6-9-2016

TrusNet: Peer-to-Peer Cryptographic Authentication

Adrian Bedard
Santa Clara University

Jonathan Bedard
Santa Clara University

SANTA CLARA UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

Date: June 9, 2016

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Adrian Bedard
Jonathan Bedard

ENTITLED

TrusNet: Peer-to-Peer Cryptographic Authentication

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

[illegible]
Thesis Advisor

[illegible]
Department Chair

TrusNet: Peer-to-Peer Cryptographic Authentication

by

Adrian Bedard
Jonathan Bedard

Submitted in partial fulfillment of the requirements
for the degree of
Bachelor of Science in Computer Science and Engineering
School of Engineering
Santa Clara University

Santa Clara, California
June 6, 2016

TrusNet: Peer-to-Peer Cryptographic Authentication

Adrian Bedard
Jonathan Bedard

Department of Computer Engineering
Santa Clara University
June 6, 2016

ABSTRACT

Originally, the Internet was meant as a general purpose communication protocol, transferring primarily text documents between interested parties. Over time, documents expanded to include pictures, videos and even web pages. Increasingly, the Internet is being used to transfer a new kind of data which it was never designed for. In most ways, this new data type fits in naturally to the Internet, taking advantage of the near limit-less expanse of the protocol. Hardware protocols, unlike previous data types, provide a unique set security problem. Much like financial data, hardware protocols extended across the Internet must be protected with authentication. Currently, systems which do authenticate do so through a central server, utilizing a similar authentication model to the HTTPS protocol. This hierarchical model is often at odds with the needs of hardware protocols, particularly in ad-hoc networks where peer-to-peer communication is prioritized over a hierarchical model. Our project attempts to implement a peer-to-peer cryptographic authentication protocol to be used to protect hardware protocols extending over the Internet.

The TrusNet project uses public-key cryptography to authenticate nodes on a distributed network, with each node locally managing a record of the public keys of nodes which it has encountered. These keys are used to secure data transmission between nodes and to authenticate the identities of nodes. TrusNet is designed to be used on multiple different types of network interfaces, but currently only has explicit hooks for Internet Protocol connections.

As of June 2016, TrusNet has successfully achieved a basic authentication and communication protocol on Windows 7, OSX, Linux 14 and the Intel Edison. TrusNet uses RC-4 as its stream cipher and RSA as its public-key algorithm, although both of these are easily configurable. Along with the library, TrusNet also enables the building of a unit testing suite, a simple UI application designed to visualize the basics of the system and a build with hooks into the I/O pins of the Intel Edison allowing for a basic demonstration of the system.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

In the past 15 years, the Internet has transitioned from a method of exchanging simple text documents to an information super-highway, carrying images, video, files and embedded datagrams around the world at unprecedented speeds. In the Internet's early years, hierarchical security protocols were implemented for the exchange of sensitive financial data. These protocols have since been extended to include many modern websites and server-client communication. However, as embedded protocols are increasingly extended over IP, in what is known as the "Internet of Things," the hierarchical security model has begun to break down. Increasingly, Internet communications are occurring between peers, extending old hardware protocols in insecure ways.

1.2 Current Solutions

Current solutions fall into two broad camps; hierarchical and Peer-to-Peer. Often, the hierarchical model is re-adapted for the Internet of Things. CoAP (Constrained Application Protocol) offers secure networking for the Internet of Things. CoAP uses an authentication system similar to HTTPS, in which public key cryptography is used to exchange symmetric keys and sign message hashes to establish identity. Additionally, CoAP targets cheap systems with limited CPU and RAM. However, CoAP is hierarchical, thus it does not offer Peer-to-Peer authentication. Nodes authenticate with the server, but not each-other (Bormann 2015).

1.2.1 Peer-to-Peer

Peer-to-Peer technologies, such as ZigBee, 6LoWPAN, and Bluetooth, allow for easy connection between peers. Such communication is often encrypted to prevent eavesdropping, but both ZigBee and Bluetooth are notorious for having issues with authentication and spoofing (Patel 2015, Niem 2002). Even Wifi has historically had problems with authentication and spoofing. This is especially relevant as Wifi nodes are increasingly used as local access points, such as Parrot's fleet of drones.

1.2.2 Hierarchical

Hierarchical protocols require connection to a server over IP to authenticate. This limitation forces many Internet of Things applications to use un-authenticated short-range protocols instead because hierarchical authentication protocols cannot work over such patch-work ad-hoc networks. Peer-to-Peer technologies, however, have the opposite problem. Peer-to-Peer technologies allow local connection, without the burden of sending packets to the server. However, Peer-to-Peer protocols do not authenticate, leaving them open to spoofing and impersonation. Such attacks are extremely common in Peer-to-Peer connections, and are the source of many security breaches in the emerging Internet of Things.

1.3 Proposal

Our project has secured these peer-to-peer communications through a novel form of peer authentication, which allows the expanding Internet of Things to communicate robustly and securely across networks of all types. We have leveraged existing authentication technology at the Peer-to-Peer level preventing both spoofing and impersonation attacks. Unlike Client-Server protocols, our protocol does not require a connection to IP for authentication, allowing for extension over ad-hoc networks.

Chapter 2

Project Requirements

Requirements are a set of statements which describe the needs and desires of the entities using the system. In the case of this project, a set of requirements for the underlying networking library, TrusNet, has been separated from a set of requirements for the demonstration application. The demonstration application will be used to visualize and demonstrate the networking protocol on embedded systems, and will actually consist of a suite of related applications communicating together.

2.1 TrusNet Library

The TrusNet Library is the core of the project, and provides the structural solution to the authentication problem in peer-to-peer networks. This library is designed to be linked and utilized by other applications, including the Demonstration Application.

2.1.1 Functional Requirements

The Functional Requirements of the TrusNet Library describe what the library must be and be capable of. These requirements serve as the primary description of the unique core capabilities of the TrusNet library.

- API for expansion
    - Adds new methods of hardware communication such as Bluetooth or ZigBee
- Prevent Active Attacks
    - Man-in-the-Middle

    - Node Spoofing

- Peer-to-Peer
    - No hierarchy required
    - Locally managed security
- Integrates with WiFi

2.1.2 Non-Functional Requirements

The Non-Functional Requirements of the TrusNet Library describe qualities the library will have. These qualities help determine the types of systems and applications where TrusNet will be the best-suited networking solution.

- Light-weight
    - Minimal outside dependencies
    - Low memory usage
- Simple to set up
- Runtime-stable
- Fast after authentication
- Smooth inter-platform communication

2.2 Demonstration Application

The Demonstration Application links to TrusNet and provides a visual representation of the library in action. This application will consist of multiple different nodes allowing for clear execution common use cases in the TrusNet Library.

2.2.1 Functional Requirements

The Functional Requirements of the Demonstration Application describe what the suite of applications must be and do. Since the Demonstration Application contains TrusNet, it adopts most of the Functional Requirements of TrusNet as well, with the exception of the expansion API.

- Utilize the TrusNet Library
- Peer-to-Peer Protocol
  - Bluetooth
  - ZigBee
- Multiple Nodes
  - UI Integrated Node
  - Headless Nodes
- Multi-platform

2.2.2 Non-Functional Requirements

The Non-Functional Requirements of the Demonstration Application describe qualities the application suite will have. Similarly, the Demonstration Application implicitly adopts all of the Non-Functional Requirements of the TrusNet Library.

- Demonstrate the core features of TrusNet
- Intuitive UI
- Simple set-up
- Engaging

2.3 Design Constraints

The Design Constraints describe the limitations on design that the developers have determined as a design choice. The Design Constraints apply to both the TrusNet library and the Demonstration Application.

- Written in C++
- Run on 32 and 64 bit architecture
- Cross Platform
  - Windows
  - Unix
    - Ubuntu
    - Mac OSX
    - Yocto (Headless)

Chapter 3

Use Cases

A use case describes how a type of user will experience the system. This section is separated into the use cases for normal functionality, hostile interaction and demonstration cases.

3.1 Normal API

Use cases in this section come from the TrusNet library itself. Many of the more complicated uses cases rely off of a combination of use cases from this section.

3.1.1 Set-Up

The set-up use case initializes the TrusNet library. All subsequent cases rely off of each node legally setting up the TrusNet library.

- Actor: Each node
- Goal: Prepare code for the use of the TrusNet API.
- Preconditions:
  1. API is available
- Postconditions:
  1. API can be used
- Steps:

    1. Load API into build path
    2. Include headers
    3. Generate keys

3.1.2 Authentication

The authentication case gives TrusNet its security. Each node must authenticate its connection with all other nodes it intends to transmit data to, or receive data from.

- Actor: TrusNet API
- Goal: Authenticate a node
- Preconditions:
    1. Unauthenticated node
- Postconditions:
    1. Node is authenticated
- Steps:
    1. Exchange keys
    2. Reference keys against internal and external key libraries
    3. Verify node
- Exceptions:
    1. Note is rejected

3.1.3 Transmit

All data is exchanged using this case, with the exception of the messages exchanged for authentication. Even network meta-data is exchanged through the transmit case.

- Actor: TrusNet API call
- Goal: Send a piece of data to another node

- Preconditions:
  1. Receiver node authenticated
- Postconditions:
  1. Data transmitted
- Steps:
  1. Call transmit function in code
  2. Data is encrypted and transmitted to given node
- Exceptions:
  1. Receiver node is not known

3.1.4 Receive

This case pairs with the "Transmit" case. As with "Transmit," all data is received through this case, except for authentication data.

- Actor: TrusNet API
- Goal: Receive and verify data
- Preconditions:
  1. Data has been received
- Postconditions:
  1. Data is provided to a program
- Steps:
  1. Authenticate received data
  2. Decrypt data
  3. Provide data to user program

- Exceptions:
    1. Data is invalid
    2. Sender node is unauthorized

3.1.5 Shut-down

This case prevents nodes from attempting to communicate with a node no longer attached to the network. Note that connections will time out if a node shutting down does not notify the network.

- Actor: User
- Goal: Let other nodes know that this node is no longer available
- Preconditions:
    1. Call shut-down
- Postconditions:
    1. Node is shut down and no longer assigned data
- Steps:
    1. End all broadcasts
    2. Broadcast shut-down signal to adjacent nodes

3.1.6 Permission Shifting

In the TrusNet system, some nodes are headless. This means that headless nodes must allow their permissions and network settings to be modified by another node.

