How to build App Inventor from the MIT sources

MIT Center for Mobile Learning

August 2012

This document contains instructions for how to build App Inventor from the sources and run it on a local machine or upload it to AppEngine.

1. Introduction

App Inventor for Android lets people create apps for android phones by piecing together programming blocks in a web browser. This document is a guide to building and deploying your own App Inventor service using the source code from MIT. It assumes that you've already had some experience using App Inventor to build apps, e.g., with the MIT App Inventor service. Now that you've used App Inventor, you might want to build your own service for personal use, or to share with a few friends, or as a prelude to working on App Inventor system development.

NOTE: Most development work in App Inventor has been done with Sun Java 6. The document points towards certain deficiencies in support for Java 7 in the different platforms. This is changing with the release of OpenJDK 7, but its support is not complete, so we still recommend Sun Java 6.
Feel free to use OpenJDK 7 and if you encounter any issues, please contact the Open Source Group.

1.1 Supported platforms

Building and deploying an App Inventor server can be done using MacOSX, GNU/Linux, and Windows operating systems, and we have done this extensively at MIT. There are instructions on building with Windows at the end of this document, but If you experience any problems, please contact the Open Source Group.

1.2 Hosting your service on App Engine, or running it locally

App Inventor is a Java service built on Google App Engine. As with any App Engine service, you can deploy your App Inventor service to Google's App Engine infrastructure at appspot.com, so that anyone on the Web can access it, provided you give them permission. If you are planning to do that, we suggest that you get some experience with App Engine and with Java App Engine services in Java. Google provides tutorials that show you how to install App Engine for Java and create simple services.

You can also can run the App Inventor service on your local machine using the App Engine "dev server" for personal testing and debugging. It is also possible to make your dev server available to small numbers of users on your local network, but the developer server is not designed to handle large loads, and we do not recommend using it to provide reliable App Inventor service to more than a handful of users.

1.3 Storing your users' projects

Your App Inventor service will maintain storage for individual user projects, both source code and compiled apps (APK files). This storage hosted on Google App Engine if you deploy your App Inventor service there, or held on your local machine if you use the dev server. In either case bear in mind that this storage is part of ***your*** service: If your service goes away, or you clear the database, the stored user projects will go away. It's a good idea to advise your users to download backup copies of their projects to their individual machines in case you need to rebuild your service.

1.3 Login and authentication

Your users will need to log in to your App Inventor service and provide a user name or other identifier. There are currently only two options for login:

1. If you host the service on appspot.com, then users should log in with their Google accounts.
2. If you host the service on a local server, then users can log in with any name at all, **and *there is no authentication (no passwords)***. As a consequence, users can read and write each other's projects, provided they know the login name. So this option will be currently useful only for your individual testing or with a small number of people who work together closely.

1.4 You'll need to supply a build server

The App Inventor service uses an auxiliary *Build Server* that handles the computationally intensive work of compiling and packaging apps. If you're running App Inventor on your local machine, that machine can act as its own build server. If you deploy your App Inventor service on at appspot.com, you'll need to obtain and configure a machine to run as the build server. The build server machine must be a MacOS or GNU/Linux machine that is accessible to the machine that hosts your App Inventor service. In particular, for a service running on appspot.com, the build server must be on the public Internet so that it can be accessed by AppEngine. It's also possible to designate a separate build server machine for a local (not on appspot.com) App Inventor service.

Sun Java 6 is, at this time, the safestoption to run the Build Server. This is due to the fact that the resultant package for the application has to be signed through the 'jarsigner' utility, only available with the Sun JDK in version 6. At this time it is not possible to run the Build Server with other JDKs such as Open JDK 6 (even though Open JDK is default installation for Ubuntu Linux). Things seem to be changing with the new release of Open JDK 7 which is now sponsored by Oracle, but not enough testing has been done with this distribution to make it a default recommendation. We hope that this changes in the near future.

2. Building App Inventor (MacOS)

This section describes how to build and run the App Inventor service on machines running the MacOS operating system. You should read through this section even if you plan to use GNU/Linux or Windows, since the instructions are similar.

