Prototype Android Scouting App Using MIT App Inventor 2

About this document

This document explores the use of MIT App Inventor 2 to create Android apps for use by the Team 100 Scouting team. An example application was created by Laura Rhodes in December 2015 to evaluate:

- the ease of use of App Inventor by beginning programmers
- how quickly could an app be put together
- how easily can an app be modified
- can we reproduce (as a minimum) the functionality of the aging Nintendo DS system currently being used by Team 100 for scouting.

Background

Why do we Scout?

The main purpose of *scouting* is to gather quantitative and qualitative data about other teams' robots to:

- identify potential alliance partners whose capabilities complement our own
- to provide this information in a easily and quickly digestible form for use by the team captain during the alliance selection process
- to develop real-time game play strategies
    - strategies that will maximize the capabilities of the robots/teams on our alliance
    - strategies to defeat capabilities of the robots/teams on the opposing alliance
- glean ideas for future robot designs, techniques, procedures

Pit Scouting

Pit Scouting refers to the gathering of data about other robots based upon up-close observation and conversations with the other teams' members in the pit area. This activity mainly takes place in the pit area on the first day of the competition, typically a practice day only with no qualification matches. Typically, Team 100 has done pit scouting with a paper-based system. A key part of pit scouting is taking photographs of each of the robots. These photographs serve several purposes:

- Reduced-resolution versions are loaded into our scouting devices (NDS) to help identify robots on the field.
- Photos are useful during team discussions about potential alliance partners. These discussions usually take place during the evening of the second day of qualifications after a full day of qualification rounds.
- Many robot design ideas can be gleaned from the photos.

Field Scouting

Field Scouting refers to the gathering of data about other teams/robots based upon observations of their performance on the field during the qualification matches. For the past several years, Team 100 has used a scouting system based on the Nintendo DS. We use 6 NDS's, one for each of the six robots on the field. Student scouters rotate in to take data on each of the robots during each of the qualification matches.

Shortly before the conclusion of the qualification matches, the data collected on each of the NDS's is extracted and processed. Formatted data is provided to the team captain for use during alliance selection. In the past few years, we have used Tableau software (donated to FIRST teams) to provide advanced visualization of our scouting data.

Why abandon NDS Scouting System

The Nintendo DS system has stood the team well for many years, but it is time to move on. Our "fleet" of NDS's is aging – batteries no longer hold their charge, some of the devices are lost or broken. Software on the NDS platform is difficult. The programming language is a fairly low-level "C". In addition, the system was not designed to be user programmable. Nintendo does not support the development tools. We used the DevKitPro tool chain with the PAlib Nintendo DS library functions. These tools have very limited support documentation. We had to defeat the built-in security on the NDS using a SuperKey to bypass the security on the DS, allowing non-Nintendo approved cards to be run, a Super Card to take an SD card and turn it into a GBA card and a separate SD card. To extract the information stored on an NDS, it is necessary to turn off the device, remove the Super Card, extract the SD card, and read its contents with an SD card reader on a laptop computer.

Extensive preparations need to be made to prepare a configuration file for each NDS and a detailed match list needs to be carefully prepared and loaded onto the NDS in order to use them for taking data. In addition, the data output needs to be extensively processed off of the NDS in order to be of use.

Why Android?

Android is the most popular operating system in the world. More than 80% of new smartphones use the Android operating system. Team 100 owns two Nexus 7 Android tablets, donated by Google. We feel confident that, if we develop our capability to create an Android scouting app, we can get 4 more tablets donated. The larger screen size of the tablets (compared to the NDS) makes for a better scouting experience. The development environments for Android are well supported and documented. No "quasi-legal" devices are needed.

A secondary objective is to give our Team 100 students the opportunity to develop marketable skills in the development of mobile apps. More advanced students will have the opportunity to explore the development and use of data bases and also cloud services.

Nintendo DS App

History

Our current NDS scouting system was originally developed for the 2009 FRC season by Team 100 student, Nicholas Carlini, when he was a senior on the team. In 2010, Nicholas came back as a mentor and extensively revised the system so that it could be easily modified without rebuilding the device's firmware. This newer system could be modified for each year's new game, simply by changing the contents of a configuration file loaded on the SD card inserted in the NDS. Since that time, only small revisions have been made to the base software on the NDS. New configuration files were developed for each year's FRC game. Support utilities (in Java and in Python) were developed to process the match data (which needed to be stored on the NDS) and to process the output file data to make it useful for alliance selection. This tool chain was patched together and required a student expert to develop and operate.

Using NDS Devices to Scout

Prior to the Competition

In the weeks leading up to an FRC competition, the scouting team had to design and prepare the NDS configuration file based upon the current year's game and input from the drive team and other students. This configuration file was then tested on a GameBoy emulator and on the actual NDS devices. All of the team's students were trained in the use of the devices so they would be ready to scout in the real matches. All of the NDS's SD cards were loaded with the application file, the configuration file, and any old output files were deleted. Paper forms were developed for pit scouting and for "qualitative scouting" at the event and sufficient paper copies were printed.

At the Event

- Take Photos of Each Team at Regional on a separate phone or digital camera
- Paper Pit Scouting
- Obtain Match Schedule from the FIRST website
- Format Match Schedule file
- Load Files onto six NDS's
- Clear out data file from six NDS's
- Take data during all Qualification rounds

At the conclusion of the Qualification Rounds

- Extract output data files from six NDS's
- Post process Data
- Provide team captain with data for alliance selection

Android Scouting

Team 100 History

In 2014, Google donated two Nexus 7 Android tablets to Team 100. A number of students were very eager to learn to develop Android apps. They soon realized that Android development using the (then newly released) Android Studio was very difficult. Significant progress was made, but nothing approaching a usable scouting app was competed in the 2014-15 FRC season.

App Inventor to the Rescue?

In the fall of 2015, FTC students were given a new control system based on Android phones both on the robot and at the driver station. They were given two choices in development environments – Android Studio and App Inventor. The FTC students at Woodside used Android Studio, even though most of them had no programming experience. In preparation for being a Control Systems Advisor for an FTC qualifier, I determined that I should be knowledgeable in both development environments. I tried out App Inventor and was impressed with the speed at which apps could be developed and the approachability of the environment, with its drag-and-drop programming, to students new to programming. I thought that App Inventor may be a simpler development platform for a scouting application and decided to build a test application that reproduced the functionality of the NDS scouting app. In one morning's time, I was able to create an Android app in App Inventor that did about 90% of the NDS functionality. Over the next week, I cleaned up this app a bit and this document gives the results of this investigation.

What is App Inventor

App Inventor is a web-based development tool jointly created by Google and MIT. It allows a user to quickly turn an idea into an app without the steep learning curve of more traditional app creation processes. It transforms the complex language of text-based coding into visual, drag-and-drop building blocks. App Inventor lives on the Internet at http://appinventor.mit.edu/explore/. There are many online resources available to learn App Inventor, from books to YouTube tutorials. App Inventor has two main operating modes. In the **Designer** mode, one creates the user interface by dragging components onto the screen. In the **Blocks** mode, the event handlers for the actions to be taken when the user interacts with the screen's buttons, text boxes, and other components, are written in a graphical language similar to Snap or Scratch. Once an app is developed, it can be transferred to an Android device over Wi-Fi or over a USB cable connected to the PC. An app can also be tested in an Android emulator running on the PC.

App Inventor Benefits

- Apps can be developed quickly
- Students familiar with Snap (used in the Intro To Computer Science classes at Woodside) will immediately feel at home with the Blocks editor in App Inventor.
- A large body of tutorials are available
- The system provides hooks to access media, the camera, the web, local and cloud-based data bases, the file system – so very sophisticated applications can be developed.

App Inventor Limitations

All is not perfect. Some things are lost with the simplicity of App Inventor.

- The number of types of screen widgets available is limited
- You cannot create custom widgets or use more complicated Android features such as fragments.
- Cannot create screen widgets dynamically – they must be added to the screen at design time (although they can be hidden until they are needed).
- Multiple screen projects are difficult to maintain.
- It's difficult to have multiple developers on the same app

App Inventor Scouting Prototype

I chose to reproduce the 2012 game's NDS scouting app because it was well documented (In our Team-100-Robot-Code-Archive GitHub repository) and because I had the match schedule and robot photos from the SVR regional from that year. Because of the limitations of App Inventor not being able to create screen widgets dynamically, I did not use the configuration-file-based approach, but instead just created a full custom app for that FRC year. I re-used the same file structures for the match schedule input file and the data output file. I would recommend better file structures for future versions. The system used files (for match schedules and robot photos) dragged onto the Android device into a "Scouting" folder accessible from Windows Explorer when the Android device is plugged into a USB port on the PC. It's also feasible transferring these files into the Android directly using a file system app, a USB-stick and USB-On-the-Go (OTG) cable to transfer files – No Need for Wi-Fi. Wi-Fi communication is often disallowed at FRC events.