- Actor: User Node
- Goal: Change permissions of headless node
- Preconditions:
    1. Authenticated connection

    2. User node has appropriate permission

- Postconditions:
    1. Headless node permission changes
- Steps:
    1. User node specifies permission through UI
    2. Permission change order sent
    3. Headless node changes permission
- Exceptions:
    1. User node lacks permission status

3.1.7 Rebuild-Key

Over time, public-keys are at risk for compromise. Because of this, it is recommended that every 6-12 months, each node re-generates their key pair.

- Actor: User
- Goal: Regenerate verification keys
- Postconditions:
    1. Keys are regenerated
- Steps:
    1. Call key regenerate
    2. Save old key

3.2 Hostile Interactions

Because the TrusNet library is a security library, some users and use cases are hostile. This section of use cases outlines some common uses cases where a user is hostile.

3.2.1 Eavesdropping

If a hostile node is eavesdropping, it attempts to read messages passing between two nodes communicating through the TrusNet protocol. Figure 3.1 visualizes this attack.

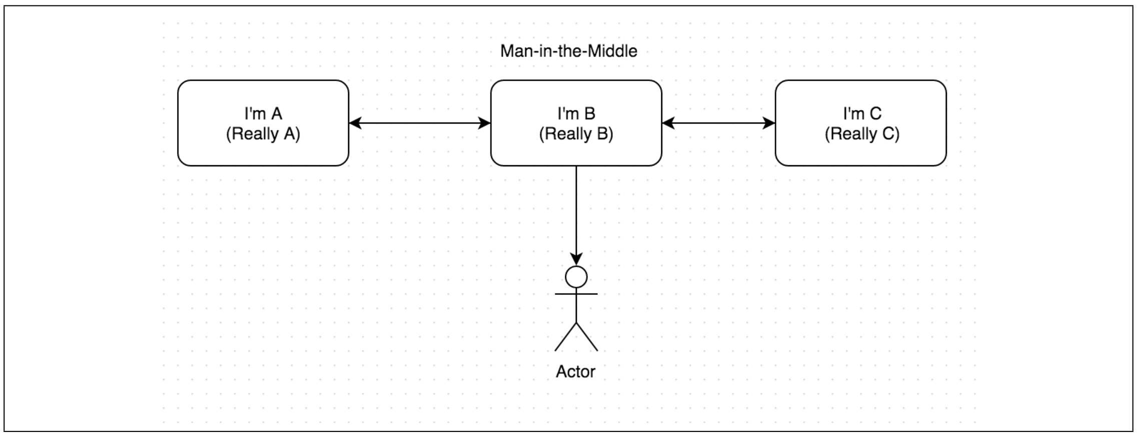


Figure 3.1 Eavesdropping

- Actor: Externally controlled node
- Goal: Demonstrate TrusNet's eavesdropping protections
- Preconditions:
    1. Network is operational
- Postconditions:
    1. Network integrity is demonstrated
- Steps:
    1. Begin eavesdropping
    2. Attempt to decrypt messages

3.2.2 Node Spoofing

During a node-spoofing attack, a hostile actor attempts to impersonate a node on the TrusNet network. Figure 3.2 demonstrates this attack. When a man-in-the-middle attack essentially combines a spoofing attack with an eavesdropping attack.

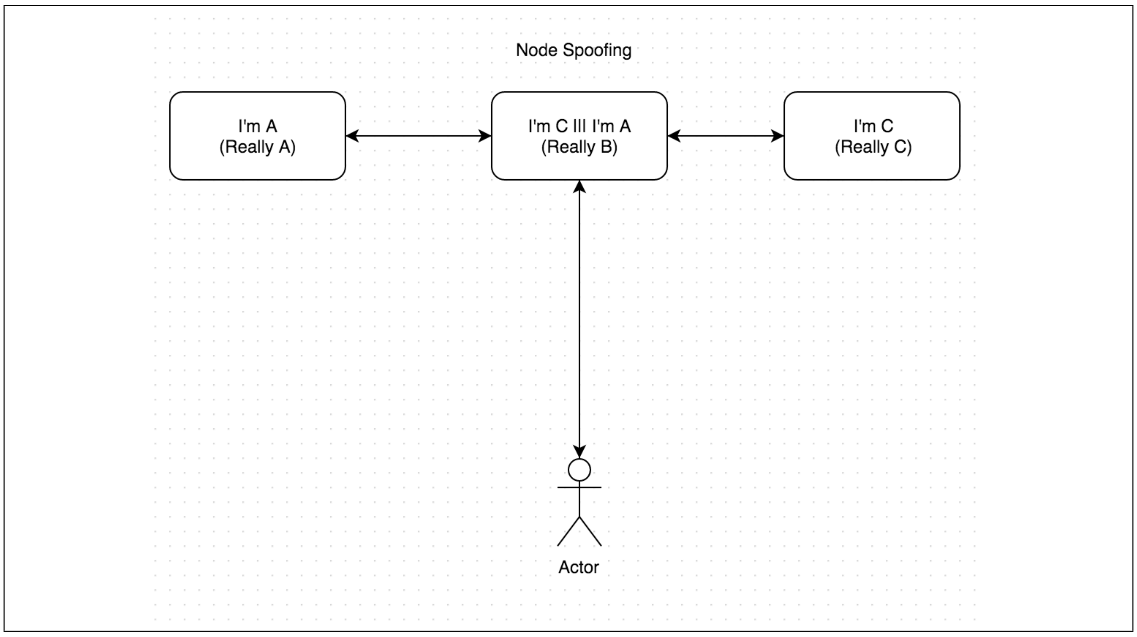


Figure 3.2 Node spoofing attack

- Actor: Externally controlled node
- Goal: Demonstrate TrusNet's spoofing protections
- Preconditions:
  1. Network is active
- Postconditions:
  1. Network integrity is demonstrated
- Steps:
  1. Activate spoofing
  2. Attempt to decrypt
  3. Attempt to fake data transmission

3.2.3 DOS Attack

A denial of service attack aims to flood TrusNet with bogus messages, preventing the exchange of legitimate messages. Figure 3.3 outlines this particular use case.

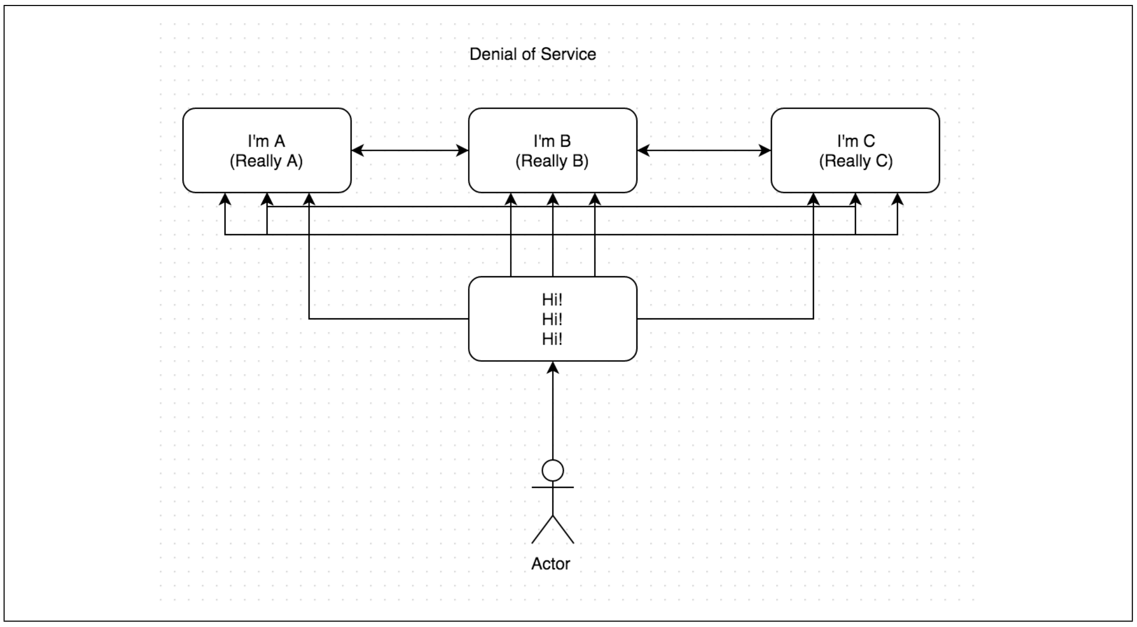


Figure 3.3 Denial of service attack

- Actor: Externally controlled node
- Goal: Demonstrate TrusNet's DOS protections
- Preconditions:
    1. Network is active
- Postconditions:
    1. Network integrity is demonstrated
- Steps:
    1. Spam network with data
    2. Note how other nodes ignore and reroute around DOS node

3.2.4 Permission Spamming

Permission spamming is an attempt to subvert the security demanded by the permission shifting use case. Figure 3.4 outlines how this attack works.

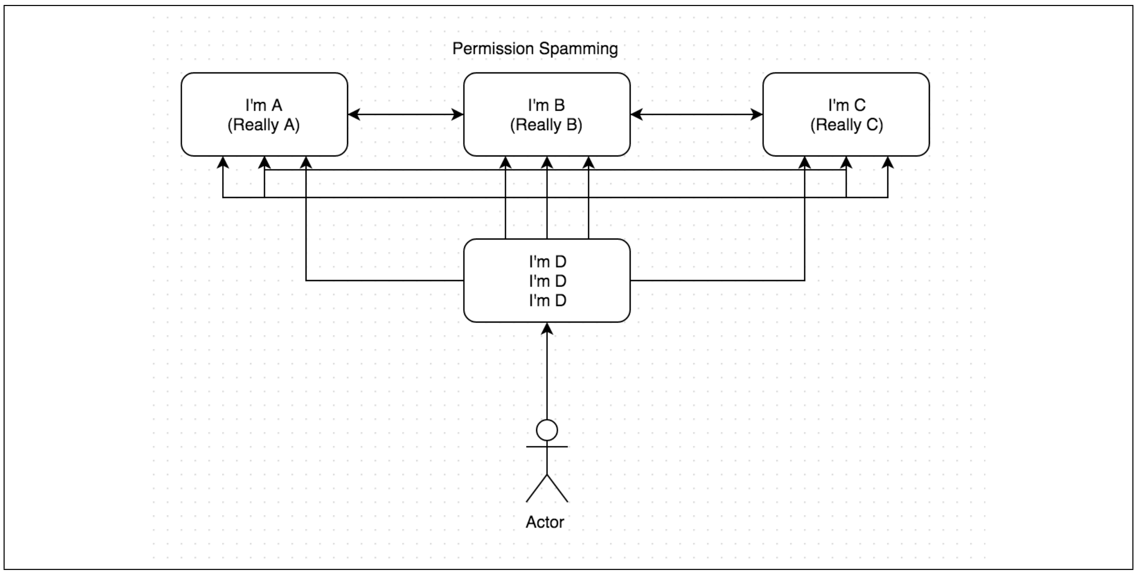


Figure 3.4 Permission spamming attack

- Actor: Externally controlled node
- Goal: Demonstrate TrusNet's permission spamming protections
- Preconditions:
    1. Network is active
- Postconditions:
    1. Network integrity is demonstrated
- Steps:
    1. Spam network with fake permission data
    2. Observe how the fake data is ignored by other nodes

3.3 Demonstration

The demonstration use cases are the cases which will visualize the underlying functionality of the library. The use cases depend heavily off of the Normal API use cases, and implicitly demonstrate the Hostile Interaction use cases.