Once you've built the App Inventor service, you can run it on a local server or deploy it to appspot.com. You'll also need to provide a build server, either by using your App Inventor machine itself as the build server or by providing a separate machine .

2.1 Setting up your MacOSX system: Required software

Before starting to work with App Inventor, you'll need following software:

- Java: We have been using Sun Java 6. We do not (yet) recommend using Java 7: Most of the building might work, but you won't be able to deploy on the dev server, because Google App engine does not yet support Java 7; and there might be other quirks. You can check your version by typing "java -version" at a shell.
- Apache ant: 1.8. 1 or 1.8.2. Type "ant -version" at a shell to check your version. If you are not running the right version, you may have to get it from the web
- bash: You should be running bash as your shell.

These should all be standardly included with the Mac OS. In addition, you'll need

- Git: We suggest that you manage your source code using the **git** distributed revision control system. Most Linux distributions come with Git you can also download and build it from http://git-scm.com.
- Google AppEngine Java SDK. Get and install the Java version of the SDK from https://developers.google.com/appengine/downloads. Make a note of the folder where you install it. We'll refer to this folder as the ***appengine SDK folder*** in the instructions below. (If you are planning to do some development using Eclipse and the Android plugin, you can install the SDK directly from the plugin).

2.2 The App Inventor source code

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

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

This will create a folder named "appinventor-sources" where the sources (and a copy of the repository) will reside. If you have problems with this command, please visit the MIT CML github's site at https://github.com/mit-cml.

We'll refer to this folder throughout the rest of these instructions as ***the appinventor parent folder***.

The appinventor parent folder also contains a file called *sample-.gitignore*. This file should be copied to another file simply named *.gitignore* which will keep generated files from being added to the git index (note that the file will be hidden in the finder and terminal after renaming. You can view it passing the flag *-a* to the *ls* command: *ls -al*).

2.3 Building App Inventor

With everything set up as above, connect to the appinventor parent folder. It contains a subfolder called *appinventor*, and we'll refer to this folder as ***the appinventor folder*** Cd to that folder and run the command

```
ant clean
```

Cleaning isn't necessary every time you build, but it's a good idea to clean each time before you build in case some junk has crept into the various build directories.

After you've cleaned, you can build App Inventor by running the command

```
ant
```

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

2.4 Configuration options

If you're building App Inventor to deploy on your local machine for testing or for personal use, you're ready to run the system. If you're deploying to appspot.com, you'll need to set up a build server and also specify some configuration options, both for the build server and for the App Inventor service itself. We'll describe those options below, but if this is your first time building App Inventor, you should just go ahead now and try it for yourself on your local machine.

3. Running App Inventor on your local machine (MacOS)

This section shows how to run App Inventor on your local machine for development and testing. Before you try to use App Inventor, you'll need to have installed the App Inventor setup software, just as with using MIT App Inventor. The setup software is the same for users and developers: If you've already been using MIT App Inventor on your computer, you needn't do anything extra to use your local version.

3.1 Launching App Inventor

You can launch your local App Inventor service by connecting to the appinventor folder and running the command (on a single line)

```
<your appengine SDK folder>/appengine-java-sdk-1.6.4.1/bin/dev_appserver.sh
--port=8888 --address=0.0.0.0 appengine/build/war/
```

Where <*your appengine SDK folder*> is the path to where you installed the App Engine SDK, and you replace "1.6.4.1" with the number of the Appengine SDK version you are running. (It's a good idea to periodically download the latest version of the SDK. Also make sure the dev_appserver.sh file is executable.)

You can change the port number and the locations of the AppEngine SDK and App Inventor directory to suit your system configuration. The *--address* flag is an AppEngine feature that lets other computers connect to your server over the Web.

The shell window will print status information as the server runs. You'll probably want to maintain this as a log for troubleshooting. For convenience, you may want set the server up to run detached, and save its output to a log, with a command line (all on one line) like:

```
nohup
<your-appengine-SDK-folder>/appengine-java-sdk-1.6.4.1/bin/dev_appserver.sh
--port=8888 --address=0.0.0.0 <your-appinventor-folder>/appengine/build/war/ >
serverlog.out &
```

You'll find it useful to create a shell script that runs this command, to save yourself the bother of typing it over and over.

