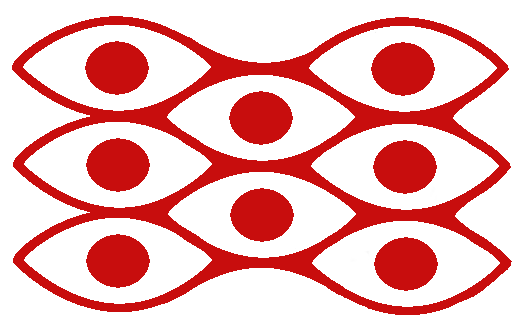
Department of Computer Science and Engineering  
The University of Texas at Arlington



**Detailed Design Specifications**

Team: Team Argus

Project: Lynx- PixelSense Secure Transfer System

Team Members:   
*Shamikul Amin*

*Brandon Deen*

*Brian Hasty*

*Keyur Patel*

*Seth Skocelas*

# Table of Contents

[Document Revision History 5](#_Toc413185658)

[Table of Figures 6](#_Toc413185659)

[List of Tables 7](#_Toc413185660)

[1. Introduction 8](#_Toc413185661)

[1.1 Purpose and Use 8](#_Toc413185662)

[1.2 Project Scope 8](#_Toc413185663)

[2. Architecture Overview 10](#_Toc413185664)

[2.1 PixelSense Super Layer 11](#_Toc413185665)

[2.2 Tablet/Lynx Super Layer 13](#_Toc413185666)

[2.3 Transfer Super Layer 14](#_Toc413185667)

[2.4 Module Decomposition 16](#_Toc413185668)

[2.5 Module Producer-Consumer Matrix 22](#_Toc413185669)

[3. System Hardware Description 23](#_Toc413185670)

[3.1 Arduino-Based Microcontroller 23](#_Toc413185671)

[3.2 IR LED 24](#_Toc413185672)

[3.3 Resistors - 1/4 Watt - 5% 25](#_Toc413185673)

[3.4 Phototransistor 26](#_Toc413185674)

[3.5 Micro-USB 2.0 On-The-Go (OTG) Adapter 27](#_Toc413185675)

[3.6 3-Feet USB 2.0 Cable 28](#_Toc413185676)

[3.7 MicroSD card breakout board+ 29](#_Toc413185677)

[3.8 MicroSD Memory Card 30](#_Toc413185678)

[3.9 Nexus 7 31](#_Toc413185679)

[3.10 Samsung SUR40 32](#_Toc413185680)

[3.11 Conceptual Hardware Diagram 33](#_Toc413185681)

[4. System Software Description 34](#_Toc413185682)

[4.1 PixelSense Windows Application 34](#_Toc413185683)

[4.2 Android Application 34](#_Toc413185684)

[4.3 Arduino Sketch 34](#_Toc413185685)

[5. PixelSense Super Layer 35](#_Toc413185686)

[5.1 Presentation Layer 35](#_Toc413185687)

[5.2 Data Processing Layer 42](#_Toc413185688)

[6. Tablet Super Layer 51](#_Toc413185689)

[6.1 Presentation Layer 51](#_Toc413185690)

[6.2 Data Processing Layer 60](#_Toc413185691)

[6.3 Data Storage Layer 70](#_Toc413185692)

[7. Transfer Super Layer 75](#_Toc413185693)

[7.1 SDK Layer 75](#_Toc413185694)

[7.2 PixelSense Transfer Layer 83](#_Toc413185695)

[7.3 Lynx Transfer Layer 87](#_Toc413185696)

[8. Quality Assurance 95](#_Toc413185697)

[8.1 Testing Modules 95](#_Toc413185698)

[8.1.2 Testing Data Emulator 95](#_Toc413185699)

[8.2 Unit Testing 95](#_Toc413185700)

[8.3 Component Testing 96](#_Toc413185701)

[8.4 Integration Testing 96](#_Toc413185702)

[8.5 Test Cases 96](#_Toc413185703)

[9. Requirements Mapping 97](#_Toc413185704)

[9.1 Requirements Traceability Matrix 97](#_Toc413185705)

[9.2 Requirement Traceability Analysis 98](#_Toc413185706)

[10. Acceptance Plan 99](#_Toc413185707)

[10.1 Packaging and Installation 99](#_Toc413185708)

[10.2 User Acceptance Testing 99](#_Toc413185709)

[10.3 Acceptance Criteria 99](#_Toc413185710)

[11. Appendix 101](#_Toc413185711)

[11.1 Java Libraries including JDBC 101](#_Toc413185712)

[11.2 PixelSense SDK 101](#_Toc413185713)

[11.3 Android SDK (Minimum API 17) 101](#_Toc413185714)

[11.4 Arduino GPIO 101](#_Toc413185715)

# Document Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Revision Number | Revision Date | Description | Rationale |
| 1.0 | 3/3/15 | DDS Final Submission | Post Huber/Presentation edits |
| .75 | 2/12/15 | DDS Draft | Initial Creation of the Document |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Table of Figures

[Figure 2‑1: Architecture Diagram 11](#_Toc413184913)

[Figure 2‑2: Module Decomposition Diagram 16](#_Toc413184914)

[Figure 2‑3: Producer-Consumer Matrix 22](#_Toc413184915)

[Figure 3‑1: Teensy 3.1 23](#_Toc413184916)

[Figure 3‑2: IR LEDs 24](#_Toc413184917)

[Figure 5‑1: Data Renderer Module 35](#_Toc413184918)

[Figure 5‑2: Input Validation Module 37](#_Toc413184919)

[Figure 5‑3: Message Formatter for Display Module 39](#_Toc413184920)

[Figure 5‑4: Message Formatter for Processing Module 41](#_Toc413184921)

[Figure 5‑5: Data Analyzer Module 43](#_Toc413184922)

[Figure 5‑6: Process Display Module 44](#_Toc413184923)

[Figure 5‑7: Generate Graphics Module 46](#_Toc413184924)

[Figure 5‑8: State Updater Module 47](#_Toc413184925)

[Figure 5‑9: Transfer Controller Module 49](#_Toc413184926)

[Figure 6‑1: Data Renderer Module 51](#_Toc413184927)

[Figure 6‑2: Input Validator Module 54](#_Toc413184928)

[Figure 6‑3: Message Formatter for Processing Module 56](#_Toc413184929)

[Figure 6‑4: Message Formatter for Display Module 58](#_Toc413184930)

[Figure 6‑5: Data Analyzer Module 60](#_Toc413184931)

[Figure 6‑6: Process Display Module 61](#_Toc413184932)

[Figure 6‑7: Generate Graphics Module 63](#_Toc413184933)

[Figure 6‑8: State Updater Module 65](#_Toc413184934)

[Figure 6‑9: Transfer Controller Module 66](#_Toc413184935)

[Figure 6‑10: Request Controller Module 68](#_Toc413184936)

[Figure 6‑11: Convert to Data Module 70](#_Toc413184937)

[Figure 6‑12: Convert to Query Module 71](#_Toc413184938)

[Figure 6‑13: Database Manager Module 73](#_Toc413184939)

[Figure 7‑1: Graphics Transfer Connector Module 75](#_Toc413184940)

[Figure 7‑2: Data Converter Module 77](#_Toc413184941)

[Figure 7‑3: USB Serial Connection Manager Module 79](#_Toc413184942)

[Figure 7‑4: Data Converter Module 81](#_Toc413184943)

[Figure 7‑5: Binary Converter Module 83](#_Toc413184944)

[Figure 7‑6: Pixel Group Module 85](#_Toc413184945)

[Figure 7‑7: Infrared Sensor Module 86](#_Toc413184946)

[Figure 7‑8: Phototransistor Module 87](#_Toc413184947)

[Figure 7‑9: LED Module 91](#_Toc413184948)

[Figure 7‑10: Binary Converter Module 93](#_Toc413184949)

# List of Tables

[Table 3‑2: LED Specifications 24](#_Toc413184950)

[Table 5‑1: Data Renderer Interfaces 36](#_Toc413184951)

[Table 5‑2: Input Validation Interfaces 37](#_Toc413184952)

[Table 5‑3: Message Formatter for Display Interfaces 39](#_Toc413184953)

[Table 5‑4: Message Formatter for Processing Display Interfaces 41](#_Toc413184954)

[Table 5‑5: Data Analyzer Interfaces 43](#_Toc413184955)

[Table 5‑6: Process Display Interfaces 45](#_Toc413184956)

[Table 5‑7: Generate Graphics Interfaces 46](#_Toc413184957)

[Table 5‑8: State Updater Interfaces 47](#_Toc413184958)

[Table 5‑9: Transfer Controller Interfaces 49](#_Toc413184959)

[Table 6‑1: Data Renderer Interfaces 52](#_Toc413184960)

[Table 6‑2: Input Validation Interfaces 54](#_Toc413184961)

[Table 6‑3: Message Formatter for Processing 56](#_Toc413184962)

[Table 6‑4: Message Formatter for Display Interfaces 58](#_Toc413184963)

[Table 6‑5: Data Analyzer Interfaces 61](#_Toc413184964)

[Table 6‑6: Process Display Interfaces 62](#_Toc413184965)

[Table 6‑7: Generate Graphics Interfaces 63](#_Toc413184966)

[Table 6‑8: State Updater Interfaces 65](#_Toc413184967)

[Table 6‑9: Transfer Controller Interfaces 67](#_Toc413184968)

[Table 6‑10: Request Controller Interfaces 68](#_Toc413184969)

[Table 6‑11: Convert to Data Interfaces 70](#_Toc413184970)

[Table 6‑12: Convert to Query Interfaces 72](#_Toc413184971)

[Table 6‑13: Database Manager Interfaces 73](#_Toc413184972)

[Table 7‑1: Graphics Transfer Connector Interfaces 76](#_Toc413184973)

[Table 7‑2: Data Converter Interfaces 77](#_Toc413184974)

[Table 7‑3: USB Serial Connection Manager Interfaces 79](#_Toc413184975)

[Table 7‑4: Data Converter Interfaces 81](#_Toc413184976)

[Table 7‑5: Binary Converter Interfaces 84](#_Toc413184977)

[Table 7‑6: Pixel Group Interfaces 85](#_Toc413184978)

[Table 7‑7: Infrared Sensor Interfaces 86](#_Toc413184979)

[Table 7‑8: Phototransistor Interfaces 88](#_Toc413184980)

[Table 7‑9: LED Interfaces 91](#_Toc413184981)

[Table 7‑10: Binary Converter Interfaces 93](#_Toc413184982)

[Table 8‑1: Test Cases 96](#_Toc413184983)

# Introduction

The purpose of this document is to detail the architecture design for Lynx, the PixelSense Secure Transfer System. This document will describe in detail the meta-architecture, layers and subsystems of all the components of the system, showing the guiding principles we will use to drive this project. We will also detail any OS dependencies as well as the testing considerations we have made while describing this architecture.

Below is a general description of the system we are designing, discussing the purpose, use and scope of our project. It also contains the key requirements we needed to take into account while developing this architecture.

## 1.1 Purpose and Use

The PixelSense Secure Transfer System is a tablet case with a series of sensors allowing secure transmission between the PixelSense table and the device (the Lynx) connected to the tablet. The benefit of this transfer method is that communication cannot be intercepted due to physical limitation, thus making the connection secure. When the Lynx is placed on the table with compatible software loaded, the table will be able to recognize the Lynx, and be able transfer data when needed. Included with the Lynx will be an SDK that will allow users to develop their own applications for both the tablet and the PixelSense table in order to use it however they see fit. With the demonstration application we’re developing for the table, a casino game, the Lynx will provide a way to securely store casino chips, and also be used as a secondary screen when playing the game. For example, if you were playing poker, you can display the cards on the tablet where no other player can see them as opposed to showing them on the table.

Since we will be developing an SDK for the Lynx, it can be used for many other purposes in tandem with the PixelSense table. Below are some ideas that could be implemented using the SDK we create:

* A health information tracker that stores information on the device, and can only be displayed when the device is placed on the PixelSense table. A doctor can then add or remove information using the table as he/she sees fit.
* An enhanced chess game where the transfer device serves as a chess piece that can store information regarding player habits.
* A degree plan tracker that can store advising information about a student, and when the device is placed on the table, the student’s degree plan and academic information would be displayed for the advisor, which the advisor could edit as he/she sees fit.

## Project Scope

For this project, we’ll be delivering the Lynx, a device that connects to an Android 4.0 tablet in order to transfer small amounts of data optically between the tablet and the PixelSense table it is placed on, provided that compatible software is loaded on both devices. Included with the Lynx will be an SDK that will allow the users to develop their own applications for both the tablet and the PixelSense table in order to use it however they see fit.

For demonstration purposes, our group will be developing a casino game on the PixelSense table, and our secure transfer system attached to a tablet will serve as a poker chip counter, securely storing your chips so you can move from table to table in order to play different casino games.

# Architecture Overview

The following diagram shows the architecture within the entire system which shows each layer, subsystem and the data flow between each of the components. The diagram shows the flow between each subsystem and layer, this in turn depicts how each subsystem interacts with each other in order to function correctly. By creating this diagram it allows a better understanding of the functions in each subsystem and how it propagates all the way up to a higher level, this in turn creates a sound and understandable system.

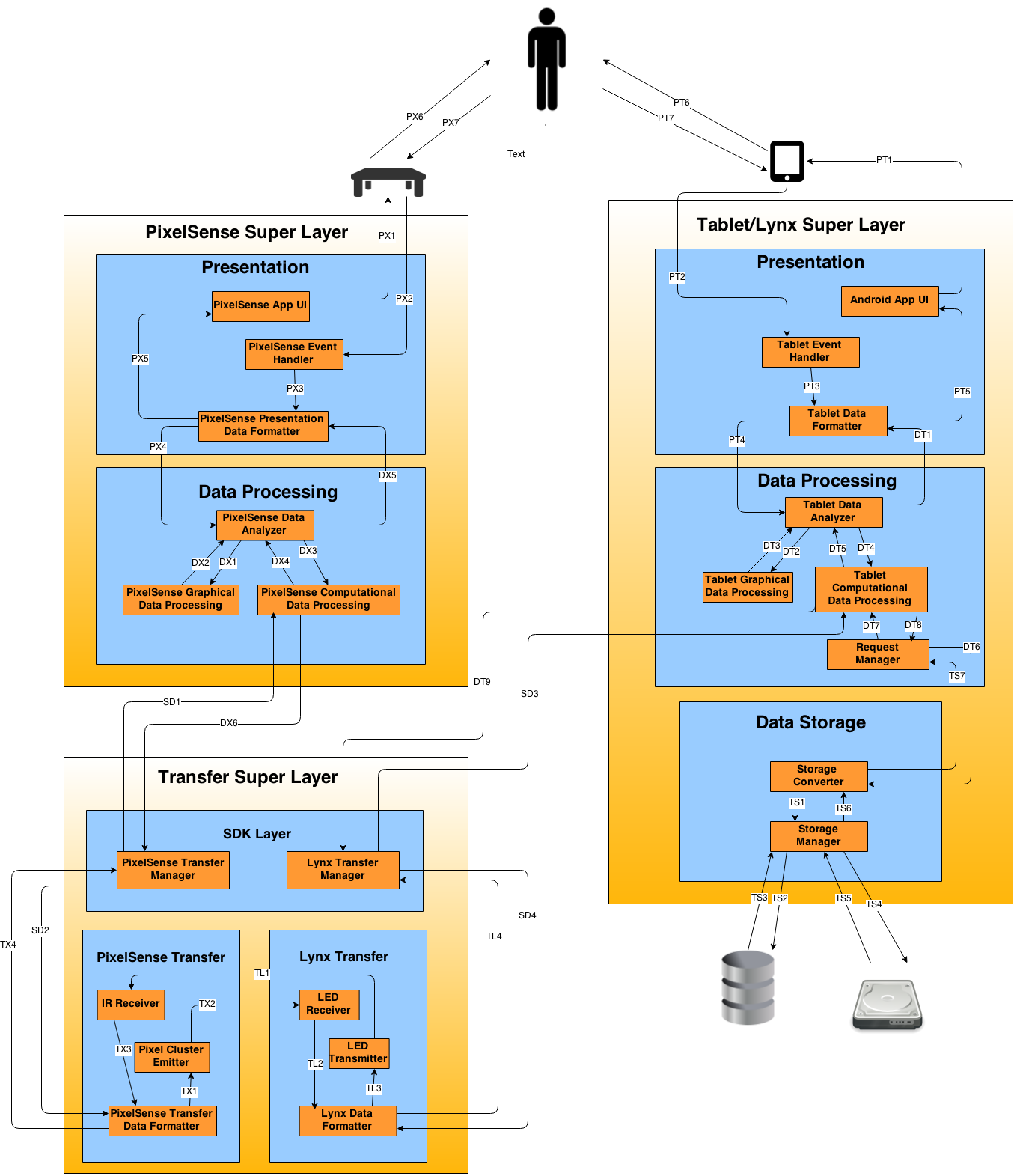


Figure 2‑1: Architecture Diagram

## 2.1 PixelSense Super Layer

The purpose of this super Layer is to serve the applications on the PixelSense Table. The main function is to get the data from the user on the PixelSense table and process it to forward it to the transfer layer. This super layer also serves the purpose of displaying the data back to the user after it is being transferred back from a different super layer. Having a super layer is useful showing the different parts of both systems as well as showing them as one system as a whole.

### 2.1.1 Presentation Layer

The purpose of the PixelSense presentation layer is to present some form of information to the user. It also accepts input from the user which would trigger the chain of events that will correspond to a scenario which will fulfill the needs of the user. This level will serve as the main source of input from the user to the system as well as presentation to the user from the system. Information sent to this layer is initially formatted and then transferred for processing. Similarly, information received by this device is formatted for presentation purposes.

### 2.1.1.1 PixelSense App UI

PixelSense App UI subsystem will display the graphical as well as computational content to the PixelSense table. This is the only subsystem which handles the output of the data to the user PixelSense table. The responsibility of this subsystem is to display the graphical as well as computational content to the user via PixelSense Table.

### 2.1.1.2 PixelSense Event Handler

PixelSense Event Handler subsystem will determine the type of data to be parsed into the processing layer. The responsibility of this subsystem is to detect the type of action that has occurred through the input controller. It will trigger the response to send to Data Formatter.

### 2.1.1.3 PixelSense Presentation Data Formatter

PixelSense Data Formatter subsystem will accept the input from the Event Handler subsystem. The formatter will then pre-process the data and translate it into a form that is acceptable for Data process Layer. The responsibility of this subsystem is to accept the input in various forms and translate it into acceptable form. The formatter should structure the data in such a way that it is easy to process. It also accepts the analyzed data from the system and processes it for it to be displayed to the user.

### 2.1.2 Data Processing Layer

The purpose of this subsystem is to accept the data from the Presentation Layer and process it. It also serves the purpose of sending data back to the presentation layer after processing. This Layer contains a data formatter which analyzes the incoming or outgoing data and formats according to the need of receiving layer. After analysis, if the data is non-graphical, it is being processed and sent to the PixelSense Transfer Manager, which forwards the data to the Transfer Layer.

### 2.1.2.1 PixelSense Data Analyzer

PixelSense Data Analyzer subsystem interacts with Data Formatter subsystem of Presentation layer. It basically analyzes the formatted data and identifies if it is graphical data or non-graphical data. After separation of data, the graphical data will be sent to the graphical data processor and non-graphical data will be sent to the computational data formatter. It also sends back the analyzed data to the data formatter in the presentation layer.

### 2.1.2.2 PixelSense Graphical Data Processing

PixelSense Graphical Data Processing subsystem interacts with Data Analyzer subsystem of Data Processing layer. This subsystem is responsible for receiving data from the PixelSense Data Analyzer subsystem. It then processes the data needed for any graphical operations and then sends the processed data back to the PixelSense Data Analyzer subsystem.

### 2.1.2.3 PixelSense Computational Data Processing

This subsystem will receive data from the PixelSense Data Analyzer subsystem, process that information, and then forwards the processed information to the PixelSense Transfer Controller subsystem in SDK layer as relevant computational (non-graphical) data. This subsystem also processes the data coming from the Lynx device and sends the processed data back to the PixelSense Data Analyzer subsystem depending for the display purposes.

## 2.2 Tablet/Lynx Super Layer

The purpose of this super layer is to enclose and distinguish between the three internal layers. The main functionality is to show the dataflow between the three internal layers, and between the Lynx and the tablet.

### 2.2.1 Presentation Layer

The purpose of this layer is to present the data on the Lynx to the user, and handle the user’s inputs. The layer will display the data on the Lynx, a log of recent transactions, and buttons the user can interact with via the tablet’s touch screen. Each interaction from the user is handled, formatted and transferred to the Data Processing Layer.

#### 2.2.1.1 Android App UI

This subsystem provides a user interface for the Tablet/Lynx system, allowing the user to both visualize what data is currently on the Lynx but also allowing the user to interact with the PixelSense application using the tablet.