3.3.1 Normal Operation

During normal operation, the demonstration platform will control an external robotics platform through the TrusNet library. Figure 3.5 visualizes this use case.

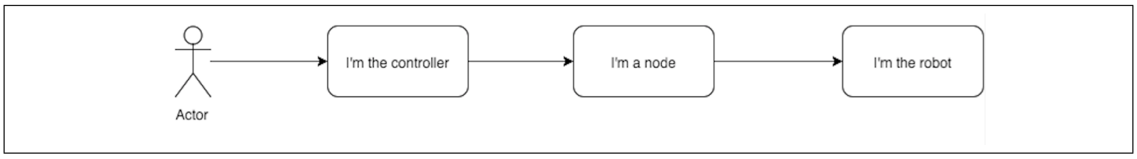


Figure 3.5 Normal operation

- Actor: Demonstrator
- Goal: Operate robot
- Preconditions:
    1. System is set-up
- Postconditions:
    1. System is demonstrated
- Steps:
    1. Operate joystick
    2. Observe robot react

3.3.2 Ad-Hoc Demonstration

This use case demonstrates the ad-hoc meshing abilities of the TrusNet library. This use case is outlined in Figure 3.6.

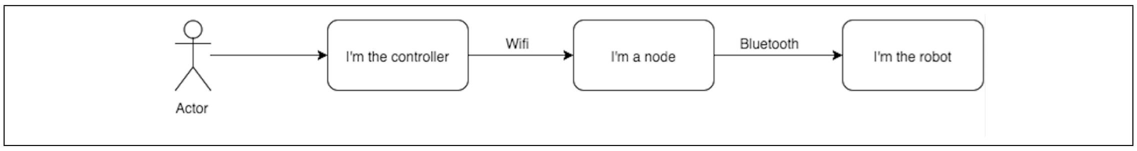


Figure 3.6 AD-HOC network

- Actor: Demonstrator

- Goal: Demonstrate ad-hoc nature of network
- Preconditions:
  1. System is operational
- Postconditions:
  1. Network adaptability demonstrated
- Steps:
  1. Deactivate part of joystick node
  2. Note how control remains as TrusNet reroutes signal through other nodes

3.3.3 Node Spoofing

In order to demonstrate the security of the system, this use case will attempt to spoof the identity of a valid node in the network, as shown in Figure 3.7. In order for this use case to succeed, the spoofed node must be rejected.

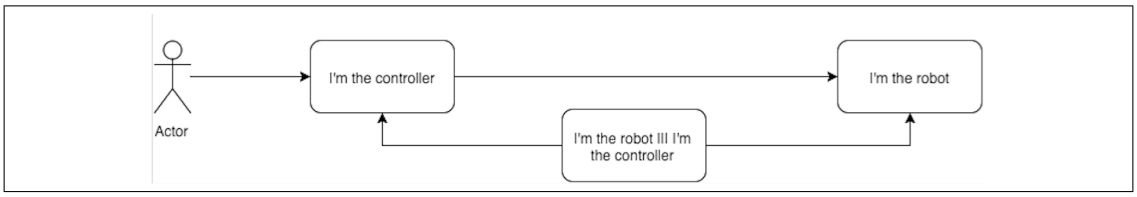


Figure 3.7 Attempting to spoof a node

- Actor: Demonstrator
- Goal: Demonstrate spoofing protections
- Preconditions:
  1. Network is operational
- Postconditions:
  1. Network integrity is demonstrated

- Steps:
    1. Activate spoof node as an intermediate node
    2. Note how signal is still broadcast despite spoofed node

Chapter 4

Conceptual Model

4.1 Overview

Our solution will consist of two Primary systems: the connection library and the network extension system. The connection library will allow a developer to connect a variety of devices. Any device running this library will also act as an additional node to further strengthen the surrounding network. The network extension system allows the user to add an additional method of communication as an available connection type.

4.2 Connection Library

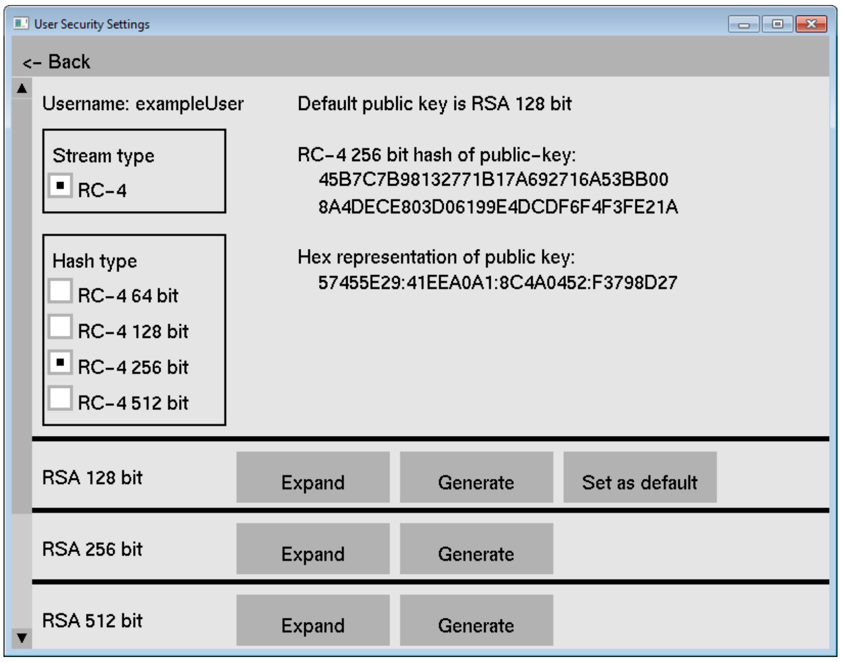
The connection library is a general purpose networking library, allowing for communication across multiple hardware mediums with fantastic security. In terms of usability, the system will function similar to TCP/IP packet transmission. This ensures that other programmers using our code will experience an easy transition. The library will be written in C++, for C++. As such, including the library will be a case of including a header file. This is the primary product of our overall project.

4.3 Network Extension System

The network extension system allows a user to add fundamentally new communication methods to the system. This helps to future proof the standard. Additionally, this allows for devices to communicate with other devices even if two given devices to not share a single hardware medium. In order to implement this, there will be a set of functions which must be implemented in C++. Once implemented, this new hardware option can be duplicated and shared, allowing for further optimization and refinement. This is a secondary element in the overall project.

4.4 Demonstration Project

The demonstration project is a simple system of nodes and devices designed to help visualize the function of our network. The demonstration involved a windows computer, an OSX computer, and two Edisons. Both Edisons were robot mounted. We demonstrated indirect communication, low latency, and network flexibility. Additionally, this helped us to better explain the unique advantages our system provides. The robot and nodes themselves do not represent a significant innovation beyond simply demonstrating other project developments.



Username: exampleUser

Stream type
[✓ RC-4]

Hash type
RC-4 64 bit
RC-4 128 bit
[✓ RC-4 256 bit]
RC-4 512 bit

Default public key is RSA 128 bit

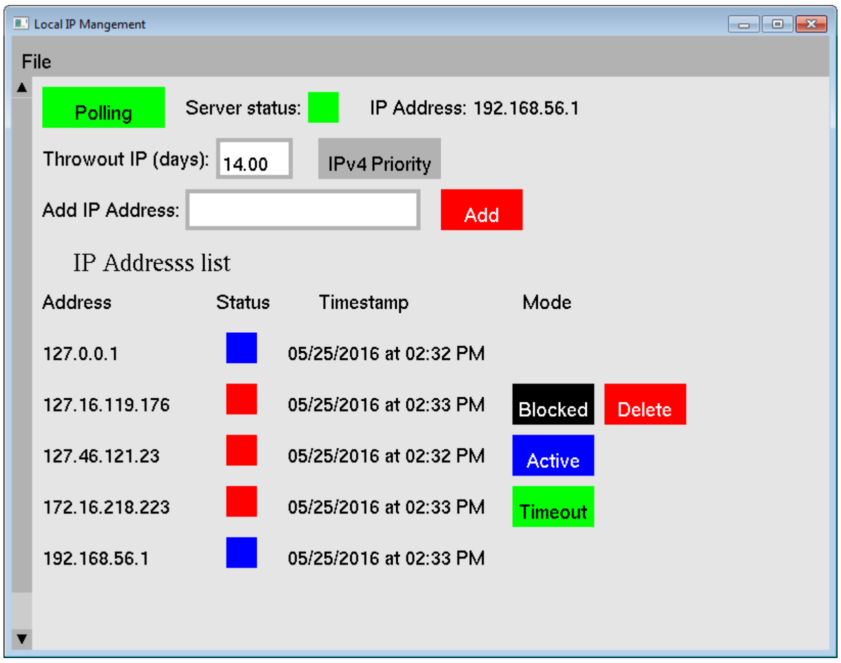
RC-4 256 bit hash of public-key:
45B7C7B98132771B17A692716A53BB00
8A4DECE803D06199E4DCDF6F4F3FE21A

Hex representation of public key:
57455E29:41EEA0A1:8C4A0452:F3798D27

RSA 128 bit
RSA 256 bit
RSA 512 bit

Figure 4.1 Key management

The demonstration project will visualize the operation of the TrusNet library. Figure 4.1 shows an example of the interface used to visualize the management of keys facilitated through the TrusNet library. Similarly, Figure 4.2 demonstrates how IP addresses are managed through the demonstration user interface. This user interface allows a user to see all connections for a given node and the status of said connections.