Run this command and wait about 30 seconds while the system starts up. (You can follow the progress in the log.) Then browse to localhost:8888 and you should see App Inventor running. People on your local network should also be able to use it by browsing to <*your IP address*>:8888.

Test the system by creating some projects. This should be an almost completely working App Inventor system.. You should be able to open the Blocks editor, connect the phone or the emulator, download and upload files -- in short, everything except packaging apps. You'll need a buildserver to package apps.

3.2 Launching a build server on your local machine

You'll need to set up a build server. You can use your local machine as the build server. You can also designate a different machine as a buildserver, as described below in in *Setting up a Build Server*. To start and run the build server on your local machine, open a new shell, connect to the buildserver subdirectory of the App Inventor directory

```
cd your appinventor directory
cd buildserver
```

and run

```
ant RunLocalBuildServer
```

If all is successful, you should see output that ends in

```
[java] Server running
```

As with the App Inventor server, you'll probably want to maintain a log, and also run detached, with a command line like:

```
cd ~/appinventor-build/appinventor/buildserver
nohup ant RunLocalBuildServer > ../../buildserver-log.out &
```

When you build a project, it's useful to watch the buildserver output to see the steps it goes through in packaging and to verify that things are running. Packaging can take a long time, especially if you are using a small machine for the build server.

You might want to create a shell script that runs this command together with the command above that launches App Inventor, so you can launch App Inventor and start the build server with a single command for testing and debugging.

You can verify that your build server is running by looking at the log, or by viewing some status information with a web browser. Browsing to

```
localhost:9990/buildserver/vars
```

will show you some status information about the server and the machine it's running on. Also, browsing to

```
localhost:9990/buildserver/health
```

should respond *OK* if the server is running.

4. Building and Launching App Inventor (GNU/Linux)

This section describes how to build and run the App Inventor service on machines running the GNU/Linux operating system. Once you've built App Inventor, you can run it on your machine's dev server for testing and development or you can deploy it to appspot.com. You'll also need to create a build server. The steps here are almost identical to those for MacOS.

4.1 Setting up your GNU/Linux system: Required software

Before starting with App Inventor, make sure your server includes the following software:

- Java: We have tested build using Sun Java 6. Other versions might work, but we haven't tested these. We recommend using Sun Java as opposed to OpenJDK. Feel free to experiment with OpenJDK for building App Inventor, but there may be incompatibilities. In particular, when you are using App Inventor to create apps, the Blocks Editor will not work under OpenJDK due to the fact that the needed utility to sign jar files "*jarsigner*" is not provided with OpenJDK. Once you've installed Java, check that Java and Java Web Start both work by trying the standard App Inventor "test your Java configuration" tests at http://beta.appinventor.mit.edu/learn/setup/index.html. If this doesn't work, you might need to reinstall Java.

- Apache ant: You may need to install Apache ant, or you may need to update your installed version to 1.8.1 or 1.8.2. Check your version.

- bash: You should be running bash as your shell. To make ant work more conveniently, under GNU/Linux, add the following to your .bashrc file, making the necessary changes to reflect the actual directories where you've installed Java and ant:

```
export ANT_HOME=/usr/share/ant/apache-ant-1.8.2
export JAVA_HOME=/usr/local/buildtools/java/jdk
export PATH=${PATH}:${ANT_HOME}/bin
```

- git: We suggest that you manage your source code using the **git** distributed version control system. You can find an appropriate distribution at http://git-scm.com. Many Linux distributions also come with git as part of the distribution.

- Google AppEngine Java SDK. Get and install the latest Java version of the SDK from developers.google.com/appengine/downloads.html.

4.2 Building App Inventor and Launching it on your local machine (GNU/Linux)

Once you have the required software installed, the steps for getting the source code, building App Inventor, and launching it on your local machine for testing and development are the same as for MacOS described above.