#### 2.2.1.2 Tablet Event Handler

This subsystem takes tablet inputs from the user and passes them on so that the system can update the UI as well as handle any application processing needed.

#### 2.2.1.3 Tablet Data Formatter

This subsystem formats the data received from the Data Processing Layer and pushes it up to the Android App UI, as well as formatting data from the Tablet Event Handler subsystem pushing it to the Data Processing Layer to be processed.

### 2.2.2 Data Processing Layer

The purpose of this layer is to process the data from the Presentation layer, the Data Storage Layer, and the Transfer Super Layer. This layer will accept an input of data from one of the previously stated layers, process that data, and then transfer it to the receiving layer of the transaction.

#### 2.2.2.1 Tablet Data Analyzer

This subsystem analyzes data to be processed or data that has just been processed. If data needs to be processed the subsystem will determine whether it is graphical or computational (general) processing and send the data to those respective subsystems. If the data has already been processed the subsystem will push the processed data to the Presentation Layer.

#### 2.2.2.2 Tablet Graphical Data Processing

This subsystem processes all graphical data, such as generating images for cards, and any other visual data.

#### 2.2.2.3 Tablet Computational Data Processing

This subsystem will process all non-graphical data, make requests to store or retrieve data in the database, and send data to the Transfer Super Layer.

#### 2.2.2.4 Request Manager

This subsystem handles requests for data stored in the database, sending requests to the Data Storage Layer.

### 2.2.3 Data Storage Layer

The purpose of this layer is to manage and store transaction logs. This layer will receive a request from the Data Processing layer, then send that request on the Storage Manager Layer which will return the relevant information to the Storage Conversion Layer, which will convert the data and send it to the Data Processing Layer.

#### 2.2.3.1 Storage Converter

This subsystem converts requests into queries which will be sent to the Storage Manager subsystem, and it converts the results of executed queries into relevant data to be sent back to the Request Manager Subsystem.

#### 2.2.3.2 Storage Manager

This subsystem connects to the database, executes the queries received from the Storage Converter subsystem and returns the results of the query to the Storage Converter subsystem.

## 2.3 Transfer Super Layer

The purpose of this super layer is to enclose and distinguish between the two internal layers. The main function is to show the transfer between the PixelSense table and the tablet/lynx device as well as the dataflow between the two internal layers. Having a super layer is useful showing the different parts of both systems as well as showing them as one system as a whole.

### 2.3.1 PixelSense Transfer Layer

The purpose of this layer is to transfer (send/receive) strictly through the PixelSense side. Here the layer will receive data from the PixelSense data processing layer, format the data and emit it through some pixel cluster configuration. Going the other way, the table will detect certain light sequences through its IR receiver, format it to binary data and then send it off to the PixelSense data processing layer for interpretation.

#### 2.3.1.1 IR Receiver

The IR receiver is designed to detect light or any object that reflects IR beams. Basically what the PixelSense has is an array of IR sensors underneath the LCD panel. From here the IR sensors will listen and detect when our object is placed or when data is being transferred via LEDs. The subsystem then takes that raw data and sends it off to the data formatter subsystem for message formatting and converting.

#### 2.3.1.2 Pixel Cluster Emitter

The Pixel Cluster Emitter is designed to flash a group of pixels sized appropriately for the phototransistors to detect the flashing pixels. How this will work is that once it receives data it will flash a group of pixels in a binary sequence so that the phototransistors will be able to detect it.

#### 2.3.1.3 PixelSense Transfer Data Formatter

The PixelSense Data Formatter is designed to take the incoming raw binary data, convert it and send it off to the data converter for further formatting. This process happens both ways so the formatter will receive data from the converter, convert it to binary and send it off to the emitter for transferring.

### 2.3.2 Lynx Transfer Layer

The purpose of this layer is to transfer (send/receive) strictly through the Lynx side. Here the layer will be receiving data from the tablet’s processing layer, it will format it and then emit it through a certain light sequence via LEDs. Going the other way, the Lynx will detect light sequences through LED light detection. It will then format it to binary data and transfer it to the tablet’s data processing layer.

#### 2.3.2.1 LED Receiver

The LED receiver will detect light through phototransistors. Its essential job is to listen for any light changes so that it can capture it to relay over to the next subsystem. Its sole purpose is just to detect the changes in light.

#### 2.3.2.2 LED Transmitter

The LED Transmitter is designed to flash a group of LEDs for the IR sensors on the PixelSense to detect the flashing LEDs. How this will work is that once it receives data it will flash the group of LEDs in a binary sequence so that the IR sensors will be able to detect it.

#### 2.3.2.3 Lynx Data Formatter

The Lynx Formatter is designed to take the incoming raw binary data, convert it and send it off to the data converter for further formatting. This process happens both ways so the formatter will receive data from the converter, convert it to binary and send it off to the emitter for transferring.

### 2.3.3 SDK Layer

The purpose of this layer is to convert the computational data from both the tablet side and PixelSense side to data to a protocol where is can be sent over optics as well as converting the data received to something the PixelSense and tablet will both understand

#### 2.3.3.1 PixelSense Transfer Manager

The purpose of the PixelSense Transfer Manager is to take the data it received and transfer it to and from the PixelSense table via the built in graphics controller. The subsystem will do this by listening for data and then transfer it over the graphics bus to and from the processing layer as well as to and from the transfer layer’s subsystems.

#### 2.3.3.2 Lynx Transfer Manager

The purpose of the Lynx Transfer Manager is to take the data it received and transfer it to and from the Android tablet via the USB serial connection. The subsystem will do this by listening for data and then transfer it over the USB serial connection to and from the processing layer as well as to and from the transfer layer’s subsystems.

## 2.4 Module Decomposition

This subsection introduces the architecture module breakdown and provides a high level description of each module. A more detailed explanation of each module will be given in later sections of the document. The figure below illustrates the detailed decomposition of the system architecture.

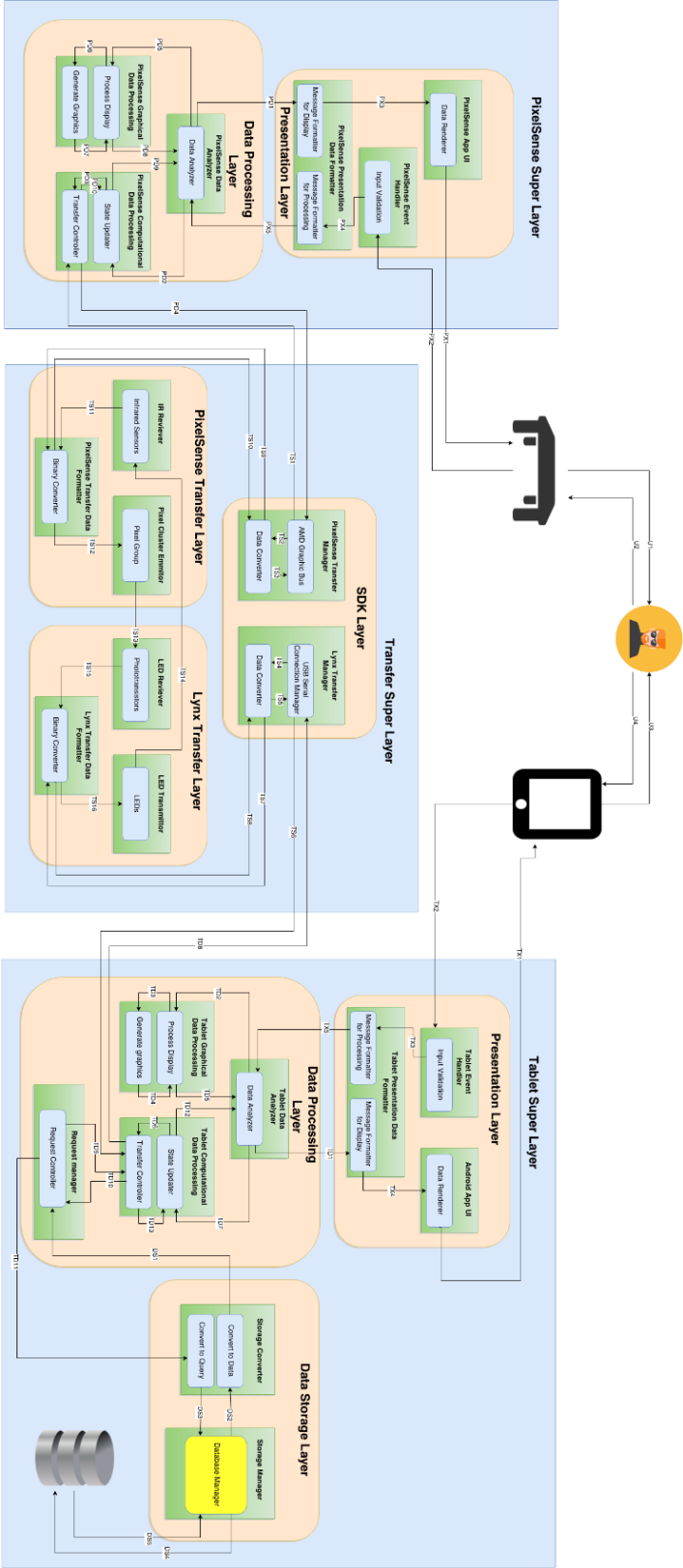


Figure 2‑2: Module Decomposition Diagram

### 2.4.1 PixelSense Super Layer

#### 2.4.1.1 Presentation Layer

The purpose of the PixelSense presentation layer is to present some form of information to the user. It also accepts input from the user which would trigger the corresponding events. This layer level will serve as main source of input from the user to the system as well as presentation to the user from the system. Information sent to this layer is initially formatted and then transferred for the processing. Similarly, information received by this device is formatted for the presentation.

##### 2.4.1.1.1 PixelSense App UI

PixelSense App UI subsystem will display the graphical as well as computational content to the PixelSense table. This is the only subsystem which handles the output of the data to the user PixelSense table.

###### 2.4.1.1.1.1 Data Renderer

This module will render graphical as well as computational data content to PixelSense Table’s screen.

##### 2.4.1.1.2 PixelSense Event Handler

PixelSense Event Handler subsystem will determine the type of data to be parsed into the process layer. This subsystem will handle the inputs of the user and forwards it to presentation data formatter.

###### 2.4.1.1.2.1 Input Validation

Input Validation module is responsible for determining data type of the user input.

##### 2.4.1.1.3 PixelSense Presentation Data Formatter

PixelSense Data Formatter subsystem will accept the input from the Event Handler subsystem. The formatter will then pre-process the data and translate it into a form that is acceptable for the Data processing Layer. The responsibility of this subsystem is to accept the input in various forms and translate it into acceptable form. The formatter should structure the data in such a way that it is easy to process. It also accepts the analyzed data from the system and processes it for the display to user.

###### 2.4.1.1.3.1 Message Formatter for Display

The Message Formatter for Display module will accept the analyzed data from Data Analyzer module, and it will format the data in such a way that it is ready for display on the table.

###### 2.4.1.1.3.2 Message Formatter for Processing

Message Formatter module for Processing will accept the validated input data from Event Handler subsystem, and it will format the data in such a way that it is ready for further processing. It then forwards formatted data to Data Analyzer for processing.

#### 2.4.1.2 Data Processing Layer

The purpose of this subsystem is to accept the data from the Presentation Layer and process it. It also serves the purpose of sending data back to the presentation layer after processing. This Layer contains a data formatter which analyzes the incoming or outgoing data and formats according to the need of the receiving layer. After analysis, if the data is non-graphical, it is being processed and sent to the PixelSense Transfer Manager, which forwards the data to the transfer Layer.

##### 2.4.1.2.1 PixelSense Data Analyzer

PixelSense Data Analyzer subsystem interacts with Data Formatter subsystem of Presentation layer. It basically analyzes the formatted data and identifies if it is graphical data or non-graphical data. After separation of data, graphical data will be sent to graphical data processor and non-graphical data will be sent to computational data formatter. It also sends back the analyzed data to data formatter in presentation layer.

###### 2.4.1.2.1.1 Data Analyzer

Data Analyzer module interacts with Presentation layer. It receives formatted data from processing message formatter module, and identifies graphical and computational data. Both forms of data are then transferred to graphical and computational data formatter respectively. This module also sends back analyzed data to message formatter for display purposes.

##### 2.4.1.2.2 PixelSense Graphical Data Processing

PixelSense Graphical Data Processing module interacts with the Data Analyzer module of the Data Processing layer. This subsystem is responsible for receiving data from the PixelSense Data Analyzer subsystem. It then processes the data needed for any graphical operations and then send the processed data back to the PixelSense Data Analyzer subsystem.

###### 2.4.1.2.2.1 Process Display

Process Display module accepts graphical data from data analyzer module, and uses that data to create graphical content using the graphics generator module.

###### 2.4.1.2.2.2 Generate Graphics

Generate Graphics module is responsible for generating game related graphics using the graphical content like card faces, table cloth, chips images, etc.

##### 2.4.1.2.3 PixelSense Computational Data Processing

This module will receive data from the PixelSense Data Analyzer module, process that information, and then forwards the processed information to the PixelSense Transfer Controller module in SDK layer as relevant computational (non-graphical) data. This module also processes the data coming from Lynx device and sends the processed data back to the PixelSense Data Analyzer module depending for the display purposes.

###### 2.4.1.2.3.1 State Updater

This module updates the state of input from the data analyzer, and makes it ready for the transfer controller to forward it to SDK layer. Data passed to this module is computational and no graphics are involved.

###### 2.4.1.2.3.2 Transfer Controller

This module is responsible for transferring computational data to the Graphics Transfer Controller inside SDK layer. This module also accepts the data coming from Lynx device and forwards it to State Updater for processing.

### 2.4.2 Tablet/Lynx Super Layer

#### 2.4.2.1 Presentation Layer

##### 2.4.2.1.1 Android App UI

###### 2.4.2.1.1.1 Data Renderer

This module will render data to be displayed to the user via the Tablet.

##### 2.4.2.1.2 Tablet Event Handler

###### 2.4.2.1.2.1 Input Validation

This module will validate and interpret the inputs from the user, given through the Tablet’s touch screen.

##### 2.4.2.1.3 Tablet Data Formatter

###### 2.4.2.1.3.1 Message Formatter for Processing

This module formats data to be sent to the Data Processing Layer.

###### 2.4.2.1.3.2 Message Formatter for Display

This module formats data received from the data processing layer, to be displayed to the user.

#### 2.4.2.2 Data Processing Layer

##### 2.4.2.2.1 Tablet Data Analyzer

###### 2.4.2.2.1.1 Data Analyzer

This module analyzes the data it receives and determines which parts need graphical processing and which parts need general computational processing.

##### 2.4.2.2.2 Tablet Graphical Data Processing

###### 2.4.2.2.2.1 Process Display

This module processes data to be displayed on the tablet.

###### 2.4.2.2.2.2 Generate Graphics

This module generates the graphics to be displayed to the user on the tablet.

##### 2.4.2.2.3 Tablet Computational Data Processing

###### 2.4.2.2.3.1 State Updater

This module updates the state of the application, processes general data.

###### 2.4.2.2.3.2 Transfer Controller

This module transfers data to the Lynx (USB Serial Connection Manager), which will be sent to the table.

##### 2.4.2.2.4 Request Manager

###### 2.4.2.2.4.1 Request Controller

This module receives requests to store and retrieve data in the database.

#### 2.4.2.3 Storage Layer

##### 2.4.2.3.1 Storage Converter

###### 2.4.2.3.1.1 Convert to Data

This module converts the result of an executed query into relevant data as per the request received from the Request Controller Module.

###### 2.4.2.3.1.2 Convert to Query

This module converts a request into a query to be executed, and passes that query on to the Database Manager Module.

##### 2.4.2.3.2 Storage Manager

### 2.4.2.3.2.1 Database Manager

This module connects to the database and executes the queries it receives, returning the result of that query to the Convert to Data module.

### 2.4.3 Transfer Super Layer

#### 2.4.3.1 SDK Layer

##### 2.4.3.1.1 PixelSense Transfer Manager

###### 2.4.3.1.1.1 Graphics Transfer Connector

The Graphics Transfer Connector module is responsible for transferring data back and forth from the PixelSense graphics module and the AMD Athlon processor.

###### 2.4.3.1.1.2 Data Converter

The Data Converter will convert the binary data to arrays of pixel location and value so that the graphics processor will know what pixels to update and where they should be updated. This will also work the other way and convert the incoming binary data from specific pixel points to raw computational data.

##### 2.4.3.1.2 Lynx Transfer Manager

###### 2.4.3.1.2.1 USB Serial Connection Manager

The USB Serial Connection Manager is responsible for transferring data back and forth between the Lynx microcontroller and the Android device over a serial connection with a baud rate of 9600.

###### 2.4.3.1.2.2 Data Converter

The Data Converter will convert the incoming data from the serial connection to binary and put it into an array so the Lynx microcontroller will know which LEDs to flash. This will also work the other way in that it will convert the raw binary input to a binary array which can be translated to English letters and words.

#### 2.4.3.2 PixelSense Transfer Layer

##### 2.4.3.2.1 IR Receiver

###### 2.4.3.2.1.1 Infrared Sensors

The Infrared Sensors will detect the Lynx device placed on the screen as well as detect the LEDs flashing for data transmission.

##### 2.4.3.2.2 IR Pixel Cluster Emitter

###### 2.4.3.2.2.1 Pixel Group

The Pixel Group module will flash a group of pixels on the screen proportional to a size of a phototransistor so that it can emulate flashing LEDs.

##### 2.4.3.2.3 PixelSense Transfer Data Formatter

###### 2.4.3.2.3.1 Binary Converter

The Binary Converter module will convert the incoming arrays to raw binary so the graphics processor can update the screen appropriately.

#### 2.4.3.3 Lynx Transfer Layer

##### 2.4.3.3.1 LED Receiver

###### 2.4.3.3.1.1 Phototransistors

The Phototransistor module is responsible to detect light and send the individual on/off states back to the Arduino.

##### 2.4.3.3.2 LED Transmitter

###### 2.4.3.3.2.1 LEDs

The LEDs will be an array of LEDs which will flash to transfer the incoming data.

##### 2.4.3.3.3 Lynx Transfer Data Formatter

###### 2.4.3.3.3.1 Binary Converter

The Binary Converter module will convert the incoming arrays to raw binary so the Arduino will know which LEDs to flash and in what specific order.

## 2.5 Module Producer-Consumer Matrix

The table below depicts the relationships between each module as well as the data flows in a more visual representation for a different perspective of the dataflow. The table below shows our data flows in the expected source and destination (producer and consumer). Consumers or the destination with a large number of outgoing or incoming flows indicate that the module is either complex or a bottleneck to the entire system as a whole. For table 2-3, it can be seen that the most of the modules are pretty well balanced with the most being 3 in/out for each Data Analyzer (PixelSense and Tablet). With that said the data analyzers can be seen as a moderate priority subsystem since only a few other module depend on it. With the moderate load on the Data Analyzer the rest of the system seems to be pretty well balanced in terms of load.