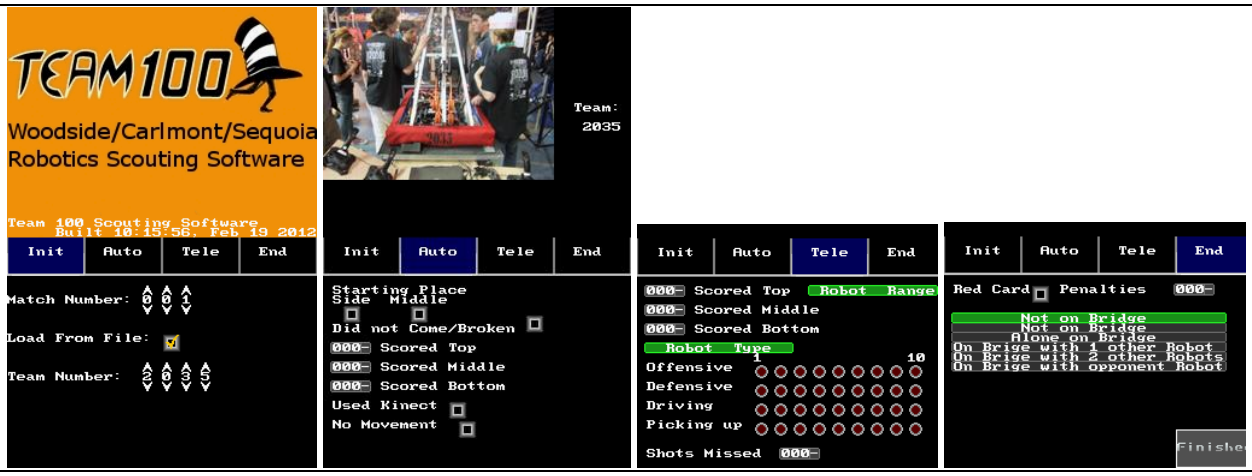
Reproduces 2012 NDS Functionality

Potential Future Work

This app was developed as a proof-of-concept. There remains much that can be significantly improved, including:

- Nicer screen formatting
- Data output post-processing
    - Separate App for Data Post-processing
    - Automatically bringing up the data in Tableau (on a laptop or on the same tablets used for scouting)
    - Separate App for Use during Alliance Selection
- Android Studio for full control, custom widgets and fragments
- Automatic Match Schedule Updates using the FMS API introduced in 2015. http://docs.frcevents2.apiary.io/#
- Separate App for Pit Scouting/Photography
- Ability to re-use data from one regional to the next
- Data accessibility by Drive Team for real-time strategy adjustments

Original DS Version



Sample Match Schedule file, "DS.txt".CSV file format (comma-separated value). Match number followed by

```
1,2073,971,2813,115,2035,1072
2,3021,2643,2473,649,8,3501
3,4171,1700,1351,114,1868,2144
4,4255,100,254,3256,256,2854
5,4186,4086,1156,766,253,840
6,597,852,751,2135,1538,670
7,3482,3022,1967,2367,3354,2489
8,1458,846,604,192,581,1280
9,4047,100,1351,675,4086,3021
10,751,840,3501,2035,4255,1868
…
```

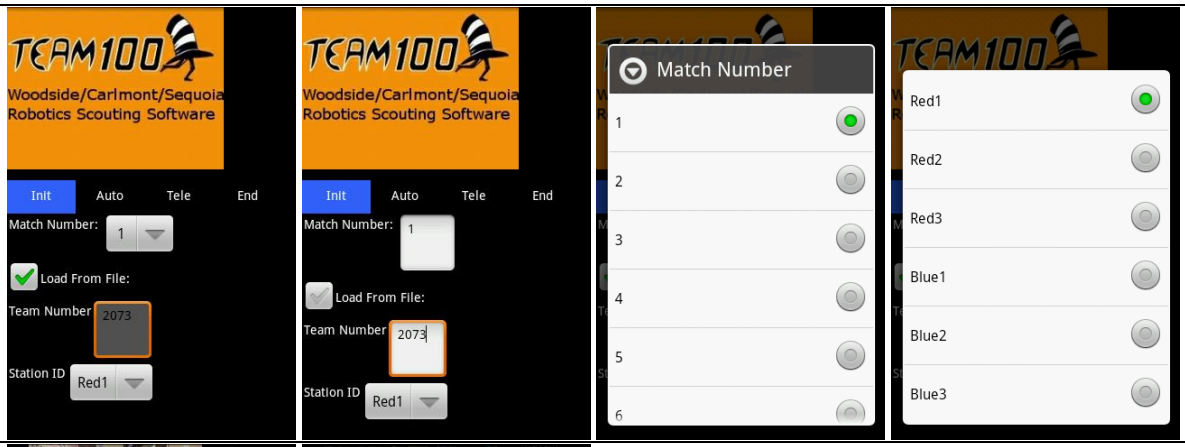
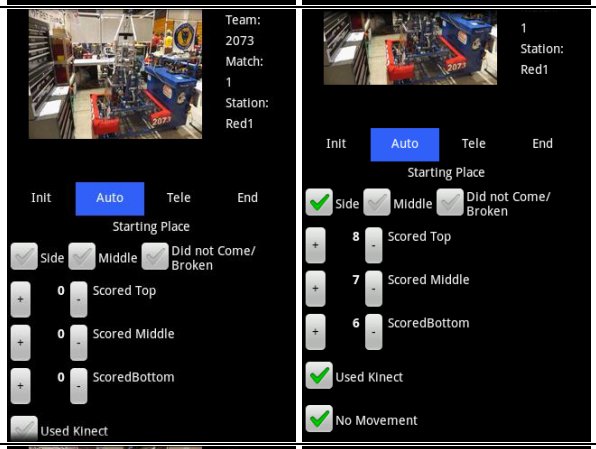
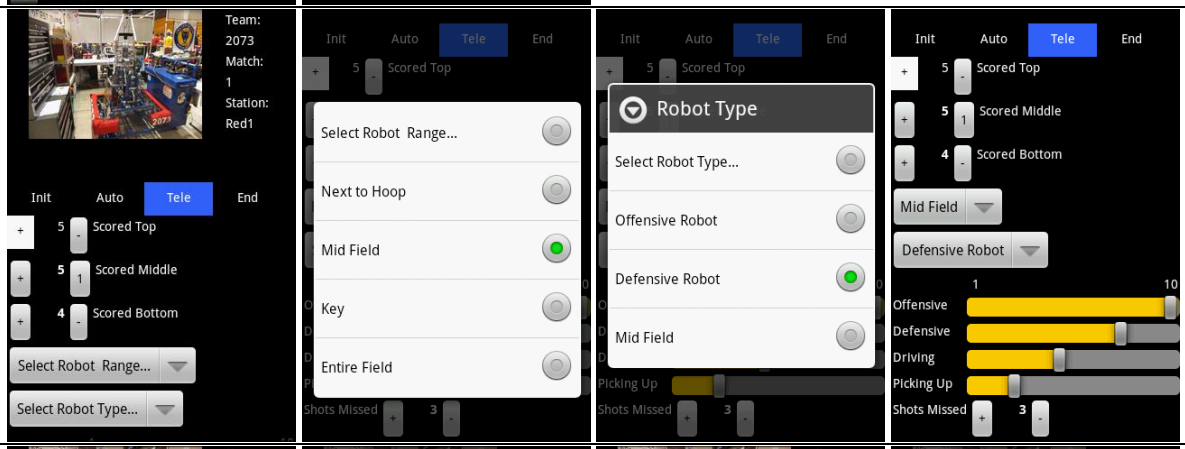
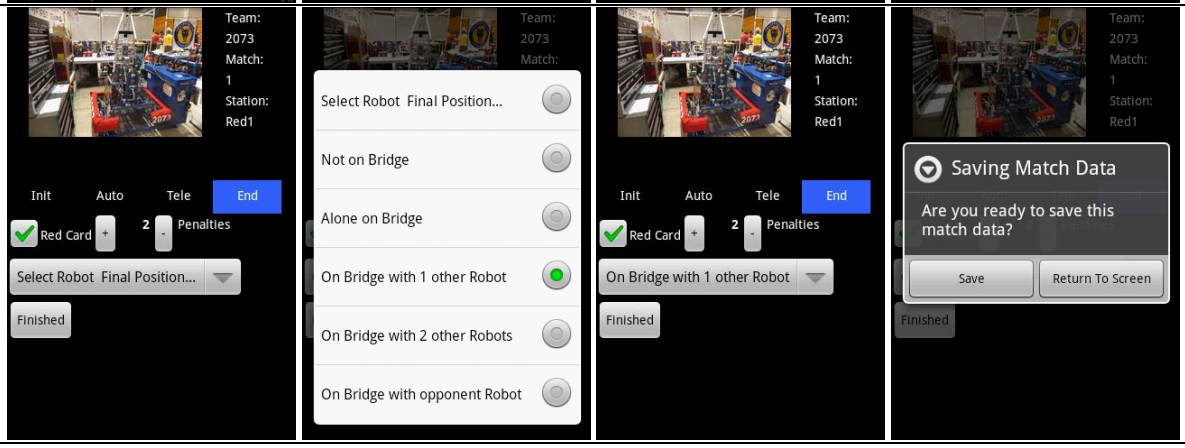
Contents of the Scouting folder on the /sdcard on the Android device. These files can be downloaded to the device using adb (Android Debug Bridge) or just dragging and dropping using Windows Explorer when the Android device is connected to the computer with a USB cable. The jpg files in the images folder are scaled to fit inside a 192 x 192 pixel square.

