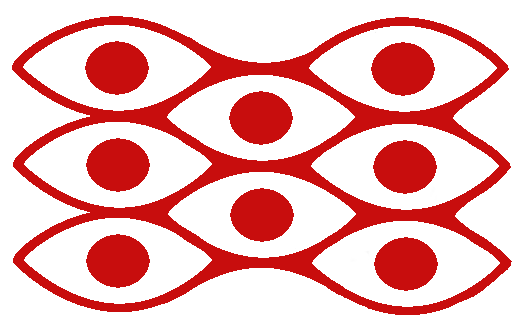
Department of Computer Science and Engineering  
The University of Texas at Arlington



**Architectural Design Document**

Team: Team Argus

Project: Lynx- PixelSense Secure Transfer System

Team Members:   
*Shamikul Amin*

*Brandon Deen*

*Brian Hasty*

*Keyur Patel*

*Seth Skocelas*

# Table of Contents

[Table of Contents i](#_Toc410296494)

[Document Revision History ii](#_Toc410296495)

[List of Figures iii](#_Toc410296496)

[List of Tables v](#_Toc410296497)

[1 Introduction 1](#_Toc410296498)

[2 Meta Architecture 4](#_Toc410296499)

[3 Layer Definitions 8](#_Toc410296500)

[4 PixelSense Super Layer 12](#_Toc410296501)

[5 Tablet/Lynx Super Layer 20](#_Toc410296502)

[6 Transfer Super Layer 32](#_Toc410296503)

[7 Inter-Subsystem Data Flows 44](#_Toc410296504)

[8 Requirements Mapping 52](#_Toc410296505)

[9 Operating System Dependencies 54](#_Toc410296506)

[10 Testing Considerations 55](#_Toc410296507)

# Document Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Revision Number | Revision Date | Description | Rationale |
| 0.5 | 12/2/2014 | ADS First Draft | Initial Creation of the Document |
| 1.0 | 1/29/2015 | ADS First Submission | Edits done to reflect comments received |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# List of Figures

[Figure 4‑1: PixelSense App UI 12](#_Toc410276820)

[Figure 4‑2: PixelSense Event Handler 13](#_Toc410276821)

[Figure 4‑3: PixelSense Data Formatter 14](#_Toc410276822)

[Figure 4‑4: PixelSense Data Analyzer 16](#_Toc410276823)

[Figure 4‑5: PixelSense Graphical Data Processing 17](#_Toc410276824)

[Figure 4‑6: PixelSense Computational Data Processing 18](#_Toc410276825)

[Figure 5‑1: Android App UI 20](#_Toc410276826)

[Figure 5‑2: Tablet Event Handler 21](#_Toc410276827)

[Figure 5‑3: Tablet Data Formatter 22](#_Toc410276828)

[Figure 5‑4: Tablet Data Analyzer 24](#_Toc410276829)

[Figure 5‑5: Tablet Graphical Data Processing 25](#_Toc410276830)

[Figure 5‑6: Tablet Computational Data Processing 26](#_Toc410276831)

[Figure 5‑7: Request Manager 27](#_Toc410276832)

[Figure 5‑8: Storage Converter 29](#_Toc410276833)

[Figure 5‑9: Storage Manager 30](#_Toc410276834)

[Figure 6‑1: IR Receiver 32](#_Toc410276835)

[Figure 6‑2: PixelSense Cluster Emitter 33](#_Toc410276836)

[Figure 6‑3: Transfer Data Formatter 34](#_Toc410276837)

[Figure 6‑4: LED Receiver 36](#_Toc410276838)

[Figure 6‑5: LED Transmitter 37](#_Toc410276839)

[Figure 6‑6: Lynx Data Formatter 38](#_Toc410276840)

[Figure 6‑7: PixelSense Transfer Manager 40](#_Toc410276841)

[Figure 6‑8: Lynx Transfer Manager 41](#_Toc410276842)

[Figure 7‑1: Dataflow Diagram 44](#_Toc410276843)

# List of Tables

[Table 1‑1: Key Requirements 3](#_Toc410212254)

[Table 2‑1: Assumptions & Design Implications 6](#_Toc410212255)

[Table 3‑1: Layer Definitions 8](#_Toc410212256)

[Table 3‑2: Architecture Diagram 11](#_Toc410212257)

[Table 4‑1: Public Interfaces (PixelSense App UI) 13](#_Toc410212258)

[Table 4‑2: Public Interfaces (PixelSense Event Handler) 14](#_Toc410212259)

[Table 4‑3: Interlayer Interfaces (PixelSense Data Formatter) 15](#_Toc410212260)

[Table 4‑4: Interlayer Interfaces (PixelSense Data Analyzer) 17](#_Toc410212261)

[Table 4‑5: Interlayer Interfaces (PixelSense Computational Data Processing) 19](#_Toc410212262)

[Table 5‑1: Public Interfaces (Android App UI) 21](#_Toc410212263)

[Table 5‑2: Public Interfaces (Tablet Event Handler) 22](#_Toc410212264)

[Table 5‑3: Interlayer Interfaces (Tablet Data Formatter) 23](#_Toc410212265)

[Table 5‑4: Interlayer Interfaces (Tablet Data Analyzer) 25](#_Toc410212266)

[Table 5‑5: Interlayer Interfaces (Tablet Computational Data Processing) 27](#_Toc410212267)

[Table 5‑6: Interlayer Interfaces (Request Manager) 28](#_Toc410212268)

[Table 5‑7: Interlayer Interfaces (Storage Converter) 29](#_Toc410212269)

[Table 5‑8: Public Interfaces (Storage Manager) 31](#_Toc410212270)

[Table 6‑1: Interlayer Interfaces (IR Receiver) 33](#_Toc410212271)

[Table 6‑2: Interlayer Interfaces (PixelSense Cluster Emitter) 34](#_Toc410212272)

[Table 6‑3: Interlayer Interfaces (PixelSense Transfer Data Formatter) 35](#_Toc410212273)

[Table 6‑4: Interlayer Interfaces (LED Receiver) 37](#_Toc410212274)

[Table 6‑5: Interlayer Interfaces (LED Transmitter) 38](#_Toc410212275)

[Table 6‑6: Interlayer Interfaces (Lynx Data Formatter) 39](#_Toc410212276)

[Table 6‑7: Interlayer Interfaces (PixelSense Transfer Manager) 41](#_Toc410212277)

[Table 6‑8:Interlayer Interfaces (Lynx Transfer Manager) 42](#_Toc410212278)

[Table 7‑1: Dataflow Definitions 50](#_Toc410212279)

[Table 7‑2: Producer-Consumer Relationship 51](#_Toc410212280)

[Table 8‑1: Requirements Traceability Matrix 53](#_Toc410212281)

# 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 the 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.

## 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 physically or wirelessly, 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 to 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 the 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 a poker chip counter, securely storing your chips so you can move from table to table in order to play different casino games.

## Key Requirements

|  |  |  |
| --- | --- | --- |
| **Requirement No.** | **Name** | **Description** |
| 3.1 | Lynx - Send data optically | The product must be able to send specifically timed flashes of light as bits of information and send them in a way a receiving unit can interpret it. |
| 3.2 | Lynx – Read data optically | The product must be able to receive specifically timed flashes of light as bits of information and interpret them correctly, storing the data if necessary. |
| 3.3 | Optical Communication SDK | A suite of functions designed to handle optical communication between capable hardware will be available in a portable SDK library. |
| 3.4 | Lynx - Serial port to transfer data collected | The Lynx will have a serial port to transfer data optically collected to the Android device connected. |
| 3.7 | Software to show optical transfer | A program will be supplied to demonstrate the capabilities of the optical communication SDK, and Lynx. The program will play a game of Blackjack, with the device holding chip information. |
| 4.1 | System support | An SDK will be provided for both Android API 17 (4.2 and up) and Microsoft Surface 2.0 (PixelSense) platforms. An SDK will also be provided for the Lynx device itself. |
| 4.7 | Lynx SDK – Send and receive data | The Device and Android SDK will provide functions that will allow the Android application developed with it to send and receive data using the Lynx device. |
| 4.8 | PixelSense SDK – Send and receive data | The PixelSense SDK will provide functions that will allow the PixelSense application developed with it to send to and receive data from the Lynx device. |
| 4.9 | PixelSense SDK - Orientation | The PixelSense SDK functions will provide the ability to determine how the Lynx is oriented on the Surface table. These functions can leverage existing functionality present in the development kit for PixelSense. |
| 4.10 | Android – Device on table | The Device and Android SDK will provide information to the user via the Android tablet connected to the Lynx when the Lynx is on the Surface table and ready to transmit or receive data. |
| 4.11 | PixelSense – Device on table | The Device and PixelSense SDK will provide information to the user via the Surface table that the Lynx is on the Surface table and ready to transmit or receive data. |
| 6.1 | Minimum Transfer Rate | Data transfer must be consistent at minimum of 200 bits/sec |

Table 1‑1: Key Requirements

# Meta Architecture

This section describes an overview our design principles and philosophy; including some key assumptions we’ve made while designing the system, our overall design principles, and structural choices that will keep this project both manageable and feasible. It will elaborate on the approach that the team has been following as well as the guiding principles which the structure was based on. It will also explain the reasoning behind the decisions and any implications. The section will then end with any tradeoffs and assumptions for our architecture.

## Architectural Vision

The PixelSense Optical Transfer Center consists of three main parts; the PixelSense table hardware, the Lynx transfer device and the developed Black Jack software on both the Android and Microsoft PixelSense table. With that said, since the main purpose of the system is to transfer data optically, it was imperative to develop a communication system between the components to be not only fast and efficient but must also be secure to avoid interceptions of data.

The team has decided to classify layers into larger “Super Layers”; A PixelSense Super Layer, Lynx Super Layer, and a Transfer Super Layer. Inside these Super layers there are smaller layers that make this system a whole. Inside the PixelSense Super Layer, the following sub-layers exist: Presentation and Data Processing. In the Lynx Super Layer, the following sub-layers exist: Presentation, Data Processing and Data Storage. And finally in the Transfer Super Layer the following sub-layers exist: PixelSense Transfer and Lynx Transfer layers. Within these sub-layers, there will be several subsystems which are the main components to the system. Starting with the PixelSense Super Layer; this layer responsible for housing all layers and subsystems in relation to the PixelSense hardware. Inside the PixelSense Super Layer we have the Presentation layer, this layer is responsible for all UI, User Input and User Output functions. After that we have the Data Processing layer, this layer takes the input and processes it to perform a certain function. Going the other way, once it receives data back it processes the data and sends it off back to the presentation for displaying. In the Lynx Super layer essentially the same process happens with the data flow. We also have the Storage Layer in the Lynx side which will handle all data being stored and saved as well as data retrieval. Finally we come to the Transfer Super Layer, inside here we have two more sub-layers where one is the PixelSense Transfer and the other is the Lynx Transfer, with that said we have both these layers will be sending a receiving data over light between each other.