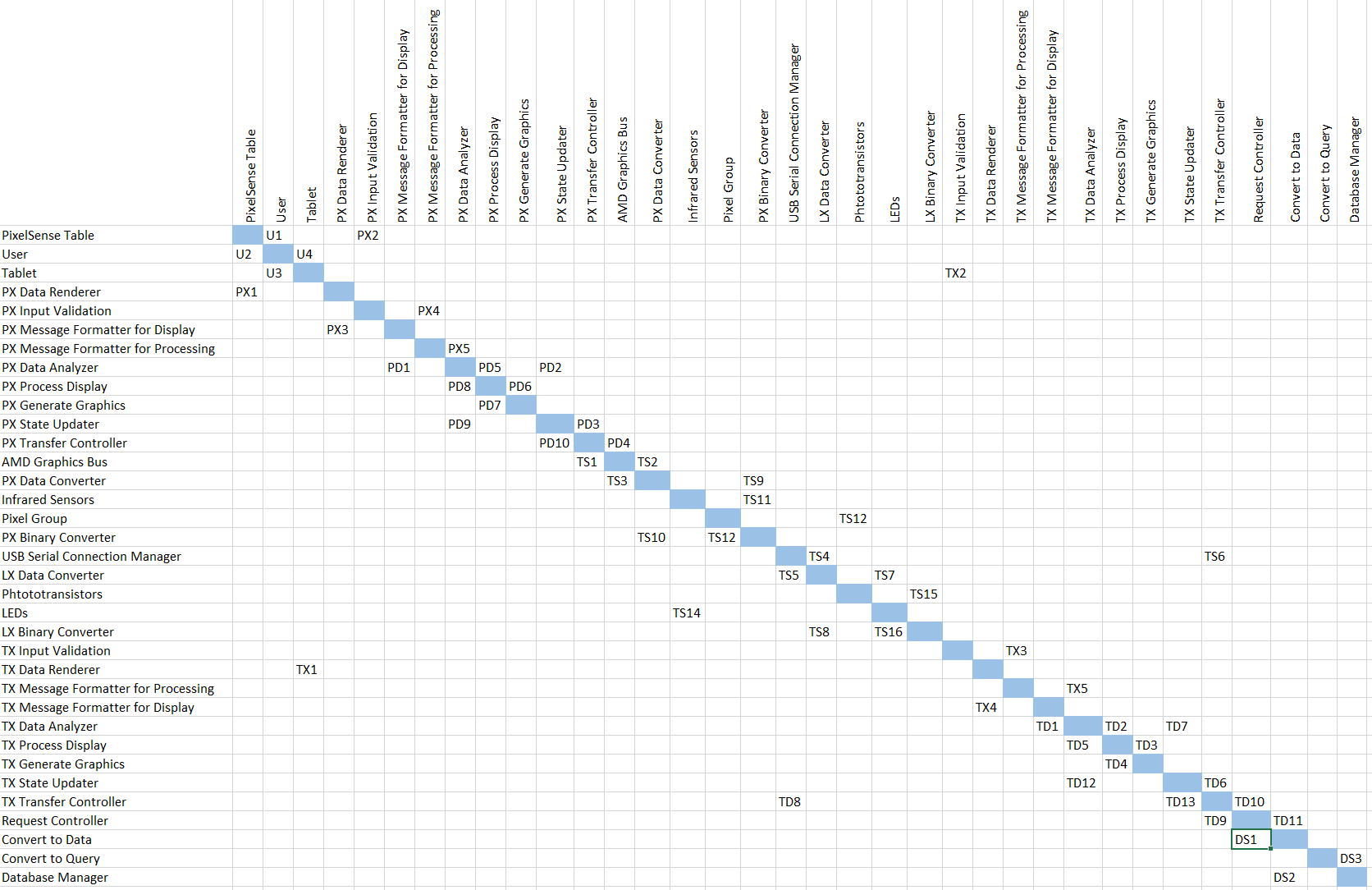


Figure 2‑3: Producer-Consumer Matrix

# 3. System Hardware Description

This section will cover each hardware component used in and in conjunction with the Lynx. This will cover the quantity, purpose, specifications and interfaces associated with each hardware component.

## 3.1 Arduino-Based Microcontroller



Figure 3‑1: Teensy 3.1

### 3.1.1 Quantity

The Lynx Transfer System will require one Arduino based Microcontroller.

### 3.1.2 Purpose

The Arduino-based Microcontroller will be the main driver and processor of the Lynx Transfer System, reading in inputs from the phototransistors and the USB interface, processing that data and transferring the data either back through the USB, or by lighting up the LEDs connected to the outputs. The Arduino-based Microcontroller will be powered through the USB interface by the Android tablet connected to it, and provides power to the components connected to it.

Specific specifications will be determined when we select a final device.

### 3.1.3 Interfaces

The Arduino Uno R3 will interface with the Android tablet through the USB port and a Micro USB to USB OTG cord. The Arduino will also receive input from the Shift Registers connected to the phototransistors, receiving bits indicating if the white light is being received or not. The Arduino will output to both the Android tablet using the USB port, and outputting bits to the Counter Shift Registers, to power the LEDs to send information to the PixelSense table.

## 3.2 IR LED



Figure 3‑2: IR LEDs

### 3.2.1 Quantity

The Lynx Transfer System will require 16 of these LEDs.

### 3.2.2 Purpose

These LEDs will allow us to send light to the PixelSense table so it can read data from our tablet optically. The LED will represent a bit, when on, it will represent 1, and when off and the table is in reading mode, it will represent 0.

Specifications:

|  |  |
| --- | --- |
| Diameter | 5mm |
| Casing | Clear |
| Light Color | White |
| Forward Voltage | 3.0 – 3.2V |

Table 3‑1: LED Specifications

### 3.2.3 Interfaces

The IR LEDs will interface with the Arduino by receiving current from it in order to light it up. The output of the LED will be infrared light that will represent a binary 1 to the PixelSense table and when not shining, it will represent a binary 0.

## 3.3 Resistors - 1/4 Watt - 5%



Figure 3‑3: Various Resistors

### 3.3.1 Quantity

We will use a varying quantity of resistors depending on our hardware design.

### 3.3.2 Purpose

These resistors will allow us to limit the amount of current flowing through electronic components within the Lynx Transfer System. To do this, we will be using resistors of different Ohm ratings depending on the component connected to it.

Specifications:

|  |  |
| --- | --- |
| Resistance: | Varying (1KΩ,10KΩ, 333Ω) |
| Power Rating | ¼ W |
| Tolerance | 5% |
| Resistance: | Varying (1KΩ,10KΩ, 333Ω) |

Table 3‑1: Resistor Specifications

### 3.3.3Interfaces

The interfaces for the resistors will be dependent on which electrical components will need them, which will be determined in our final implementation..

## Phototransistor



Figure 3‑4: Phototransistors

### 3.4.1 Quantity

The Lynx Transfer System will use 16 of these phototransistors.

### 3.4.2 Purpose

These phototransistors will allow us to detect light emitting from the PixelSense table, thus using it to transfer bits from the table to the Lynx.

Specific specifications will be determined when we select a final device.

### 3.4.3 Interfaces

The phototransistor will receive input in the form of light from the PixelSense table. The light will be translated into current by the phototransistor, and passed through to a NPN transistor to magnify that current to a level that can be read by the Arduino. The phototransistor will receive power from the Arduino’s 5V pin.

## 3.5 Micro-USB 2.0 On-The-Go (OTG) Adapter



Figure 3‑5: Micro USB OTG (On-The-Go) Cable

### 3.5.1 Quantity

The Lynx Transfer System will need one of these adaptors.

### 3.5.2 Purpose

This Micro USB to OTG adaptor will allow us to connect the Android tablet we use to the USB port of the Arduino (using a Micro USB to USB cable) on the Lynx so we can transfer data between the two devices.

Specifications:

|  |  |
| --- | --- |
| Input: | USB 2.0 Micro-B Male |
| Output: | USB 2.0 Type A Female |

Table 3‑2: Micro USB OTG Cable Specifications

### 3.5.3 Interfaces

This adaptor will connect to the Android tablet using the USB 2.0 Micro-B Male input and connect to the Arduino on the Lynx with USB Male A to USB Male B cable on the USB 2.0 Type A Female output of this adaptor.

## 3.6 3-Feet USB 2.0 Cable



Figure 3‑6: USB A-B Cable

### 3.6.1 Quantity

The Lynx Transfer System will need one of these cables.

### 3.6.2 Purpose

The purpose of this cable is to connect Arduino on the Lynx Transfer System to the Micro USB OTG adaptor connected to the Android tablet. Once connected, this cable will allow us to transfer data between the two devices.

Specific specifications will be determined when we select a final device.

### 3.6.3 Interfaces

This cord will connect to the adaptor connected to the Android tablet using the USB 2.0 Male A input and connect to the Arduino on the Lynx with the output of this cable.

## 3.7 MicroSD card breakout board+

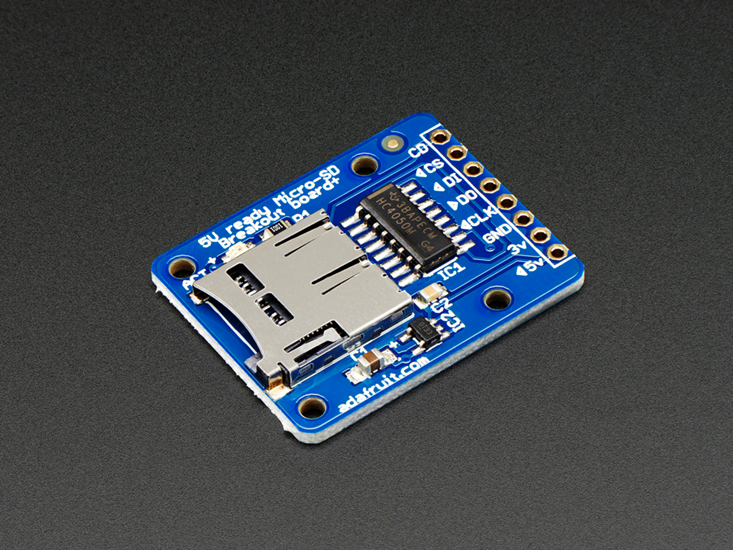


Figure 3‑7: MicroSD Shield for Arduino

### 3.7.1 Quantity

The Lynx Transfer System will need one of these Micro SD Card shields.

### 3.7.2 Purpose

This Micro SD Card shield will allow us to store data we receive from the table and store it on a Micro SD Card for processing. This will allow for easy access of our stored data.

Specific specifications will be determined when we select a final device.

### 3.7.3 Interfaces

The Micro SD Card breakout board will connect to the Arduino through the digital input pins, which will allow it to read and write to the Micro SD card plugged into the breakout board.

## 3.8 MicroSD Memory Card



Figure 3‑8: 2GB MicroSD Card

### 3.8.1 Quantity

The Lynx Transfer System will need one of these MicroSD cards

### 3.8.2 Purpose

This MicroSD card will be our storage for any data that is sent to the Android tablet through the Lynx Transfer System.

Specific specifications will be determined when we select a final device.

### 3.8.3 Interfaces

The MicroSD card will be placed within the breakout board attached to the Arduino, which will allow the Arduino to read and write to the MicroSD card.

## 3.9 Nexus 7



Figure 3‑9: Nexus 7 Tablet

### 3.9.1 Quantity

The Lynx Transfer System will need one Nexus 7.

### 3.9.2 Purpose

We will be using the Nexus 7 to interact with the Lynx Transfer System. The tablet will allow us to see the information being sent from the PixelSense table through Lynx, modify that data, and use the controls on the touch screen of the tablet to activate the Lynx to send data back to the table if needed.

Specifications

|  |  |
| --- | --- |
| Processor: | 1.5 GHz Snapdragon S4 –Quad Core |
| RAM: | 2 GB DDR2 SDRAM |
| Hard Drive: | 16 GB SATA Drive |
| Port: | Micro USB 2.0 |
| Size (LWH): | 7.87 inches, 4.49 inches, 0.34 inches |
| Weight: | 10.24 ounces |
| Wireless Type: | 802.11bgn |
| Battery | 3,950 mAh Lithium Ion |

Table 3‑3: Nexus 7 Tablet Specifications

### 3.9.3 Interfaces

We will interface with the Nexus 7 by connecting a Micro USB to OTG adaptor to it, allowing us to plug in a USB Male A to Male B cord, which we will also plug into the Arduino’s USB port on the Lynx. This will allow us to transfer data between the Lynx and the Nexus 7. We will also being receiving inputs in the form of touch on the Nexus 7’s touch screen. This will allow the user to use the controls we give them on the Android program we will be writing for the transfer system.

## 3.10 Samsung SUR40



Figure 3‑10: Microsoft PixelSense 2.0 SUR40

### 3.10.1 Quantity

The Lynx Transfer System will need one Samsung SUR40 in order to function.

### 3.10.2 Purpose

This is the device we will be reading from optically and transferring data to. This device will run our blackjack software as a demonstration of the Lynx’s capabilities, showing that we can transfer data to and from the table and an Android tablet connected to Lynx optically.

Specifications:

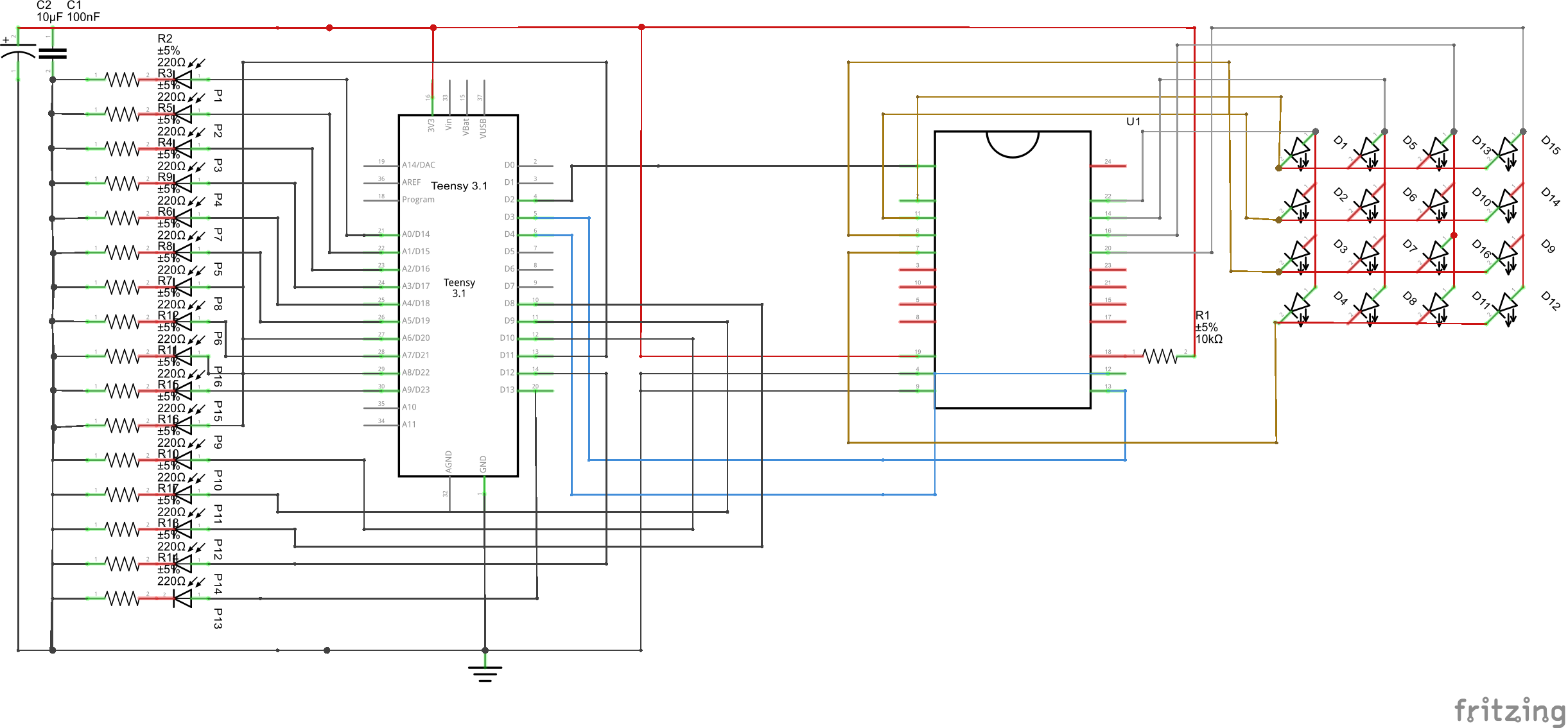
|  |  |
| --- | --- |
| H-Scanning Frequency | 30 - 81KHz |
| Maximum Pixel Frequency | 148.5 MHz |
| V-Scanning Frequency | 56~75 Hz |
| Diagonal Size | 40" |
| Light Source Type | CCFL |
| Type | a-Si TFT active matrix |
| Resolution | 1920 x 1080 |
| Pixel Pitch(mm) | 0.46125 x 0.46125 mm |
| Active Display Area | 885.6 x 498.15 mm |
| Brightness (Typical) | 300 cd/㎡ |
| Response Time (G-to-G) | 8.0 ms |
| Contrast Ratio | 2,000:1 |
| Display Colour | 16.7 M |
| CPU | AMD Athlon X2 Dual-Core 245e |
| HDD | 320 GB HDD |
| RAM | 4 GB DDR3 |
| OS | Microsoft Windows 7 Pro with PixelSense |

Table 3‑4: Microsoft PixelSense SUR40 Specifications

### 3.10.3 Interfaces

This device will interface with the user using the touch screen capabilities it has. We will provide touch controls that will allow the user to use the blackjack software provided to them. In addition, whenever the Lynx is on, connected to an Android tablet, and placed on the table, the table will be able to send and receive data from the Lynx device optically.

## 3.11 Conceptual Hardware Diagram



# 4. System Software Description

This section will introduce and explain the software technologies and solutions used in the three parts of our whole system; PixelSense Windows Application, the Android application and the Arduino sketch. The specific programming languages and development frameworks/designs will be provided. The basic descriptions below will provide sufficient information to understand the platforms and environment that the product will run on. All third party tools and libraries used will be included in the appendix of this document.

## 4.1 PixelSense Windows Application

The PixelSense Windows Application will be the main program which will be running the Blackjack game as well as initiate the communication between our devices. This application will be utilizing the AMD graphics processor to update the game screen and since it will be a native PixelSense application, the program will also have unrestricted access to the table’s sensors such as the infrared grid sensors underneath the screen. The PixelSense application will be written in C# and use the Microsoft PixelSense 2.0 SDK to access the various hardware features easily and will be exclusively developed for Windows 7 Professional (NT 6.1, Build 7600) and on Visual Studio 2010. The application will also incorporate the Microsoft XNA Framework which includes some of the XBOX development tools as well as the Windows Presentation Foundation (WPF) framework. Using all these tools the PixelSense Windows Application will be able to communicate with the Lynx device back and forth and update the application based on the received data from the Lynx device.

## 4.2 Android Application

The Android Application will be the primary interface that the user will be communicating with on the Android tablet. On this application the user will be able to place bets, stand and input various Blackjack game options. The Android application will developed for Android 4.2.2+ (Jelly Bean, API 17) and will incorporate the Android external USB libraries to communicate with the Lynx device. The application will also use the OpenGL ES 2.0 library for all graphics rendering and animations. In light of this the team has decided to develop the application in a native android environment (for more flexibility than Cordova) which will include the Java and XML programming languages. The application environment we will be using is the Google provided Android Studio and the gradle build system. The final output will be an APK file which will be installable on any android compatible android device with USB Host capability and have an operating system version of 4.2.2 and up. Based on this our team will maintain all the rights to any code produced as a result of using this platform.

## 4.3 Arduino Sketch

The Arduino Sketch will be the main program running on our Arduino microcontroller. The sketch will be responsible for receiving and transferring data back and forth between the PixelSense table and the Android device. The Arduino microcontroller will be a part of our Lynx device. The sketch will incorporate external SD card libraries in order to use the flash storage not part of the microcontroller.

# 5. PixelSense Super Layer

The purpose of this super Layer is to serve the applications on PixelSense Table. The main function is to get the data from the user on the PixelSense table and process it to forward it to the transfer layer. This super layer also serves the purpose of displaying the data back to user after it is being transferred back from different super layer. Having a super layer is useful showing the different parts of both systems as well as showing them as one system as a whole.

## 5.1 Presentation Layer

The purpose of the PixelSense presentation layer is to present some form of information to the user. It also accepts input from the user which would trigger the corresponding events. This layer level will serve as main source of input from the user to the system as well as presentation to the user from the system. Information sent to this layer is initially formatted and then transferred for the processing. Similarly, information received by this device is formatted for the presentation.

### 5.1.1 PixelSense App UI

PixelSense App UI subsystem will display the graphical as well as computational content to the PixelSense table. This is the only subsystem which handles the output of the data to the user PixelSense table.

#### 5.1.1.1 Data Renderer

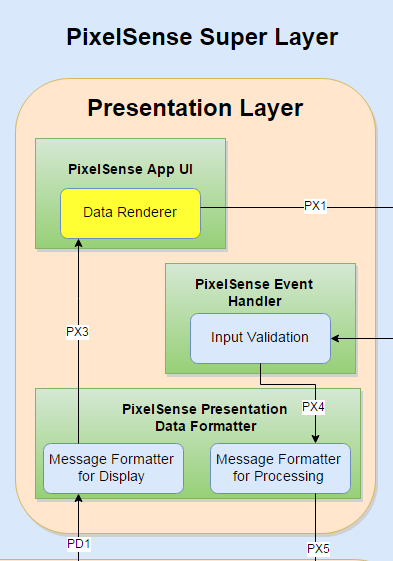


Figure 5‑1: Data Renderer Module

**Prologue:** This module will render graphical as well as computational data content to PixelSense Table’s screen**.**

**Interfaces:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Sink** | **Input to Sink** | **Return from Sink** |
| Message Formatter For Display | Data Renderer | Formatted String Message, Images | None |
| Data Renderer | User | Displayed String Message, Images | None |

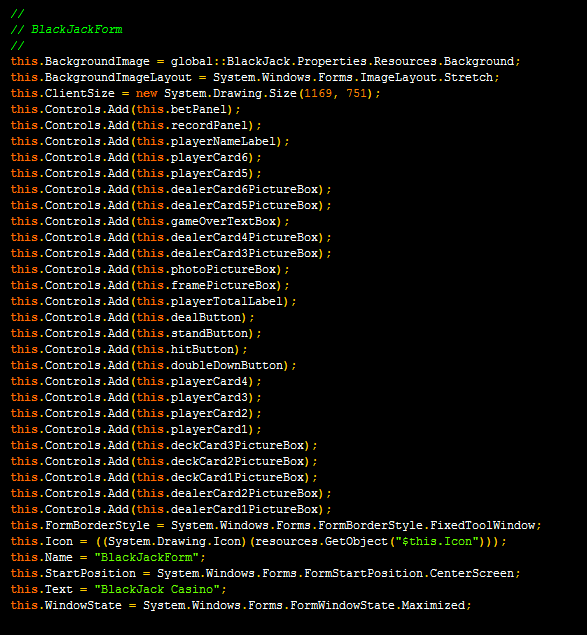
Table 5‑1: Data Renderer Interfaces

**External Data Dependencies:** User to be able to see the formatted message.

**Internal Data Dependencies: N/A**

**Object Descriptions: N/A**

**Pseudo Code:**

****

**Quality Assurance:**

This Module will be tested by updating game data on the table. All team mates are expected to test it once by playing game before deployment.