5. Setting up a build server

The instructions above showed how to deploy App Inventor on a local machine, which functions both as the App Inventor service and the build server. If you are deploying the App Inventor service on Google App Engine at appspot.com., you'll need to provide your own build server. This section describes how to set up a build server. The following section will show how to use this build server in conjunction with appspot.com to deploy a service that can be used by anyone on the Web (and you can also set up access restrictions).

The instructions here are for build server machines must be running the Linux or MacOS. Setting up build servers for Windows are described at the end of this document. The build server machine must be on the public internet so it is reachable from appspot.com. For example, you can't set it up so that it runs behind a firewall.

The steps in creating a buildserver are

1. Create the software for the buildserver machine
2. Install the software on the buildserver machine and start the server, as described in this section.

Section 6 will explain how to launch your service on App Engine and configure it to use your build server. Section 7 will discuss some configuration options you can specify for build server and for the App Inventor service itself.

5.1 Create the buildserver software package

To create the buildserver software, first build app inventor as explained above in section 2. But

don't go on the launch the service as in section 3. Instead, open a new shell, connect to the buildserver subdirectory of the App Inventor folder

```
cd your appinventor directory
cd buildserver
```

and run

```
ant BuildDeploymentTar
```

This should generate, in the appinventor folder, a file:

```
build/buildserver/BuildServer.tar
```

Create a new folder on your desktop called `for-BuildServer` and copy `BuildServer.tar` to that new folder.

Look in the App Inventor folder for a subdirectory called `misc`. In it, there should be a file called `launch-buildserver`. Copy `launch-buildserver` to the `for-Buildserver` folder. The `for-buildserver` folder now contains the software package to be installed on the buildserver machine.

5.2 Install the software on the buildserver machine and start the build server.

The buildserver machine must be reachable from the public internet.
Log on to that machine and pick a directory to hold the buildserver software. We'll refer to this folder as the ***buildserver folder***. Place `Buildserver.tar` and `launch-buildserver` in the buildserver folder.

Connect to the buildserver folder, and run the command

```
tar -xf BuildServer.tar
```

This should produce a folder called `lib`.

Finally, in the buildserver folder, run

```
./launch-buildserver
```

See below for some of the options you might want to specify for launch-buildserver

Running launch-buildserver will start the build server and create a log file `buildserver-log.out`. If you view the this log, you should see startup messages, ending in "server running". Later, as you package apps with your App Inventor service, you can examine this log file to watch the build progress and to help diagnose build errors.

> *Note*: launch-buildserver attempts to save the buildserver log file in the directory /home/buildserver. This will fail if there is no such directory on the buildserver machine. You can change this target directory with one of the buildserver options (see below).

To check that the buildserver is running, use a browser to visit the URL

> `http://`*<your buildserver machine>*`:9990/buildserver/vars`

Where *<your buildserver machine>* is the hostname (or IP address) of the buildserver machine. If the build server is running, you should see a page with status information. If you don't see this, check the buildserver log for clues as to what might be wrong.

6. Deploying App Inventor on Google App Engine

Once you've gotten experience running the App Inventor service on your local machine, you can deploy your own service Google AppEngine at appspot.com and make it available to anyone on the web. This section describes how to deploy the service on Google App Engine from machines running the MacOS or GNU/Linux operating system. Instructions are the same for both systems. Deploying on Windows is discussed at the end of this document.

The instructions here assume that, as described above in sections 2 and 3, you've already built and launched App Inventor on your local system, and that you know how to set up a build server,

6.1 Issues to consider before deploying on App Engine

Here are some things to keep in mind about deploying on App Engine. These considerations are especially important if you are going to provide the system as a service for others to use, rather than just for personal experimentation.

6.1.1 App Engine hosting; getting an Application ID

App Inventor runs as a Java web service on Google App Engine. As with any App Engine service,

you deploy App Inventor as a hosted service on Google's infrastructure at appspot.com.
The instructions below are self-contained. But if you are going to be hosting a service for others, we strongly recommend that you get some experience with App Engine and Java App Engine services. There are several tutorials on the Google App Engine site that describe how to get started.

You'll need to create an application on Google App Engine to run your App Inventor service, and register an *application id* for it. You can learn how to do this from the App Engine tutorials.