- Scouting
  - DS.txt
  - images
    - 8.jpg
    - 100.jpg
    - 114.jpg
    - etc.

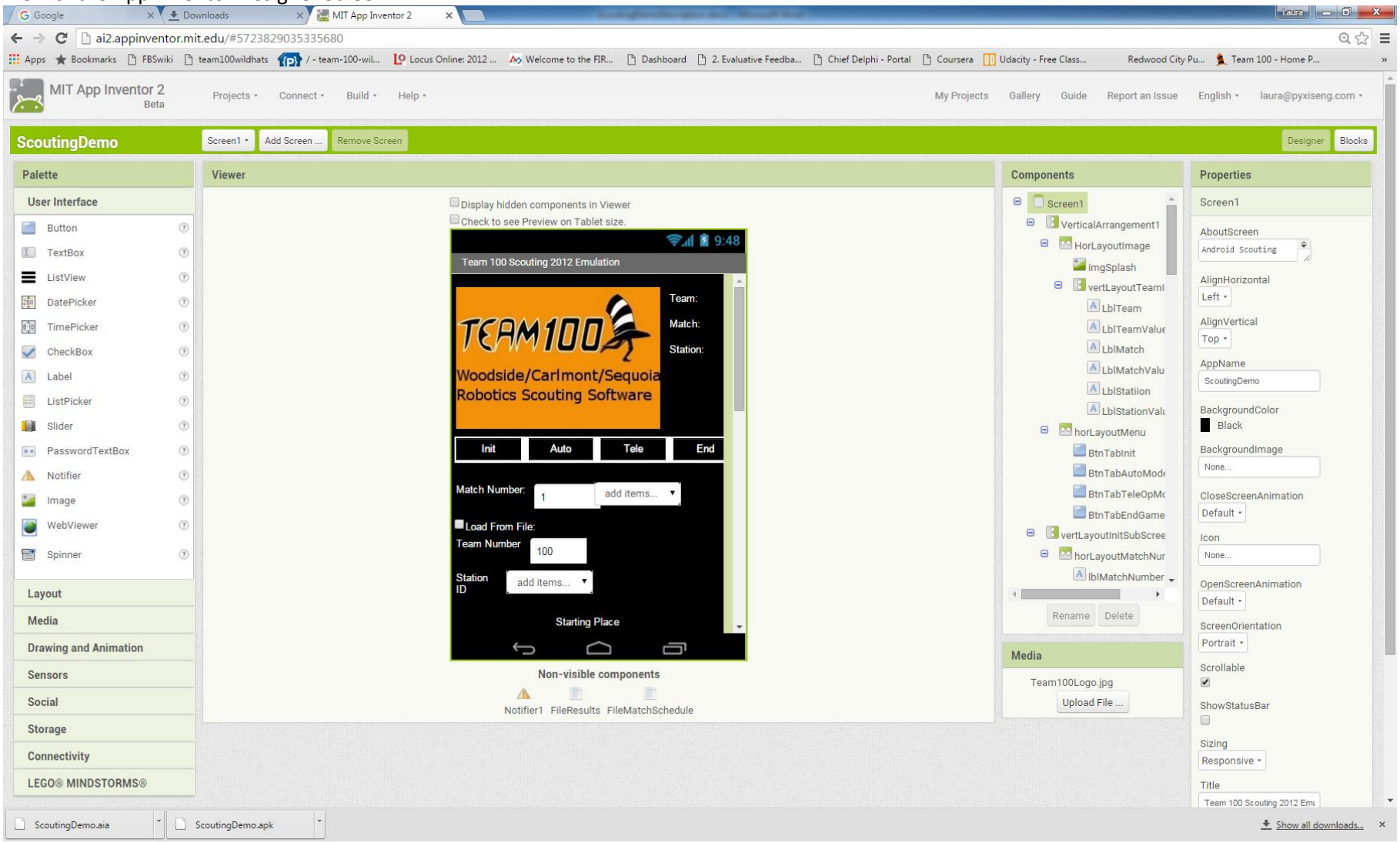
Sample Output file, DataOut.txt

```
(matchnum, 1), (teamnum, 115), (autodisabled, 0), (kinect, 1), (autoscorebot, 3),
(autoscoremid, 4), (autoscoretop, 3), (active, 0), (startposition, 2), (misses, 3),
(driverrating, 6), (manueverrating, 4), (defensiverating, 8), (offensiverating, 10),
(range, 2), (robottype, 3), (telescorebot, 7), (telescoremid, 6), (telescoretop, 5),
(penalty, 2), (bridge, 5), (redcard, 0),
(matchnum, 2), (teamnum, 649), (autodisabled, 0), (kinect, 0), (autoscorebot, 3),
(autoscoremid, 2), (autoscoretop, 1), (active, 0), (startposition, 2), (misses, 10),
(driverrating, 4), (manueverrating, 5), (defensiverating, 2), (offensiverating, 1),
(range, 5), (robottype, 4), (telescorebot, 6), (telescoremid, 5), (telescoretop, 4),
(penalty, 1), (bridge, 3), (redcard, 0),
…
```

Sample Android App screen shots:

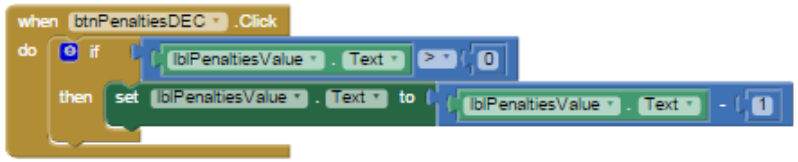
Init Screen	
Auto Screen	
TeleOp Screen	
End Screen	

View of the App Inventor Designer Screen

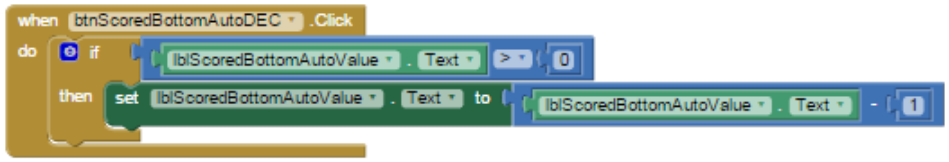


View of the App Inventor's blocks (10000 ft. view)

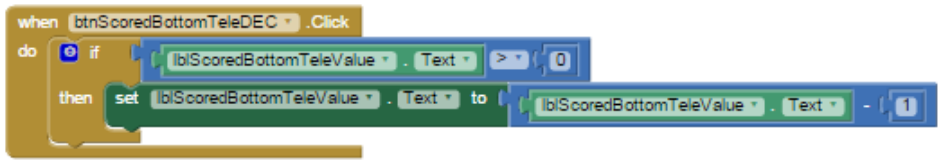
Close-ups of each of these blocks is given in the following pages.


when btnPenaltiesDEC .Click
do if lblPenaltiesValue . Text > 0
then set lblPenaltiesValue . Text to lblPenaltiesValue . Text - 1