### 5.1.2 PixelSense Event Handler

PixelSense Event Handler subsystem will determine the type of data to be parsed into the Data Processing layer. This subsystem will handle the inputs of the user and forwards it to presentation data formatter.

#### 5.1.2.1 Input Validation

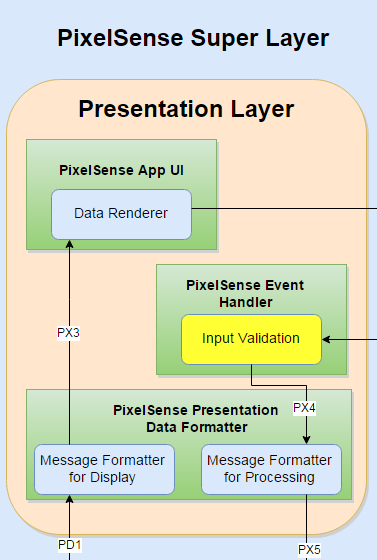


Figure 5‑2: Input Validation Module

**Prologue:** Input Validation module is responsible for determining data type of the user input.

**Interfaces:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Sink** | **Input to Sink** | **Return from Sink** |
| User | Input Validation | User Event | None |
| Input Validation | Message Formatter for Processing | Input String Message | None |

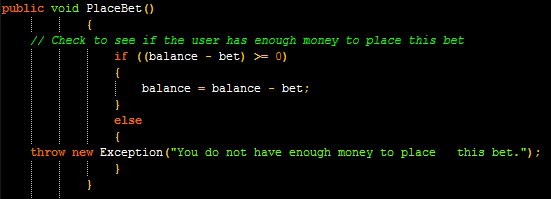
Table 5‑2: Input Validation Interfaces

**External Data Dependencies:** User Input from the PixelSense Table

**Internal Data Dependencies:** Surface 2.0 SDK

**Object Descriptions: N/A**

**Pseudo Code:**

****

**Quality Assurance:** This module will be tested by entering touch inputs/ Lynx inputs through table to the system. All team mates are expected to test this module by playing game before deployment.

### 5.1.3 PixelSense Presentation Data Formatter

PixelSense Data Formatter subsystem will accept the input from the Event Handler subsystem. The formatter will then pre-process the data and translate it into a form that is acceptable for Data Processing Layer. The responsibility of this subsystem is to accept the input in various forms and translate it into acceptable form. The formatter should structure the data in such a way that it is easy to process. It also accepts the analyzed data from the system and processes it for the display to user.

#### 5.1.3.1 Message Formatter for Display

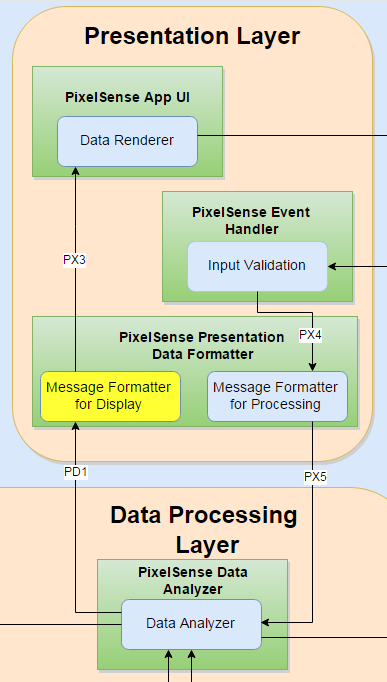


Figure 5‑3: Message Formatter for Display Module

**Prologue:** Message Formatter for display for module will accept the analyzed data from Data processing layer, and it will format the data in such a way that it is ready for display on the table.

**Interfaces:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Sink** | **Input to Sink** | **Return from Sink** |
| Data Analyzer | Message Formatter for Display | Analyzed string data, Images | None |
| Message Formatter for Display | Data Renderer | Formatted string data for display, Images | None |

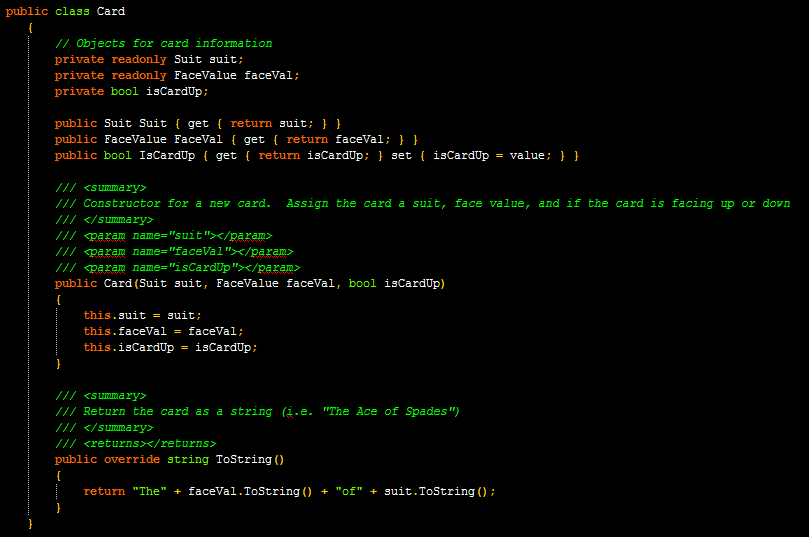
Table 5‑3: Message Formatter for Display Interfaces

**External Data Dependencies: N/A**

**Internal Data Dependencies:** Surface 2.0 SDK

**Object Descriptions: N/A**

**Pseudo Code:**

****

**Quality Assurance:** This module will be tested by formatting analyzed data into message that can be processed by graphical and general processing components.

#### 5.1.3.2 Message Formatter for Processing

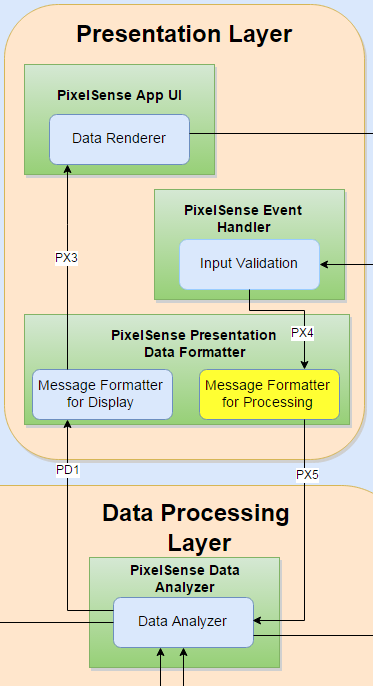


Figure 5‑4: Message Formatter for Processing Module

**Prologue:** Message Formatter for processing module will accept the validated input data from Event Handler subsystem, and it will format the data in such a way that it is ready for further processing. It then forwards formatted data to Data Analyzer for processing.

**Interfaces:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Sink** | **Input to Sink** | **Return from Sink** |
| Input Validation | Message Formatter for processing | Validated input string data | None |
| Message Formatter for processing | Data Analyzer | Formatted string data for processing | None |

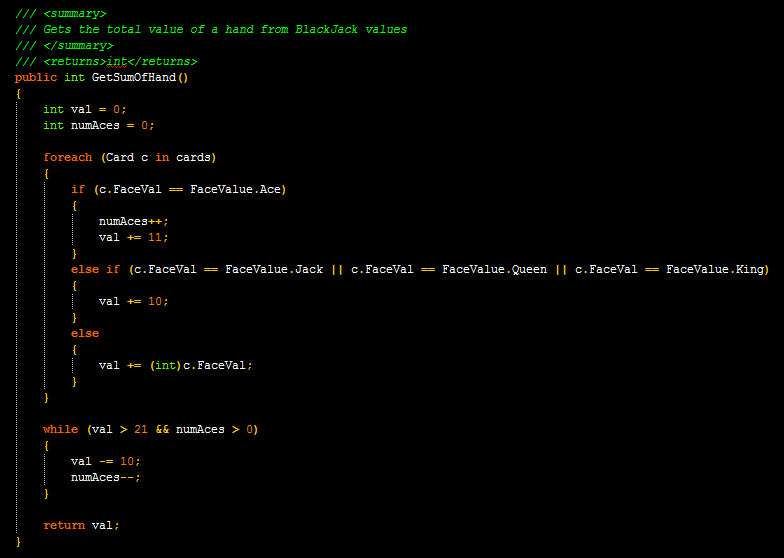
Table 5‑4: Message Formatter for Processing Display Interfaces

**External Data Dependencies: N/A**

**Internal Data Dependencies:** Surface 2.0 SDK

**Object Descriptions: N/A**

**Pseudo Code:**

****

**Quality Assurance:** This module will be tested by seeing if the data coming in data renderer is ready to be displayed or not.

## 5.2 Data Processing Layer

The purpose of this subsystem is to accept the data from the Presentation Layer and process it. It also serves the purpose of sending data back to the presentation layer after processing. This layer contains a data formatter which analyzes the incoming or outgoing data and formats according to the need of the receiving layer. After analysis, if the data is non-graphical, it is being processed and sent to the PixelSense Transfer Manager, which forwards the data to the transfer Layer.

### 5.2.1 PixelSense Data Analyzer

PixelSense Data Analyzer subsystem interacts with Data Formatter subsystem of Presentation layer. It basically analyzes the formatted data and identifies if it is graphical data or non-graphical data. After separation of data, graphical data will be sent to graphical data processor and non-graphical data will be sent to computational data formatter. It also sends back the analyzed data to data formatter in presentation layer.

#### 5.2.1.1 Data Analyzer

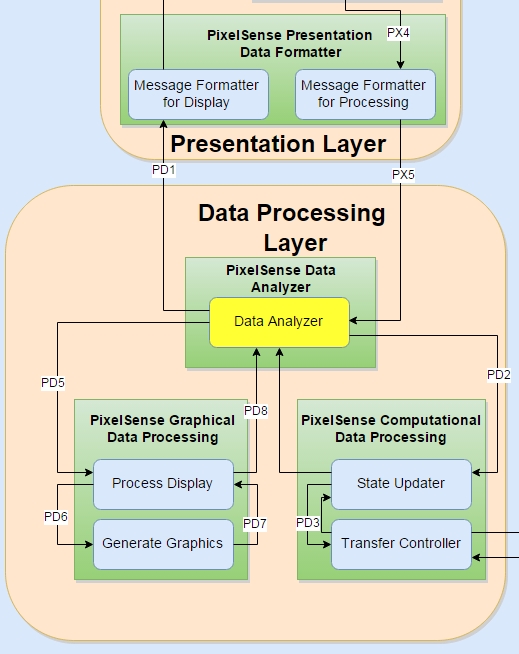


Figure 5‑5: Data Analyzer Module

**Prologue:** Data Analyzer module interacts with Presentation layer. It receives formatted data from processing message formatter module, and identifies graphical and computational data. Both forms of data are then transferred to graphical data formatter and computational data formatter respectively. This module also sends back analyzed data to the Message Formatter for Display for display purposes.

**Interfaces:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Sink** | **Input to Sink** | **Return from Sink** |
| Data Analyzer | Message Formatter for Display | Analyzed string data | None |
| Message Formatter for Processing | Data Analyzer | Formatted string data | None |
| Process Display | Data Analyzer | Graphics generated data, Images | Analyzed string Data |
| State Updater | Data Analyzer | Computed integer data | Analyzed integer Data |

Table 5‑5: Data Analyzer Interfaces

**External Data Dependencies: N/A**

**Internal Data Dependencies:** Surface 2.0 SDK

**Object Descriptions: N/A**

**Pseudo Code:**

****

**Quality Assurance:** This module will be tested by seeing if the data coming in data renderer is ready to be analyzed or not.

### 5.2.2 PixelSense Graphical Data Processing

PixelSense Graphical Data Processing subsystem interacts with the Data Analyzer subsystem of the Data Processing layer. This subsystem is responsible for receiving data from the PixelSense Data Analyzer subsystem. It then processes the data needed for any graphical operations and then send the processed data back to the PixelSense Data Analyzer subsystem.

#### 5.2.2.1 Process Display

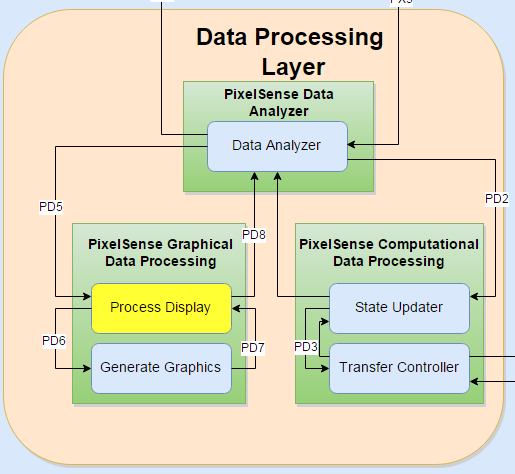


Figure 5‑6: Process Display Module

**Prologue:** Process Display module accepts graphical data from data analyzer module, and uses that data to create graphical content using Generate Graphics module.

**Interfaces:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Sink** | **Input to Sink** | **Return from Sink** |
| Process Display | Data Analyzer | Graphics generated data, Images | Analyzed string data |
| Generate Graphics | Process Display | Images | Raw data request query |

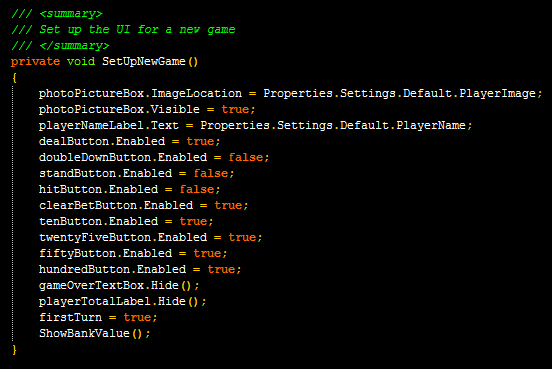
Table 5‑6: Process Display Interfaces

**External Data Dependencies: N/A**

**Internal Data Dependencies:** Surface 2.0 SDK

**Object Descriptions: N/A**

**Pseudo Code:**

****

**Quality Assurance:**

This module will be tested by generating a graphic image and passing it onto the Process Display module.

#### 5.2.2.2 Generate Graphics

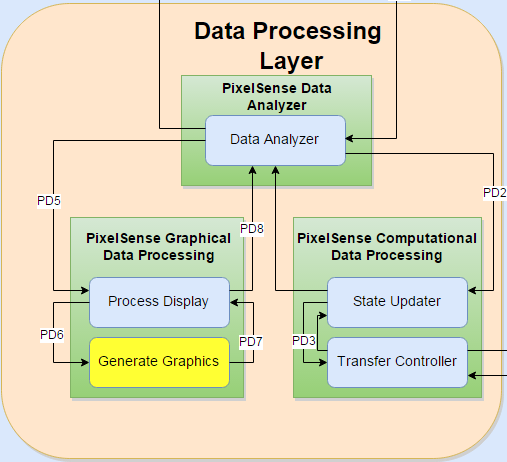


Figure 5‑7: Generate Graphics Module

**Prologue:** Generate Graphics module is responsible for generating game related graphics using the graphical content like card faces, table cloth, chips images, etc.

**Interfaces:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Sink** | **Input to Sink** | **Return from Sink** |
| Generate Graphics | Process Display | Images | Raw Data request query |

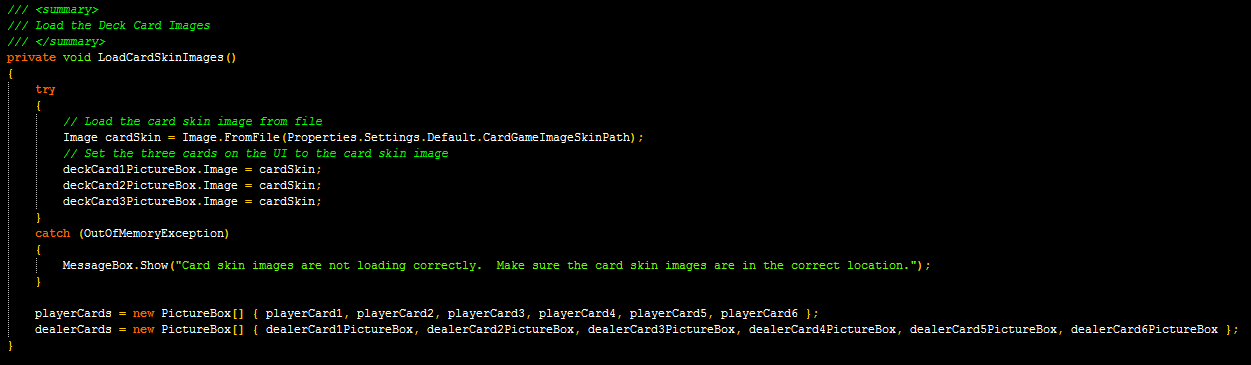
Table 5‑7: Generate Graphics Interfaces

**External Data Dependencies: N/A**

**Internal Data Dependencies:** Surface 2.0 SDK

**Object Descriptions: N/A**

**Pseudo Code:**

****

**Quality Assurance:** This module will be tested by seeing the proper card & game graphics on the table.

### 5.2.3 PixelSense Computational Data Processing

This module will receive data from the PixelSense Data Analyzer module, process that information, and then forwards the processed information to the PixelSense Transfer Controller module in SDK layer as relevant computational (non-graphical) data. This module also processes the data coming from Lynx device and sends the processed data back to the PixelSense Data Analyzer module depending for the display purposes.

#### 5.2.3.1 State Updater



Figure 5‑8: State Updater Module

**Prologue:** This module updates the state of input from the data analyzer, and makes it ready for the transfer controller to forward it to SDK layer. Data passed to this module is computational and no graphics are involved.

**Interfaces:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Sink** | **Input to Sink** | **Return from Sink** |
| State Updater | Data Analyzer | Computed integer data after updating state | Analyzed integer data |
| Transfer Controller | State Updater | Integer data | Computed integer data after updating State |

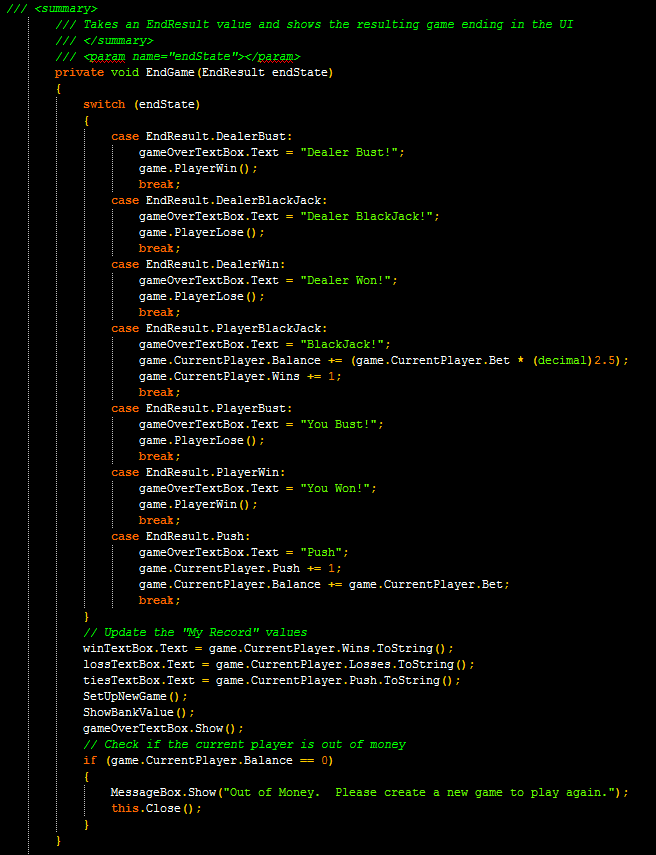
Table 5‑8: State Updater Interfaces

**External Data Dependencies:** Valid Id for Graphic Images

**Internal Data Dependencies:** Surface 2.0 SDK

**Object Descriptions:** N/A

**Pseudo Code:**

****

**Quality Assurance:** This module will be tested by seeing the state of the transferred data inside Transfer super layer.