Local IP Mangement

File

Polling Server status: 🟩 IP Address: 192.168.56.1

Throwout IP (days): 14.00 IPv4 Priority

Add IP Address: Add

IP Addresss list

Address	Status	Timestamp	Mode	
127.0.0.1	🟦	05/25/2016 at 02:32 PM		
127.16.119.176	🟥	05/25/2016 at 02:33 PM	Blocked	Delete
127.46.121.23	🟥	05/25/2016 at 02:32 PM	Active	
172.16.218.223	🟥	05/25/2016 at 02:33 PM	Timeout	
192.168.56.1	🟦	05/25/2016 at 02:33 PM		

Figure 4.2 IP Address management

Chapter 5

Component State Charts

The component state charts describe the various states that the system can be in. In the TrusNet library, each state determines the functionality TrusNet will make available to a given node. TrusNet has two large classifications of state: Authentication states and permission states.

5.1 Authentication

Node authentication is the primary purpose of the TrusNet library. Connections between nodes have 5 authentication states:

- NULL State
- Disconnected
- Connected
- Secured
- Authenticated

The relationship between these states is shown, at a high level, in Figure 5.1. These states describe how each node tracks its connection with any other node in the network. The specific nature of each state will be over-viewed in the sub-sections below.

5.1.1 NULL State

The NULL state is the implicit state for the connection between any two nodes members of the same network, if they have no record of each-other. The NULL state is never explicitly specified,

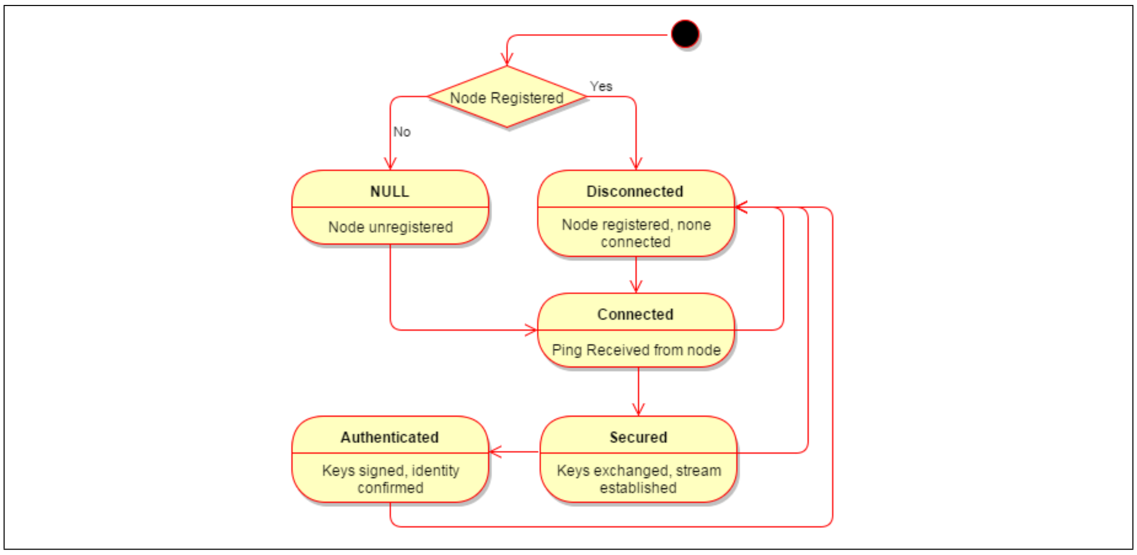


Figure 5.1 TrusNet authentication states

the moment nodes become aware of the existence of another node, the connection pair enters into either the disconnected or connected state.

5.1.2 Disconnected

In this state, two nodes are aware of the existence of each-other, and have previous record of a connection. The disconnected state implies that, at some point, the two nodes who are members of the connection state have been connected. The disconnected state does not imply that the connection between the two member nodes was ever secured or authenticated, meaning the saved public keys could be spoofed.

5.1.3 Connected

Two nodes enter into the connection state after they trade ping messages. A ping message contains the public key and name of the source node, the required information to enter the secure state. If an error occurs, or the connection times out, the node pair returns to the disconnected state.

5.1.4 Secured

The secured connection state guarantees the connection between two nodes is immune to eavesdropping. This is achieved by exchanging a stream-cipher seed through the public keys of each node, and using the stream-cipher to encrypt all futures messages passed between the node pair. If this stream is broken, or the connection times out, the node pair will return to the disconnected state.

5.1.5 Authenticated

After a secured stream is established, the pair request a signature from the other, confirming that both nodes have the private key pair to the advertised public key. Once this state is achieved, the connecting between the two nodes is considered both secure and authentic. It is only in the authenticated state that the TrusNet library will be able to send messages to a node and receive messages from it. If the authentication fails, the secured stream is broken or the connection times out, the node pair will return to the disconnected state.

5.2 Permissions

Permissions determine how much control a given node has over another given node. Each node keeps track locally of its own permission status on connected nodes, as well as record of the permission status of connected nodes with itself. These tracked permissions are functional permissions, nodes also keep local record of desired permissions as well, see Figure 5.2 for details.

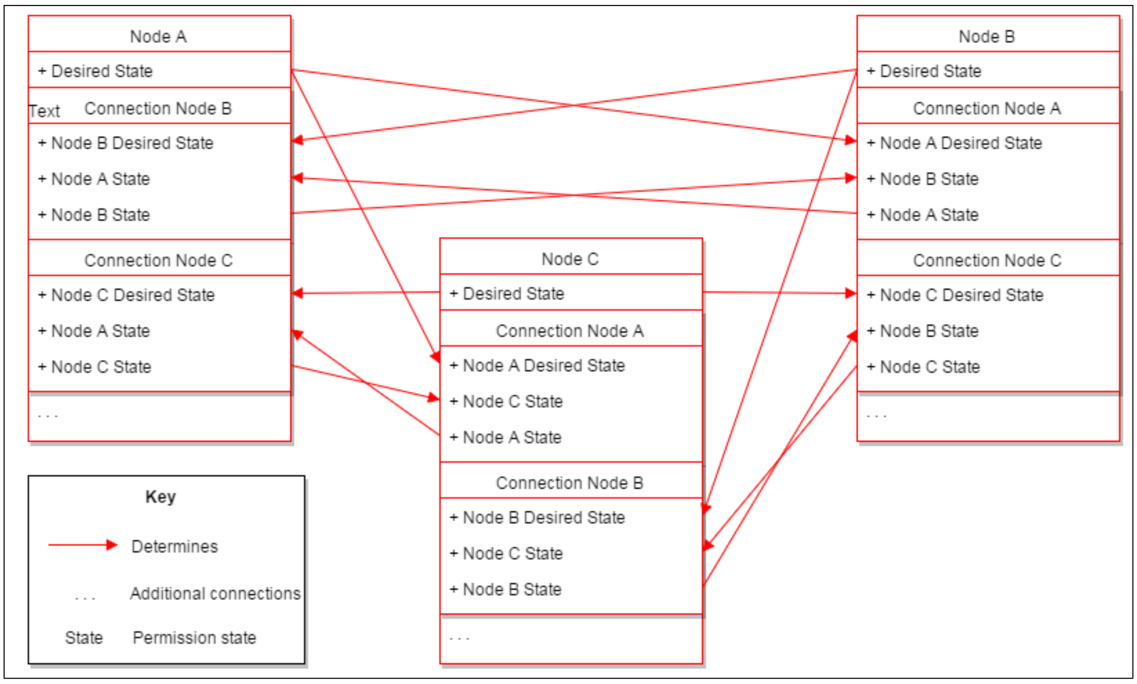


Figure 5.2 TrusNet permission status local tracking

5.2.1 Desired and Functional Permissions

Functional permissions are how nodes track the current state of node permissions in all connection pairs involving themselves. Desired permissions are the permission state which a node requests connected nodes to place it in. The functional permission of a node can never be higher than the desired permission, although functional permissions are frequently lower than a node's desired permission.

5.2.2 Permission Levels

Permission levels are tiered, with each successively higher level increasing the amount of control a given node has over the node which has granted the permission. Because of this tiered model, higher level permissions retain control granted at lower levels. For example, a node which is a member of a connection which grants it "Sudo" state retains the abilities of the "Master" and "Parent" state as well. Likewise, a node in the "Child" state will not allow modification of attributes outside of TrusNet, as "Servant" was already granted control of such modification.

Permissions often have a reciprocal permission, although these reciprocal permissions are in-

Table 5.1 TrusNet Permissions

Category	Permission	Description	Reciprocal
Admin	Sudo	Can modify all attributes of partner, even those outside the scope of the TrusNet system.	Slave
	Master	Permitted to modify all attributes which are within the scope of the TrusNet system, most notably the desired permission of a connected node.	Servant
	Parent	Controls the permission state of nodes which are connected to a connected node.	Child
Sibling	Director	Suggests permission states for nodes connected to a connected node, these suggestions are echoed from those received by a Parent node.	Relay
	Peer	Connected nodes have the same permission state, and neither can modify the settings of the other	Peer
	Relay	Allows a connected node to suggest permission states for connected nodes	Director
Remote	Child	Allows a connected node to determine the permission state of all connected nodes.	Parent
	Servant	Permits its own TrusNet attributes, most notably the its desired permission, to be modified	Master
	Slave	Allows all of its own attributes, including those outside the scope of the TrusNet system, to be modified by a partner who is granted Sudo permissions.	Sudo
Rejected	Unknown	Given no authority to modify its partner, although is permitted to forward messages over the partner through the network mesh.	—
	Suspicious	Network discovery and message-forwarding are disabled, the node is still permitted to retain its current connection.	—
	Hostile	All attempted connections will be immediately rejected. IP address or other network identifier will be temporarily blacklisted.	—

clusive, meaning that a connection pair need not take advantage of all of the permissions provided. As an example, two "Slave" nodes may connect to each-other, both granting each-other "Slave" permissions, even though neither node can actually modify the attributes of its partner because neither node has "Sudo" permissions. Table 5.1 contains a more detailed description of the functionality provided by each permission level.