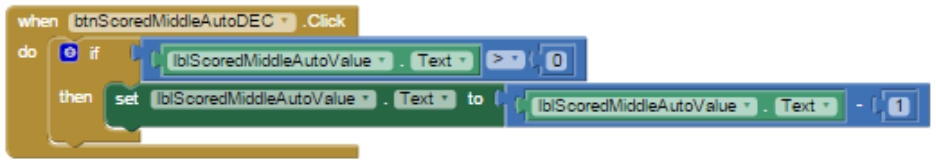

when btnPenaltiesINC .Click
do set lblPenaltiesValue . Text to lblPenaltiesValue . Text + 1

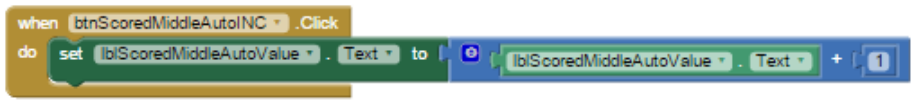

when btnScoredBottomAutoDEC .Click
do if lblScoredBottomAutoValue . Text > 0
then set lblScoredBottomAutoValue . Text to lblScoredBottomAutoValue . Text - 1

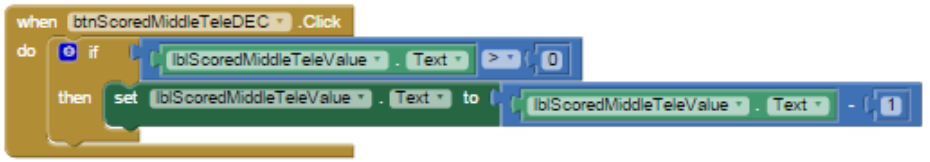

when btnScoredBottomAutoINC .Click
do set lblScoredBottomAutoValue . Text to lblScoredBottomAutoValue . Text + 1

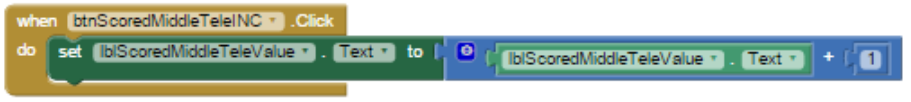

when btnScoredBottomTeleDEC .Click
do if lblScoredBottomTeleValue . Text > 0
then set lblScoredBottomTeleValue . Text to lblScoredBottomTeleValue . Text - 1

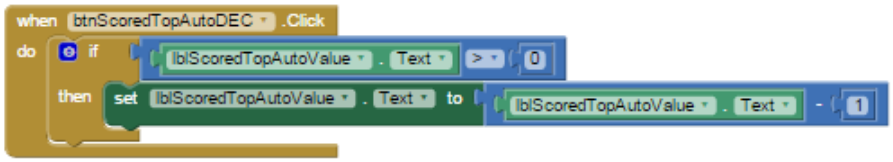

when btnScoredBottomTeleINC .Click
do set lblScoredBottomTeleValue . Text to lblScoredBottomTeleValue . Text + 1


when btnScoredMiddleAutoDEC .Click
do if lblScoredMiddleAutoValue . Text > 0
then set lblScoredMiddleAutoValue . Text to lblScoredMiddleAutoValue . Text - 1

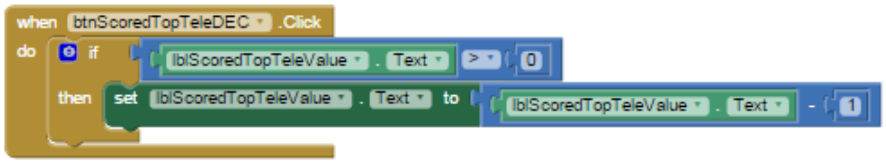

when btnScoredMiddleAutoINC .Click
do set lblScoredMiddleAutoValue . Text to lblScoredMiddleAutoValue . Text + 1


when btnScoredMiddleTeleDEC .Click
do if lblScoredMiddleTeleValue . Text > 0
then set lblScoredMiddleTeleValue . Text to lblScoredMiddleTeleValue . Text - 1

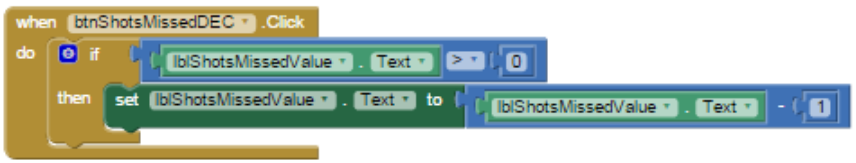

when btnScoredMiddleTeleINC .Click
do set lblScoredMiddleTeleValue . Text to lblScoredMiddleTeleValue . Text + 1


when btnScoredTopAutoDEC .Click
do if lblScoredTopAutoValue . Text > 0
then set lblScoredTopAutoValue . Text to lblScoredTopAutoValue . Text - 1


when btnScoredTopAutoINC .Click
do set lblScoredTopAutoValue . Text to lblScoredTopAutoValue . Text + 1

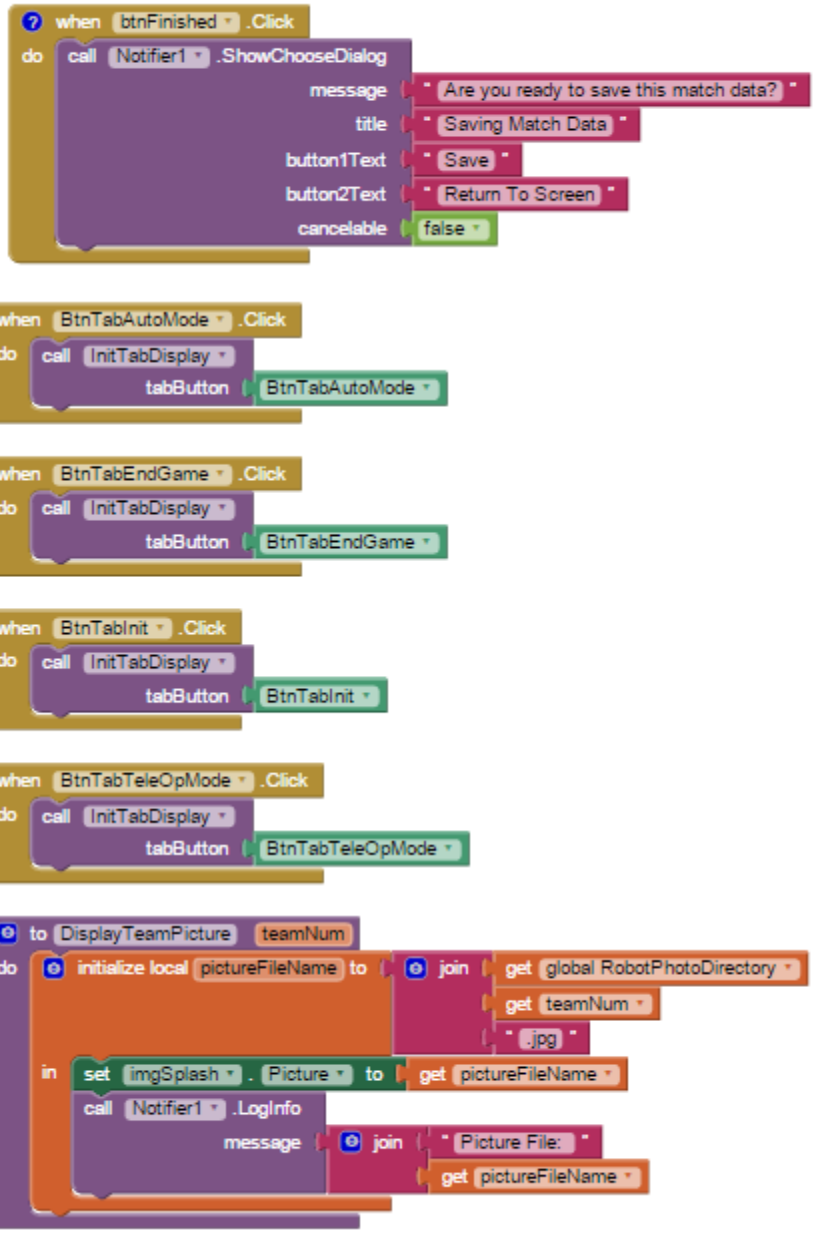

when btnScoredTopTeleDEC .Click
do if lblScoredTopTeleValue . Text > 0
then set lblScoredTopTeleValue . Text to lblScoredTopTeleValue . Text - 1


when btnScoredTopTeleINC .Click
do set lblScoredTopTeleValue . Text to lblScoredTopTeleValue . Text + 1

