fill the questionnaire for your app rating. Follow this by selecting the ‘Save Questionnaire’ and then choose the ‘Calculate Rating’ option to see your app rating on the Play Store. The last thing to finalize your app’s content rating is to click on ‘Apply’.

7. Fix App Pricing and Distribution



This application is: PAID [✓ FREE]

To publish paid applications, you need to set up a merchant account. Learn more

App Availability: Your app is currently available on the Play Store.

PUBLISH UNPUBLISH

To remove your app from the Play Store, select Unpublish.

Countries *: MANAGE COUNTRIES

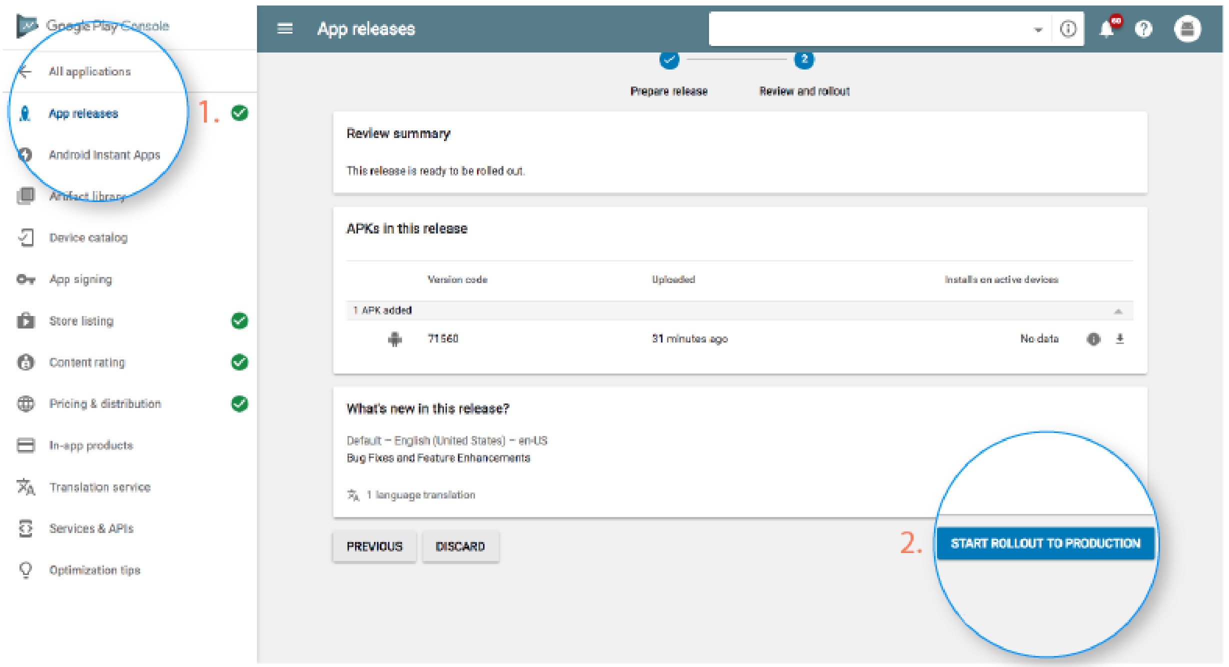
Available in 142 countries + Rest of the world.

Now, you have to be clear about what countries your app is going to be available in. The point to note here is that Google doesn't support publishing an app for all regions. The app will be published in selected countries instead of world-wide.

Moreover, assigning a price to your app is crucial. If you want your app to be free, make sure that this decision is permanent, as Google does not allow you to convert free apps into paid ones. Although, the price of the app can be altered.

To do all this, go to the Pricing and Distribution tab in the menu, and then make a choice whether your app is going to be Free or Paid. You may now select the countries you want your app to be released. Additionally, if your application is suited for children under the age of 13, you may select the option of 'Yes' for Primary Child-Detected. If otherwise is the case, simply select 'No'. Similarly, select the options for allowing ads into your application.

8. Finally, Publish the Application



Review summary

This release is ready to be rolled out.

APKs in this release

Version code	Uploaded	Installs on active devices
1 APK added		
71560	31 minutes ago	No data

What's new in this release?

Default – English (United States) – en-US
Bug Fixes and Feature Enhancements

1 language translation

Once you are confirmed about everything being correct, take the last step of this guide on how to upload an app on Play Store, i.e, add the application to the platform. You need to go back to the ‘App Releases’ tab and then select ‘Manage Production’ followed by ‘Edit Release’. After this, click on ‘Review’ and then choose ‘Start rollout to production’ option. To bring this process to an end select the ‘Confirm’ option and Voila! You have successfully uploaded the app to the Google Play Store for free.

All there is left to do now is to just wait for your application to get approved. It generally took about two hours for your application to get reviewed. But with Google Play’s updated privacy policy, it will now take hours and even days for the same, encouraging mobile app development companies to create even more flawless applications that get selected instantly. So, hold your excitement in the place and just wait.

How To Get Your App Featured On Play Store?

Your job to make sure the app gains popularity and thousands of downloads doesn't end here. After successful execution of the steps regarding how to upload an app to Google Play Store, it is now time to get it featured on Play Store.

There are certain practices such as user interaction and visual design services, working on the latest technologies, localization, etc. that helps your app to get featured. Getting featured on Google Play can benefit your app to an extent that it increases the attention of users on your app by multi-folds.

References

http://ai2.appinventor.mit.edu

Assessment

Based on the final project where ideas will be of students, design will be of students and programming will be done by students. Best three apps will be uploaded on play store