The permissions defined in Table 5.1 are made available outside of the TrusNet library, and can be utilized by applications using the TrusNet library. When using the TrusNet library, an application can define which permissions a node begins with, along with how to treat newly connected nodes.

Note that these permissions essentially define the level of security that the TrusNet library defines, if a node sets its default permission state too loosely, the authentication provided may be moot.

Chapter 6

System Sequence Diagram

The diagrams in this section describe how the system transfers between states. The sequences described form the foundation for the function of the TrusNet library. This chapter focuses on the TrusNet library, as the demonstration application will merely be an implementation of the more general library.

6.1 Connection Security and Authentication

The end-to-end security and authentication model is over-viewed, at a high level, in Figure 6.1. The security and authentication process used in TrusNet provides the basic authenticity assumptions used when interacting with a TrusNet mesh network. As such, the security and authentication model provides the primary functionality of the TrusNet library.

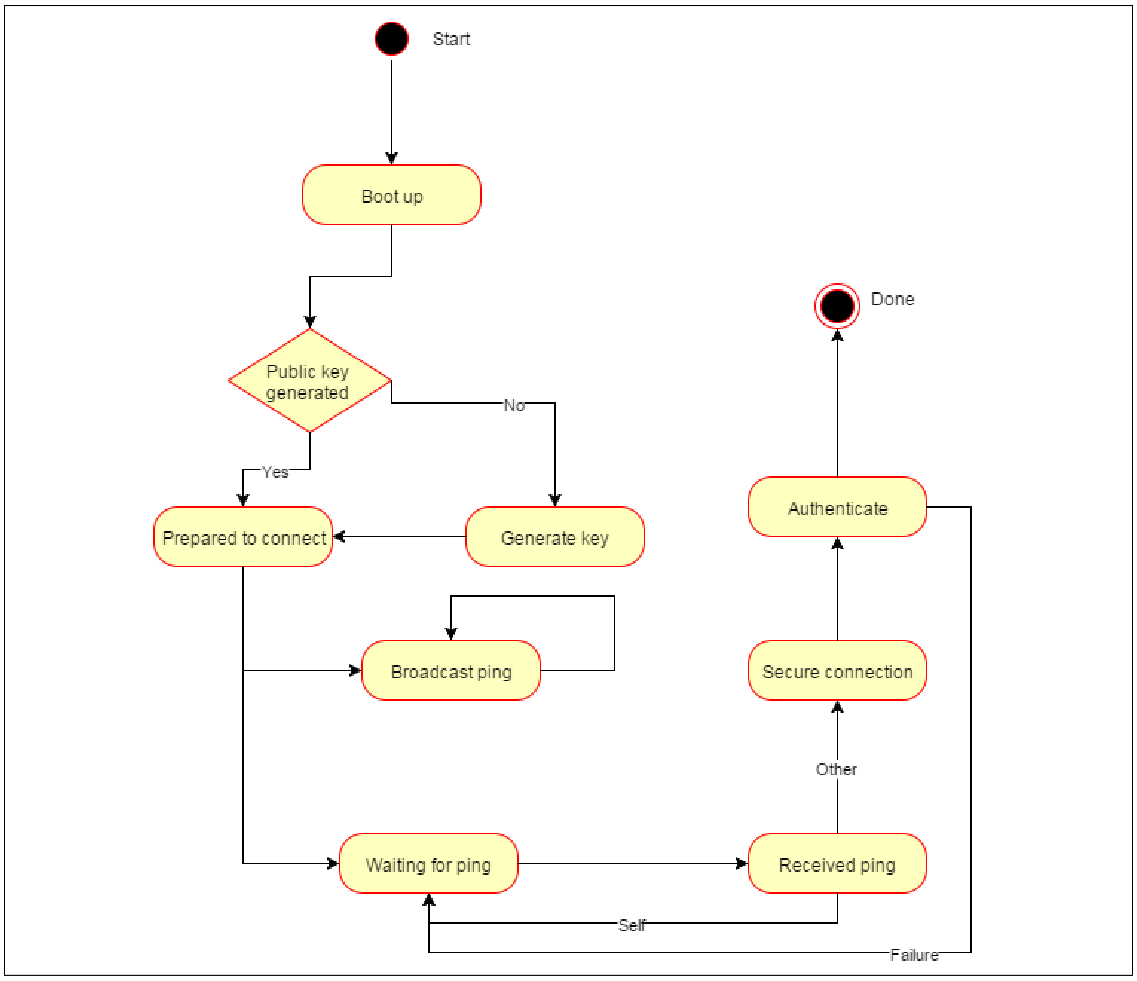


Figure 6.1 Securing and authenticating a connection

6.1.1 Key Generation

In the TrusNet library, nodes generate their own public-private key pairs. This is because the TrusNet library is designed to be peer-to-peer, and any central key repository would be hierarchical. Depending on the public key algorithm used, the key space will be of a different size. The purpose of this section is to demonstrate that randomly generating public keys provides sufficient security. Generally, public key pairs are comprised of two prime numbers or prime number derivatives combined together. In order to generalize key generation we will use the notation:

$$Key_p = Prime\ derivative\ number$$

$$Key_q = Prime\ derivative\ number$$

$$Key_{pair} = Public\ Key\ Pair$$

$$Key_p\ (op)\ Key_q = Key_{pair}$$

Where "(op)" is the operation used to derive the public key from Key_p and Key_q, which is unique to the public key algorithm used. Generally, for an n bit public key:

$$2^{\frac{n}{2}-1} < Key_p < 2^{\frac{n}{2}}$$

$$2^{\frac{n}{2}-1} < Key_q < 2^{\frac{n}{2}}$$

$$2^{n-2} < Key_{pair} < 2^n$$

Because of the Prime Number Theorem (Selberg 1949), we can accurately estimate the number of primes up to a given number with natural log:

$$p = set\ of\ primes$$

$$p_n = p < 2^n$$

$$|p_n| \simeq \frac{2^n}{log_e(2^n)}$$

We can use this to approximate the number of available Key_p and Key_q's there are in any Key_pair of n bits with the equation below:

$$p_q = p < 2^{\frac{n}{2}}$$

$$p'_q = p < 2^{\frac{n}{2}-1}$$

$$|p_q| \simeq \frac{2^{\frac{n}{2}}}{log_e(2^{\frac{n}{2}})}$$

$$|p'_q| \simeq \frac{2^{\frac{n}{2}-1}}{log_e(2^{\frac{n}{2}-1})}$$

$$|q_{space}| \approx |p_q| - |p'_q|$$

$$|q_{space}| \approx \frac{2 * 2^{\frac{n}{2}-1}(n-4)}{n(n-2)log_e(2)}$$

Now that the q_{space} has been defined, we can define the number of available Key_p and Key_q's there are in a few well known key sizes:

$$q_{512} \approx 3.24995 * 10^{74}$$

$$q_{1024} \approx 1.88530 * 10^{151}$$

$$q_{2048} \approx 1.26513 * 10^{305}$$

$$q_{512} \approx 2^{247}$$

$$q_{1024} \approx 2^{502}$$

$$q_{2048} \approx 2^{1013}$$

In cryptanalysis, any Key_{pair} which shares a Key_p or Key_q with any other Key_{pair} is trivial to compromise. Given the approximate number of available keys, it is simple to calculate the probability of this happening:

$$Key'_{col} = Collision\ chance\ for\ any\ key\ pair$$

$$q_{col} = \frac{1}{|q_{space}|}$$

$$Key'_{col} \approx 4 * q_{col}$$

$$Key'_{col} \approx \frac{4}{|q_{space}|}$$

Given the probability of any two keys colliding, we can calculate the probability that, given Keys$_{tot}$ number of keys generated in a key space, any of that set of keys will collide using the Poisson approximation:

$$Key_{col} = Chance\ of\ any\ collision\ in\ key\ space$$

$$Keys_{tot} = Total\ number\ of\ keys\ generated$$

$$Key_{col} = \frac{\frac{1}{Key'_{col}}!}{\frac{1}{Key'_{col}}^{Keys_{tot}}(\frac{1}{Key'_{col}} - Keys_{tot})!}$$

This equation can be used to calculate the probability of any collision over the key-space. This probability is astronomically low, the equation itself has a limit of 0 as the key-space moves toward infinity.

6.1.2 Secure Connection

Securing a connection involves using public key cryptography to exchange a symmetric stream key used to initialize a stream cipher which encrypts all future messages in the connection. Each node must secure its own connection with every node it intends on communicating with. Figure 6.2 provides a visualization of the process.

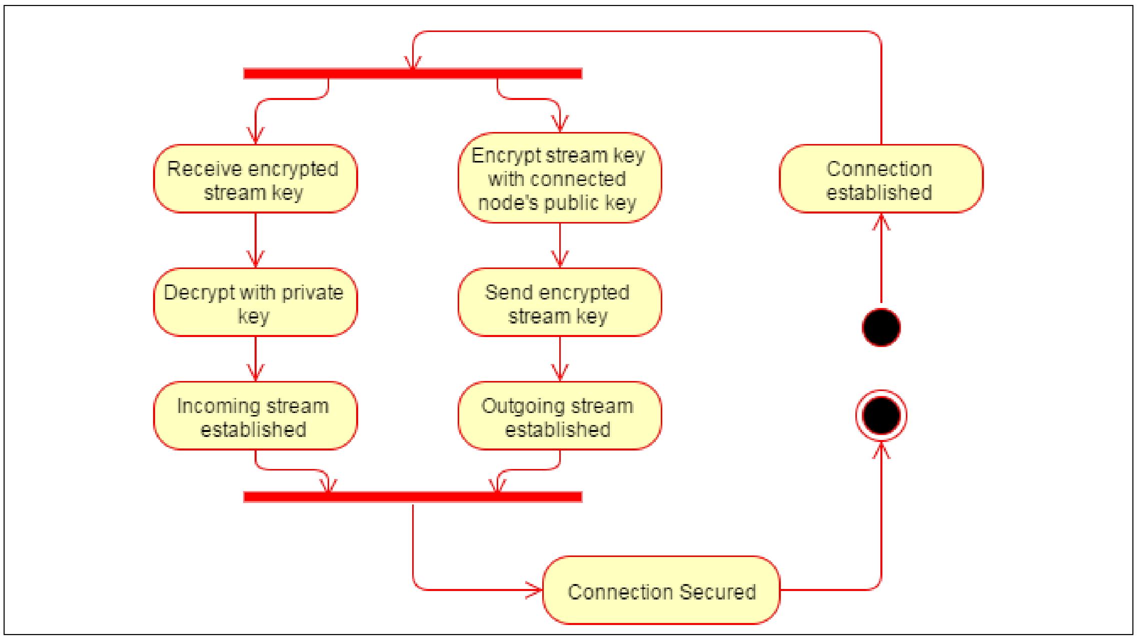

Receive encrypted stream key
Encrypt stream key with connected node's public key
Connection established
Decrypt with private key
Send encrypted stream key
Incoming stream established
Outgoing stream established
Connection Secured

Figure 6.2 Securing a connection

6.1.3 Authenticate Connection

Connections are authenticated through digital signatures. The hash of a signature message is signed by a node to verify the node possess the private key pair of the advertised public key. Figure 6.3 details the authentication process. Note that if a node fails the authentication test, the node is either hostile or using the TrusNet library improperly.