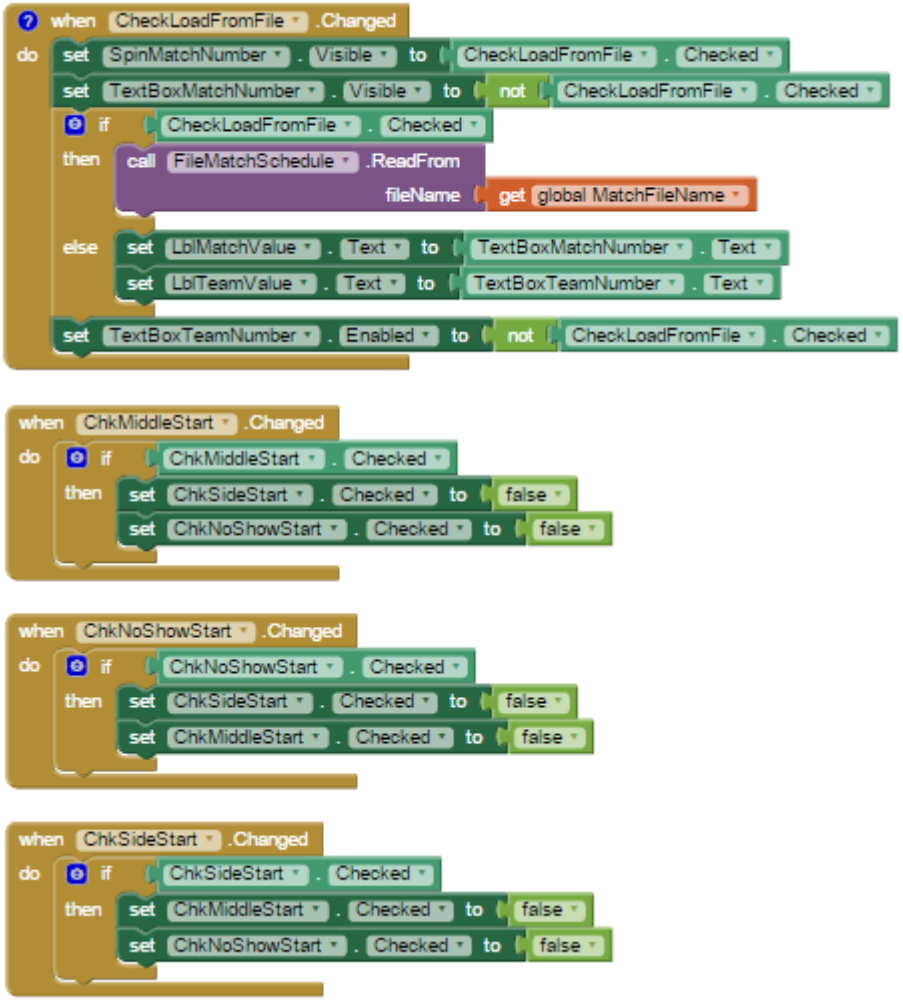

when btnShotsMissedDEC .Click
do if lblShotsMissedValue . Text > 0
then set lblShotsMissedValue . Text to lblShotsMissedValue . Text - 1


when btnShotsMissedINC .Click
do set lblShotsMissedValue . Text to lblShotsMissedValue . Text + 1

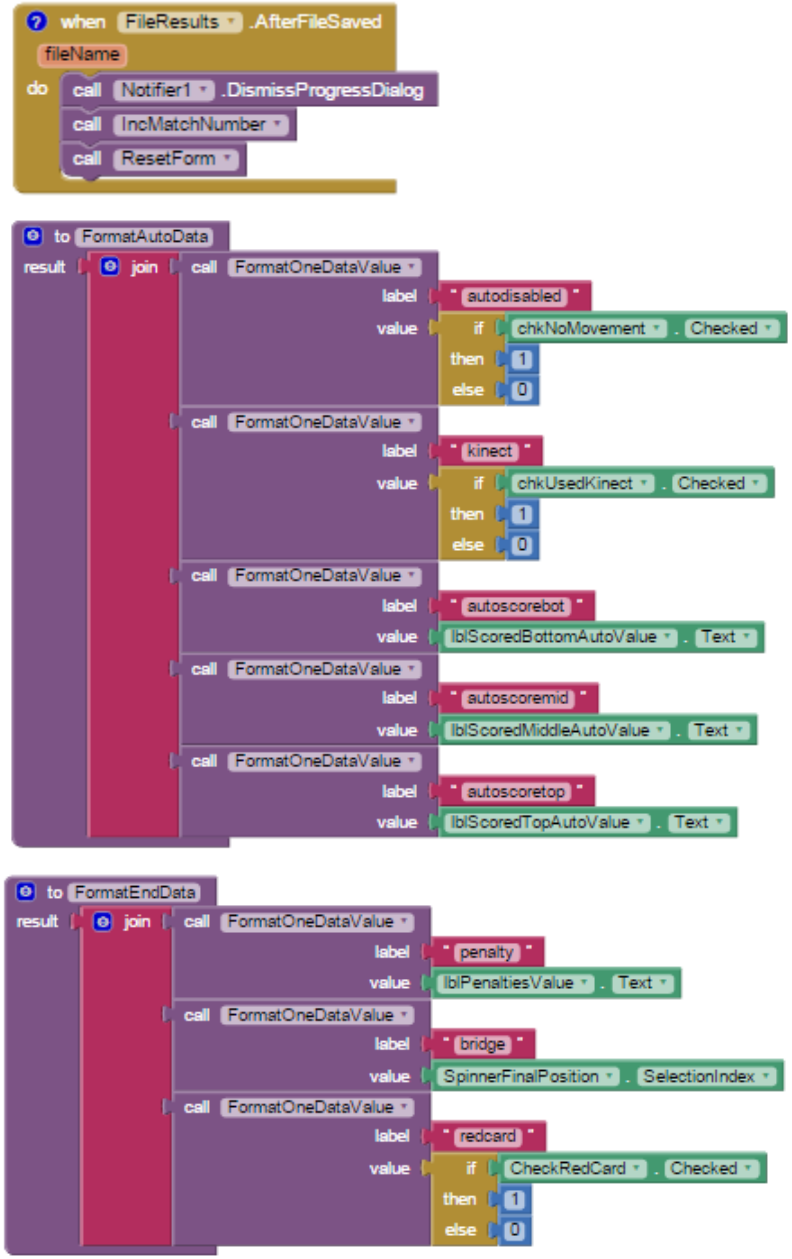
When the Finished button is pressed, it brings up a dialog asking if the user wants to save the match data to the output file.



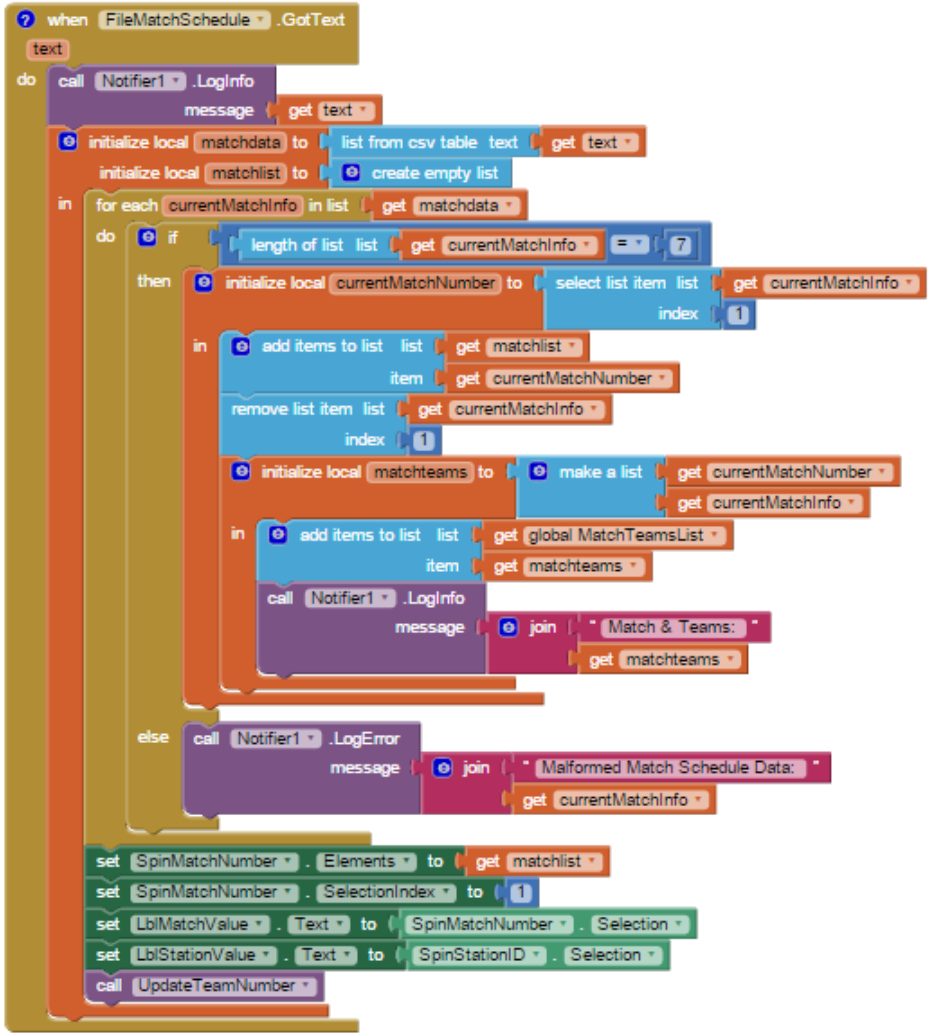
When the Load From File checkbox is selected, the match schedule file is read and the team number is automatically determined.