6.1.2 User login and authentication

The local App Inventor service you created above had no user authentication. In contrast, the service on App Engine requires people to log in with a Google ID. In principle, anyone on the web with a Google ID can use your service, which is probably not what you want. To restrict access, App Inventor lets you configure a *whitelist* , where the only people who can use the service are those whose IDs you include on the whitelist. You can set up the whitelist when you set the configuration options.

6.1.3 Storage for user projects

The App Inventor service will maintain storage for individual user projects, both source code and compiled apps (APK files), of Google App Engine data bases. Remember that this user project storage is part of ***your*** service: If you clear the database, then the stored user projects will go away. If you are experimenting with the service, and also providing it for others, you should advise your users to maintain backup copies of their projects in their individual machines in case you need to rebuild the service and clear your database.

6.1.4 Build server

In contrast to a local service, where the local App Inventor service can also be the build server, you must provide a separate build server machine for an App Inventor service hosted on App Engine. The build server machine must be on a public network where Google App Engine can reach it.

6.1.5 AppEngine charges for use

Google charges for App Engine use, beyond a certain amount. This limit should be adequate for personal use or in providing a service for a few users. But with more than that, you may find yourself running up against App Engine's free use limit. If you are running a service for others to use, you should experiment with small user groups and keep monitoring the load.

6.2 Building and deploying App Inventor and the Build Server

Deploying to App Engine has two main parts:

1. build App Inventor and upload to App Engine
2. set up a build server and configure your App Inventor service to use it

There are also configuration options for the App Inventor service and the build server that you may want to set. (See below)

6.2.1 Build App Inventor and deploy it on appspot.com

Build App Inventor just as you did above in section 2. But don't go on to run it on your local machine.

6.2.2 Deploy your service to appspot.com

Connect to the App Inventor directory on your local machine and run the command (all on one line)

```
<your-appengine-SDK-folder>/bin/appcfg.sh -A <your-application-id> update
appinventor/appengine/build/war
```

where <*your-appengine-SDK-folder*> is the path to where you installed the App Engine Java SDK, and <*your-application-id*> is the application id you created with Google App Engine.

> Note: You may want to create a script with the command line above, to avoid having to retype it in the future.

Now you should be able to browse to <your-application-id>.appspot.com and see your App Inventor service running there. You should be able to use it, open the blocks editor, and work interactively with the phone or emulator -- everything but packaging apps.

6.2.3 Set up a build server and configure your App Inventor service to use it

To package apps, you'll need a build server. Set up and launch a build server as described above in section 5, and test that the server is running and is reachable from the public internet.
Now connect to the appinventor folder on your local machine find the file

```
appengine/build/war/WEB-INF/appengine-web.xml
```

This assumes you've already built App Inventor on that machine. *Warning:* Make sure you edit the file in the folder `appengine/build/war` and not the version of the file in `appengine/war`.

Change the line that reads

```
<property name="build.server.host" value="localhost:9990" />
```

to replace "localhost" with the name for your Build Server machine, specifying port 9990, for example, `value="mybuildmachine.myschool.edu:9990"`.

> Note: Keep the port at 9990. It is possible to change to a different port, but this requires also making some other changes, and changing the port where the buildserver is listening See the section below on *Changing the Build Server Port Number*.

Re-deploy your App Inventor service using the same appcfg.sh command as above, and test that you can package an app. If something goes wrong, use your browser to check the build server's /buildserver/vars page to verify that your build server is running and that your App Inventor service can reach it over the network. It might also help to look at the information shown if you click the *Debugging* link on the App Inventor Designer page.

7. Configuration options for the App Inventor service

This section describes some useful options you can configure for your App Inventor service.

7.1 Setting up a white list

The App Inventor server can configured to use a whitelist: only users on the list will be permitted to log in. Others will get a message saying that the server is not available. The whitelist is located in the file

`<your-appinventor-folder>/appengine/build/war/WEB-INF/whitelist`

Edit this file to list the user addresses you want to permit, one address per line. Don't forget to include your own address.