## Guiding Principles

### Ease of Use

The PixelSense Optical Transfer Center will have a very smooth and clear user interface which will allow the user to interact with the system. Since the system is very user driven, it is imperative that the system is very easy to use so that the user is not confused. This applies to both the Windows application as well as the Android application, the applications should not be cluttered with too many things, all pieces should be easily identifiable as well as the user interaction parts should also be easily accessible. Since this is mainly a Black Jack application, the amount of screens to navigate through will be very minimal which will reduce the complexity of the application from having too many options and buttons that the user may curiously click and not know how to get back to the main screen. The end goal for this is to have a system that is simple and yet powerfully effective at the same time for the highest user satisfaction and usability.

### Simplicity

Due to the nature of our product (a simple Black Jack application), the overall system as well as the architecture should be very simple. By increasing the size and complexity of the system and user interface, the system will be more vulnerable to failures and issues. With that said, the team has decided to implement the KISS principle and believe that simplicity on this project is key to it being successful. With that said the architecture should not have any unnecessary subsystems as well as too many redundant data flows, having too much redundancy will cause our system to be bottlenecked at some point.

### Reliability

The PixelSense Optical Transfer Center should be very robust in both the hardware and software. By having prebuilt hardware in terms of the PixelSense Table, Android device and the Microcontroller board, the team can be confident of having solid and reliable hardware pieces. With that said anything built by the team (ie the actual Lynx), should not be held in any lower standard and should be built with the same confidence of build quality in mind. Now going onto the software side, the software shall be thoroughly tested to avoid holes and broken links through data flows. By saying that, good programming conventions must be followed to avoid common mistakes. With that said, the system as a whole should run error free and the end-user should not run into any errors or issues that would impede the usability of the system.

## Key Assumptions and Tradeoffs

This section includes the assumptions and Tradeoffs that the team considered to be true before designing the system. The structure of this system will rely on these assumptions and tradeoffs to be both correct and constant throughout the duration of the project, through construction and delivery.

### Communications Between At Least Two Heterogeneous Devices

|  |  |
| --- | --- |
| **Assumptions** | **Design Implications** |
| Communication should be at the minimum between two heterogeneous devices | When fully implemented, the system facilitates communication between a PixelSense table, and at least one Lynx device. The system is not designed to have two Lynx devices communicate directly with each other, or for two PixelSense tables to share data directly. |
| Data Under Complete System Control During Transfer | From the moment data is sent from a front end user interface on either the PixelSense table or Lynx device, until it reaches the other system’s user interface, the system will have total control over the format and configuration of the sent data. |

Table 2‑1: Assumptions & Design Implications

### Tradeoffs

While designing the system architecture, the team had come across several alternatives to get to a viable system design. The list below shows the tradeoffs that have occurred once the team has chosen an end design.

Initially the team had decided on using a single system with many layers and subsystems. We quickly found out that model would not be very efficient since a lot of duplicate subsystems and layers were popping up. With that said, as a team consensus and some advice the team had taken the direction of implementing a Super Layer system, where there would be three major Super Layers and within each Super Layer, there would be a smaller system. The tradeoff here is that the system is no longer a single system, but a multisystem and by having a multisystem, this created much more data flows from inter-system transfers. But the end goal seemed to be better for the team as it created a more organized, simple and concise overall system and the data flow now is much easier to follow.

Another tradeoff that had occurred was the removal of an input and output layer. Initially the team thought that that it would be wise to have separate input and output layers in order to accommodate data entry and exit. But once the team built it like that, it was quickly seen that there were redundant subsystems and that the input and output layers themselves were quite small and seemed like it did not warrant for their own layers. With that said the team decided to remove those layers and just incorporate it into the presentation layer. This in turn proved to be better for the team as this removed the number of overall subsystems the architecture had.

## Design Principles

In this section, the core philosophy behind the team’s design, the reasons behind these major design decisions are explained, and why the team ultimately decided upon these ideals.

### Super Layers and Discrete Design

Since this project is about communication between distinct objects, data flows between defined architecture layers will be intrinsically linked to which object that layer belongs to. Layers representing components from the same object will be much more dependent to each than two layers from distinct objects. The exception to this observation are the layers from each object that directly relates to optical data transfer.

The team has decided to classify layers into larger “Super Layers”; A PixelSense Super Layer, Lynx Super Layer, and a Transfer Super Layer. Classifying layers in this manner allows the team to examine dependencies between layers, as well as discrete objects in the system, allowing us to reduce communication between objects, increasing modularity. Super Layers also allow the team to identify parts of the system that can be easily developed in parallel. Layers existing in different Super Layers should be able to be developed independently of each other. Layers that communicate between Super Layers would have to be developed last, but system implementation should be reserved for only implementing objects under this design, instead of every individual component

### Subcomponents: Two Way, In-Out Pipeline

To reduce the complexity of communication between objects, each subcomponent in a respective layer is delegated to a specific step in the data transfer pipeline. Each subcomponent expects one specific input format from one end, and outputs 1 specific output format for the next component in the chain. Since this communication needs to be two-directional, the format that a subcomponent outputs may also be taken in as input from the opposite direction, and vice-versa. A component handles exactly one format of data from another component. Handling the data in this manner will reduce the types of data each subcomponent has to handle and translate. Rather than having one subcomponent handle complete translation, this process allows for a more granular approach to design, implementation, and testing.

# Layer Definitions

The diagram below depicts a high level description of our system by breaking it down into several layers. Since our whole system contains two smaller systems, the diagram has been split up into a non-traditional way. This is done by having super layers and within each super layer, there will be smaller layers and subsystems. In total we will have three super layers and eight sub layers. The super layers are labeled as: PixelSense, Tablet/Lynx, and Transfer. The sub layers in the PixelSense layer are: Presentation and Data Processing. The sub layers in the Tablet/Lynx layer are: Presentation, Data Processing and Storage Layer. The sub layers in the transfer layer are: Lynx, PixelSense Transfer, and SDK. This architecture is based on the guiding principles in section 2.2. A description of each layer and sub layer will be provided and explained in the sections below.

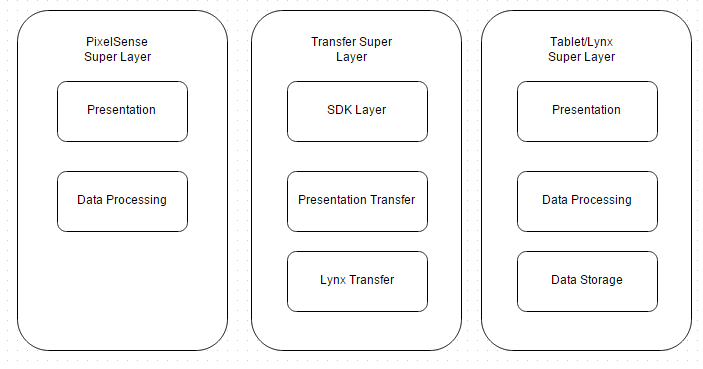


Table 3‑1: Layer Definitions

## 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 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.

### 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

### Data Processing Layer

The purpose of this layer 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 PixelSense Transfer Manager which forwards data to transfer Layer.

## 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

### 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

### 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.

### 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 to the Data Storage Layer which will return the relevant information to the Data Processing Layer.

## 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

### 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.

### 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.

### 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

## Detailed Architecture Diagram

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 of them interact 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.

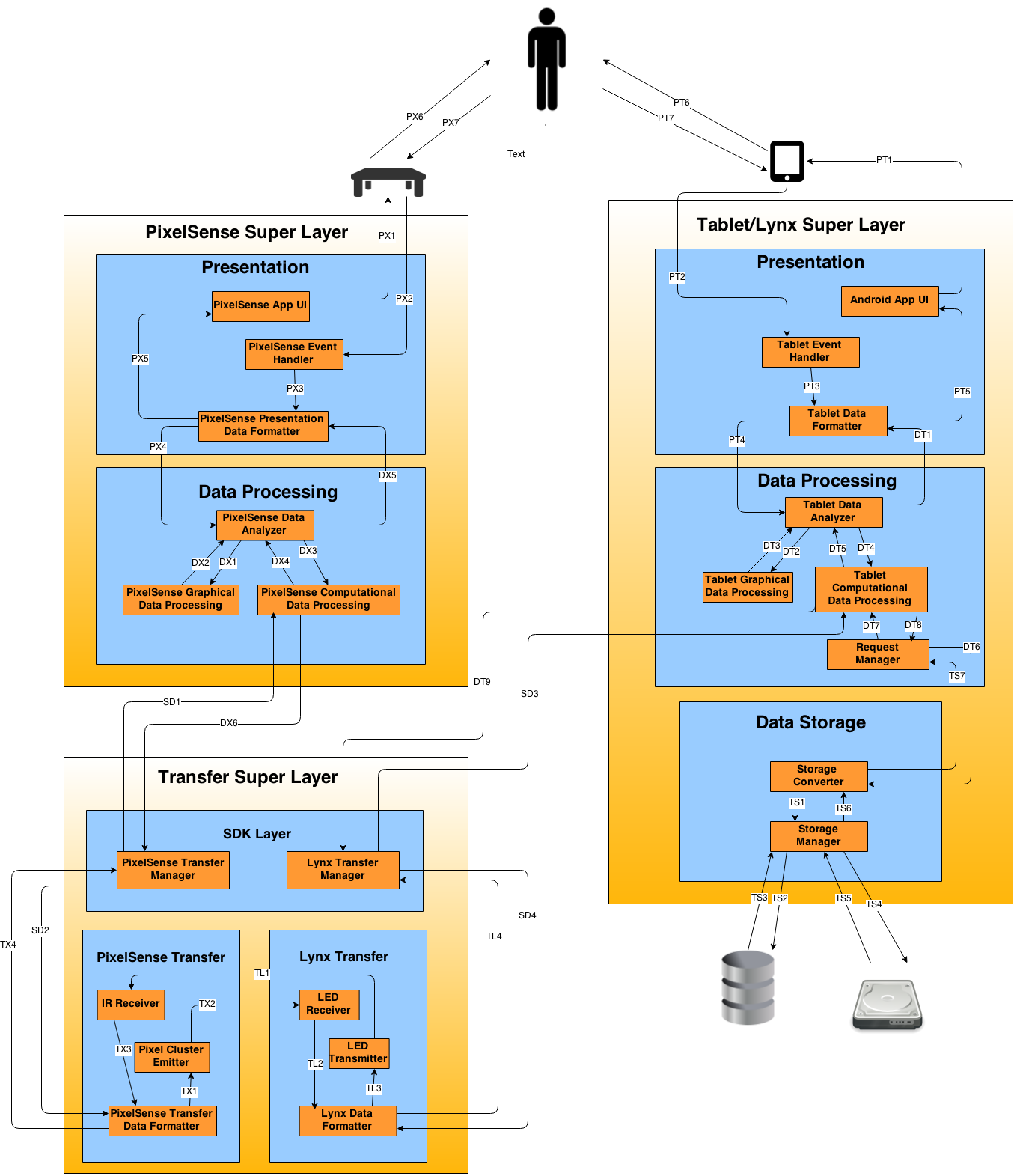


Table 3‑2: Architecture Diagram

# PixelSense Super Layer

## Presentation Layer

The following section will describe the PixelSense Presentation Layer in more detail. The purpose of the layer is to take input from the user via demo application on PixelSense Table. Subsystems inside the layer will be listed and elaborated on

### PixelSense App UI

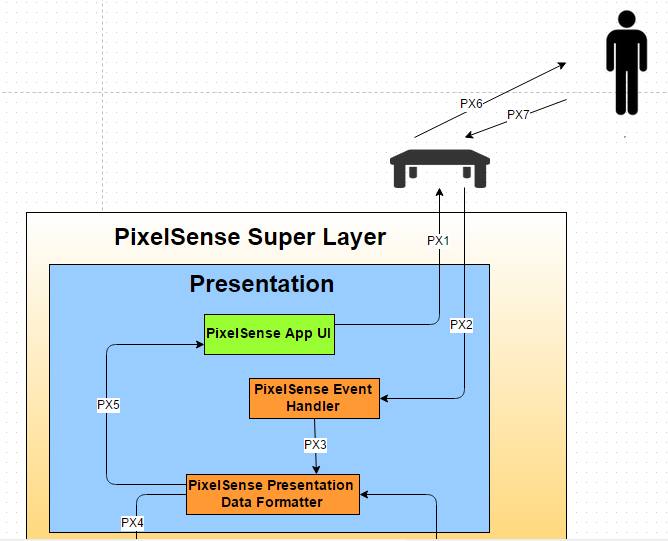


Figure 4‑1: PixelSense App UI

#### General Description

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.