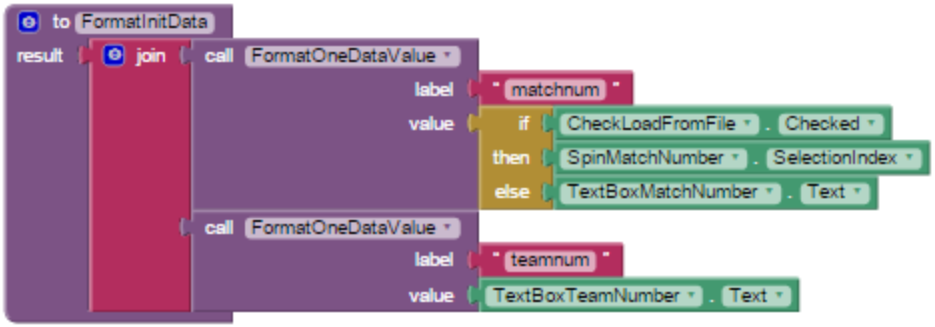


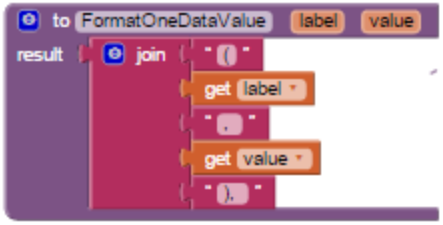
After the match results have been saved, the match number is automatically incremented and the Init screen displayed.

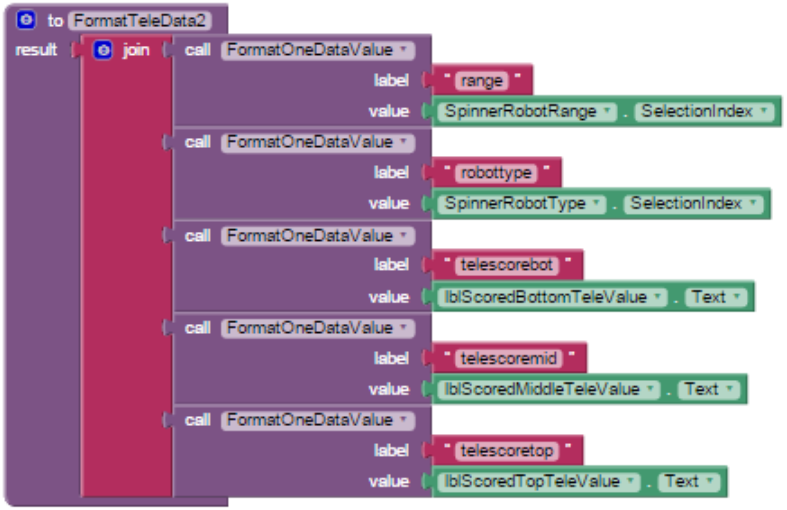


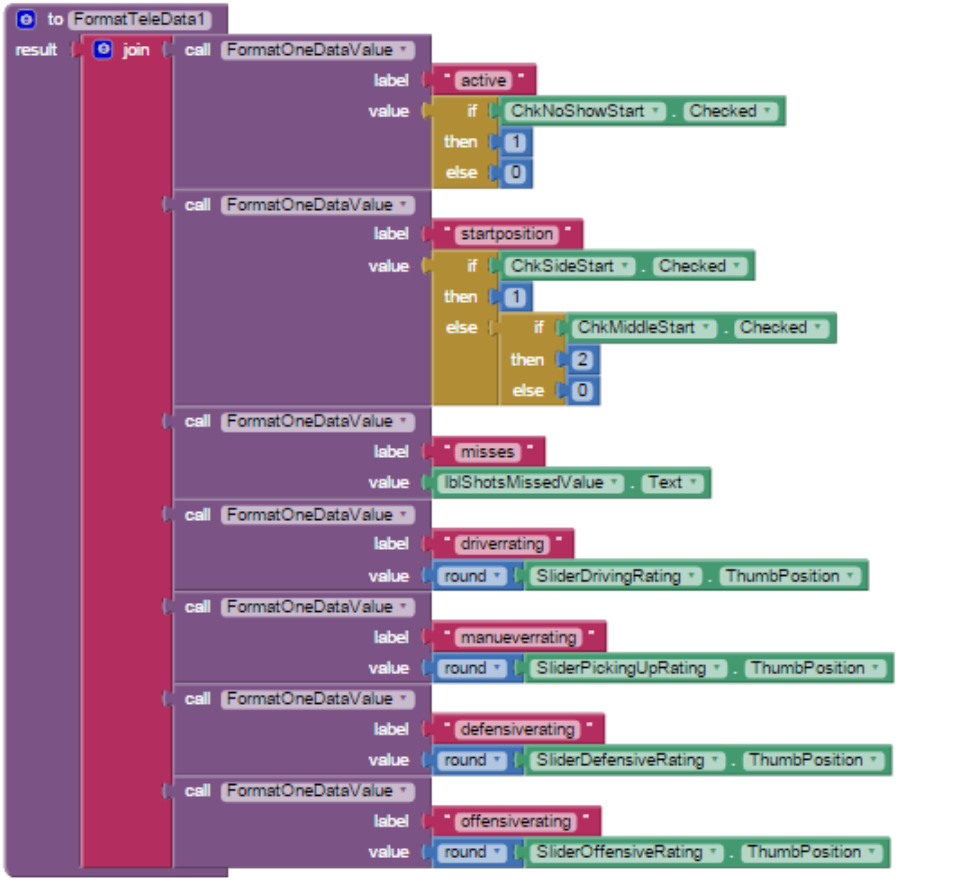
After the Match Data file has been read, it parses the data to initialize the database and the match number spinner selector.