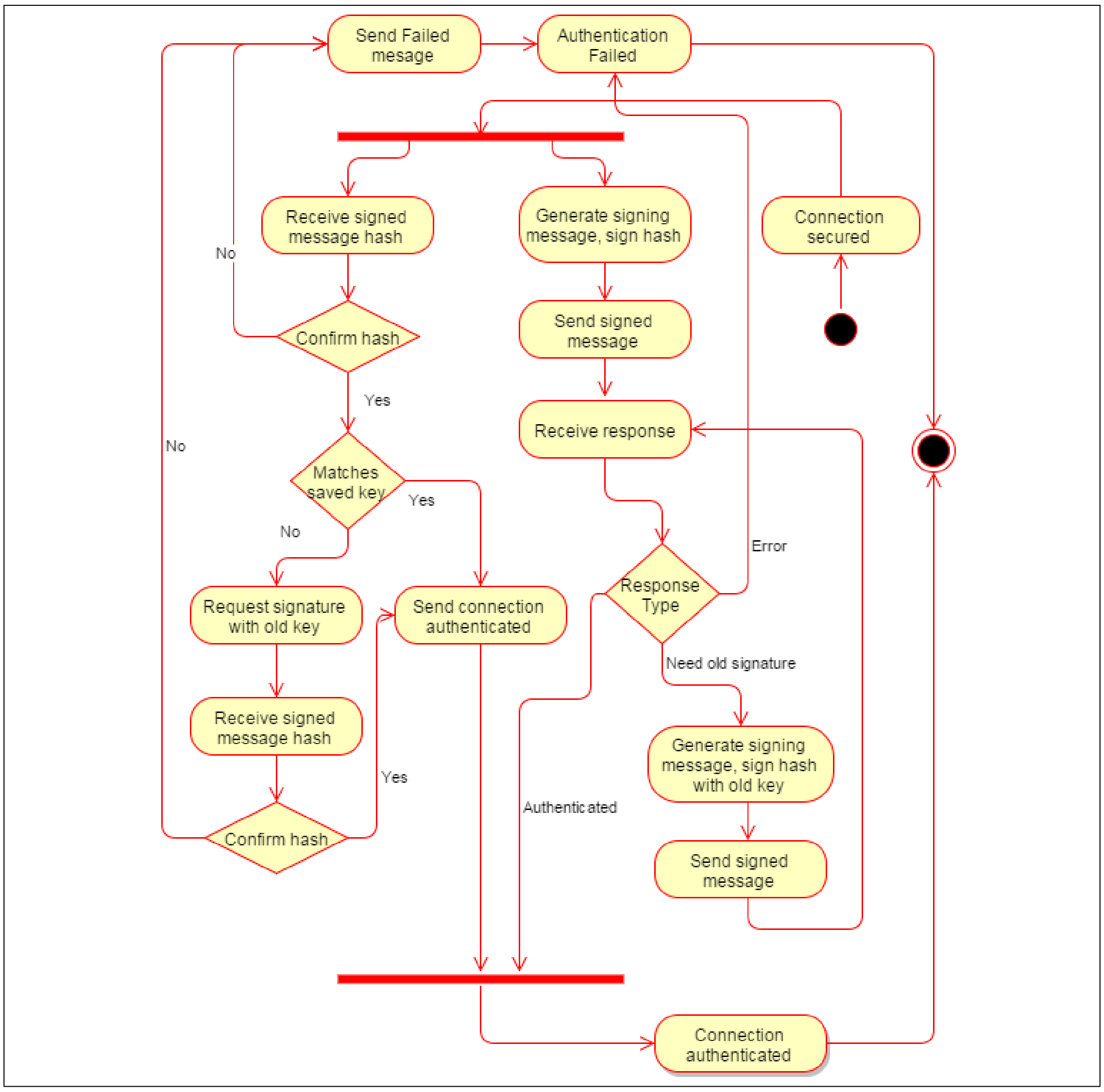


Figure 6.3 Authenticating a connection

6.2 Network Management

The TrusNet library defines a sub-network with string based addressing which operates on top of existent networks. TrusNet, unlike the Internet in general, defines and secures this sub-net across an area of any size. The sub-net is entirely virtual, and is achieved through encrypting and forwarding messages through the sub-net.

6.2.1 Addressing

TrusNet addresses are string based and two tiered. Each node belongs to a group, defined by a 20 character string. Nodes cannot communicate with nodes outside their group through the TrusNet protocol. Inside a group, each node is further identified by a 20 character string. This string, a node's name, is the TrusNet address of node. Any network protocol specific routing is abstracted out, such that node send and receive messages based on name and group only. Figure 6.4 shows this addressing framework.

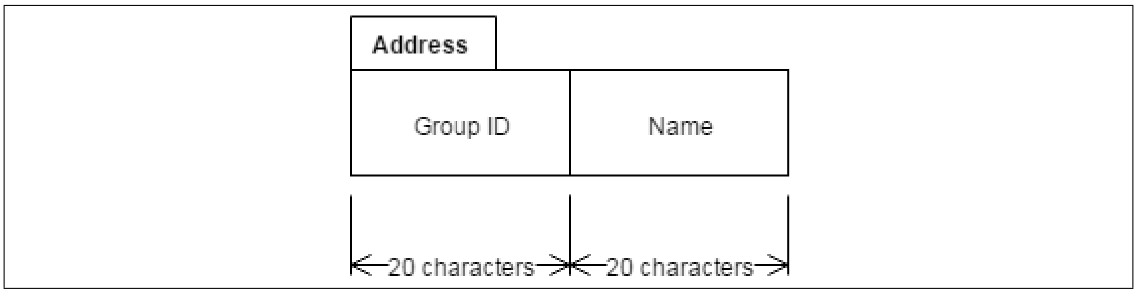


Figure 6.4 TrusNet addressing

6.2.2 Mesh Network

TrusNet allows nodes to forward messages through other nodes. This allows nodes to communicate over diverse networks, which is especially important if a node either lacks a network interface or is outside of the range of a particular interface. In Figure 6.5, node A, C and D are all connected over Wifi. Node A, B and D are all connected over Bluetooth. Inorder for node C to communicate with Node B, it must forward messages through node A.

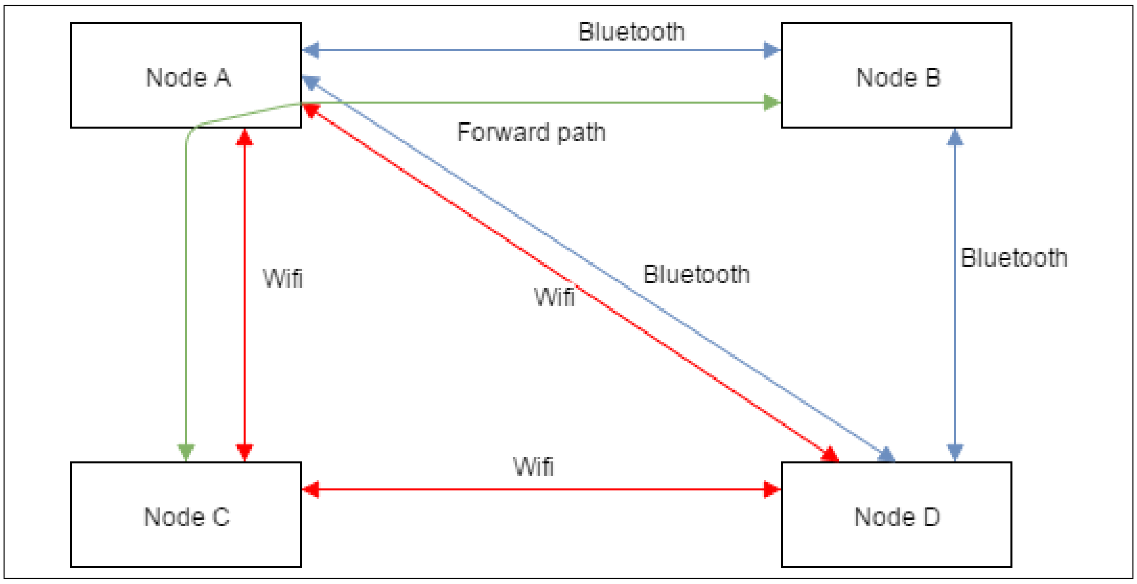


Figure 6.5 Mesh network

Chapter 7

Architectural Diagram

Our system design revolves around several basic steps. First the actors program calls the TrusNet API to send a message. The TrusNet API calls on available hardware to format and encrypt the message. This message is then sent into the mesh of various nodes until it arrives at the target device. This target device will then use available hardware and the TrusNet API to decode the message and provide it to a user program. The above Figure 7.1 shows the connections of a sample network.