#### 5.2.3.2 Transfer Controller

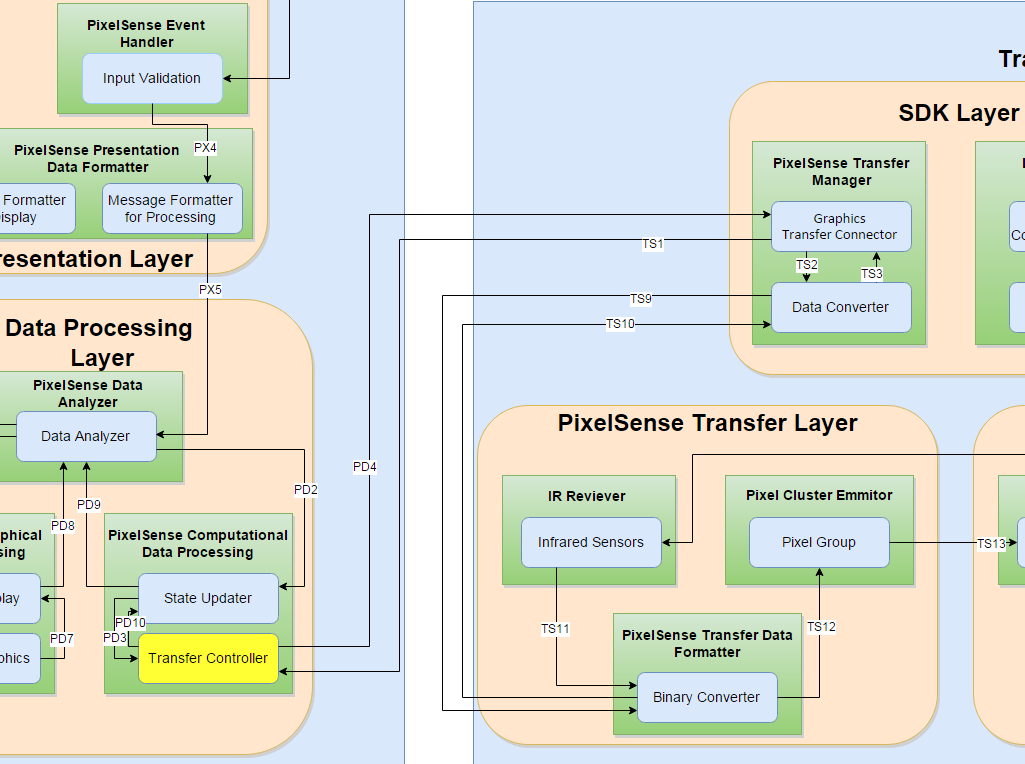


Figure 5‑9: Transfer Controller Module

**Prologue:** This module is responsible for transferring computational data to the Graphics Transfer Connector inside SDK layer. This module also accepts the data coming from Lynx device and forwards it to State Updater for processing.

**Interfaces:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Sink** | **Input to Sink** | **Return from Sink** |
| Transfer Controller | State Updater | Computed integer data | Computed data after updating State |
| Graphics Transfer Connector | Transfer Controller | Graphics Data, Images | Computed data |

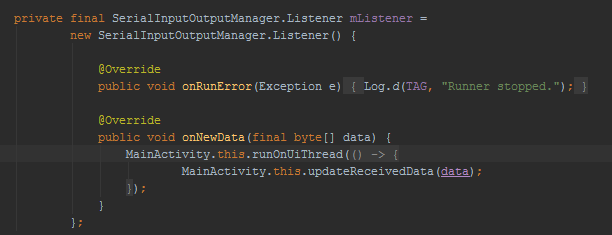
Table 5‑9: Transfer Controller Interfaces

**External Data Dependencies: N/A**

**Internal Data Dependencies:** Surface 2.0 SDK

**Object Descriptions:** N/A

**Pseudo Code:**

****

**Quality Assurance:** This module will be tested by seeing if the data coming in from Graphic Transfer Controller is ready to be transferred or not.

# 6. Tablet Super Layer

The purpose of this super layer is to enclose and distinguish between the three internal layers. The main functionality is to show the dataflow between the three internal layers, and between the Lynx and the tablet

## 6.1 Presentation Layer

The purpose of this layer is to present the data on the Lynx to the user, and handle the user’s inputs. The layer will display the data on the Lynx, a log of recent transactions, and buttons the user can interact with via the tablet’s touch screen. Each interaction from the user is handled, formatted and transferred to the Data Processing Layer.

### 6.1.1 Android App UI

This subsystem provides a user interface for the Tablet/Lynx system, allowing the user to both visualize what data is currently on the Lynx but also allowing the user to interact with the PixelSense application using the tablet.

#### 6.1.1.1 Data Renderer

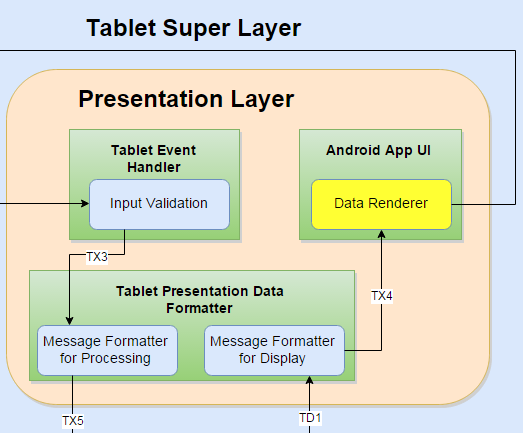


Figure 6‑: Data Renderer Module

**Prologue:** This module is responsible for displaying to the user what data is currently on the Lynx, as well as displaying buttons for application commands, and a log of recent transactions made between the Tablet/Lynx and the PixelSense table.

**Interfaces:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Sink** | **Input to Sink** | **Return from Sink** |
| Tablet Data Renderer | User | Display | None |
| Message Formatter for Display | Tablet Data Renderer | User Event | None |

Table 6‑: Data Renderer Interfaces

**External Data Dependencies: N/A**

**Internal Data Dependencies:** Android API 17

**Object Descriptions: N/A**

**Pseudo Code:**

****

**Quality Assurance:** This module will be tested by displaying the users chip count and appropriate application commands on the tablet.

### 6.1.2 Tablet Event Handler

This subsystem takes tablet inputs from the user and passes them on so that the system can update the UI as well as handle any application processing needed.

#### 6.1.2.1 Input Validator

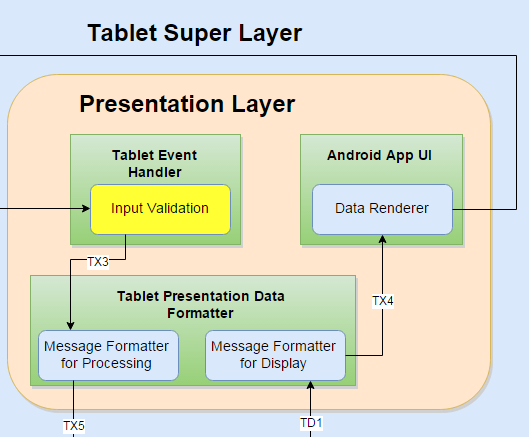


Figure 6‑: Input Validator Module

**Prologue:** This module is responsible for taking the user’s inputs and passing them along to the Tablet Data Formatter module for further processing.

**Interfaces:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Sink** | **Input to Sink** | **Return from Sink** |
| User | Input Validator | User input (String, int) | None |
| Input Validator | Message Formatter for Processing | Event ID (int) | None |

Table 6‑: Input Validation Interfaces

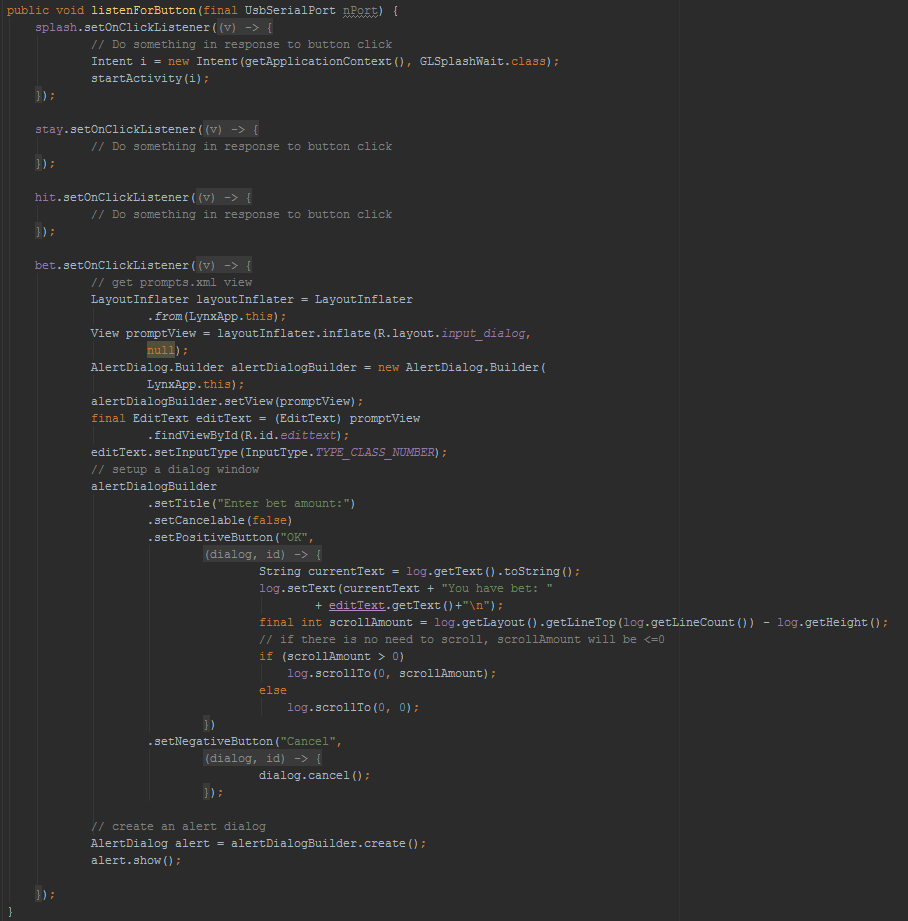
**External Data Dependencies: N/A**

**User input: N/A**

**Internal Data Dependencies:** Android API 17

**Object Descriptions: N/A**

**Pseudo Code:**

****

**Quality Assurance:** This module will be tested by ensuring the clicking of buttons on the tablet display correspond to an input event.

### 6.1.3 Tablet Data Formatter

This subsystem formats the data received from the Data Processing Layer and pushes it up to the Android App UI, as well as formatting data from the Tablet Event Handler subsystem pushing it to the Data Processing Layer to be processed.

#### 6.1.3.1 Message Formatter for Processing

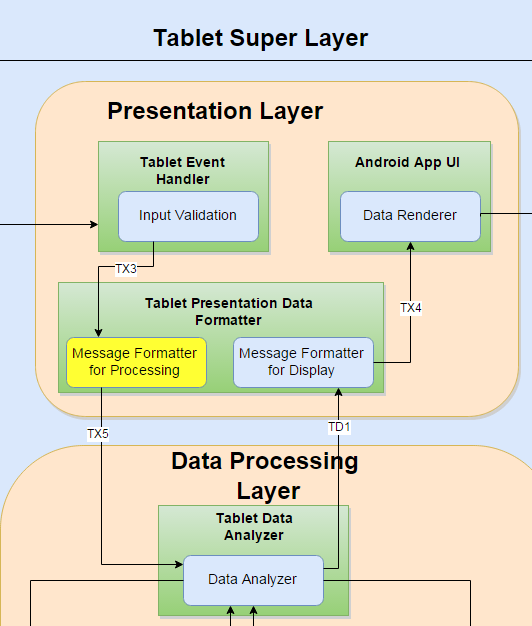


Figure 6‑: Message Formatter for Processing Module

**Prologue:** This module is responsible for formatting data received from the Tablet Event Handler. The module also pushes the formatted data up to the Data Renderer module for display to the user.

**Interfaces:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Sink** | **Input to Sink** | **Return from Sink** |
| Tablet Data Formatter for Processing | Tablet Data Analyzer | Formatted User input (String, int) | None |
| Tablet Data Analyzer | Tablet Data Formatter for Processing | User input (String, int) | None |

Table 6‑: Message Formatter for Processing

**External Data Dependencies: N/A**

**Internal Data Dependencies:** Android API 17

**Object Descriptions:** **N/A**

**Pseudo Code:**



**Quality Assurance:** This module will be tested by formatting an android event into message that can be processed by graphical and general processing components.

#### 6.1.3.2 Message Formatter for Display

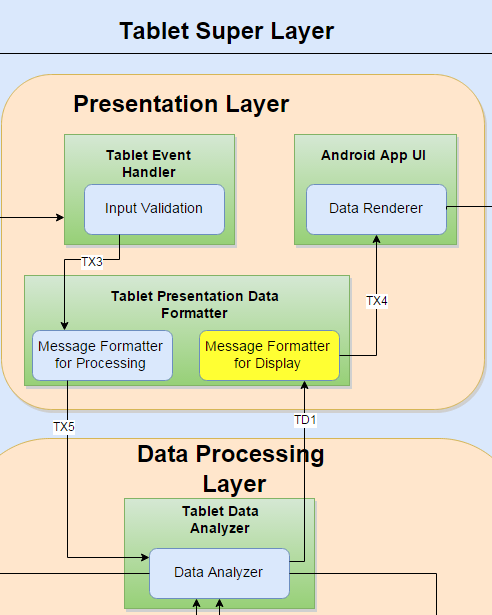


Figure 6‑: Message Formatter for Display Module

**Prologue:** This module is responsible for formatting processed data to be displayed to the user. It will receive the data from the Tablet Data Analyzer and push it to the Data Renderer Module.

**Interfaces:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Sink** | **Input to Sink** | **Return from Sink** |
| Tablet Data Formatter for Display | Data Renderer | Android Event | None |
| Tablet Data Analyzer | Tablet Data Formatter for Display | Data for next Event (Android Event) | None |

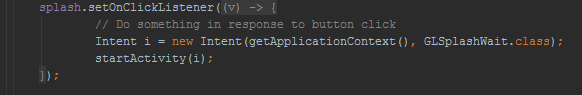
Table 6‑: Message Formatter for Display Interfaces

**External Data Dependencies: N/A**

**Internal Data Dependencies:** Android API 17

**Object Descriptions: N/A**

**Pseudo Code:**

****

**Quality Assurance:** This module will be tested by creating a message to be rendered by the Data Renderer.

## 6.2 Data Processing Layer

The purpose of this layer is to process the data from the Presentation layer, the Data Storage Layer, and the Transfer Super Layer. This layer will accept an input of data from one of the previously stated layers, process that data, and then transfer it to the receiving layer of the transaction.

### 6.2.1 Tablet Data Analyzer

This subsystem analyzes data to be processed or data that has just been processed. If data needs to be processed the subsystem will determine whether it is graphical or computational (general) processing and send the data to those respective subsystems. If the data has already been processed the subsystem will push the processed data to the Presentation Layer.

#### 6.2.1.1 Data Analyzer

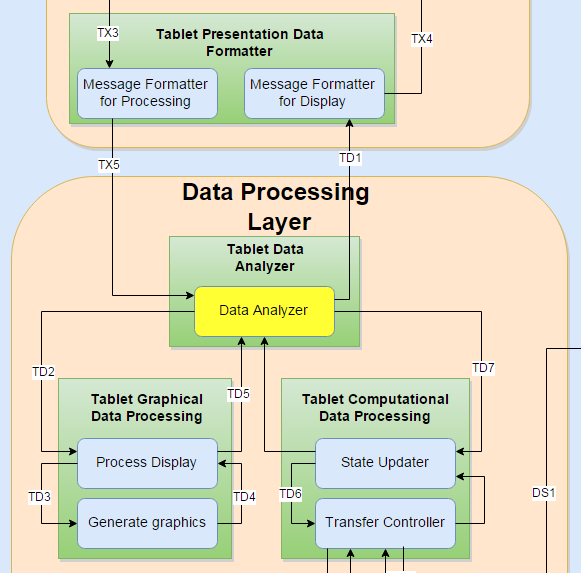


Figure 6‑: Data Analyzer Module

**Prologue:** This module is responsible for analyzing data to be processed or data that has just been processed. If data needs to be processed the subsystem will determine whether it is graphical or computational (general) processing and send the data to those respective subsystems. If the data has already been processed the subsystem will push the processed data to the Presentation Layer.

**Interfaces:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Sink** | **Input to Sink** | **Return from Sink** |
| Tablet Data Analyzer | Message Formatter for Display | Processed android event (Android Event) | None |
| Tablet Data Analyzer | State Updater | Formatted data for processing  (Android Event) | None |
| Tablet Data Analyzer | Process Display | Formatted data for processing (String) | None |
| Process Display | Tablet Data Analyzer | Android Event | None |
| Message Formatter for Processing | Tablet Data Analyzer | Formatted data for processing  (Android Event) | None |

Table 6‑: Data Analyzer Interfaces

**External Data Dependencies: N/A**

**Internal Data Dependencies:** Android API 17

**Object Descriptions: N/A**

**Pseudo Code:**

****

**Quality Assurance:** This module will be tested by sending it Android events to be split among the processing subsystems and by taking the processed data and passing it back to the proper module in the Data Formatter subsystem.

### 6.2.2 Tablet Graphical Data Processing

This subsystem processes all graphical data, such as generating images for cards, and any other visual data.

#### 6.2.2.1 Process Display

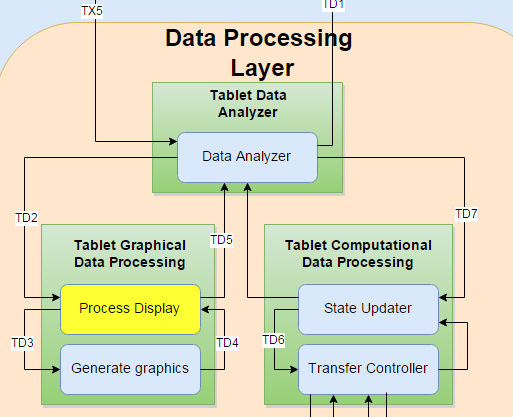


Figure 6‑: Process Display Module

**Prologue:** This module is responsible for taking data passed down from the Data Analyzer module and processing the display that will be passed back to the Data Analyzer module. Any graphics that need to be generated will be sent to the Generate Graphics module, which will send back the generated graphics.

**Interfaces:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Sink** | **Input to Sink** | **Return from Sink** |
| Process Display | Generate Graphics | Graphic ID (int) | None |
| Process Display | Data Analyzer | Android Display (Android Activity) | None |
| Data Analyzer | Process Display | Formatted data for graphical processing (Android Activity) | None |
| Generate Graphics | Process Display | Graphic (image) | None |

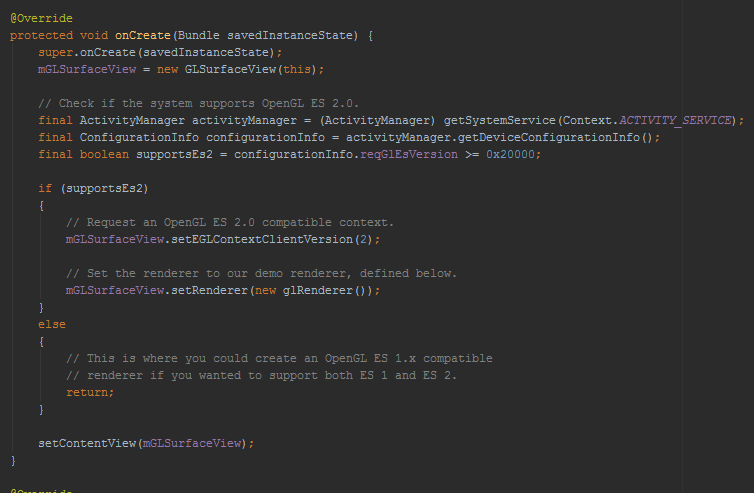
Table 6‑: Process Display Interfaces

**External Data Dependencies: N/A**

**Internal Data Dependencies:** Android API 17

**Object Descriptions: N/A**

**Pseudo Code:**

****

**Quality Assurance:** This module will be tested by processing data for a new display.

#### 6.2.2.2 Generate Graphics

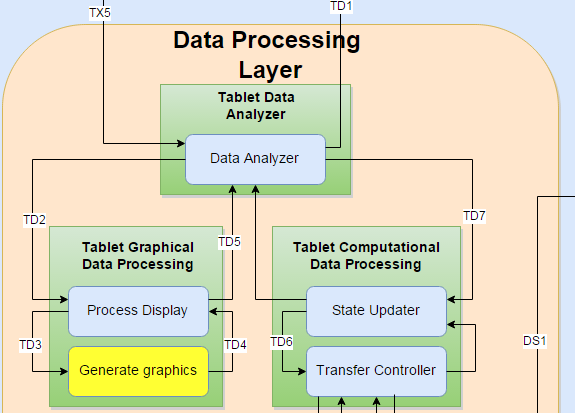


Figure 6‑: Generate Graphics Module

**Prologue:** This module is responsible for generating the graphics to be displayed to the user. The Process Display module will request a graphic by id, and this module will generate it and send the graphic back to the Process Display Module.