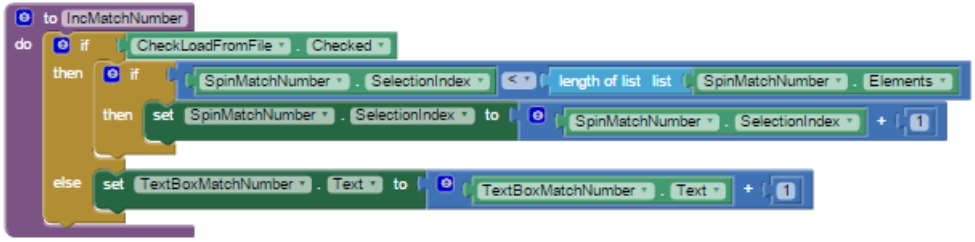


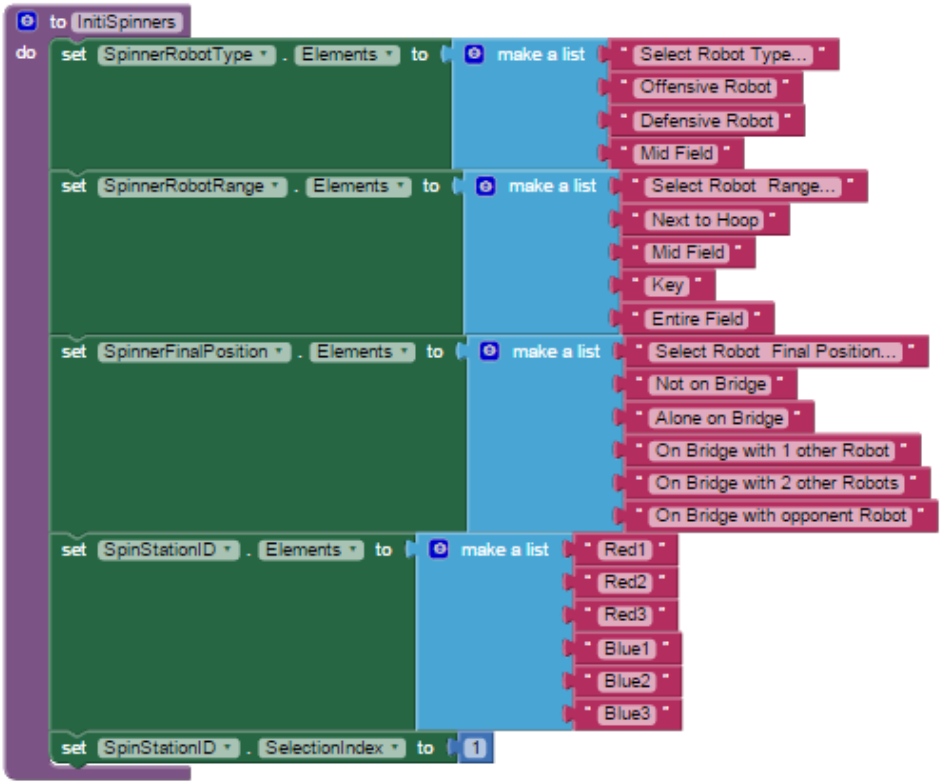

to FormatInitData
result
join
call FormatOneDataValue
label " matchnum "
value if CheckLoadFromFile . Checked
then SpinMatchNumber . SelectionIndex
else TextBoxMatchNumber . Text
call FormatOneDataValue
label " teamnum "
value TextBoxTeamNumber . Text

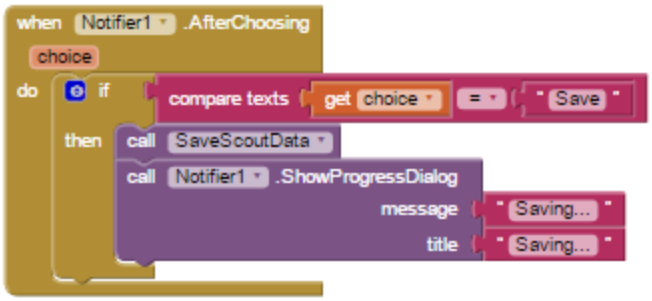

to FormatOneDataValue label value
result
join
" ("
get label
" , "
get value
"), "


to FormatTeleData2
result
join
call FormatOneDataValue
label " range "
value SpinnerRobotRange . SelectionIndex
call FormatOneDataValue
label " robottype "
value SpinnerRobotType . SelectionIndex
call FormatOneDataValue
label " telescorebot "
value lblScoredBottomTeleValue . Text
call FormatOneDataValue
label " telescoremid "
value lblScoredMiddleTeleValue . Text
call FormatOneDataValue
label " telescoretop "
value lblScoredTopTeleValue . Text

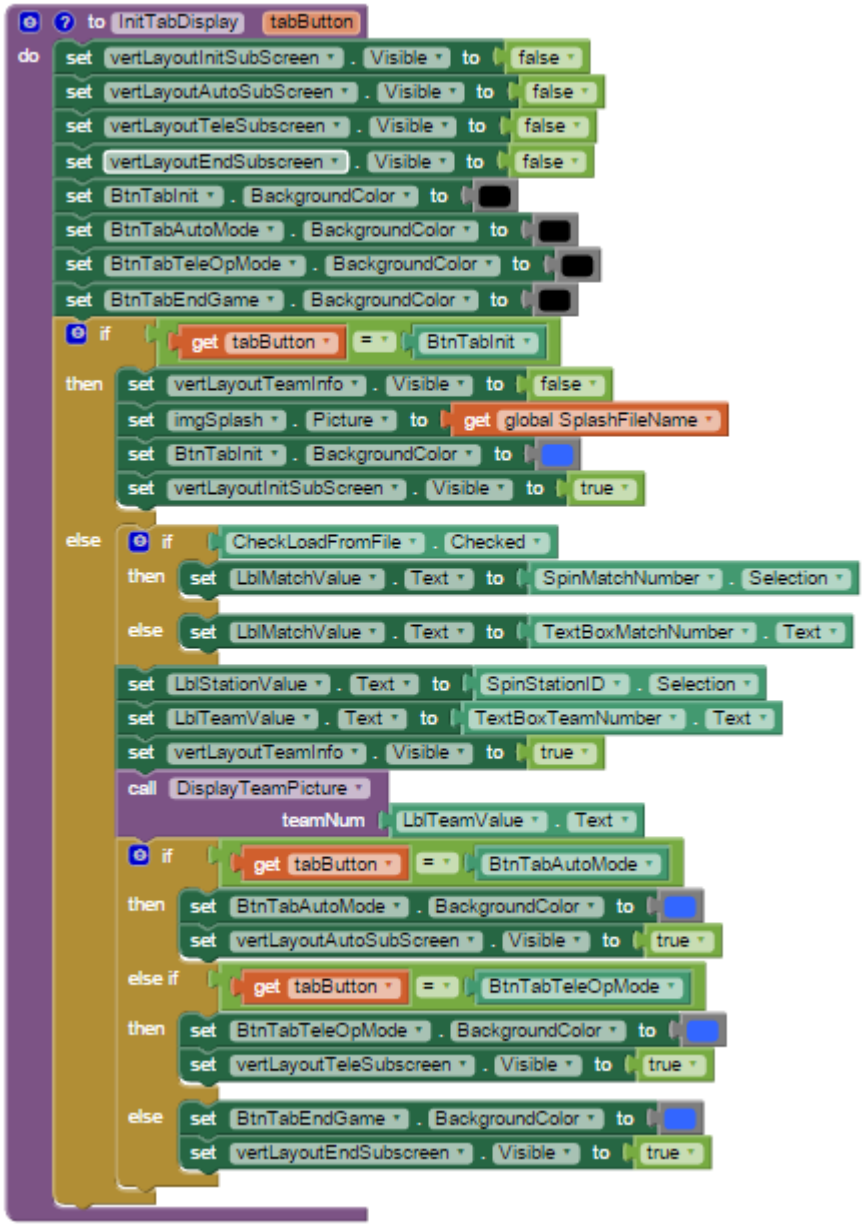

to FormatTeleData1
result
join
call FormatOneDataValue
label " active "
value if ChkNoShowStart . Checked
then 1
else 0
call FormatOneDataValue
label " startposition "
value if ChkSideStart . Checked
then 1
else if ChkMiddleStart . Checked
then 2
else 0
call FormatOneDataValue
label " misses "
value lblShotsMissedValue . Text
call FormatOneDataValue
label " driverrating "
value round SliderDrivingRating . ThumbPosition
call FormatOneDataValue
label " manueverrating "
value round SliderPickingUpRating . ThumbPosition
call FormatOneDataValue
label " defensiverating "
value round SliderDefensiveRating . ThumbPosition
call FormatOneDataValue
label " offensiverating "
value round SliderOffensiveRating . ThumbPosition


to IncMatchNumber
do if CheckLoadFromFile . Checked
then if SpinMatchNumber . SelectionIndex < length of list list SpinMatchNumber . Elements
then set SpinMatchNumber . SelectionIndex to SpinMatchNumber . SelectionIndex + 1
else set TextBoxMatchNumber . Text to TextBoxMatchNumber . Text + 1