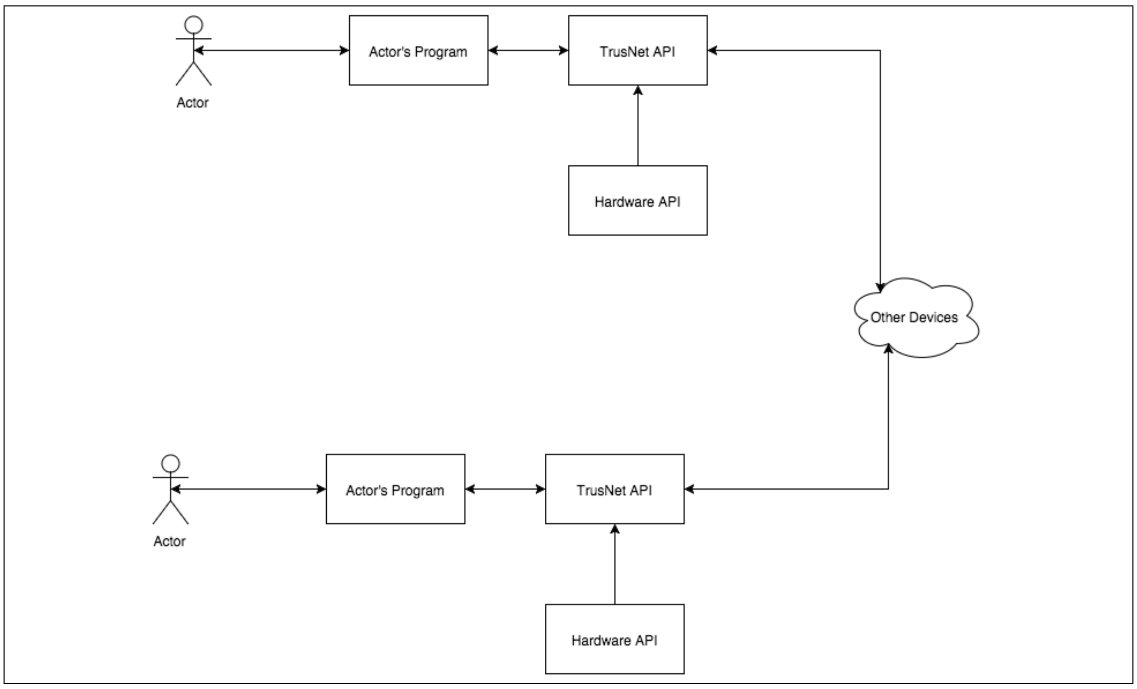


Figure 7.1 Overall design for a network

Chapter 8

Technologies Used

Technologies used describes the existing technologies used to in the project as well as the ones used to support and create the project.

8.1 Build

These are the technologies used to build and compile the system. Note that both gcc and MVCC are the broad compiler category, both were used to compile C++ code.

- CMake
- gcc
- Scripting
  - Batch
  - Shell
- MVCC

8.2 Support

Support technologies are used for organization, documentation and debugging of code. Note that in some cases, support technologies are closely related to build technologies, most notably IDEs.

- GitHub

- LaTeX
- Visual Studios
- XCode

8.3 Hardware

Hardware technologies refers to the platforms which have been used to both develop and run the project. The system will be primarily developed on Windows.

- Intel Edison
- Raspberry Pi
- Mac
- Windows

8.4 Runtime

Runtime technologies are the libraries and languages used during the runtime of the projects. This project will use C++ as its primary language.

- C
- C++
- C++ 11
- openGL

8.5 Algorithms

The list of algorithms provided here is, by no means, an exhaustive record of all algorithms used. Rather, the algorithms used includes algorithms implemented by the project.

- RSA
  Public key encryption algorithm, used for exchanging keys and signing message hashes

- RC4

  Stream cipher used to encrypt large blocks of data

- AVL Tree

  Balanced binary search tree used to store data during runtime

Chapter 9

Design Rationale

The design rationale justifies technologies used. This section is split into Build, Support, Hardware, Runtime and Algorithms.

9.1 Build

Technologies chosen for building the TrusNet project were effected, primarily, by our design constraints. For our compilers, we have chosen those natively supported by our chosen operating systems. Additionally, the compilers chosen are among the most popular C++ compilers, meaning they are well tested and well used.

Because of the diversity of the compilers chosen, CMake was the most obvious choice for the primary build system, as it supports a wide range of compilers on a wide range of platforms. To automate miscellaneous file movement and calling of CMake, shell and batch scripting are used for Unix and Windows respectively, as these languages are natively supported.

9.2 Support

For source control, GitHub is used to store and manage all code and documentation. Git is the industry standard for source control, and automatically releases TrusNet as an open-source library.

Because it is both modular and makes type-setting easy, LaTex is the tool our team for documentation. LaTex allows each library to contain its own documentation, but for a central document to additionally display that documentation.

Lastly, Visual Studios and XCode are two IDEs supported by CMake that provide extensive debugging and programming aids. Both these IDEs are the standards for Windows and OSX development, respectively.

9.3 Hardware

The hardware platforms were chosen for a variety of reasons. There are two classes of hardware that run the TrusNet demonstration applications: UI platforms and headless platforms. Generally, UI platforms were chosen because:

- Widely used
- OpenGL Support
- Developers posses systems

Headless systems were chosen because:

- GPIO Support
- Inexpensive
- Internet of Things targets
- Unix based

9.4 Runtime

C and C++ are the base of the TrusNet system because this family of languages is supported by most devices, both embedded and otherwise, on the market today. Furthermore, C and C++ combined balance speed and complexity, allowing for a system which both runs quickly and has a high degree of complexity. C++ 11 has been integrated into the project to expedite development, taking advantage of some of the newer features integrated into C++ 11.

OpenGL is used to construct the UI for a number of reasons. First, OpenGL provides a consistent interface across all platforms. Second, OpenGL provides an immense amount of control to developers, allowing for construction of non-standard visual elements. Lastly, an OpenGL UI implementation gives developers access to the basic source code of the UI, allowing the TrusNet library and the visual interface a much higher degree of coupling than would be permitted with the use of a third-party library.

9.5 Algorithms

RSA and RC4 were chosen as the initial cryptographic algorithms because the pair is simple and well researched. RSA provides the key exchange and signature, while RC4 provides the stream cipher encrypting the messages passed between two nodes.

The Datastructures library relies heavily off of AVL trees because AVL trees combine fast insertion and deletion with ability to traverse a set of data. Both vectors and arrays have limited insertion and deletion capabilities, and hashes have difficulty traversing the set of data contained in them.

Chapter 10

Development Milestones

This project revolved around the development of our connection protocol. As such, the groundwork of the project took most of 2015, followed by a major rebuild of several elements in early 2016 and a quick set of updates towards the end of the year, bringing additional elements online. From initial connection to our final release took almost 5 months, with total development taking almost the entire year. Table 10.1 on the next page notes the most important points in our development process. Please consult figures A.1, A.2 and A.3 in the appendix for the gant charts used to plan our development timeline.

Table 10.1 Key milestones

Milestone	Date	Details
Development begins	June 10th, 2015	We began serious work on TrusNet at the end of our junior year. Working through the summer and fall allowed us to take on a more ambitious project than we would have otherwise been capable of.
Initial connection	December 12th, 2015	This was the first time our protocol connected two devices. This was under highly controlled circumstances, but served as a proof of concept for our future work.
Edison freeze	March 9th, 2016	The Intel Edisons were unusable for a several month period due to problems with the operating system's total size. When a working OS was finally made available, we quickly locked each of our Edisons to this OS as well as cleaned and reloaded all additional functions we had added. The OS of each Edison was unchanged since this point.
1.0 release	April 24th, 2016	The 1.0 release was the first build with each of the features critical to demonstration. This version of the protocol included executables which passed the necessary data, but did not yet include the files to read the joystick or move the robot.
1.1 release	May 5th, 2016	The 1.1 release added joystick functionality, as well as a UI for the joystick.
1.2 release	May 7th, 2016	The 1.2 release added robot functionality, as well as a UI for the robot.
1.3 release	May 10th, 2016	The 1.3 release fixed numerous bugs uncovered in the previous releases, as well as added IP propagation, allowing for a reliable and helpful demonstration.

Chapter 11

Project Problems

This sections contains analysis of certain problems which we encountered with the development of the TrusNet protocol. Note that early in project development, potential risks were concretely outlined and analyzed. Consult chapter B in the appendix for these tables.

11.1 Software Problems

As with any software project, we had numerous issues involving the code we had written. Aside from bugs brought on by logical errors in our programming, we had several software issues caused by compilers and language properties. The Intel Edison uses a different C compiler than Windows and OSX systems. The differences in compilers meant that some static variables were stored in different orders depending on platform. This in turn meant that several critical classes were formatting data in different manors depending on platform. When different platforms tried to connect, the out of order memory allocation meant that messages could not be read between platforms.

Another critical error was brought on by the need to constantly save time stamps. The Edisons are constantly saving information, but when our protocol closes, saves may be in progress. Every time an Edison was closed, even if not a crash, there is a chance that a file may be corrupted. When this occurs, the system can no longer operate and must be cleaned manually. While we have taken some steps to minimize the likelihood of this curring there is still a danger of corruption on shutdown.

Lastly, we had a problem with our destructors and virtual inheritance. When a class is virtually inheriting two or more classes, which in turn inherit from a higher class (I will call the inheriting class CB, the virtually inherited classes C1 and C2, and the highest class CM for sake of explanation) and the destructor is called, CB and CM are properly destroyed. However, C1 and C2 are destroyed in random order. C2's destructor would fail if run second. Every time we deleted a CB object, we had a 50% chance of crashing our system.

11.2 Hardware Problems

We also had a selection of hardware centered commands, most involving the Edisons. For a several month period, the flash tool Intel offered refused to work with any image we had. Additionally, the older flash tools we had stored refused to install the latest OS image, mainly because the latest image (which contained a critical upgrade to make we needed) was larger than the available memory on an Edison. We had initially hoped to begin work on the demo unit in January, but were only able to bring the Edisons online in mid April.

The provided Yocto image for the Edisons lacked several critical commands we used heavily for scripting and building. We ended up downloading and installing bash and git manually on each Edison, as well as installing Intel's IoT libraries. Additionally, the school's network forced us to manually add each Edison's IP Address to our personal accounts though a convoluted process.

Once built, our robot had a significant problem with power usage. Our initial wiring setups cause the Edisons to frequently lose power entirely. We eventually built two independent battery systems for the robot so that should the motor draw to much current, the Edison's power would be mostly. We still have voltage swings, but they are now well within the operating parameter of the Edison.

11.3 Team Problems

Both members of this project have clear opinions on software design, which did not always coincide. As such, numerous arguments about the implementation of various elements in our system broke out. To deal with these disagreements, we maintained a strong vision of our final goal, as well as avoided criticizing each other, only the ideas put forward. Additionally, the initial division of roles established a clear hierarchy in most arguments. Only a handful of disagreement lasted more than a day, as one voice would eventually convince the other of a particular decision.