**Interfaces:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Sink** | **Input to Sink** | **Return from Sink** |
| Process Display | Generate Graphics | Graphic id (int) | None |
| Generate Graphics | Process display | Graphic (image) | none |

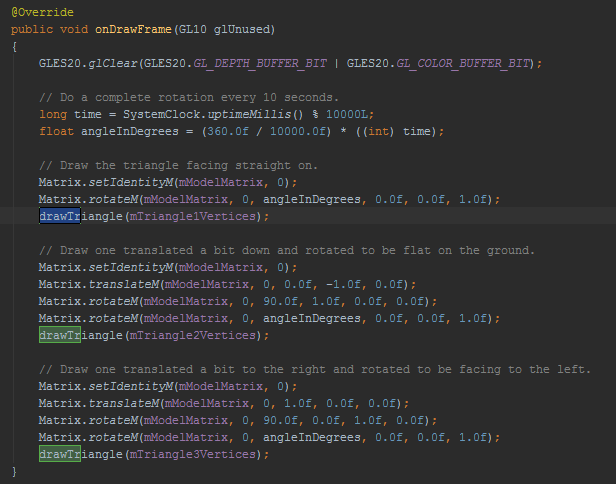
Table 6‑: Generate Graphics Interfaces

**External Data Dependencies:** Valid graphic id

**Internal Data Dependencies:** Android API 17

**Object Descriptions: N/A**

**Pseudo Code:**

****

**Quality Assurance:** This module will be tested by generating a graphic image and passing it onto the Process Display module.

### 6.2.3 Tablet Computational Data Processing

This subsystem will process all non-graphical data, make requests to store or retrieve data in the database, and send data to the Transfer Super Layer.

#### 6.2.3.1 State Updater

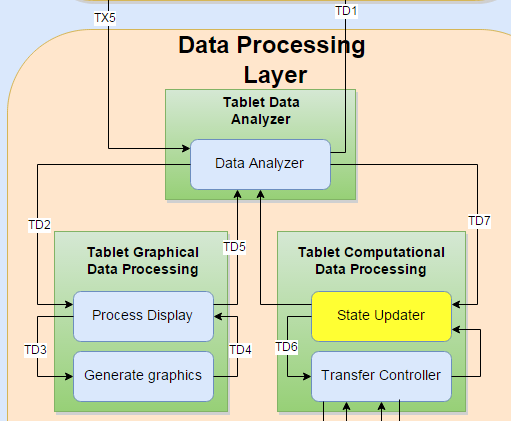


Figure 6‑: State Updater Module

**Prologue:** This module is responsible for updating the state of the application and processing any data received from the Data Analyzer module. Once the data is processed and the state is updated the data will be passed back to the Data Analyzer module. Any data that needs to be sent to the PixelSense table via the Lynx will be passed onto the Transfer Controller module.

**Interfaces:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Sink** | **Input to Sink** | **Return from Sink** |
| State updater | Transfer Controller | Processed data to transfer (String, int) | None |
| Data Analyzer | State Updater | Data to be processed  (String, int) | None |
| State Updater | Data Analyzer | Processed data  (String, int) | None |

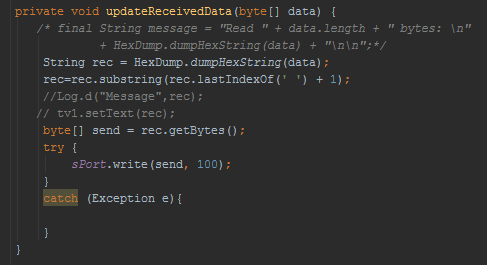
Table 6‑: State Updater Interfaces

**External Data Dependencies: N/A**

**Internal Data Dependencies:** Android API 17

**Object Descriptions: N/A**

**Pseudo Code:**

****

**Quality Assurance:** This module will be tested by processing data and updating the state of the application.

#### 6.2.3.2 Transfer Controller

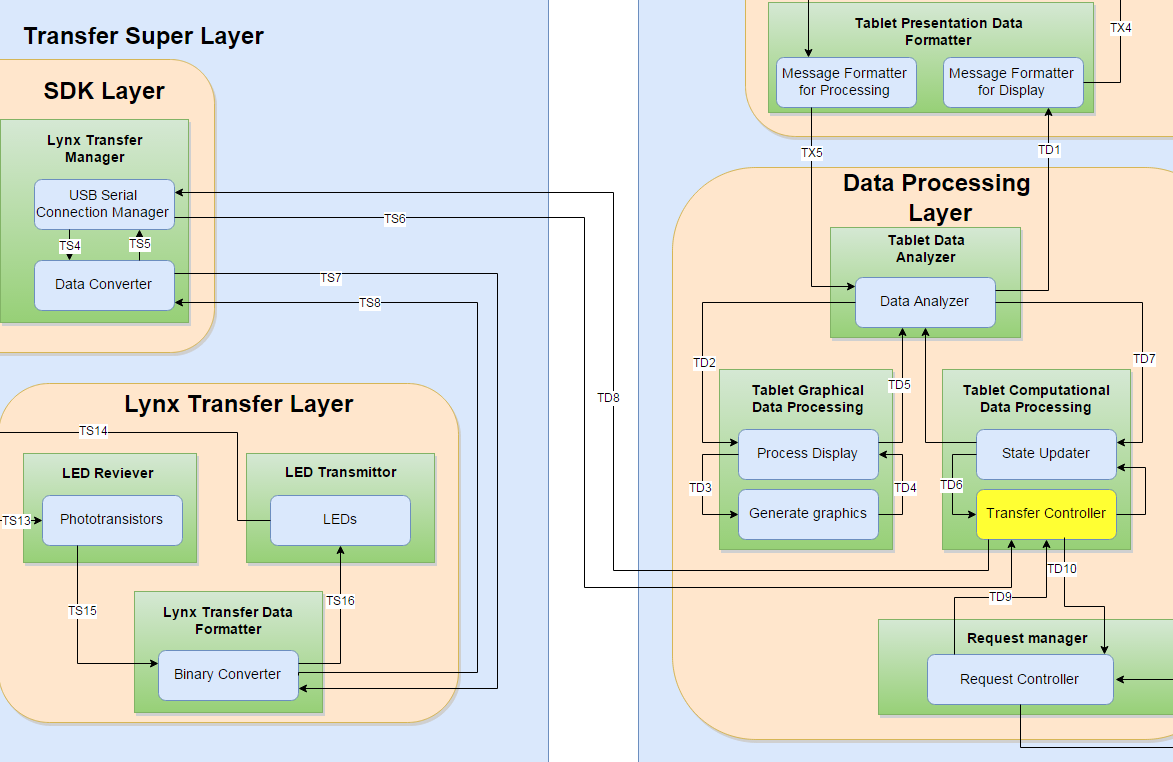


Figure 6‑: Transfer Controller Module

**Prologue:** This module is responsible for sending data that needs to be transferred to the PixelSense to the Transfer Layer. This module also requests to store the data to be transferred or data received in the database by making a request via the Request Controller module.

**Interfaces:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Sink** | **Input to Sink** | **Return from Sink** |
| Transfer Controller | USB Serial Connection Manager | Data to be transferred  (String, int) | None |
| Transfer Controller | State Updater | Processed data  (String, int) | None |
| Transfer Controller | Request Controller | Request to store/retrieve data  (String) | None |
| USB Serial Connection Manager | Transfer Controller | Transferred data  (String, int) | None |
| State Updater | Transfer Controller | Data to be transferred (String, int) | None |
| Request Controller | Transfer Controller | Retrieved data  (String, int) | None |

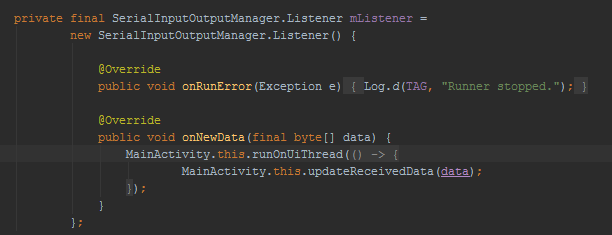
Table 6‑: Transfer Controller Interfaces

**External Data Dependencies: N/A**

**Internal Data Dependencies:** Android API 17

**Object Descriptions: N/A**

**Pseudo Code:**

****

**Quality Assurance:** This module will be tested by sending and receiving data to the USB Serial Connection Manager.

### 6.2.4 Request Manager

This subsystem handles requests for data stored in the database, sending requests to the Data Storage Layer.

#### 6.2.4.1 Request Controller

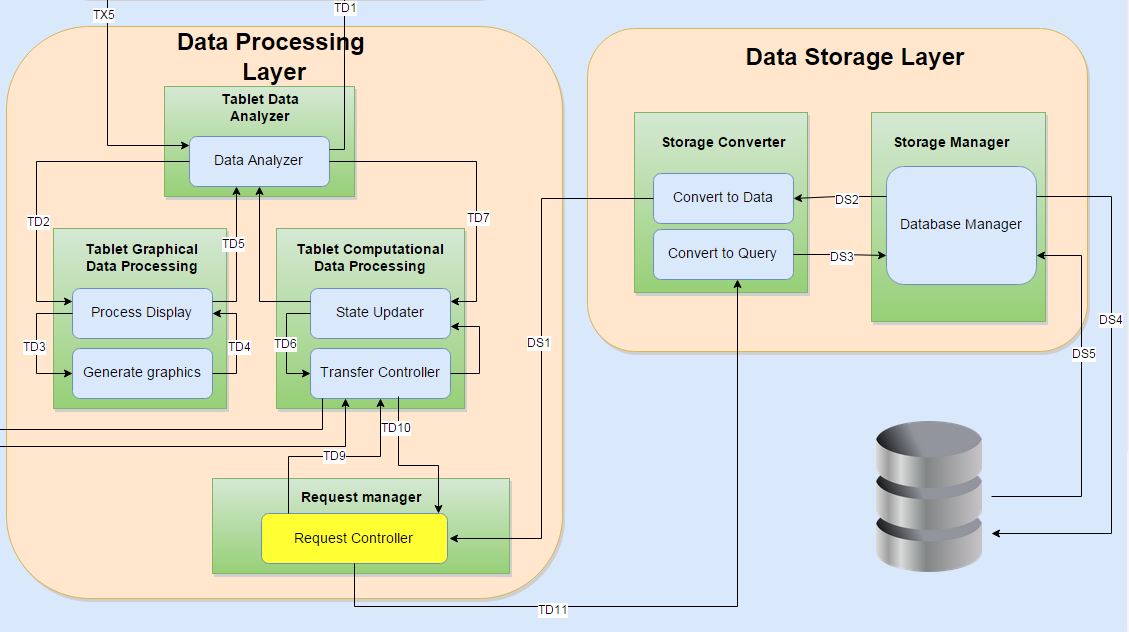


Figure 6‑: Request Controller Module

**Prologue**: This module is responsible for taking requests, from the Transfer Controller module, to store or retrieve data and passing them along to the Convert to Query. If there was a request to retrieve data this module will receive that data from the Convert to Data module, and then pass that data to the Transfer Controller module.

**Interfaces:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Sink** | **Input to Sink** | **Return from Sink** |
| Request Controller | Convert to Query | Request (String) | None |
| Request Controller | Transfer Controller | Request (String) | None |
| Convert to Data | Request Controller | Requested data (String) | None |
| Transfer Controller | Request Controller | Requested data (String) | None |

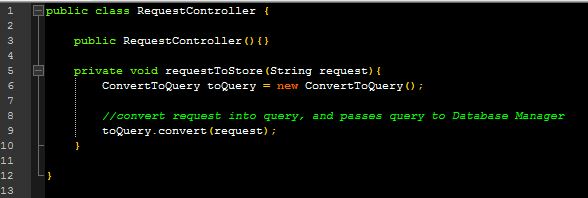
Table 6‑: Request Controller Interfaces

**External Data Dependencies: N/A**

**Internal Data Dependencies:** Android API 17

**Object Descriptions:** **N/A**

**Pseudo Code:**

****

**Quality Assurance:** This module will be tested by transferring requests to convert to Query module, and receiving data from Convert to Data module.

## 6.3 Data Storage Layer

The purpose of this layer is to manage and store transaction logs. This layer will receive a request from the Data Processing layer, then send that request on the Storage Manager Layer which will return the relevant information to the Storage Conversion Layer, which will convert the data and send it to the Data Processing Layer.

### 6.3.1 Storage Converter

This subsystem converts requests into queries which will be sent to the Storage Manager subsystem, and it converts the results of executed queries into relevant data to be sent back to the Request Manager Subsystem.

#### 6.3.1.1 Convert to Data

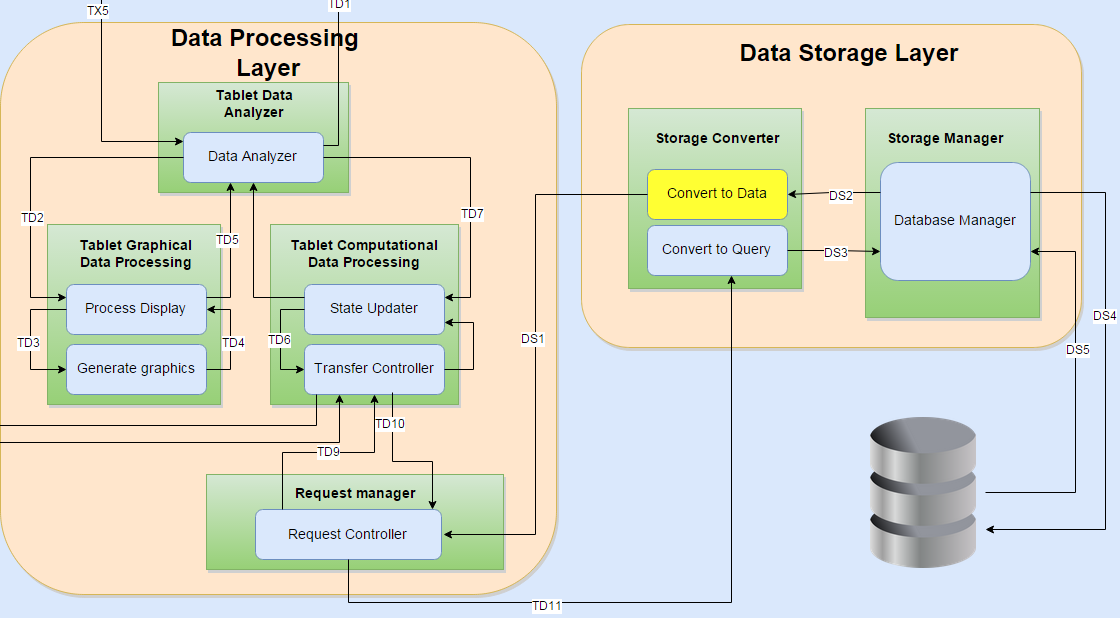


Figure 6‑: Convert to Data Module

**Prologue:** This module is responsible for taking the results of a query received from the Database Manager module and converting or formatting it into relevant data. Once the data has been converted it will be passed to the Request Controller module.

**Interfaces:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Sink** | **Input to Sink** | **Return from Sink** |
| Database Manager | Convert to Data | Result of query (ResultSet) | None |
| Convert to Data | Request Controller | Requested data (String) | None |

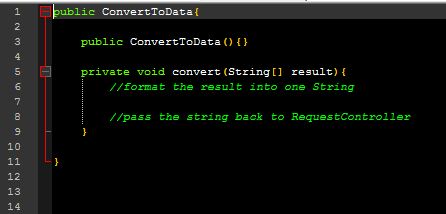
Table 6‑: Convert to Data Interfaces

**External Data Dependencies: N/A**

**Internal Data Dependencies:** Android API 17, SQLite

**Object Descriptions: N/A**

**Pseudo Code:**

****

**Quality Assurance:** This module will be tested by passing in the results of a executed query and converting it into formatted data.

#### 6.3.1.1 Convert to Query



Figure 6‑: Convert to Query Module

**Prologue:** This module is responsible for taking the request received from the Request Manager module, converting that request into a SQLite query and passing that query onto the Database Manager module.

**Interfaces:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Sink** | **Input to Sink** | **Return from Sink** |
| Request Controller | Convert to Query | Request to store/retrieve (String) | None |
| Convert to Query | Database Manager | SQLite Query (String) | None |

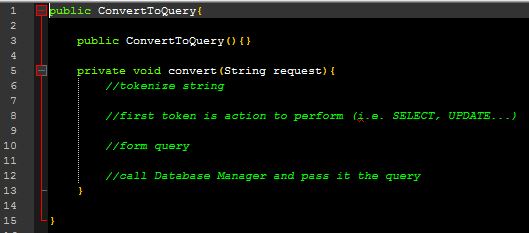
Table 6‑: Convert to Query Interfaces

**External Data Dependencies: N/A**

**Internal Data Dependencies:** Android API 17**,** SQLite

**Object Descriptions: N/A**

**Pseudo Code:**

****

**Quality Assurance:** This module will be tested by converting a request into a SQLite query.

### 6.3.2 Storage Manager

This subsystem connects to the database, executes the queries received from the Storage Converter subsystem and returns the results of the query to the Storage Converter subsystem.

#### 6.3.2.1 Database Manager

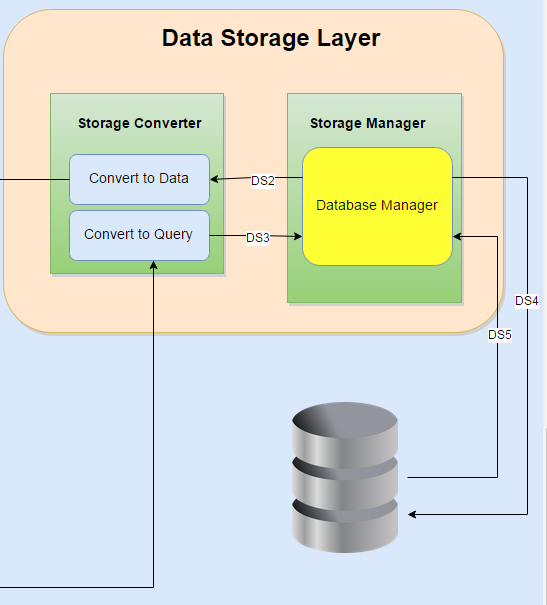


Figure 6‑: Database Manager Module

**Prologue:** This module is responsible for connecting to the SQLite database, executing the query received from the Convert to Query module, and then pass the results of that query onto the Convert to Data module.

**Interfaces:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Sink** | **Input to Sink** | **Return from Sink** |
| Database Manager | Convert to Data | Result from query (ResultSet) | None |
| Database Manager | SQLite Database | Query (String) | None |
| Convert to Query | Database Manager | SQLite query (String)) | None |
| SQLite Database | Database Manager | Result from query (ResultSet) | None |

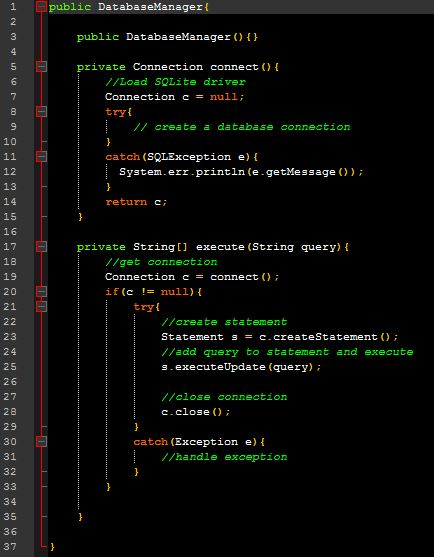
Table 6‑: Database Manager Interfaces

**External Data Dependencies:** SQLite

**Internal Data Dependencies:** Android SDK

**Object Descriptions: N/A**

**Pseudo Code:**

****

**Quality Assurance:** This module will be tested by connecting to a database and executing a test query.

# 7. Transfer Super Layer

The purpose of this super layer is to enclose and distinguish between the two internal layers. The main function is to show the transfer between the PixelSense table and the tablet/lynx device as well as the dataflow between the two internal layers. Having a super layer is useful showing the different parts of both systems as well as showing them as one system as a whole.

## 7.1 SDK Layer

The purpose of this layer is to convert the computational data from both the tablet side and PixelSense side to binary data over a protocol where it can be sent over optics as well as converting the data received to something the PixelSense and tablet will both understand.

### 7.1.1 PixelSense Transfer

This subsystem is responsible for transferring data back and forth between the Lynx device and the PixelSense table.