#### Assumptions

PixelSense App UI must be able to handle various types of display inputs from PixelSense Presentation Data Formatter.

#### Responsibilities

The responsibility of PixelSense App UI subsystem is to display the graphical as well as computational content to the user via PixelSense Table.

#### Inter-Layer Interfaces

None

#### Public Interfaces

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Description** | **Info Required** | **Info Returned** |
| displayPresentationData | Displays the data to the table. | None | Data |

Table 4‑1: Public Interfaces (PixelSense App UI)

### PixelSense Event Handler Subsystem

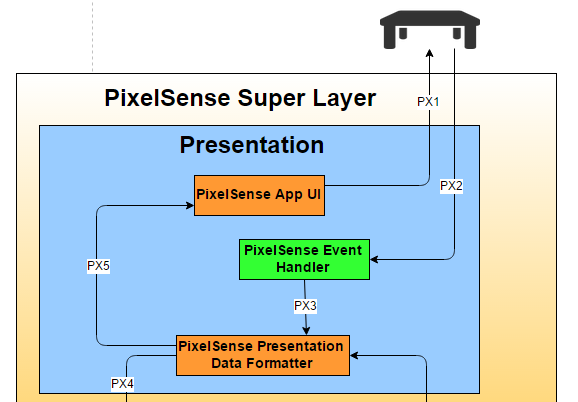


Figure 4‑2: PixelSense Event Handler



#### General Description

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

#### Assumptions

PixelSense Event Handler subsystem must be able to handle various types of data inputs from user.

#### Responsibilities

The responsibility of PixelSense Event Handler 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.

#### Inter-Layer Interfaces

None

#### Public Interfaces

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Description** | **Info Required** | **Info Returned** |
| getData | receives raw input from user | None | None |

Table 4‑2: Public Interfaces (PixelSense Event Handler)

### PixelSense Presentation Data Formatter Subsystem

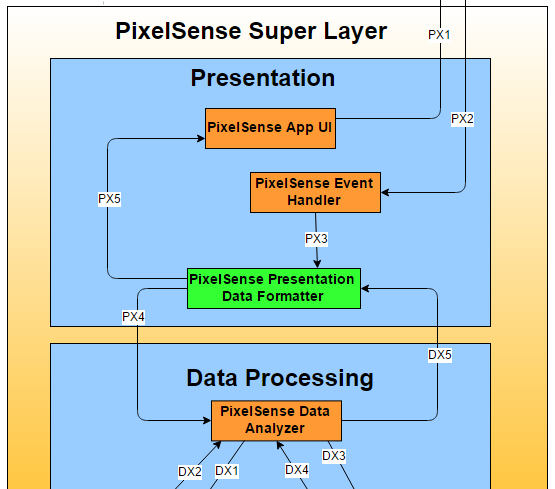


Figure 4‑3: PixelSense Data Formatter

#### General Description

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.

#### Assumptions

PixelSense Data Formatter subsystem must be able to handle various types of data inputs from User.

#### Responsibilities

The responsibility of PixelSense Data Formatter 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.

#### Inter-Layer Interfaces

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Description** | **Info Required** | **Info Returned** |
| getAnalyzedData | Gets Analyzed data from Data Analyzer subsystem in Data Processing Layer for display purposes. | computational data | None |
| sendFormattedData | Sends formatted data to Data Analyzer for further data process. | None | Formatted data |

Table 4‑3: Interlayer Interfaces (PixelSense Data Formatter)

#### Public Interfaces

None

## Data Processing Layer

### PixelSense Data Analyzer

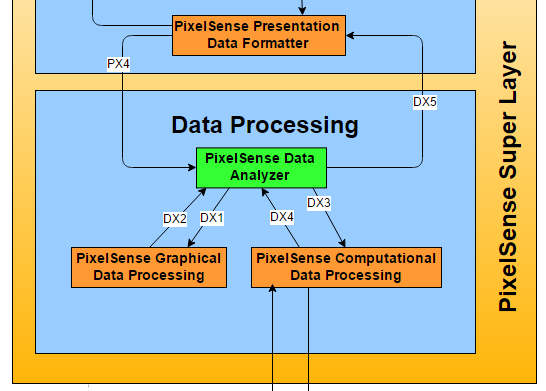


Figure 4‑4: PixelSense Data Analyzer

#### General Description

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.

#### Assumptions

Data coming from PixelSense Data Formatter must be properly formatted in order to separate two types of data.

#### Responsibilities

Responsibility of PixelSense Data Analyzer is to separate graphical and computational input and send it to respective subsystems.

#### Inter-Layer Interfaces

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Description | Info Required | Info Returned |
| getFormattedData | Gets formatted data from Presentation Layer. | Formatted data | None |
| sendAnalyzedData | Sends analyzed data to Data formatter for display purposes. | None | analyzed data |

Table 4‑4: Interlayer Interfaces (PixelSense Data Analyzer)

#### Public Interfaces

None

### PixelSense Graphical Data Processing

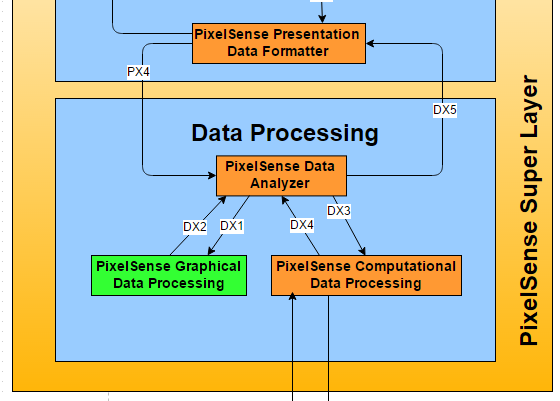


Figure 4‑5: PixelSense Graphical Data Processing

#### General Description

PixelSense Graphical Data Processing subsystem interacts with Data Analyzer subsystem of Data Processing layer. This subsystem gets the graphical data and dumps it as it no longer required for the further process.

#### Assumptions

Data coming from Data Analyzer must be graphical data

#### Responsibilities

This subsystem is responsible for receiving data from the PixelSense Data Analyzer subsystem. Then it processes the data needed for any graphical operations and then sends the processed data back to the PixelSense Data Analyzer subsystem.

#### Inter-Layer Interfaces

None

#### Public Interfaces

None

### PixelSense Computational Data Processing

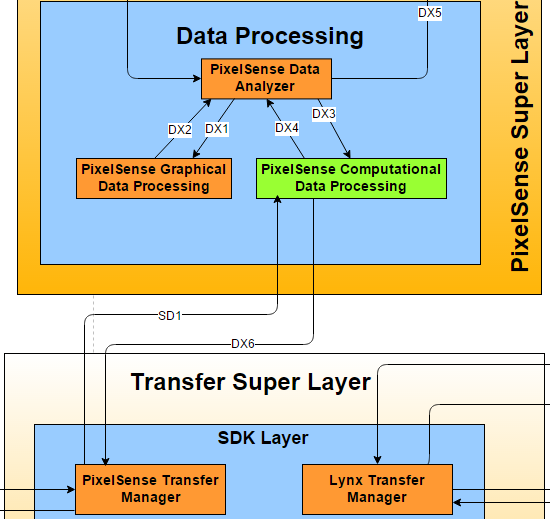


Figure 4‑6: PixelSense Computational Data Processing

#### General Description

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.

#### Assumptions

Data coming from Data Analyzer must be in computational. Also, data coming from transfer controller must be in binary format.

#### Responsibilities

This subsystem processes the data needed for any computational operations and then sending the processed data back to the PixelSense Data Analyzer subsystem or the PixelSense Transfer Controller in SDK layer subsystem depending on where the data is coming.

#### Inter-Layer Interfaces

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Description** | **Info Required** | **Info Returned** |
| sendData | Sends computation data to PixelSense Transfer Layer. | None | Binary Data |

Table 4‑5: Interlayer Interfaces (PixelSense Computational Data Processing)

#### Public Interfaces

None

# Tablet/Lynx Super Layer

## Presentation Layer

### Android Application UI

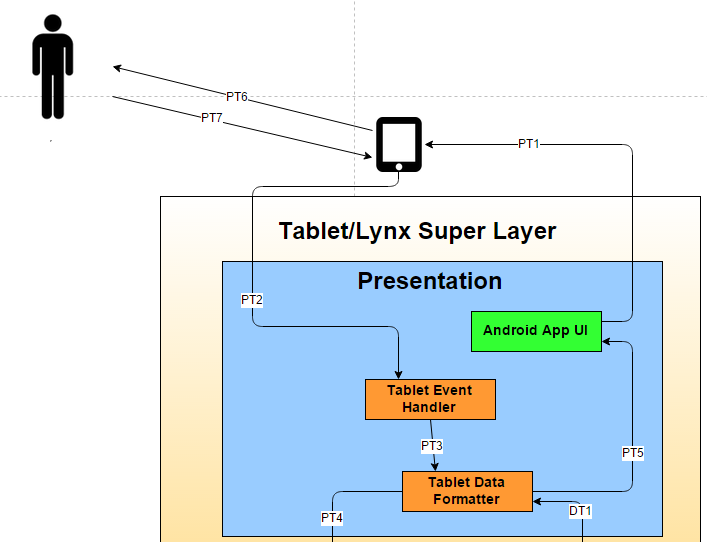


Figure 5‑1: Android App UI

#### General Description

This subsystem will provide a graphical user interface for the Lynx/Tablet application. This subsystem will display a username, chip count, Blackjack game command options, and a transaction log. The user will be able to interact with the game commands via touch screen.

#### Assumptions

The Android tablet is functioning properly and running the required version of Android operating system. The blackjack command buttons are relevant to the current game situation (e.g. Bet is displayed when it is time to bet, or Hit/Stay when it’s the user’s time to hit or stay). The transaction log will display at most the fifteen most recent transactions.

#### Responsibilities

The application UI will display the following: username, the chip count, Blackjack application command buttons, and a transaction log. As well as refreshing the graphical user interface as new data is received from the Tablet Data Formatter subsystem.