Now turn on the whitelist restriction: Find the file

```
<your-appinventor-folder>/appengine/build/war/WEB-INF/appengine-web.xml
```

Edit the line that reads

```
<property name="use.whitelist" value="false"/>
```

to change “false” to “true”.

You can leave the whitelist restriction off if you wish. Bear in mind that if you do so, anyone will be able to connect to your service.

7.2 Message of the day

You can configure App Inventor specify a message of the day (MOTD), that users will see at the upper right of the designer. This feature is turned off by default, and we suggest that you keep it off.

7.2.1 Enabling the message of the day

But if you do want to display a MOTD for your service, find the file

```
<your-appinventor-folder>/appengine/build/war/WEB-INF/appengine-web.xml
```

Change the line that reads

```
<property name="motd.check.interval.secs" value="0" />
```

to make the value an interval for checking is there is a new MOTD. For example, a value of 900 would check every 15 minutes. A value of 0 disables the MOTF.

We suggest that you keep the checking interval long, or keep the MOTD disabled, because the constant checking puts load on the App Engine service, which will increase your instance’s usage statistics and may put you past the free limit of App Engine services.

7.2.2 Setting the text for the message of the day

Do this step only if you chose to enable the MOTD .

Once your App Inventor service is running, you can specify the text for the MOTD by opening the

administration console for your service (from appspot.com) and selecting Data Viewer on the left (under data). Use the "By kind" dropdown list to select MotdData and open up the item with id=1. Under the Content section, you'll see a box for Value. You can edit the contents of that box to create the new MOTD text. You can make this MOTD change at any time. You don't have to restart the service in order for it to take effect.

7.3 Terms of Service

People are asked to agree to terms of service the first time they connect to your appinventor service. The terms of service are an HTML document located at

`<your-appinventor-folder>/appengine/build/war/Ya_tos_form.html`

and you can edit these as you wish.

To configure App Inventor to now ask for people to agree to terms of service at all, go to the appinventor folder and find the file

`<your-appinventor-folder>/appengine/build/war/WEB-INF/appengine-web.xml`

Edit the line that reads

```
<property name="require.tos" value="true" />
```

to change "true" to "false".

8. Configuration options for the Build server

This section describes configuration options for the build server. Generally, these are specified as command-line options to launchbuildserver.

8.1 Version number matching

Each App Inventor release has a version number (e.g., "61"). You'll find this version shown at the bottom right of the designer screen in the browser, at the bottom left of the blocks editor window, and at the beginning of the Build Server log. These versions must match, or else App Inventor won't build projects. When you update to a new release, be sure to update both the build server and the App Inventor service. If you are using someone else's Build Server, you must work with them to both use the same version.

You can configure App Inventor to not enforce version number matching, something that might be

useful when you are testing new versions.

To turn off version number matching, go to the appinventor folder and find the file

`<your-appinventor-folder>/appengine/build/war/WEB-INF/appengine-web.xml`

Edit the line that reads

```
<property name="build.send.git.version" value="true" />
```

to change “true” to “false”.

8.2 Checking the Build Server Status Information

To check whether the Build Server is running, browse to the URL

http: <address:port>/buildserver/health

where <address:port> is the name and port where you deployed the Build Server. If the server responds “OK”, then it is running and accepting build requests.

If you browse to

http: <address:port>/buildserver/vars

you’ll see information about the status of the server including memory usage and load average. There’s also information about the number of jobs built and rejected and the number of simultaneous builds, which might be useful in tuning your server to match the load (see below).

8.3 Changing the Build Server Port Number

The build server port is set by default at 9990. You should leave it set at 9990 if you can, but if you really need to change it, you can do it as follows:

First, as described above under “Specify the Build Server machine”: Go to the appinventor folder and find the file

`<your-appinventor-folder>/appengine/build/war/WEB-INF/appengine-web.xml`

Change the line that reads

```
<property name="build.server.host" value="localhost:9990" />
```

to the correct hostname and port.

Then launch the Build Server to use this port number by running launchbuildserver with the command line option

`--port n`

where *n* is the port number. You should use your browser to check the build server status and verify that the it is listening on the correct port.