#### 7.1.1.1 Graphics Transfer Connector

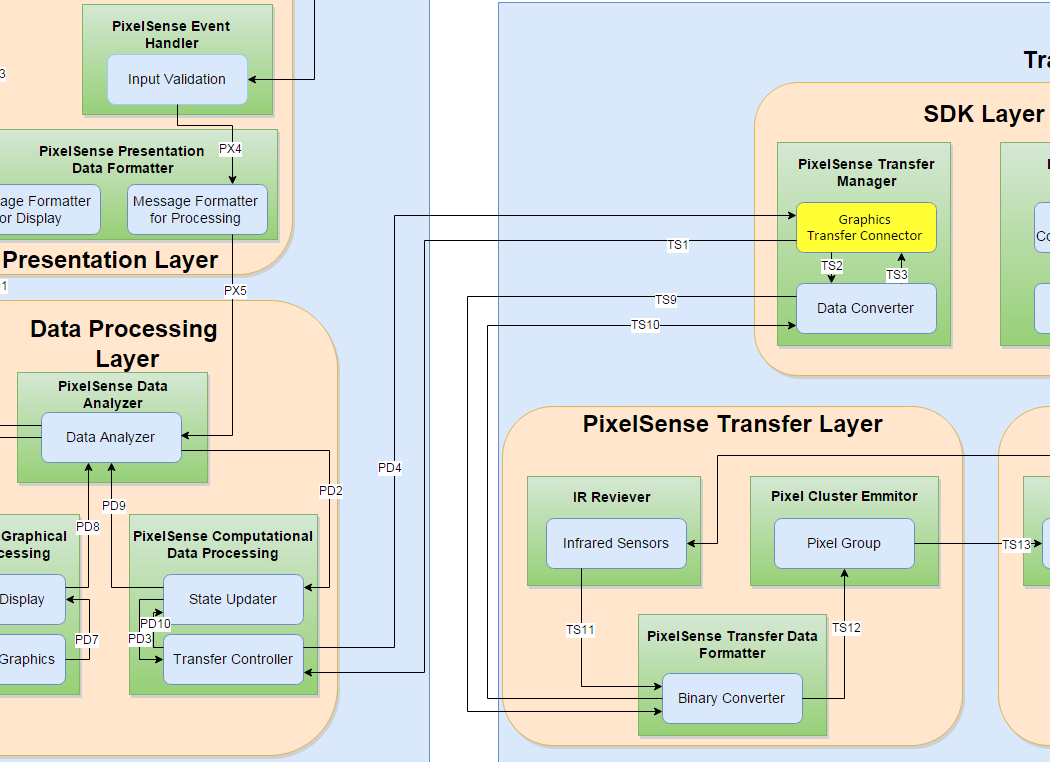


Figure 7‑1: Graphics Transfer Connector Module

**Prologue:** The Graphics Transfer Connector will listen for data from the CPU and pass that data to the Graphics processor to be converted to graphical arrays. The Bus also transfers data the other way and listens for touch and Infrared input from the graphics processor and transfers it to the CPU for further processing.

**Interfaces:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Sink** | **Input to Sink** | **Return from Sink** |
| Transfer Controller | Graphics Transfer Connector | CPU function call to refresh screen | N/A |
| Graphics Transfer Connector | Transfer Controller | Graphic images to display | N/A |
| Graphics Transfer Connector | Data Converter | Graphics locations and values | N/A |
| Data Converter | Graphics Transfer Connector | Graphics Pixel Array | N/A |

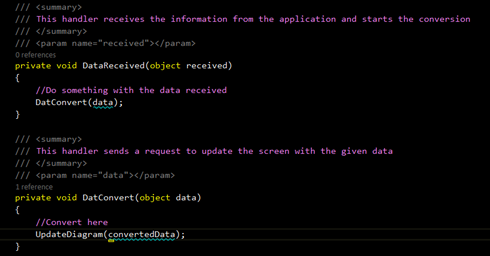
Table 7‑1: Graphics Transfer Connector Interfaces

**External Data Dependencies:** Pixel location and values on which groups to update.Graphics array input that corresponds to data input via the Lynx device.

**Internal Data Dependencies:** PixelSense XNA Framework**,** PixelSense SDK 2.0**,** AMD Catalyst Hardware data information

**Object Descriptions: N/A**

**Pseudo Code:**

****

**Quality Assurance:** The development team will send dummy data to be flashed on the screen in the sequence that we choose. The other test will be getting location and raw images of devices as well as touch point information from the screen. Doing this will validate that the module is indeed working as intended by sending and receiving data from the touch/display panel of the PixelSense table.

#### 7.1.1.2 Data Converter

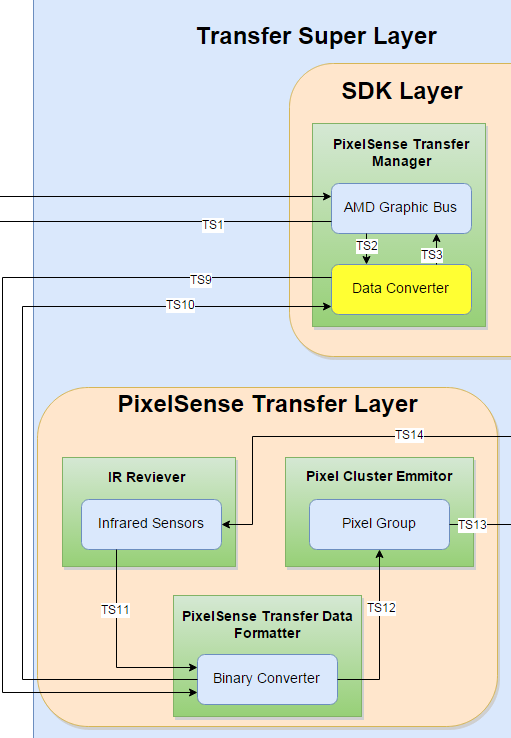


Figure 7‑2: Data Converter Module

**Prologue:** The Data converter will listen for from the Graphics Bus and convert it into graphical information such as pixel location and pixel values (pixel flashing). The data converter also converts the detected information from the infrared sensors and detects the state (received or not) of those certain locations that we predefine.

**Interfaces:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Sink** | **Input to Sink** | **Return from Sink** |
| Binary Converter | Data Converter | Raw Binary Pixel changes from IR | N/A |
| Data Converter | Binary Converter | Array of Pixel locations and RGB values | N/A |
| Data Converter | Graphics Transfer Connector | Array of Pixel locations and RGB values | N/A |
| Graphics Transfer Connector | Data Converter | Graphics locations and values | N/A |

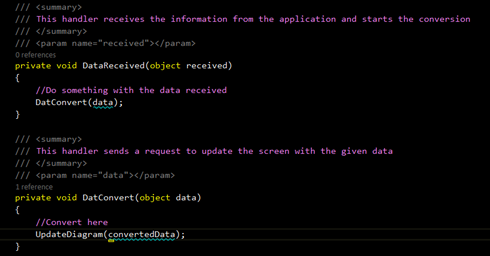
Table 7‑2: Data Converter Interfaces

**External Data Dependencies:** Pixel location and values on which groups to update.Graphics array input that corresponds to data input via the Lynx device.

**Internal Data Dependencies:** PixelSense XNA Framework**,** PixelSense SDK 2.0**,** AMD Catalyst Hardware data information

**Object Descriptions: N/A**

**Pseudo Code:**

****

**Quality Assurance:** To test this the team will send a predefined array to the post function and see if it will update the screen. The other test would be to touch certain parts of the screen and see if we can get pixel data back (location and state changes).

### 7.1.2 Lynx Transfer

This subsystem is responsible for transferring data back and forth between the Lynx device and the Android tablet.

#### 7.1.2.1 USB Serial Connection Manager

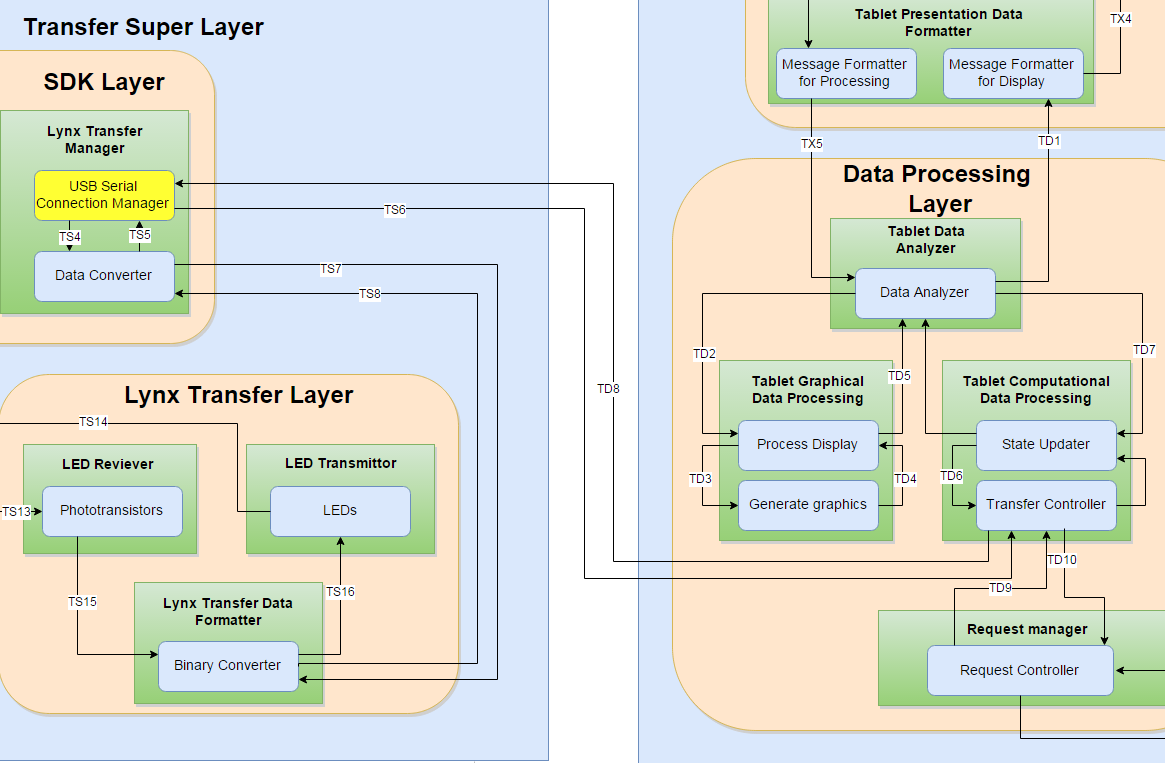


Figure 7‑3: USB Serial Connection Manager Module

**Prologue:** The USB Serial Connection Manager will listen for data from the Android tablet and pass that data to the Arduino microprocessor to convert it into a sequence for transmission. The Manager also transfers data the other way and listens for state changes from the phototransistor and passes it back to the Android Tablet.

**Interfaces:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Sink** | **Input to Sink** | **Return from Sink** |
| USB Serial Connection Manager | Data Converter | Received Byte Array | N/A |
| Data Converter | USB Serial Connection Manager | Byte Array | N/A |
| USB Serial Connection Manager | Transfer Controller | ASCII Characters | N/A |
| Transfer Controller | USB Serial Connection Manager | ASCII Characters | N/A |

Table 7‑3: USB Serial Connection Manager Interfaces

**External Data Dependencies: N/A**

**Internal Data Dependencies:** Android API 17**,** Arduino Bootloader**,** USB Serial Connection

**Object Descriptions: N/A**

**Pseudo Code:**

****

**Quality Assurance:** The team has decided to test this by sending numbers and strings over serial and see if the flash the LEDs accordingly. To compliment this the team will also send some input from the Arduino and see if the android device can read the input. If the Android can read the correct input and the Arduino flashes the LEDs accordingly then we can be sure that the module is functioning correctly.

#### 7.1.2.2 Data Converter

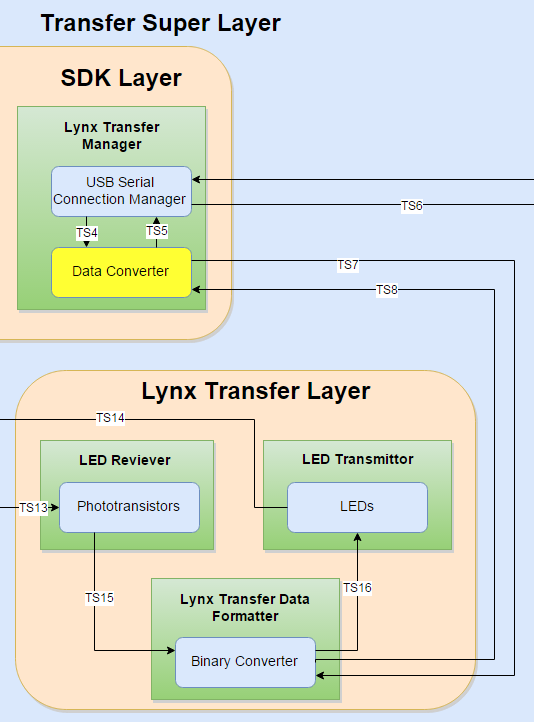
****

Figure 7‑4: Data Converter Module

**Prologue:** The Data converter will listen for data from the USB Serial Connection Manager and convert it readable data. The data converter also converts the detected information from the photo transistors and detects the state (received or not) of those certain locations that we predefine.

**Interfaces:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Sink** | **Input to Sink** | **Return from Sink** |
| Binary Converter | Data Converter | Raw Binary values detected | N/A |
| Data Converter | Binary Converter | ASCII Sequence Array | N/A |
| Data Converter | USB Serial Connection Manager | Byte Array | N/A |
| USB Serial Connection Manager | Data Converter | Byte Array | N/A |

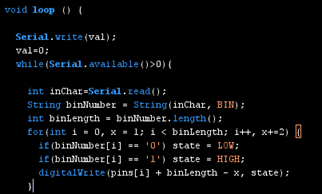
Table 7‑4: Data Converter Interfaces

**External Data Dependencies: N/A**

**Internal Data Dependencies:** Arduino Serial

**Object Descriptions: N/A**

**Pseudo Code:**

****

**Quality Assurance:** To test this the team will see if the data sent back and forth are the same. For example if we send the word “BET” from the android side it should receive the word “BET” on the Arduino side and vice versa.

## 7.2 PixelSense Transfer Layer

The purpose of this layer is to transfer (send/receive) strictly through the PixelSense side. Here the layer will receive data from the PixelSense data processing layer, format the data and emit it through some pixel cluster configuration. Going the other way, the table will detect certain light sequences through its IR receiver, format it to binary data and then send it off to the PixelSense data processing layer for interpretation.

### 7.2.1 PixelSense Transfer Data Formatter

This subsystem is responsible to format the data appropriately so that the data can be understood by the next subsystem, the main priority is for the data to be converted to and from binary.

#### 7.2.1.1 Binary Converter

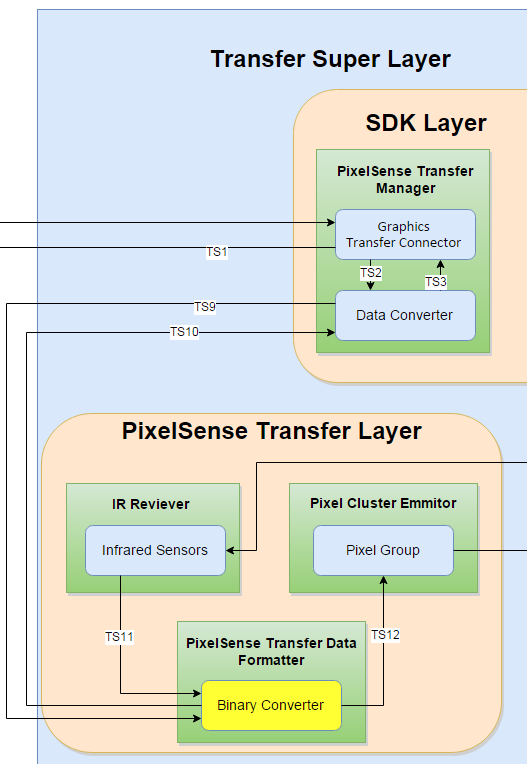


Figure 7‑5: Binary Converter Module

**Prologue:** The purpose of this module is to convert the data coming in and out to binary or from binary. This module will be the final module between the hardware receiving and transmitting modules so it will end up formatting the data so that either the Pixel Group or the data converter will understand properly.

**Interfaces:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Sink** | **Input to Sink** | **Return from Sink** |
| Binary Converter | Data Converter | Touch meta information | N/A |
| Infrared Sensors | Binary Converter | Raw touch data | N/A |
| Data Converter | Binary Converter | Pixel location information | N/A |
| Binary Converter | Pixel Group | Binary data for pixel placement | N/A |

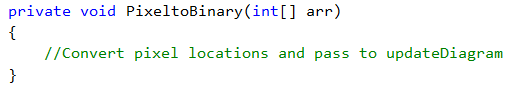
Table 7‑5: Binary Converter Interfaces

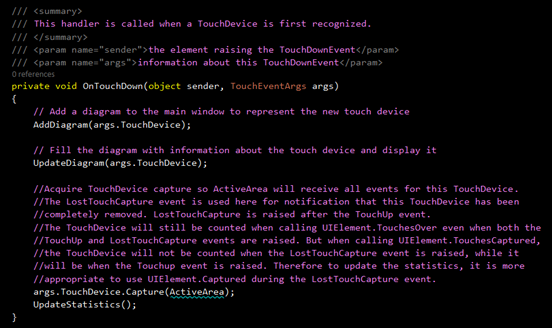
**External Data Dependencies: N/A**

**Internal Data Dependencies:** PixelSense XNA Framework**,** PixelSense SDK 2.0**,** AMD Catalyst Hardware data information

**Object Descriptions: N/A**

**Pseudo Code:**

****

****

**Quality Assurance:** To test this module the team will send different data types (integer, string, etc.) and send it to be converted to binary, if the screen flashes the pixel groups in a specific sequence then we will know that the module is working.

### 7.2.2 Pixel Cluster Emitter

The purpose of this module is to flash specific parts of the screen in a sequence that the Lynx device will understand.

#### 7.2.2.1 Pixel Group

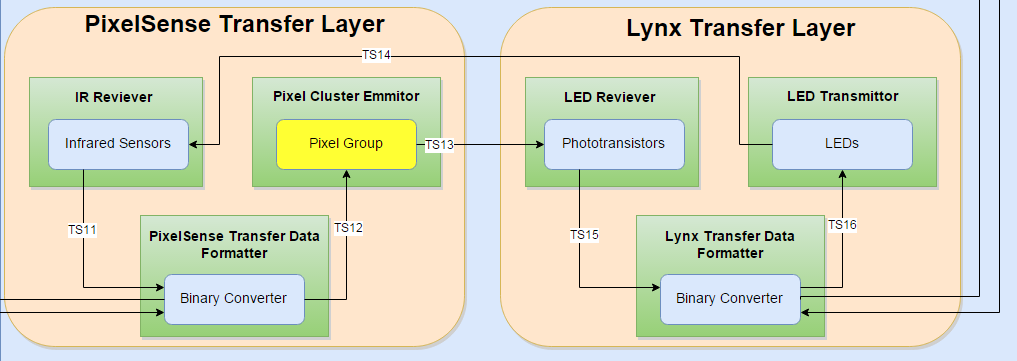


Figure 7‑6: Pixel Group Module

**Prologue:** This module will flash a group of pixels in a sequence that the Lynx device will understand. It will get a certain sequence from the Binary converter and execute it so that the Pixel Group will flash in a way that the phototransistors will be able to detect.

**Interfaces:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Sink** | **Input to Sink** | **Return from Sink** |
| Binary Converter | Pixel Group | Binary Pixel Information | N/A |
| Pixel Group | Phototransistors | Light Sequence | N/A |

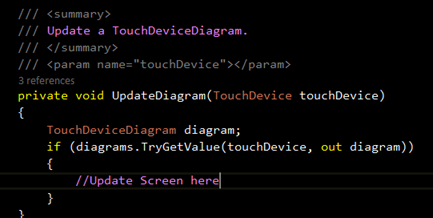
Table 7‑6: Pixel Group Interfaces

**External Data Dependencies: N/A**

**Internal Data Dependencies:** PixelSense XNA Framework**,** PixelSense SDK 2.0**,** AMD Catalyst Hardware data information

**Object Descriptions: N/A**

**Pseudo Code:**

****

**Quality Assurance:** The team has decided to test this by sending some dummy strings, if the screen flashes the certain areas in proper sequence then the module will be validated and check working.

### 7.2.3 IR Receiver

This subsystem is responsible for getting the raw input from the Infrared grid in the PixelSense.