#### Inter-Layer Interfaces

None

#### Public Interfaces

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Description** | **Info Required** | **Info Returned** |
| getPresentationData | Recieves formatted data | Computational Data | None |
| displayPresentationData | Displays the data to the tablet. | None | Data |

Table 5‑1: Public Interfaces (Android App UI)

### Tablet Event Handler

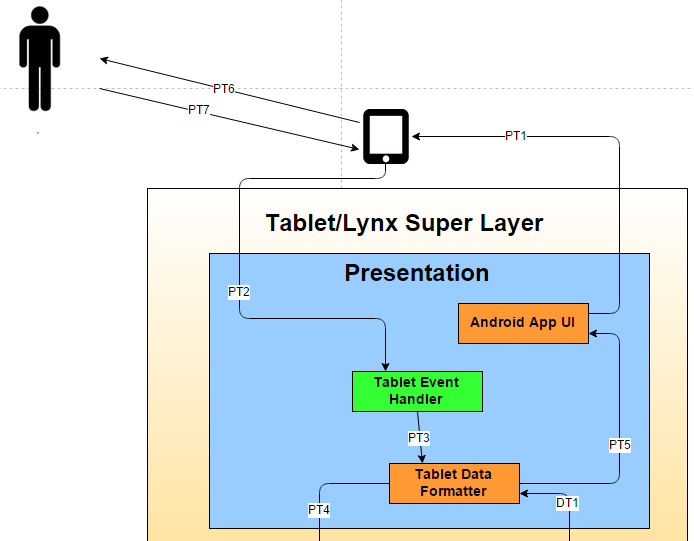


Figure 5‑2: Tablet Event Handler

#### General Description

This subsystem handles input from the user and sends said input to the Tablet Data Formatting subsystem. This subsystem forms a valid Android event from the user input received from the user.

#### Assumptions

The inputs received from the user are valid Android events, or valid inputs given the blackjack game options (bet, hit, or stay), at valid times in the game.

#### Responsibilities

The Tablet Event Handler subsystem receives an input from the user, then determines how that corresponds to the Application UI subsystem, and sends that data to the Data Formatting subsystem.

#### Inter-Layer Interfaces

None

#### Public Interfaces

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Description** | **Info Required** | **Info Returned** |
| getInputData | Receives raw input from user | Input Data | None |
| sendInputData | Sends the input data to event handler | None | Input Data |

Table 5‑2: Public Interfaces (Tablet Event Handler)

### Tablet Data Formatter

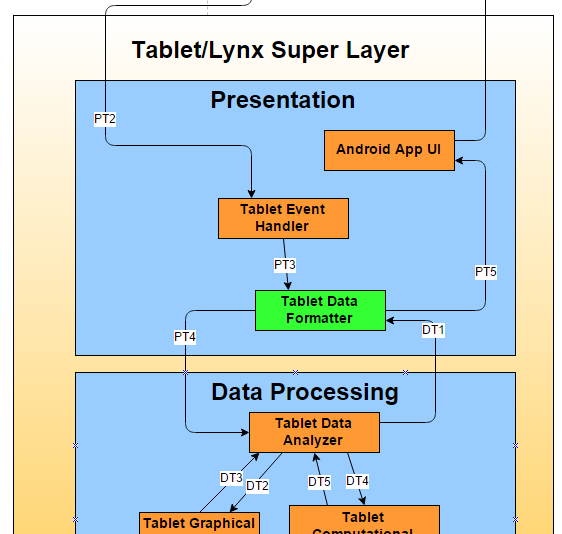


Figure 5‑3: Tablet Data Formatter

#### General Description

This subsystem formats data that will be sent to the Android Application UI and the Data Processing Layer. The data that is formatted is received from either the Tablet Event Handler or the Data Processing Layer.

#### Assumptions

The Android Application UI is able to present the data that is formatted and sent to it from this subsystem. All data that needs to be processed or displayed to the user is tagged accordingly so that it may be sent to the correct subsystem.

#### Responsibilities

This subsystem formats the data it is given from either the Event Handler subsystem or the Data Transfer subsystem and then sends it to the Application UI or to the Data Transfer subsystem. This subsystem is responsible for sending data to the Data Processing Layer and the Data Formatting subsystem, as well as, receiving data from the Data Processing Layer and the Data Formatting subsystem.

#### Inter-Layer Interfaces

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Description** | **Info Required** | **Info Returned** |
| getAnalyzedData | Gets Analyzed data from Data Analyzer subsystem in Data Processing Layer for display purposes. | computational data | None |
| sendFormattedData | Sends formatted data to Data Analyzer for further data process. | None | Formatted data |

Table 5‑3: Interlayer Interfaces (Tablet Data Formatter)

#### Public Interfaces

None

## Data Processing Layer

### Tablet Data Analyzer

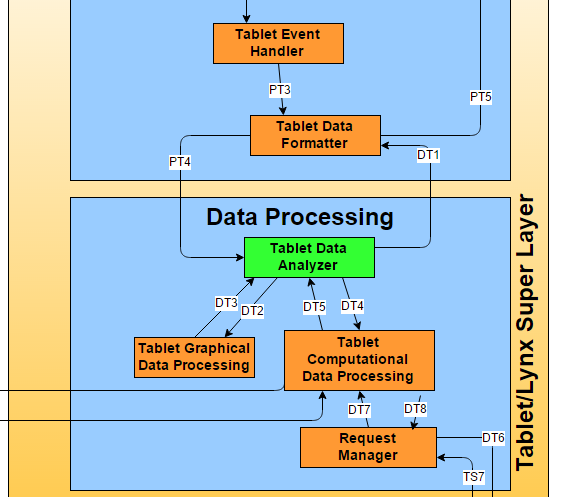


Figure 5‑4: Tablet Data Analyzer

#### General Description

This subsystem analyzes data to be sent to Tablet Graphical Data Processing, Tablet Computational Data Processing, or the Tablet Data Formatter (Presentation Layer) subsystems. The data to be analyzed is received from the same subsystems stated previously.

#### Assumptions

All data is properly tagged so that it is sent to the correct subsystem.

#### Responsibilities

This subsystem is responsible for analyzing data received from the Presentation Layer and then sending that data to either the Tablet Graphical Data Processing subsystem or the Tablet Computational Data Processing subsystem. Once that data is processed and sent back to this subsystem, it is sent to the Presentation Layer to be displayed to the user.

#### Inter-Layer Interfaces

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Description** | **Info Required** | **Info Returned** |
| getFormattedData | Gets formatted data from Presentation Layer. | Formatted data | None |
| sendProcessedData | Returns the processed data to the Presentation Layer | None | Processed data |

Table 5‑4: Interlayer Interfaces (Tablet Data Analyzer)

#### Public Interfaces

None

### Tablet Graphical Data Processing

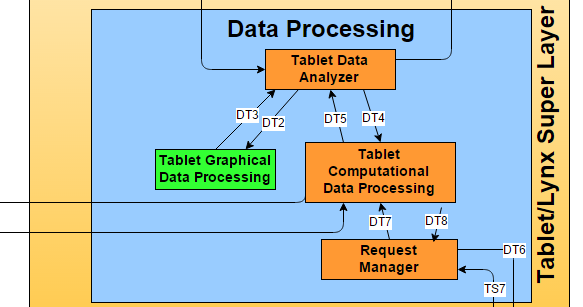


Figure 5‑5: Tablet Graphical Data Processing

#### General Description

This subsystem will receive data from the Tablet Data Analyzer subsystem, process that information, and then return the processed information to the Tablet Data Analyzer subsystem as relevant graphical data.

#### Assumptions

All graphical operations can be handled by the tablet’s Android operating system. All data sent to this subsystem can be processed by this subsystem.

#### Responsibilities

This subsystem is responsible for receiving data from the Tablet Data Analyzer subsystem. Then processing the data needed for any graphical operations and then sending the processed data back to the Tablet Data Analyzer subsystem.

#### Inter-Layer Interfaces

None

#### Public Interfaces

None

### Tablet Computational Data Processing

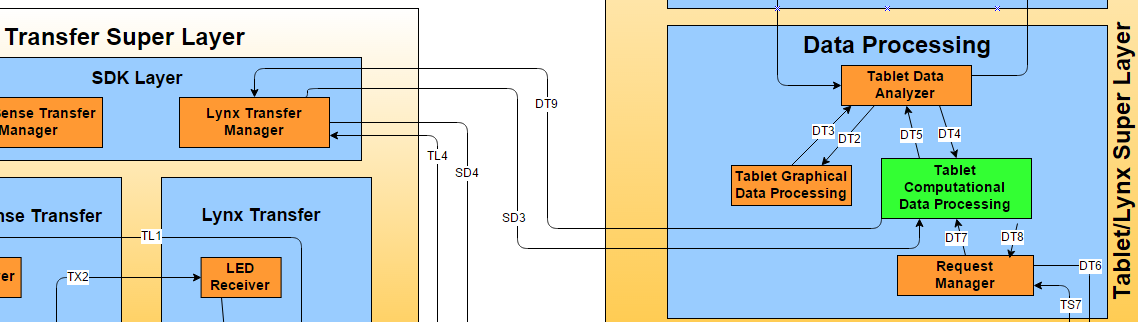


Figure 5‑6: Tablet Computational Data Processing

#### General Description

This subsystem will receive data from the Tablet Data Analyzer subsystem and the Lynx Transfer Manager subsystem, process that information, and then return the processed information to the Tablet Data Analyzer subsystem or the Lynx Transfer Manager subsystem as relevant computational (non-graphical) data.

#### Assumptions

All data received by this subsystem to be processed can be processed computationally. Data received from the Lynx Transfer Manager subsystem is binary and data sent to the Lynx Transfer Manager is converted to binary.

#### Responsibilities

This subsystem is responsible for receiving data from the Tablet Data Analyzer subsystem or the Lynx Transfer Manager subsystem. Then processing the data needed for any computational operations and then sending the processed data back to the Tablet Data Analyzer subsystem or the Lynx Transfer Manager subsystem.

#### Inter-Layer Interfaces

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Description** | **Info Required** | **Info Returned** |
| getComputationalData | Gets computational data from computational processing subsystem. | Binary Data Data | None |
| sendData | Sends computation data to Lynx Transfer Layer. | None | Binary Data |

Table 5‑5: Interlayer Interfaces (Tablet Computational Data Processing)

#### Public Interfaces

None

### Request Manager

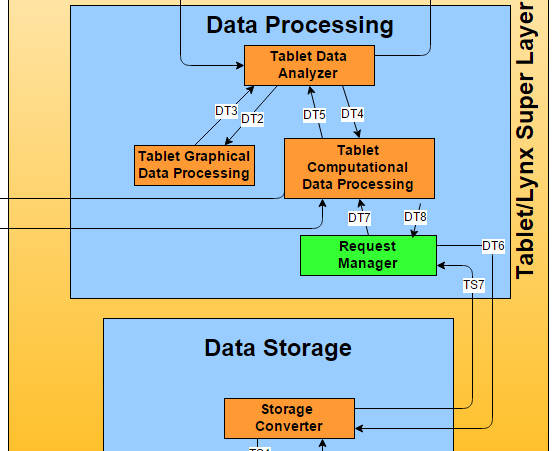


Figure 5‑7: Request Manager

#### General Description

This subsystem will manage requests for data from the Data Storage Layer. Requests will be sent to this subsystem from the Tablet Computational Data Processing subsystem. Those requests are then sent to the Data Storage Layer to be processed and retrieve, update or insert the data in the request.