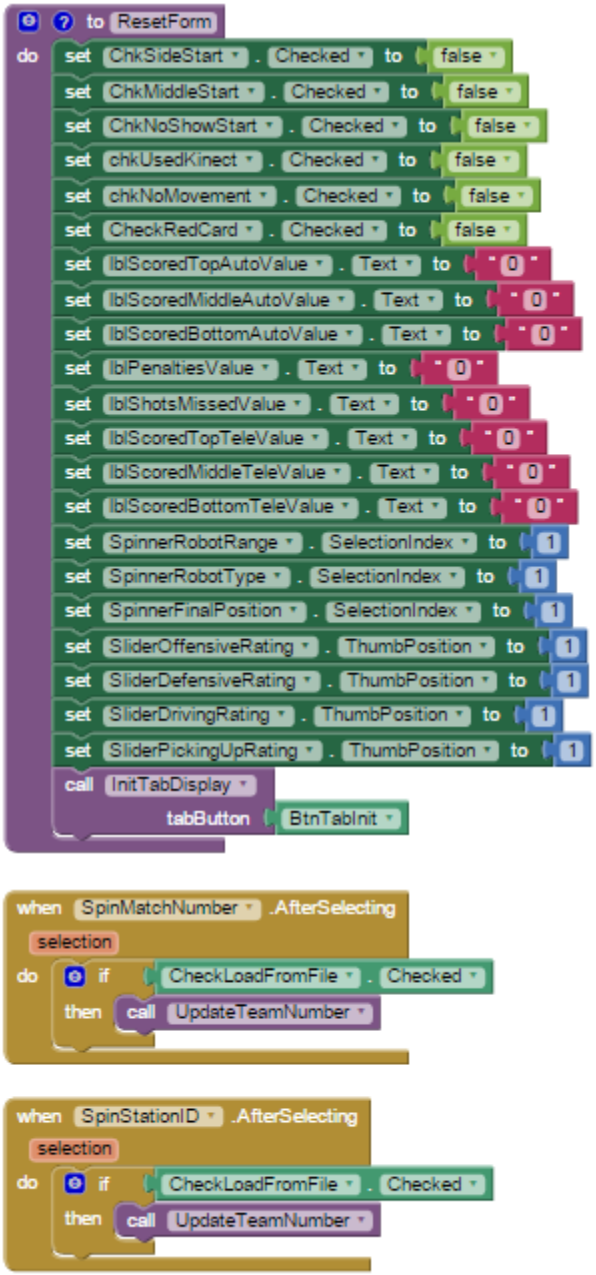




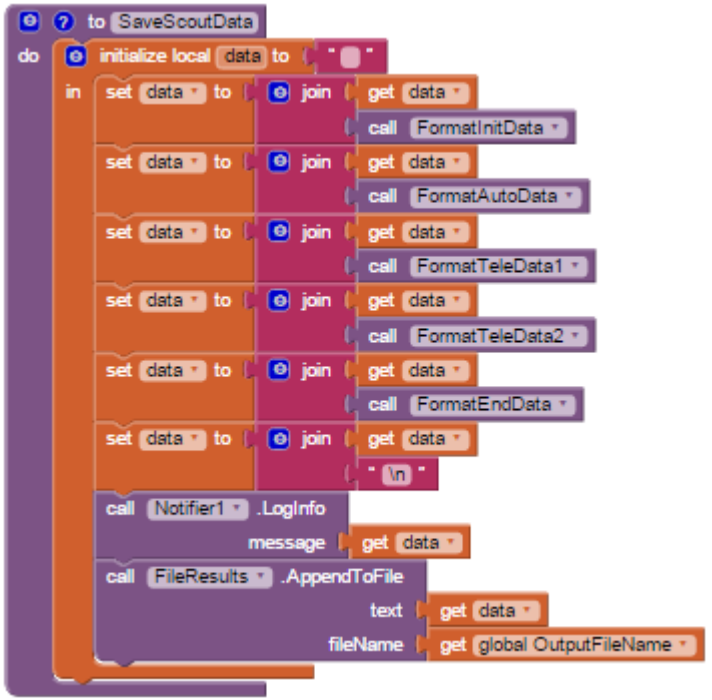
Initializes the screen based upon which tab was selected.

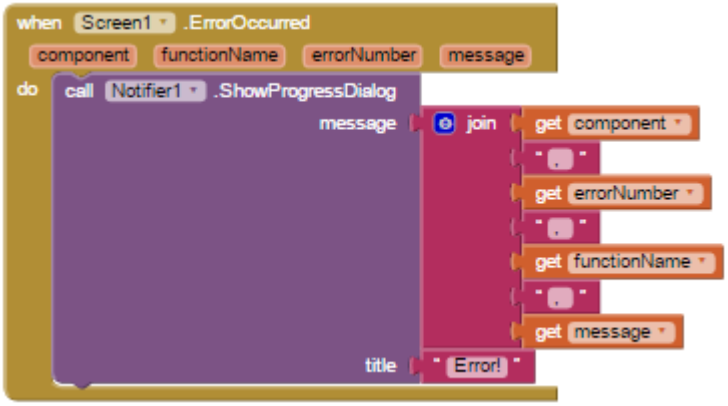


Resets the match data back to defaults - ready to start collecting data for the next match.

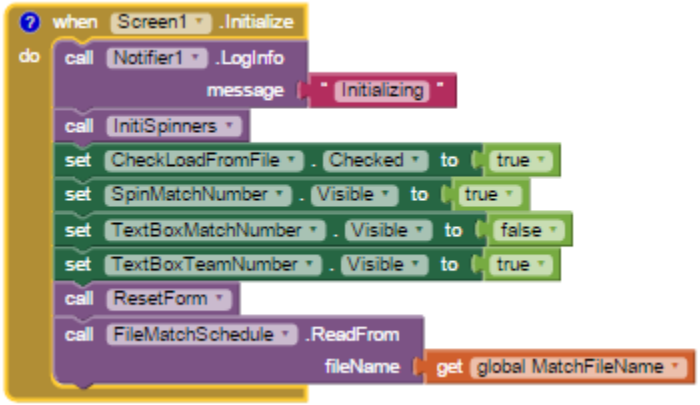


Appends the current match data to the output file.

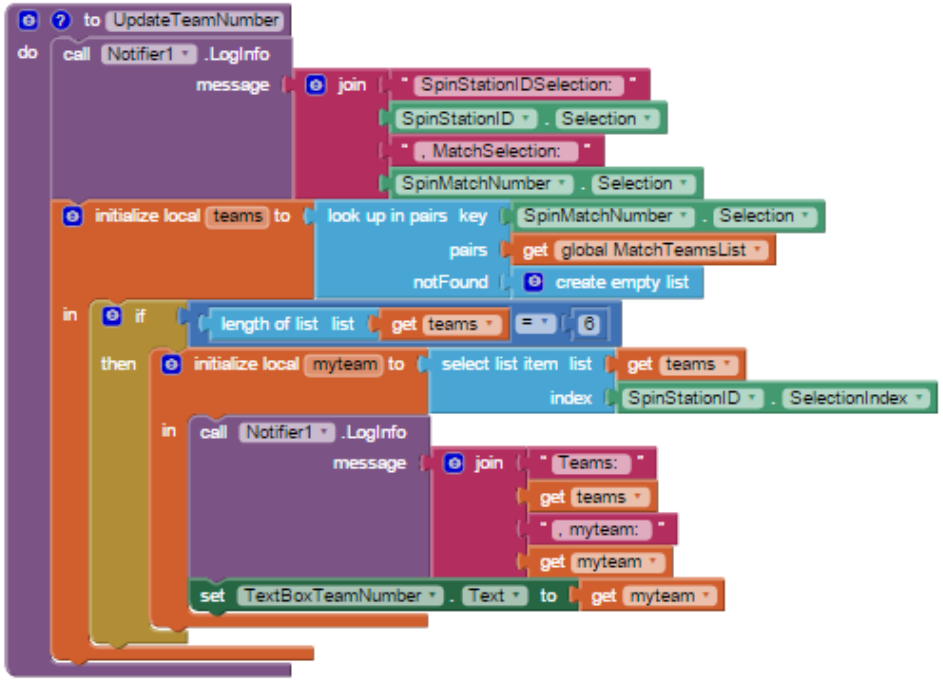




Initializes the app. Sets the elements for all the spinners, resets the form values, reads in the match schedule data base.



Determines the team number based upon the match data base, the currently selected match and the station ID.



Global Variables

Variable Name	Use
SpashFileName	Location of the SplashScreen image file
MatchFileName	The name of the file which has the list of matches and which 6 teams participate in each match.
OutputFileName	Stores the name of the file which will hold all of the scouting results.
MatchTeamsList	Holds the "data base" of the match and team numbers.
RobotPhotoDirectory	Directory location of the team photos. This directory should be in the "/sdcard" directory so that the files can be loaded via drag-and-drop from the PC.