8.4 Specifying the maximum number of simultaneous builds

Building projects is a compute-intensive task. Several users trying to build simultaneously can overwhelm a build server. Your users will notice this by the fact that projects take a very long time to build.

You can configure the Build Server to limit the number of build tasks it will work on simultaneously. If the maximum number of simultaneous builds is set to 8, for example, and there are currently 8 builds going on, the an attempt to do an additional build will get a response saying that the server is busy, with a request to try again in a few minutes. The number of rejected requests is included in the Build Server status information.

You can adjust the max number of simultaneous builds and experiment with what works, given your build server and the number of users who are working with it. The default is set to allow an unlimited number of simultaneous builds. You'll probably want to change this if you are deploying a service for public use. Something like 8 simultaneous builds is a realistic number for a reasonably powerful machine.

To change this parameter, edit the launch-buildserver script as described above for the port number, adding the option

--maxSimultaneousBuilds 8

or whatever number you want to use instead of 8. Setting the number to 0, or leaving out the optional argument, makes the maximum number of simultaneous builds unlimited.

8.5 Restricting the allowable App Engine hosts

Normally you build server will respond to any App Inventor that requests it to build. You can restrict this to a list of designated hosts with the requiredHosts option to launchbuildserver. Calling launch builder with the option

--requiredHosts ***list of hosts***

will cause the build server to respond only to hosts in the list.

8.5 Path for the Build Server log

You can change the location where the Build Server will write its log by using the logpath option:

--logpath *<path to log file>*

9. Running App Inventor locally with an external build server

Note: This isn't implemented yet. For now, if you run App Inventor on your local machine, you must also use that machine as the build server.

10. Building and launching on Windows

These are notes on how to do this. They are not comprehensive but have been tested. Feel free to contact the Open Source Group if you have any trouble.

It has been tested (build the source code, start the dev server and the build server, and package the app) in the Windows 7 Home Premium 32 bits and 64 bits and Windows 7 Ultimate 32 bits. It expects to work with all the Windows 7 versions. For Windows XP, the following instruction are the same, except for the location of the Windows System Variables.

Prerequisite

Software

1. Ant (http://ant.apache.org/bindownload.cgi)
    a. You might need to add Ant to Windows System Variables
    (For Windows 7, My Computer - Properties - Advanced System Settings - Environment Variables - System Variable)

2. GitHub (https://github.com/)
    a. You might need to add git to the Windows System Variables
    b. After installed the Github, you can clone the App Inventor source code from here.

3. AppEngine SDK (https://developers.google.com/appengine/downloads)
    a. The current supported version for MIT App Inventor is 1.6.4.1.

4. Android SDK (http://developer.android.com/sdk/index.html)
    a. Recommend to use the installer version

Notes:

The links are updated as 05/31/2012, so the links might change in the future.

When running from the Command Prompt, one might need to make sure there is no space between any letters in the directory. If there is, you need to change the directory, or get a new account name without a single space (Windows 7 doesn't support to change the account name).

Please note that the default directory will be C:\Program File\xxx, and the there is a space between the word Program and File. The recommended installation path will be a direct sub-folder in the C:\. If you follows this path, and you have existing Java JRE or JDK installed, you need to uninstall them and then do the installation, or add the new variable path to the "Path" variable under the System Variables.

Finally, you need to check whether all the required variables are set properly by entering "set" into the command prompt. If they are not, you might need to double check that you make the change using the admin account, or reboot the OS (in some rare cases).

Building

There is no change if you change the source code correctly above. You might want to double check the JAVA_HOME by enter the command "echo %JAVA_HOME% " to the command prompt. There are many compiling errors related to this issue.

During the building process, you might encounter some warning, or notes which show the directories have backward slashes attached to it. This won't affect anything, we can fix that by couple changes of the source code. We leave it as it since we want to minimize the code modifications.

If you do see some errors, or failures during the building process, go back to double check the source code and do a "ant clean" before you do the next "ant".

Launching

When running the App Engine, you should call the **.cmd** file, NOT the **.sh** file. If there are exceptions

thrown out in the command prompt at the first time, you need to do an “ant clean”, and build the entire source codes again.