#### Assumptions

All requests are valid and capable of being handled by the Data Storage Layer.

#### Responsibilities

This subsystem is responsible for managing requests for data from the Data Storage Layer, by sending requests from the Computational subsystem to the Data Storage Layer.

#### Inter-Layer Interfaces

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Description** | **Info Required** | **Info Returned** |
| getData | Sends request for stored data to the Data Storage Layer. | Request | Stored Data |
| storeData | Sends request to store data to the Data Storage Layer | Data to store | none |

Table 5‑6: Interlayer Interfaces (Request Manager)

#### Public Interfaces

None

## Data Storage Layer

### Storage Converter

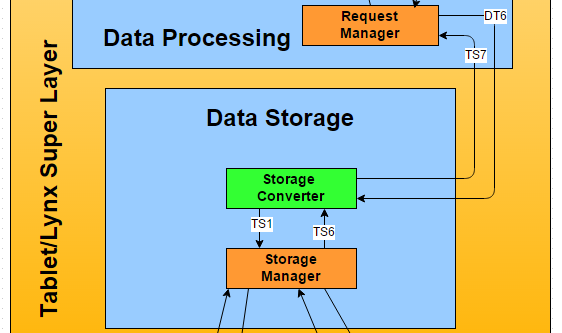


Figure 5‑8: Storage Converter

#### General Description

This subsystem converts data requests into relevant queries, submits those queries to the Storage Manager subsystem, receives the data requested from the Storage Manager subsystem, and also sends the data requested to the requesting party.

#### Assumptions

All requests to be converted into queries for a SQLite database. All requests have a corresponding database table and/or column.

#### Responsibilities

This subsystem is responsible for converting requests into queries, submitting queries to the Storage Manager, and then sending the data yielded from the query to the Data Processing Layer’s Request Handler subsystem.

#### Inter-Layer Interfaces

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Description** | **Info Required** | **Info Returned** |
| returnData | Sends requested data to the Data Processing Layer. | none | Stored Data |
| getRequest | Gets request for data from the Data Processing Layer | None | Data Request |

Table 5‑7: Interlayer Interfaces (Storage Converter)

#### Public Interfaces

None

### Storage Manager

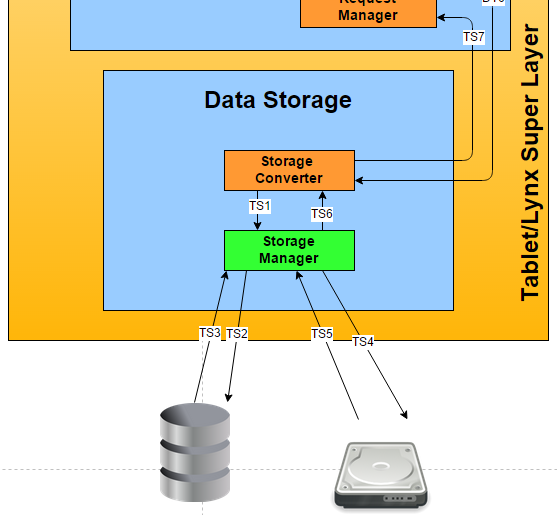


Figure 5‑9: Storage Manager

#### General Description

This subsystem manages all data stored on the Tablet that is needed for the Android application. Requests or queries are received from the Storage Converter subsystem, and the corresponding data is sent back to the Storage Converter subsystem.

#### Assumptions

A SQLite database will be used to store data and all queries sent to this subsystem will be formatted in the manner that SQLite handles them.

#### Responsibilities

This subsystem is responsible for processing the requests or queries received from the Storage Converter subsystem and returning the corresponding data to the Storage Converter subsystem.

#### Inter-Layer Interfaces

None

#### Public Interfaces

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Description** | **Info Required** | **Info Returned** |
| executeQuery | Executes query to SQLite database. | Query | Stored Data |
| returnData | Returns the data relevant to the executed query. | None | Stored Data |

Table 5‑8: Public Interfaces (Storage Manager)

# Transfer Super Layer

## PixelSense Transfer Layer

### IR Receiver

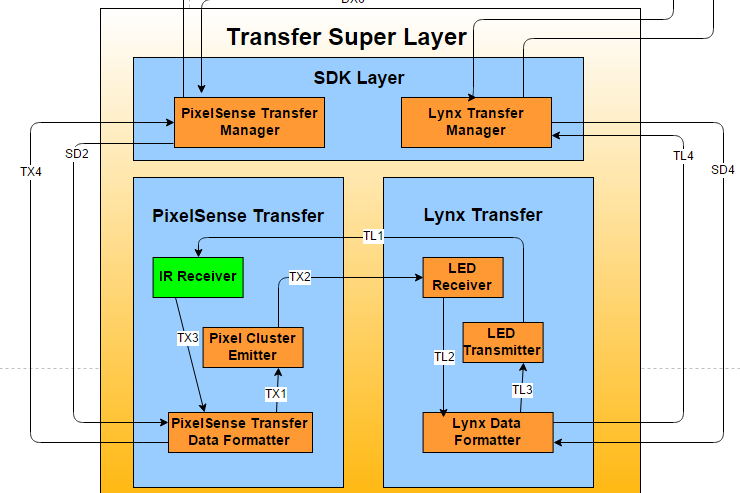


Figure 6‑1: IR Receiver

#### General Description

The IR Receiver is responsible for receiving the light emitted from the LED transmitter and hand it off to the data formatter. It only accepts data to send off.

#### Assumptions

The data inputted from the LED transmitter will be in light form and the data outputted will be in digital high or low so that the formatter can easily understand it.

#### Responsibilities

The responsibility of the IR Receiver is to receive light from the transmitter and convert it to a digital high or low, that way the data formatter can take the individual highs and lows and can output it in binary to the next subsystem.

#### Inter-Layer Interfaces

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Description** | **Information Required** | **Information Returned** |
| DetectLight | The IR Receiver will constantly look for light emissions from the transmitter and pass on a digital high or low to the formatter | On or off LEDs | None |

Table 6‑1: Interlayer Interfaces (IR Receiver)

#### Public Interfaces

None

### PixelSense Cluster Emitter

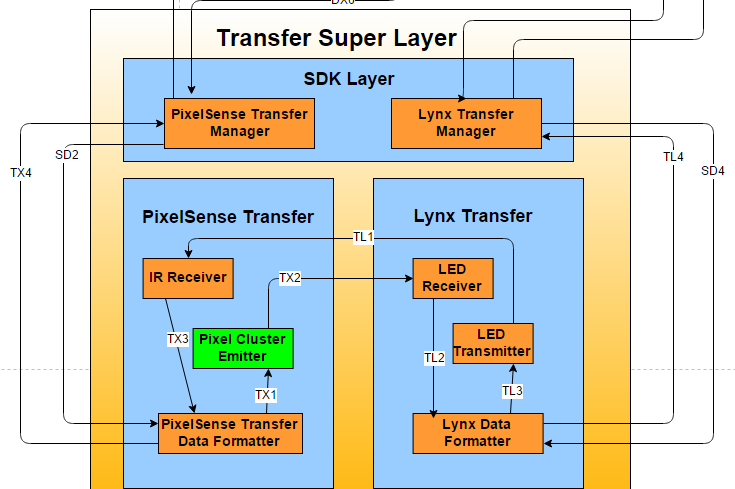


Figure 6‑2: PixelSense Cluster Emitter

#### General Description

The Pixel Cluster Emitter is responsible for emitting the received data in light format through clusters of pixels so that an LED receiver can detect it.

#### Assumptions

Input data will be binary so that the pixels can be flashed in either on or off positions.

#### Responsibilities

The Pixel Cluster Emitter is responsible for emitting the data through flashing pixels, so that the data will be transferred through light. It has no other purpose other than flashing pixels from the incoming data from the data formatter.

#### Inter-Layer Interfaces

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Description** | **Information Required** | **Information Returned** |
| FlashPixels | The Pixel Cluster Emitter will flash data to the on screen pixels so that it can be picked up by the LED receiver. | Formatter data from the PixelSense Trasnfer Data Formatter | None |

Table 6‑2: Interlayer Interfaces (PixelSense Cluster Emitter)

#### Public Interfaces

None

### PixelSense Transfer Data Formatter

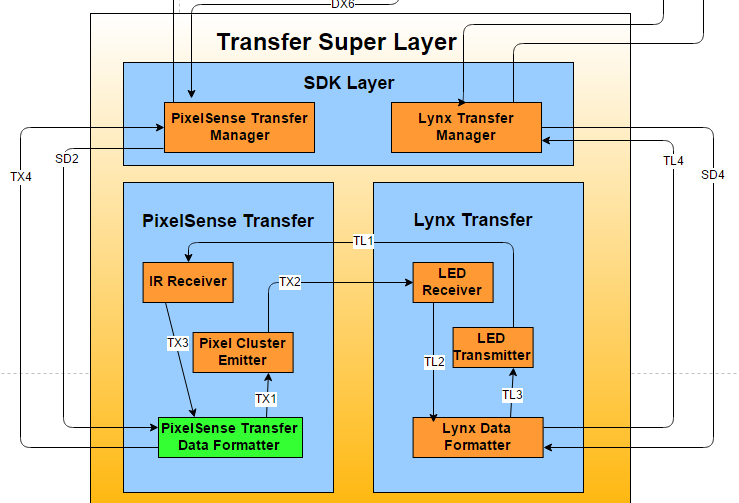


Figure 6‑3: Transfer Data Formatter

#### General Description

The PixelSense Transfer Data Formatter is responsible for formatting incoming and outgoing data to the proper format so the next subsystem can understand what the data is and what to do with it. It is essential to have this subsystem in order to reduce data corruption by having to use a common data format.

#### Assumptions

Data formatter must accept various forms of input from the IR receiver and the PixelSense Transfer Manager.

#### Responsibilities

The responsibility of the Data Formatter is to accept the expected input and translate it to an acceptable form. The formatter also structures the data in such a way that it is easy to understand for the next subsystem.

#### Inter-Layer Interfaces

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Description** | **Information Required** | **Information Returned** |
| PixelManagerGetData | The subsystem will retrieve data from the PixelSense Transfer Manager for sending. | None | None |
| PixelManagerSendData | The subsystem will send data to the PixelSense Transfer Manager for sending | None | None |

Table 6‑3: Interlayer Interfaces (PixelSense Transfer Data Formatter)

#### Public Interfaces

None

## Lynx Transfer Layer

### LED Receiver

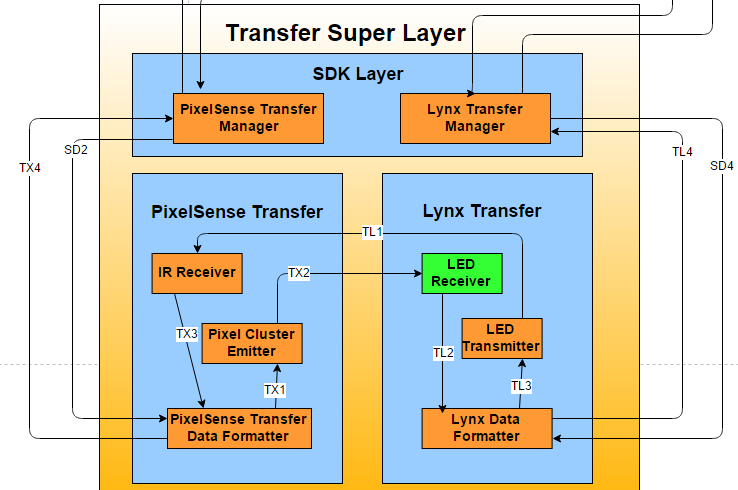


Figure 6‑4: LED Receiver

#### General Description

The LED Receiver is responsible for receiving the light emitted from the Pixel Cluster Emitter and hand it off to the data formatter. It only accepts data to send off.