#### 7.2.3.1 Infrared Sensor

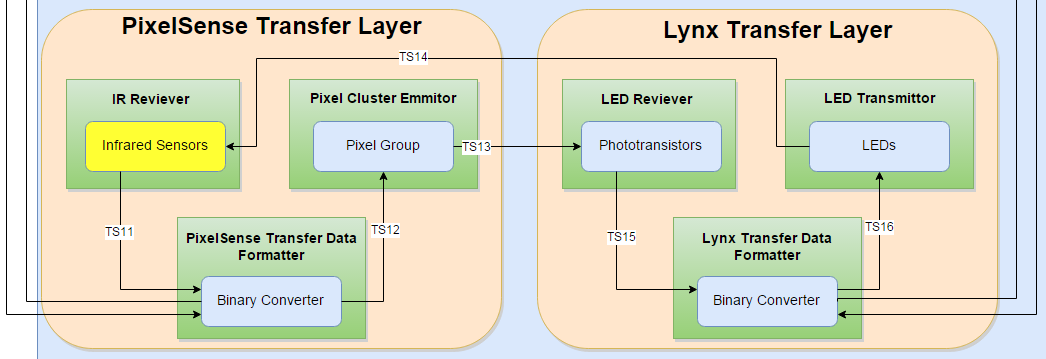


Figure 7‑7: Infrared Sensor Module

**Prologue:** The infrared sensor will detect the LEDs from the transmitter so that data can be transferred over. It will take the raw input (location and sequence) and pass it on to the Binary Converter for further formatting so the data can be understood than just being on or off states.

**Interfaces:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Sink** | **Input to Sink** | **Return from Sink** |
| LEDs | Infrared Sensors | Light Sequence | N/A |
| Infrared Sensors | Binary Converter | Raw Binary data | N/A |

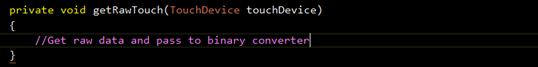
Table 7‑7: Infrared Sensor Interfaces

**External Data Dependencies:** **N/A**

**Internal Data Dependencies:** PixelSense XNA Framework**,** PixelSense SDK 2.0**,** AMD Catalyst Hardware data information

**Object Descriptions: N/A**

**Pseudo Code:**

****

**Quality Assurance:** To test this the team has decided to flash some LEDs in some sequence (such as 65 in binary to represent ‘A’) and see if the table sees the ‘A’ as well. The value flashed will be arbitrary, and if it passes multiple successful translations then the module will be noted as complete and working.

## 7.3 Lynx Transfer Layer

The purpose of this layer is to transfer (send/receive) strictly through the Lynx side. Here the layer will be receiving data from the tablet’s processing layer, it will format it and then emit it through a certain light sequence via LEDs. Going the other way, the Lynx will detect light sequences through LED light detection. It will then format it to binary data and transfer it to the tablet’s data processing layer.

### 7.3.1 LED Receiver

The LED Receiver is an array of phototransistors that will detect light changes on the PixelSense display panel. It is basically the receiving end of the data transmission stream.

#### 7.3.1.1 Phototransistors

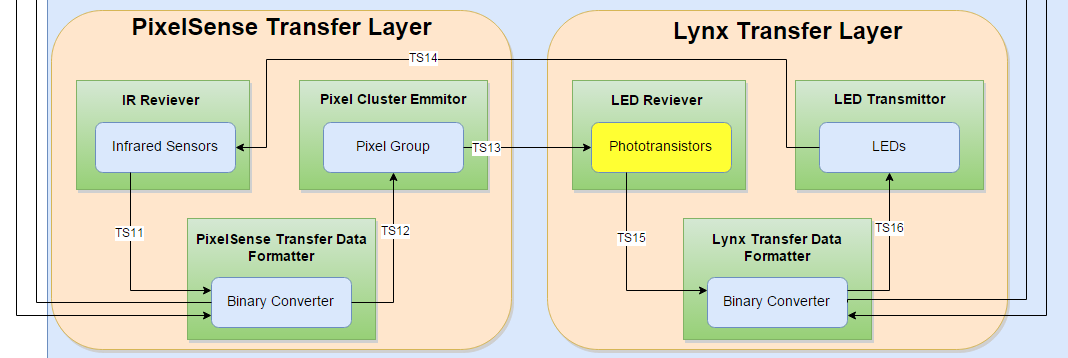


Figure 7‑8: Phototransistor Module

**Prologue:** This module is responsible for detecting light changes and it will then transmit the detected state (on or off) to the binary converter. The module has no other job and will only function has a receiving end of the data transfer process.

**Interfaces:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Sink** | **Input to Sink** | **Return from Sink** |
| Pixel Group | Phototransistors | Light | N/A |
| Phototransistors | Binary Converter | Binary States | N/A |

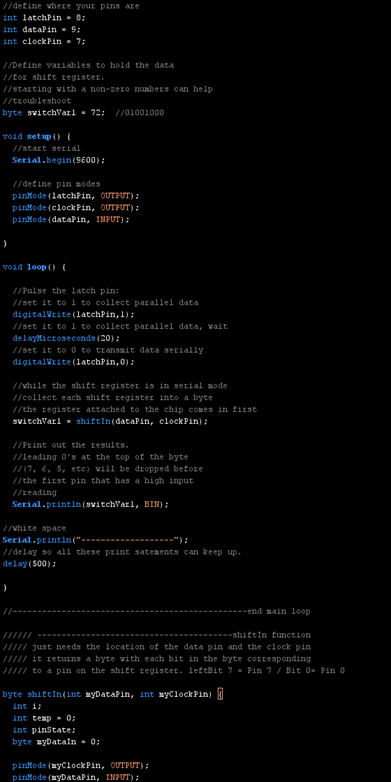
Table 7‑8: Phototransistor Interfaces

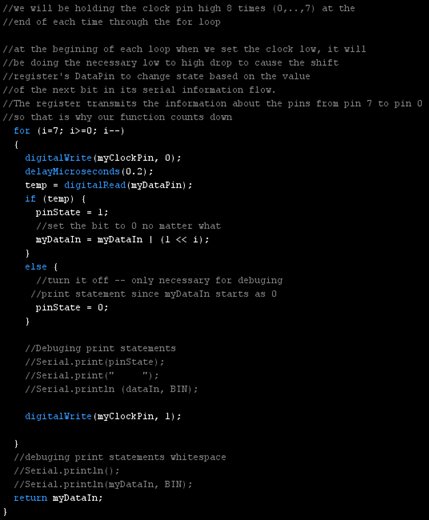
**External Data Dependencies: N/A**

**Internal Data Dependencies:** Arduino ADK

**Object Descriptions: N/A**

**Pseudo Code:**

****

****

**Quality Assurance:** To test this, the team has decided to have an 8 position dip switch and send different states through the CD401B chip and if the values come up correct then we will know the code is working properly. The next step would be to physically use the phototransistors to replicate the process. Once this succeeds then we can be sure that the module is working as intended.

### 7.3.2 LED Transmitter

The LED transmitter subsystem is responsible for sending data through a grid of LEDs. This will work by receiving binary data and will flash LEDs accordingly based on the binary sequence.

#### 7.3.2.1 LEDs

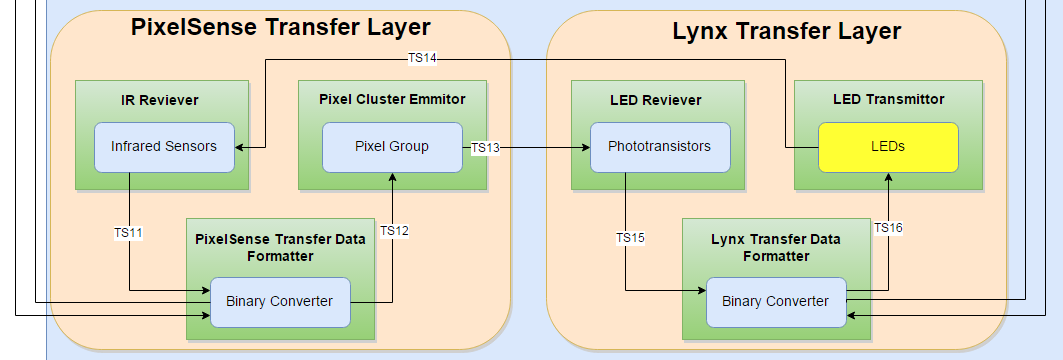


Figure 7‑9: LED Module

**Prologue:** This module is an array of LEDs which will flash in a certain binary sequence provided by the Binary Converter. So once it receives certain values it will flash our array in an 8-bit sequence with a small delay in the middle to distinguish different letters and numbers.

**Interfaces:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Sink** | **Input to Sink** | **Return from Sink** |
| LEDs | IR Receiver | Light | N/A |
| Binary Converter | LEDs | Binary Sequence | N/A |

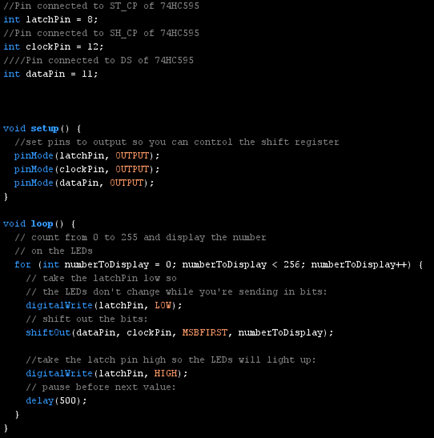
Table 7‑9: LED Interfaces

**External Data Dependencies: N/A**

**Internal Data Dependencies:** Arduino ADK

**Object Descriptions: N/A**

**Pseudo Code:**

****

**Quality Assurance:** To test this the team has decided to send specific strings over a serial connection and see of the LED array will flash them in a binary sequence that we specify.

### 7.3.3 Lynx Data Formatter

This subsystem is responsible to format the data appropriately so that the data can be understood by the next subsystem, the main priority is for the data to be converted to and from binary.

#### 7.3.3.1 Binary Converter

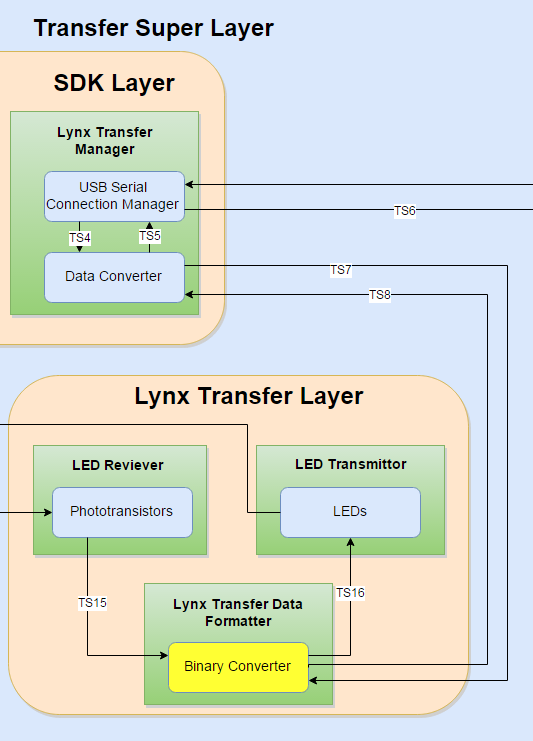


Figure 7‑10: Binary Converter Module

**Prologue:** The purpose of this module is to convert the data coming in and out to binary or from binary. This module will be the final module between the hardware receiving and transmitting modules so it will end up formatting the data so that either the Pixel Group or the data converter will understand properly.

**Interfaces:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Sink** | **Input to Sink** | **Return from Sink** |
| Phototransistors | Binary Converter | Light sequence | N/A |
| Binary Converter | LEDs | Binary sequences | N/A |
| Binary Converter | Data Converter | ASCII Array | N/A |
| Data Converter | Binary Converter | ASCII Array | N/A |

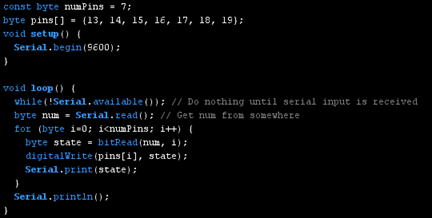
Table 7‑10: Binary Converter Interfaces

**External Data Dependencies: N/A**

**Internal Data Dependencies:** Arduino ADK

**Object Descriptions: N/A**

**Pseudo Code:**

****

**Quality Assurance:** To test this the team has decided to give it some string or integer and see if a binary sequence is outputted to the console and/or flashed onto the LED array that we have setup.

# 8. Quality Assurance

This section details the testing requirements to assure the correctness of the Lynx system. The plan involves the use of specific testing modules, to be incorporated as needed throughout the construction process. This involves basic unit testing through final implementation testing. System Verification will be handled by a list of abstract testing criteria, outside of any specific module.

## 8.1 Testing Modules

The testing modules detailed below will be used in the majority of the testing process. These modules are designed to be included in the code of any target module as a single function call that archives the change of data from the beginning of a module through the end of the module’s process. Multiple modules may be tested at once if they are called in a sequence.

### 8.1.1 Testing Data Logger

**Description:** The purpose of the Data Logger is to record the process of a module from its input, to its final output. The Data Logger keeps track of a persistent log file. A module calls the Data Logger, inputting its name, originally received input, and the data it intends to send to the next module in the communication chain. The Data Logger formats this input, along with additional meta-data (timestamp, and the function that called it) and appends it to a log file for later review.

**Input:** The input parameters passed will be a string of the function name that called it, a generic type parameter that conforms to the original input type of the calling function, and a generic type parameter that conforms the intended output of the calling function.

**Output:** This function returns nothing. It is only intended to append the formatted data from its input into a log file.

### 8.1.2 Testing Data Emulator

**Description:** The purpose of the Data Emulator is to allow for finer testing scenarios, and to reduce the reliance of the beginning of the data communication pipeline (the presentation layer modules). A module calls this function to replace the input normally supplied by a module that is not currently being tested. The Data Emulator accesses a file based on its file name, and returns the data stored in it to the calling function. The change of the data to the appropriate type is delegated to the calling function.

**Input:** The input is a string that represents a file name.

**Output:** This function returns a string representing the contents of the file.

## 8.2 Unit Testing

During unit testing, a module is isolated from the communication pipeline and is tested on its own. The module will be modified to call the Data Emulator, along with the code necessary to change the data from a string into the necessary data type. The change is placed before any computation begins. Appended to the end of a module, is a single call to the Data Logger, to record the changes made to the inputted data. Data is then visually inspected to deduce if the data was changed into the appropriate format. The function call to the next module is commented out during the test.

## 8.3 Component Testing

Component Testing will focus on the pipeline from the edges of a layer (modules with connection to other layers) to another, passing through the necessary modules in between. For each round of testing, the intended beginning must call the Data Emulator, to isolate itself from modules not being tested, and each module there-after must call the Data Logger (including the first module). The last module in the chain to be tested must comment out the next module in the chain.

## 8.4 Integration Testing

Integration testing proceeds in a similar manner to component testing, spanning layers instead of modules. Testing will include the layers of the PixelSense super layer in one test, and the Tablet/Lynx super layer in another, each with a separate log file locally stored on each system.

### 8.4.1 Super Layer Integration Testing

Integration testing will eventually combine the entire system under a single test, testing from the presentation modules of one system to the other. Since the Transfer super layer does not have an easily accessed file storage system, the entire layer will be tested at once during this phase. Changes to the data will be recorded in both log files (PixelSense log and the Table/Lynx log) since each step records both its input and output.

## 8.5 Test Cases

Test cases will comprise the testing of the entire system from an end user perspective. All cases mut be successfully completed before the project can be considered complete.

|  |  |
| --- | --- |
| Test Case | Expected Result |
| A Player can join a game of Black Jack | A Player is authenticated by the system, and relevant player data is transferred to the table. |
| A Player can play a round of Black Jack | A Player plays through a game of Black Jack, with inputs being sent the table, and receiving data output updates to chip count |
| A Player can leave the table | A Player disconnects his/her device, retaining the accurate state from the table, without data corruption |

Table 8‑1: Test Cases

# 9. Requirements Mapping

## 9.1 Requirements Traceability Matrix



## 9.2 Requirement Traceability Analysis

Based on the table above, we can determine that the design of our architecture satisfies all the key requirements given to us by our project sponsor. Because we are dealing with multiple systems, we have individual layers that share similar characteristics with other layers in different systems. This allows us to fulfill all the key requirements while still isolating unique functionality within the layers. Most of the key requirements touch the PixelSense Transfer and Lynx Transfer layers of the Transfer system, which shows us that these are the most crucial and involved layers. The Data Storage layer has little overlap with the key requirements, but is crucial for the creation of an Android application, required for storing information within the application. The requirements for “Software to show optical transfer” and “System support” cover all layers and subsystems. The software we develop must touch all subsystems in order to show the functionality of the systems, and the SDK must have functions and support for all the functionalities we create for these systems.

Each one of these layers are necessary for successful optical communication between the Lynx device and the PixelSense table. Our layers are independent; containing required functionality within subsystems in each system.

# 10. Acceptance Plan

This section will outline our plan to meet the acceptance criteria set by our project sponsor and the team at the beginning of the project. This plan will include how we will physically deliver the project, such as the packaging and the installation instructions. In addition, it will also lay out on the UAT and acceptance criteria needed to be met in order for the project to get accepted.

## 10.1 Packaging and Installation

The Lynx Transfer System will include the following items:

* 1 Lynx Transfer System (Already assembled)
* 1 Micro USB to OTG adaptor
* 1 USB Male A to Male B cord
* 1 Nexus 7

The Android software for Lynx will be provided on a USB stick as an APK file. If an Android device other than the provided Nexus 7 will be used, that device must support USB OTG through a Micro USB port, as well as Android API Level 17 or above.

The Surface 2.0 software, along with the SDK used to build the software for both Surface and Android, will be provided on a USB stick to be ran on the Samsung SUR40 table.

## 10.2 User Acceptance Testing

UAT, conducted by the project sponsor, will be performed on all software and hardware developed by the team. The plan for this testing will be outlined and discussed in our System Test Plan documentation.

## 10.3 Acceptance Criteria

In order for the Lynx project to be accepted, it must pass the following criterions that were agreed upon by the sponsor and the team. Each of the criteria listed below covers requirements that were deemed either high or critical priority by all parties involved.

* The device can securely transfer information to the PixelSense table using an array of sensors
  + Requirement 3.1 - Send Data Optically.
  + Requirement 4.7 - The system shall allow the Lynx to send and receive data
  + Requirement 4.8 - The system shall allow the PixelSense table to send and receive data
* The device can securely receive information from the PixelSense table using an array of sensors
  + Requirement 3.2 - Read Data Optically.
  + Requirement 4.7 - The system shall allow the Lynx to send and receive data
  + Requirement 4.8 - The system shall allow the PixelSense table to send and receive data
* An SDK is provided that can be used to program for the device created
  + Requirement 3.3 - All work done by the product involving optical communication should be compiled into a well-documented library
  + Requirement 4.1 - The system will support Android API 17 and Surface 2.0 platforms
* The transfer rate of the Lynx is at least 200 bits/sec
  + Requirement 5.1 – Minimum Transfer Rate
* The Lynx has a port that can be used to connect to a tablet
  + Requirement 3.4 – Device must have a serial port.
* The software is provided for the PixelSense table that demonstrates the connection between the Lynx and the table
  + Requirement 3.7 – Software will be built to show optical transfer protocol.
* Software developed for the PixelSense can detect the orientation of the Lynx
  + Requirement 4.9 - The system shall be able to determine the orientation of the Lynx
* Lynx Software developed for the PixelSense, Device, and Android Platform can detect when a Lynx device is present
  + Requirement 4.10 - The system shall notify the table that the Lynx is on it
  + Requirement 4.11 - The system shall authenticate that a valid Lynx device is placed on the PixelSense table.

# 11. Appendix

## 11.1 Java Libraries including JDBC

The system will use various external Java libraries that will help connect each different component with each other. This includes database, USB and external storage libraries which will be used by various modules in the system.

## 11.2 PixelSense SDK

The PixelSense application will be developed used the PixelSense 2.0 SDK with all the dependent frameworks such as the XNA framework and .NET framework. These libraries are crucial to the development of our product through accessing the PixelSense hardware and all low level function calls to certain components like the Infrared Sensors and Orientation Sensor.

## 11.3 Android SDK (Minimum API 17)

The android application will be using the Android SDK, the SDK will provide various libraries and function calls for accessing the hardware on the Android device. Such hardware would things including but not limited to the orientation sensor, USB Serial module and external storage solutions. The SDK is also the main development tool for creating the android application as we the team have elected to develop the application in a native android environment.

## 11.4 Arduino GPIO

The Arduino GPIO is a module in the Arduino ADK which will allow access to the Arduino’s input and output pins. There will be a total of 6 pins used, they will be 2 clock pins, 2 latch pins and 2 data pins. Each of these pins will connect to one of each shift registers (74HC595 and CD4021BE) and will send and receive data for data transmission.