#### Assumptions

The data inputted from the Pixel Cluster Emitter will be in light form and the data outputted will be in digital high or low so that the formatter can easily understand it.

#### Responsibilities

The responsibility of the LED Receiver is to receive light from the Pixel Cluster Emitter and convert it to a digital high or low, that way the data formatter can take the individual highs and lows and can output it in binary to the next subsystem.

#### Inter-Layer Interfaces

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Description** | **Information Required** | **Information Returned** |
| DetectLight | The LED Receiver will constantly look for light emissions from the transmitter and pass on a digital high or low to the formatter | On or off LEDs | None |

Table 6‑4: Interlayer Interfaces (LED Receiver)

#### Public Interfaces

None

### LED Transmitter

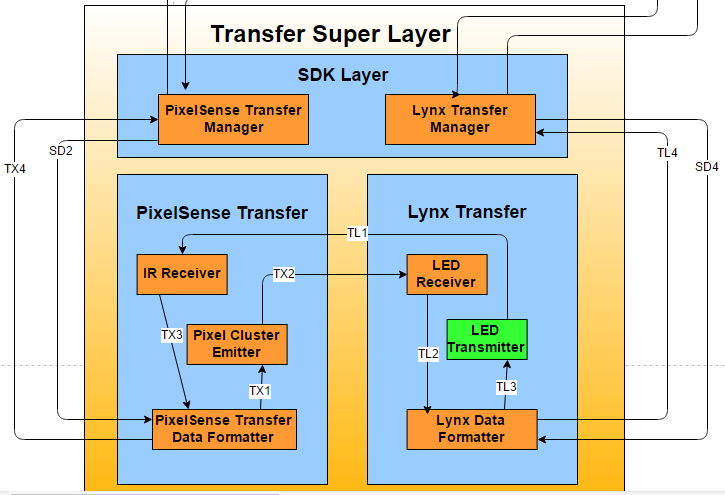


Figure 6‑5: LED Transmitter

#### General Description

The LED Transmitter is responsible for emitting the received data in light format through LEDs so that the IR receiver can detect it.

#### Assumptions

Input data will be binary so that the pixels can be flashed in either on or off positions.

#### Responsibilities

The LED Transmitter is responsible for emitting the data through flashing pixels, so that the data will be transferred through light. It has no other purpose other than flashing pixels from the incoming data from the data formatter.

#### Inter-Layer Interfaces

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Description** | **Information Required** | **Information Returned** |
| FlashLEDs | The LED Transmitter will flash data to the on screen pixels so that it can be picked up by the LED receiver. | Formatter data from the Lynx Data Formatter | None |

Table 6‑5: Interlayer Interfaces (LED Transmitter)

#### Public Interfaces

None

### Lynx Data Formatter

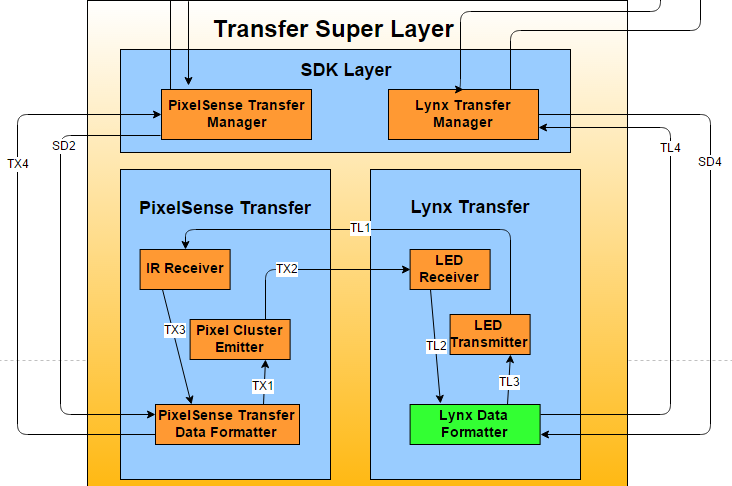


Figure 6‑6: Lynx Data Formatter

#### General Description

The Lynx Transfer Data Formatter is responsible for formatting incoming and outgoing data to the proper format so the next subsystem can understand what the data is and what to do with it. It is essential to have this subsystem in order to reduce data corruption by having to use a common data format.

#### Assumptions

Data formatter must accept various forms of input from the IR receiver and the Lynx Transfer Manager.

#### Responsibilities

The responsibility of the Data Formatter is to accept the expected input and translate it to an acceptable form. The formatter also structures the data in such a way that it is easy to understand for the next subsystem.

#### Inter-Layer Interfaces

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Description** | **Information Required** | **Information Returned** |
| LynxManagerGetData | The subsystem will retrieve data from the Lynx Transfer Manager for sending. | None | None |
| LynxManagerSendData | The subsystem will send data to the Lynx Transfer Manager for sending | None | None |

Table 6‑6: Interlayer Interfaces (Lynx Data Formatter)

#### Public Interfaces

None

## SDK Layer

### PixelSense Transfer Manager

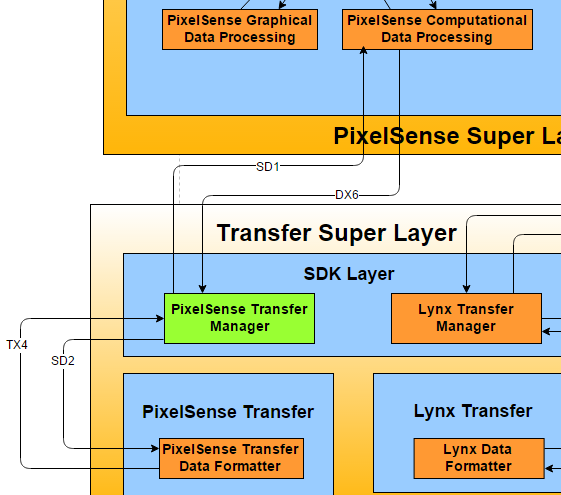


Figure 6‑7: PixelSense Transfer Manager

#### General Description

The PixelSense Transfer Manager’s purpose is to take the processed data from the Data Processing subsystem and sending it off for formatting to the Data Formatter

#### Assumptions

Processed Data must be in the appropriate digital form so the subsystem can translate to analog data.

#### Responsibilities

The responsibility of the PixelSense Transfer Manager is to send and receive data from the processing subsystem as well as the Data Formatter, it acts like a middle translator for digital-analog and vice-versa.

#### Inter-Layer Interfaces

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Description** | **Information Required** | **Information Returned** |
| GetPixelProcessedData | The subsystem will retrieve data from the PixelSense Computational Data Processing Subsystem. | None | None |
| SendPixelProcessedData | The subsystem will send data back to the PixelSense Computational Data Processing Subsystem for output feedback back to the user | None | None |
| GetPixelFormattedData | The subsystem will get back formatted data from the Data Formatter | None | None |
| SendPixelDataToBeFormatted | The subsystem will send data to the Data Formatter | None | None |

Table 6‑7: Interlayer Interfaces (PixelSense Transfer Manager)

#### Public Interfaces

None

### Lynx Transfer Manager

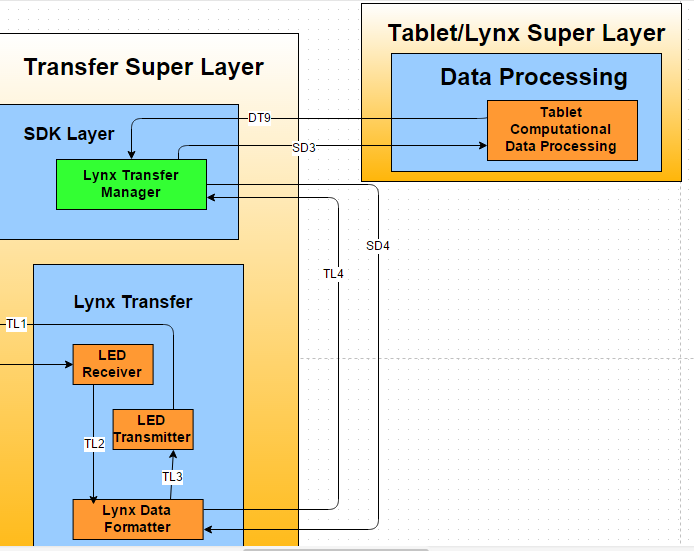


Figure 6‑8: Lynx Transfer Manager

#### General Description

The Lynx Transfer Manager’s purpose is to take the processed data from the Data Processing subsystem and sending it off for formatting to the Data Formatter

#### Assumptions

Processed Data must be in the appropriate digital form so the subsystem can translate to analog data.

#### Responsibilities

The responsibility of the Lynx Transfer Manager is to send and receive data from the processing subsystem as well as the Data Formatter, it acts like a middle translator for digital-analog and vice-versa.

#### Inter-Layer Interfaces

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Description** | **Information Required** | **Information Returned** |
| GetLynxProcessedData | The subsystem will retrieve data from the Lynx Computational Data Processing Subsystem. | None | None |
| SendLynxProcessedData | The subsystem will send data back to the Lynx Computational Data Processing Subsystem for output feedback back to the user | None | None |
| GetLynxFormattedData | The subsystem will get back formatted data from the Data Formatter | None | None |
| SendLynxDataToBeFormatted | The subsystem will send data to the Data Formatter | None | None |

Table 6‑8: Interlayer Interfaces (Lynx Transfer Manager)

#### Public Interfaces

None

# Inter-Subsystem Data Flows

In this section, the architectural layers for the whole system will be broken down and explained. The breakdown will show the individual super layers, layers and subsystems. The communication between each subsystem through data flows will also be distinguished from each other. Each subsystem is an important part and function of the layer and super layer that it resides in and the data flow between them is what makes the entire system functional.

## Data Flow Diagram

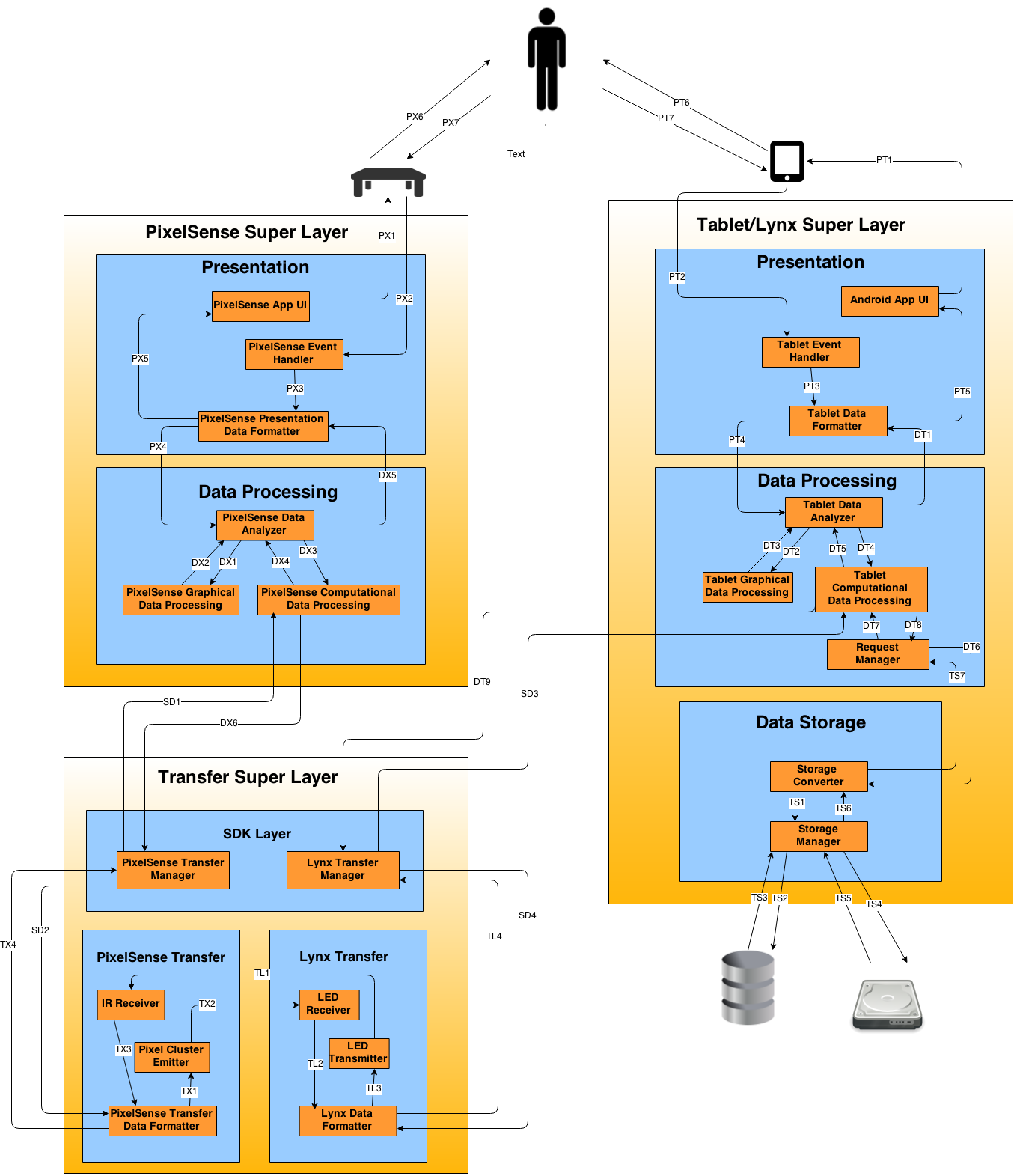
The following diagram shows the architecture within the entire system which shows each layer, subsystem and the data flow between each of the components. The diagram shows the flow between each subsystem and layer, this in turn depicts how each of them interact with each other in order to function correctly. 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 7‑1: Dataflow Diagram

## Data Flow Definitions

The table below will breakdown each of the data flows in the architectural diagram individually. It will show the source, destination and a brief description of what the dataflow contains and what it actually is. The table will show information passed between each subsystem, in turn it provides a better understanding of how the system works as a whole as well as the smaller parts.

|  |  |  |  |
| --- | --- | --- | --- |
| **Data Element ID** | **Description** | **Source** | **Destination** |
| PX1 | PixelSense App UI displays the application onto the PixelSense Hardware | PixelSense App UI | PixelSense Table |
| PX2 | The event handler receives the recognized input and takes the inputted data. | PixelSense Table | PixelSense Event Handler |
| PX3 | The data from the event handler is sent to the PixelSense Presentation Data Formatter where the data will be formatted appropriately for the next subsystem | PixelSense Event Handler | PixelSense Presentation Data Formatter |
| PX4 | After the data is formatted, it will be sent to the PixelSense Data Analyzer where it will then go to a different subsystems depending on what kind of data it is. | PixelSense Presentation Data Formatter | PixelSense Data Analyzer |
| PX5 | After the data is formatted from the PixelSense Data Analyzer, the PixelSense Data Formatter will send the formatted data to the PixelSense App UI, where there it will update the application appropriately | PixelSense Presentation Data Formatter | PixelSense App UI |
| PX6 | PixelSense Table shows the application to the User, where the user will then make some kind of reaction | PixelSense Table | User |
| PX7 | User provides an input to the system application | User | PixelSense Input Controller |
| DX1 | After the data is analyzed, the PixelSense Data Analyzer will send all data related to graphics to the PixelSense Graphical Data Processing | PixelSense Data Analyzer | PixelSense Graphical Data Processing |
| DX2 | After the graphics are calculated and computed, the graphics changes will be sent back to the PixelSense Data Analyzer | PixelSense Graphical Data Processing | PixelSense Data Analyzer |
| DX3 | From the PixelSense Data Analyzer, if it finds non-graphical data to process, then it will send the data to the PixelSense Computational Data Processing subsystem | PixelSense Data Analyzer | PixelSense Computational Data Processing |
| DX4 | After the non-graphical data is processed, it is then sent to the back to the analyzer to pair it with the graphical data. | PixelSense Computational Data Processing | PixelSense Data Analyzer |
| DX5 | Once the data is received from the graphical and non-graphical processors, the PixelSense Data Analyzer will then send the data to the PixelSense Presentation Data formatter to be put in a usable way. | PixelSense Data Analyzer | PixelSense Presentation Data Formatter |
| DX6 | After the data is processed, it is then sent to the transfer manager where the SDK translation begins for analog optical transfer. | PixelSense Computational Data Processing | PixelSense Transfer Manager |
| SD1 | After the SDK translation to digital happens, it is then sent to the data processor to be processed and understood what the received data is. | PixelSense Transfer Manager | PixelSense Computational Data Processing |
| SD2 | After the SDK translation to analog happens, it is then sent to the formatter to be formatted in a way so it can be optically transferred. | PixelSense Transfer Manager | PixelSense Transfer Data Formatter |
| TX1 | After the data is appropriately formatted, the data is then sent to the emitter where the data will be transferred optically. | PixelSense Transfer Data Formatter | Pixel Cluster Emitter |
| TX2 | The data is now transferred over from the table to the Lynx | Pixel Cluster Emitter | LED Receiver |
| TX3 | The optical data received is sent to the formatter to be formatted to the SDK will understand what it is. | IR Receiver | PixelSense Transfer Data Formatter |
| TX4 | The formatted data is now sent to the transfer manager where the data will be converted from analog to digital format so the processing subsystem will understand the data. | PixelSense Transfer Data Formatter | PixelSense Transfer Manager |
| TL1 | The data is now transferred over from the Lynx to the table | LED Transmitter | IR Receiver |
| TL2 | Once receiving the light data, the individual digital highs and lows will be sent to the Lynx Transfer Data Formatter to be formatted into something useful | LED Receiver | Lynx Data Formatter |
| TL3 | After the data is appropriately formatted, the data is then sent to the emitter where the data will be transferred optically. | Lynx Data Formatter | LED Transmitter |
| TL4 | The formatted data is now sent to the transfer manager where the data will be converted from analog to digital format so the processing subsystem will understand the data. | Lynx Data Formatter | Lynx Transfer Manager |
| PT1 | Android App UI displays the application onto the Android Hardware | Android App UI | Android Device |
| PT2 | The event handler receives the recognized input and takes the inputted data. | Android Device | Tablet Event Handler |
| PT3 | The event handler is sent to the Tablet Data Formatter where the data will be formatted appropriately for the next subsystem | Tablet Event Handler | Tablet Data Formatter |
| PT4 | After the data is formatted, it will be sent to the Tablet Data Analyzer where it will then go to a different subsystems depending on what kind of data it is. | Tablet Data Formatter | Tablet Data Analyzer |
| PT5 | After the data is formatted from the Tablet Data Analyzer, the Tablet Data Formatter will send the formatted data to the Tablet App UI, where there it will update the application appropriately | Tablet Data Formatter | Android App UI |
| PT6 | The Android device shows the application to the User, where the user will then make some kind of reaction | Android Device | User |
| PT7 | User provides an input to the Android Device | User | Android Device |
| DT1 | Once the data is received from the graphical and non-graphical processors, the Tablet Data Analyzer will then send the data to the Tablet Data formatter to be put in a usable way. | Tablet Data Analyzer | Tablet Data Formatter |
| DT2 | After the data is analyzed, the Tablet Data Analyzer will send all data related to graphics to the Tablet Graphical Data Processing | Tablet Data Analyzer | Tablet Graphical Data Processing |
| DT3 | After the graphics are calculated and computed, the graphics changes will be sent back to the Tablet Data Analyzer | Tablet Graphical Data Processing | Tablet Data Analyzer |
| DT4 | From the Tablet Data Analyzer, if it finds non-graphical data to process, then it will send the data to the Tablet Computational Data Processing subsystem | Tablet Data Analyzer | Tablet Computational Data Processing |
| DT5 | After the non-graphical data is processed, it is then sent to the back to the analyzer to pair it with the graphical data. | Tablet Computational Data Processing | Tablet Data Analyzer |
| DT6 | The request manager sends a request to the storage converter to receive some kind of data. | Request Manager | Storage Converter |
| DT7 | The requested data from storage or an error message was sent back to the processing layer. | Request Manager | Tablet Computational Data Processing |
| DT8 | The processing layer lets the request manager know it needs something from storage so that the request manager can build a request for that information. | Tablet Computational Data Processing | Request Manager |
| DT9 | The processed data is sent to the SDK Lynx Transfer Manager so that the data can begin its transition from digital data to optical analog data. | Tablet Computational Data Processing | Lynx Transfer Manager |
| TS1 | The storage converter converts the request to something the manager can understand | Storage Converter | Storage Manager |
| TS2 | Data is stored to the Database | Storage Manager | Database |
| TS3 | Data is retrieved from the Database | Database | Storage Manager |
| TS4 | Data is stored on some kind of storage medium (Non-volatile) Flash Memory) | Storage Manager | Non-Volatile Flash Memory |
| TS5 | Data is retrieved from the storage medium. | Non-Volatile Flash Memory | Storage Manager |
| TS6 | The retrieved data is sent to the storage converter for conversion | Storage Manager | Storage Converter |
| TS7 | The converter sends the appropriate data back to the request manager | Storage Converter | Request Manager |

Table 7‑1: Dataflow Definitions

## Producer-Consumer Relationship Table

The table below depicts the relationships between each subsystem between data flows in a more visually aspect for a different perspective of the dataflow.

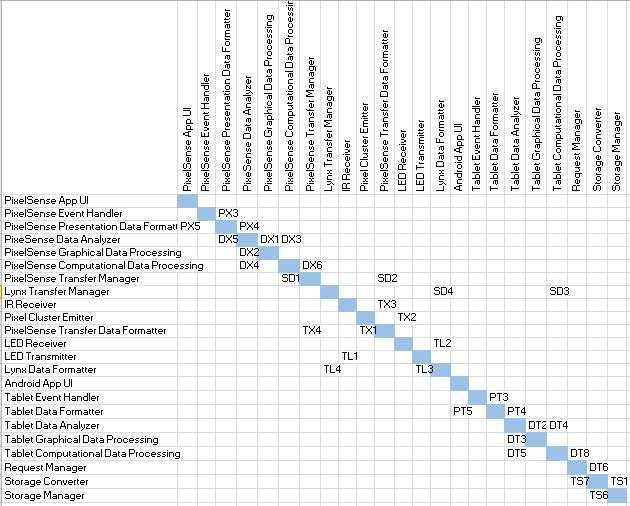


Table 7‑2: Producer-Consumer Relationship

## Producer-Consumer Relationship Analysis

The table in section 7.3 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 subsystem is either complex or a bottleneck to the entire system as a whole. For table 25, it can be seen that the most of the subsystems are pretty well balanced with the most being 3 in/out for each Data Analyzer. With that said the data analyzers can be seen as a moderate priority subsystem since only a few other subsystems 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.

# Requirements Mapping

## Requirements Traceability Matrix

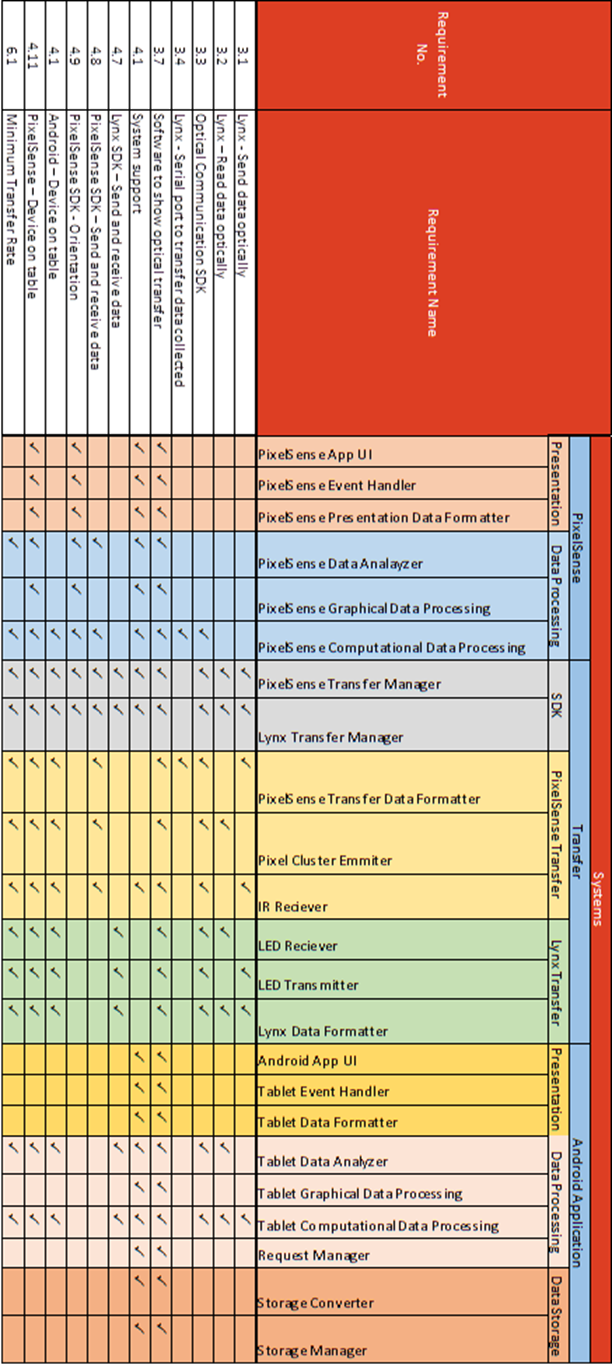


Table 8‑1: Requirements Traceability Matrix

## 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 is 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.

# Operating System Dependencies

This section contains any operating systems dependencies for each layer within the systems of our architectural design, including any external class libraries or functions which could be used. Our layers are designed to take advantage of the hardware we are working on.

## PixelSense Super Layer

Our PixelSense layers will be reliant on the functions provided by both Windows 7 operating system and the Surface 2.0 SDK designed for Windows 7. We will be using certain functions of the SDK, such as item tracking and recognition, in conjunction with functionality we create, In order to properly control our systems.

## Tablet/Lynx Super Layer

### Presentation Layer

The Presentation Layer will be dependent on the GUI options given by Android 4.0. We may also rely on external GUI libraries for Android if we find any that suit our purpose.

### Data Processing Layer

The Data Processing Layer is dependent on the functions and data processing capabilities provided by Android 4.0.

### Data Storage Layer

We will be using the SQLite database options provided by Android 4.0 in order to store data within our Data Storage Layer. We will also database viewer software on Windows PC in order to view the data stored in our database.

## Transfer Super Layer

The Transfer Layer, covering the functionality of the device we are creating (Lynx), will be dependent on the operating system of the embedded system (Arduino) we chose. This will be decided once our hardware choices are finalized

# Testing Considerations

This section describes how the team will ensure that final product adheres the architecture defined within this document. The examination criteria includes compliance with individual component criteria, defined in their own section, as well as the overarching design principles that influence the system as a whole. Described below is the overall testing objectives for this system, as well as specific criteria and considerations for each Super Layer in the system.

## Overall Considerations

**Code Compliance:** each component of each Layer will be cross-examined, its documentation as well as its eventual test output, against its definition in this document. It shall be rated on how effective it is at performing the overall task it is supposed to represent within the architecture. This criteria involves gathering appropriate input from neighboring components (or interpreting test data correctly), output correct data, and that all work is being done inside its defined role or layer (e.g. a component from the transfer layers should not be outputting to a screen under normal conditions, that is out of scope).

**Principle Adherence:** Every component must adhere to the defining principles laid out in section two of this document. When communicating between components, under normal working conditions, strict format compliance will be enforced to ensure proper granularity when translating and transferring. Also, Communication between Super Layers will be grounds for component failure unless communication is specifically mentioned in this document, in order to keep Super Layers as independent as possible.

## PixelSense Layer Considerations

Special consideration must be made when testing any component within the Super Layer’s Data Processing Layer, and those components that communicate with it. In order to properly test that components translate data from the format of the prior component to the next, these components will communicate will a diagnostic system, such as a writable file, in order to log data as is it travels through the Super Layer. During individual component testing, each one shall be configured to also run off of prepared, formatted data, to facilitate more modular testing before larger implementation. This additional communication will be purged from the system in time for delivery.

### PixelSense Presentation Layer

**Modularity:** The Presentation Layer to some extent will depend on the Data Processing Layer in that the content in the PixelSense Presentation Layer will be from the PixelSense Data Processing Layer. Other than that there should be no other dependencies for the Presentation Layer.

**Internal-System Interaction:** This layer and its subsystem will only be communicating with the Data Processing Layer of the PixelSense side.

**External-System Interaction:** This layer will directly communicate with the PixelSense Table Hardware as well as the User themselves. He/she will be able to play the game and provide various inputs.

### PixelSense Data Processing Layer

**Modularity:** The PixelSense Data Processing Layer must be a stand-alone layer and not be dependent on the internals of another layer.

**Internal-System Interaction:** This layer will receive and send data to the PixelSense Presentation Layer and the PixelSense Transfer Layer. The data flow will be bidirectional with both the Presentation Layer and Transfer Layer.

**External-System Interaction:** There are no external connections with this layer.

## Transfer Super Layer Considerations

In regards to the PixelSense layer within this Super Layer, testing may conform to overall testing criteria, and the special considerations from the PixelSense Super Layer, as these components are still natively in the table. An additional consideration is to be made for the step involving the transfer to light, which cannot be written to a file. For simplicity, components up until the light transfer will right its output to the same system in the PixelSense Super Layer. Likewise, components not dealing with light transfer directly from the Lynx transfer layer will write to the diagnostic system in the Lynx Super Layer. The components sending and receiving light will write there to opposite systems (PixelSense to Lynx, and vice-versa). This additional communication will be purged from the system in time for delivery.

### PixelSense Transfer Layer

**Modularity:** The PixelSense Transfer Layer must be a stand-alone layer and not be dependent on the internals of another layer.

**Internal-System Interaction:** This layer will receive and send data to the SDK Layer and the PixelSense Data Processing Layer. The data flow will be bidirectional with both the PixelSense Data Processing Layer and SDK Layer.

**External-System Interaction:** There are no external connections with this layer.

### Lynx Transfer Layer

**Modularity:** The Lynx Transfer Layer must be a stand-alone layer and not be dependent on the internals of another layer.

**Internal-System Interaction:** This layer will receive and send data to the SDK Transfer Layer and the Lynx/Tablet Data Processing Layer. The data flow will be bidirectional with both the Tablet/Lynx Data Processing Layer and SDK Transfer Layer.

**External-System Interaction:** There are no external connections with this layer.

### SDK Layer

**Modularity:** The SDK Layer must be a stand-alone layer and not be dependent on the internals of another layer.

**Internal-System Interaction:** This layer will receive and send data to the PixelSense Transfer Layer and the Lynx/Tablet Transfer Layer. The data flow will be bidirectional with both the Tablet/Lynx Transfer Layer and PixelSense Transfer Layer.

**External-System Interaction:** There are no external connections with this layer.

## Tablet/Lynx Super Layer Considerations

All additional considerations for the Lynx Super Layer will follow the same format as the PixelSense Super Layer considerations. The difference will be that the diagnostic system will be local to this Super Layer. This additional communication will be purged from the system in time for delivery.

### Tablet/Lynx Presentation Layer

**Modularity:** The Lynx/Tablet Presentation Layer to some extent will depend on the Lynx/Tablet Data Processing Layer in that the content in the Lynx/Tablet Presentation Layer will be from the Lynx/Tablet Data Processing Layer. Other than that there should be no other dependencies for the Presentation Layer.

**Internal-System Interaction:** This layer and its subsystem will only be communicating with the Data Processing Layer of the Tablet/Lynx side.

**External-System Interaction:** This layer will directly communicate with the Android Tablet Hardware as well as the User themselves. He/she will be able to play the game and provide various inputs.

### Tablet/Lynx Data Processing Layer

**Modularity:** The Lynx/Tablet Data Processing Layer must be a stand-alone layer and not be dependent on the internals of another layer.

**Internal-System Interaction:** This layer will receive and send data to the Lynx/Tablet Presentation Layer and the Transfer Layer. The data flow will be bidirectional with both the Lynx/Tablet Presentation Layer and Lynx/Tablet Transfer Layer.

**External-System Interaction:** There are no external connections with this layer.

### Tablet/Lynx Storage Layer

**Modularity**: The Storage Layer must be a stand-alone layer and not be dependent on the internals of another layer.

**Internal-System Interaction**: This layer will receive and send data to the Lynx/Tablet Data Processing Layer. The data flow will be bidirectional with the Lynx/Tablet Data Processing Layer.

**External-System Interaction**: The layer will externally interact with a Database as well as some flash memory for data storage.