11.4 Lessons Learned

This project taught us much, both in technical skills and project design. Technically, we learned a great deal about the struggles of working on a new platform. The constant bugs and crashes, combined with a constantly changing code base, proved a new type of challenge. We are used to uncovering bugs in our own code, this project introduced us to bugs and variance in the tools we were using to build our project. Additionally, we also had a major problem with scope creep. As the project grew, there were more and more elements we wanted to add. While we are pleased with the final result, there are many features we want to add that we simply did not have the time to work on. We were forced to decide between important features on a regular basis, which thankfully did not cause internal arguments.

In the end, we did manage to avoid many pitfalls through intelligent project architecture. We effectively managed our code through Git, and set up efficient cross-platform builds through CMake. Our project architecture also fit Doxygen well, allowing us to save development time through efficient code documentation.

Chapter 12

Ethical Analysis

12.1 Overview

Over the past few months, the ethical dilemmas involved with encryption and data-security have been in the national spotlight. For those previously unfamiliar with the issues at stake, the spat between Apple and F.B.I. concerns the unlocking of an iPhone once used by Syed Rizwan Farook, who participated in a tragic mass shooting in San Bernardino, California in December 2015 (Staff 2016b). The concerns raised in this specific case, however, are merely a snapshot in a decades long and international ethical disagreement.

This overview will, firstly, explain how TrusNet fits into the privacy verse security ethical dilemma, discussing both what the system is capable of, how it is intended to be used and possible malicious abuses of the project. Second, this overview will briefly review the international history of the legal and ethical issues of strong encryption and data-protection. Lastly, the ethics of TrusNet will be analyzed through a variety of different ethical systems and frameworks.

12.2 Privacy verses Security

The relative importance of privacy and security is a question all societies must address. There is no single, perfect answer, as no matter what is selected, issues are present. For example, a society with no privacy can maintain order extremely well. All criminals are stopped before their crimes. In theory, no murder, rape, or robbery. Unfortunately, the trade off is a loss of personal liberty. When the government has access to all information, minority groups and unpopular opinions are silenced. Innovation becomes impossible, as anything truly groundbreaking is destroyed before it can come to fruition. A society without privacy, however secure, is fundamentally flawed.

By the same token, a society with total privacy is dangerous. Syed Farook's phone is just one example in the modern world. Syed's phone could have information on additional attacks, names of conspirators, and vast amount of details only available through this phone. The inability to access this phone could hamper authorities ability to stop future attacks. However, if the phone can be unlocked, it opens another set of issues around the world. If communications can be unlocked, political dissidents in China could be targeted and killed. Movements such as the Arab Spring could be shut down before they began. If a skilled hacker stole an individuals phone, that hacker could access personal information including bank accounts, biometrics, and secure passwords. If a phone can be opened, we could access terrorists plan, but a stolen phone could be used to access infrastructure such as the power grid, forcing a great area of the nation into literal darkness. As engineers, we have a duty to consider how the technology we create can be used in both positive and negative ways.

In the creation of any particular encryption one must consider both the security of a network, as well as the encryption's ability to be used in a malicious manner. If our system allows unlimited long distance communication across millions of devices and all networks with unbreakable encryption, we will enable safe Internet usage, but at the cost of harming traditional law enforcement and anti-terrorist groups. If our security is poor, anyone with requisite knowledge could compromise our network, rendering it pointless.

12.3 TrusNet Capabilities

The TrusNet protocol is intended to be used for machine-to-machine communication on a medium-size sub-network for usage in the Internet of Things. While TrusNet can theoretically be used for a human message-exchange protocol, the deliberate size limitations on the protocol mean that such a message-exchange protocol would need to separate users into sub-groups to efficiently operate. TrusNet has two major features which relate to data security: Encryption and Authentication. These two features provide different features of the system which have both beneficial and detrimental ethical implications.

12.3.1 Encryption

Broadly, encryption refers to a mathematical operation which hides information through a key and then subsequently decrypts the data with a key, which is either equivalent or related to the first key. TrusNet leverages encryption to exchange keys and to pass data between nodes once a secure connection has been established. First, TrusNet's use of encryption helps in user authentication. Second, TrusNet's encryption protocols protect the data passed between nodes from being observed by a third party.

It is the second use of encryption which has potential ethical implications. Because the encryption algorithms used by TrusNet are strong-encryption, they cannot be read by a party which does not have the correct keys. Unlike many other systems (including, for example, the software authentication protocol used by Apple), TrusNet is not hierarchical, which means that a "master-key," or a key which can decrypt any set of data encrypted by the system, is mathematically impossible. The consequence of this is that if TrusNet were leveraged as a malicious communication protocol, the subsequent messages would be impossible for a third party to decipher.

While such strong encryption has the potential to be used maliciously, it is also crucial for the emerging Internet of Things. As our homes become more connected, thermostats, security cameras, digital locks and home-entertainment systems will all communicate with both each other and the personal devices of authorized individuals. In December 2015, for example, the digital toymaker VTech fell victim to a disastrous data breach which allowed unauthorized individuals to access account details of users (Risen 2015). In VTech's case, this compromised data even included pictures of the children using VTech's devices. This breach demonstrates the issues with failing to encrypt data in transit, especially as the data being transferred is increasingly personal as connected devices move deeper into our homes.

12.3.2 Authentication

Authentication refers to a mathematical protocol designed to ensure that a node is in possession of some unique set of data. Usually, authentication schemes rely on public-key cryptography (12.1), which allows an entity to securely prove its identity to another entity. It should be noted that the schemes which achieve this unique behavior have been rigorously mathematically tested. TrusNet leverages public-key cryptography to authenticate previously encountered nodes, requesting these nodes to prove they have the unique set of data with which they are associated.

The ethical advantages of such a system are clear, especially when electronic devices are capable of modifying the behavior of physical objects such as cars and door locks. With a robust authentication system such as TrusNet in place, these physical objects cannot be accessed in the digital realm by unauthorized entities. For the emergent Internet of Things, forcing nodes to securely and definitively identify themselves results in a safer network of devices which cannot be modified by malicious actors.

Such strong authentication, however, can have a serious dark-side if expanded on a national, or even regional, scale. Since a protocol like TrusNet forces all nodes to remain authenticated at all times, the Internet as accessed through such a system has no anonymity. While anonymity can be disastrous on embedded systems, it is crucial for media access in a free society. Because TrusNet, if expanded to a national scale, has such a clearly Orwellian feature, we have designed the system such that it's scalability remains limited. The authentication module of TrusNet, by design, becomes unusable as the system expands beyond its intended use inside a medium-size sub-network.

12.4 History of Data-Protection

Before analyzing the TrusNet project in various ethical frameworks, it is worth reviewing the history of data-protection around the world so that we may avoid the mistakes of the past. For millennia, codes and ciphers were based on writing and only rarely used to protect the most important of messages. These techniques were often time-consuming and tended to focus on military communications. While these ancient means of data-hiding were certainly pre-cursors to modern methods, it was not until the advent of simple electronic computation just before World War II that the algorithms and techniques we use to protect data today began to emerge.

12.4.1 World War II

During the Second World War, Nazi Germany developed an advanced substitution cipher based in a machine known as the Enigma machine. Famously, Allied forces performed cryptanalysis on both the Enigma machine and messages intercepted between German radio stations using the Enigma machines and managed to crack the German system. The techniques used varied based on key generation methods, the size of the messages intercepted and even the headers used in the transmitted messages (Gillogly 1995).

In the Pacific theater, American forces managed to break a number of Japanese crypto-systems, most notably JN-25 (Donovan 2012). Perhaps the most impactful consequence of this breach was the decisive Allied victory at the Battle of Midway where Japan lost four fleet carriers (Prados 1996). Later in the war, American fighters shot down Admiral Yamamoto's transport aircraft after having intercepted encrypted Japanese communications and subsequently breaking the ciphers.

In the historical examples of World War II, we see Allied code-breakers immortalized as they helped to take down the horrors of Nazi Germany and Imperial Japan. In recent years, as the United States and British governments have declassified the actions of their cryptanalysis efforts during World War II, the actions and results of some of these activities have given the public an impression that all crypto-systems can be analyzed and cracked, given enough time and effort.

12.4.2 AES/DES

The more recent history of commercial symmetric key cryptography, namely AES and DES, demonstrate that modern cryptography differs drastically from the electro-mechanical boxes used during the World Wars. In 1974, IBM developed an algorithm which was reviewed by the N.S.A. (National Security Agency) and released in 1977 as the Data Encryption Standard, or DES. Since DES is an algorithm, and not a physical device as previous crypto-systems had been, DES can be implemented on various different platforms in both hardware an software. Despite its age, and the fact that DES was formally replaced as an encryption standard for most organizations in the early 2000's, DES remains frequently used today (Smid and Branstad 1988).

As computation power increased in the 80's and 90's, the relatively small key used by DES began to be insufficient for digital cryptography. As with DES before it, the National Institute of Standards and Technologies held a contest to find the sufficient cipher to replace DES. The winner of contest was an algorithm named Rijndael, which since became known as the Advanced Encryption Standard (Selent 2010). It should be noted that AES was developed in Belgium, so the notion that any nation can adequately control the international availability of cryptographic algorithms has been shown to be quite ridiculous.

From a cryptographic perspective, both AES and DES are "secure," meaning that unlike the systems of World War II, mathematical analysis has yet to be able to decrypt AES and DES cipher texts without access to the keys. In both cases, the only known way to break into an AES or DES secured system is to iteratively attempt all possible keys until a working key is found. For the 128 bit minimum key size of AES, such a brute force attack is far beyond the capabilities of modern computing. DES, however, has a key size of only 56 bits. In the 70's when DES was developed, the 56 bit key size was sufficient. Today, in research settings, brute force attacks on DES have been successfully preformed. The equations bellow describe the difference between the size of the two keys.

$$Keysize_{AES} = 2^{128} \approx 3.4 * 10^{38}$$

$$Keysize_{DES} = 2^{56} \approx 7.2 * 10^{16}$$

The key sizes of both AES and DES have ethical implications because they hugely effect the overall security of the system, and how the systems can actually be compromised. AES is not merely 70 times stronger than DES, as a simple bit count would indicate, it is 4.7 sextillion times stronger than DES. With modern, and even foreseeable technology, AES remains an essentially unbreakable symmetric key crypto-system.

12.4.3 RSA

In 1978, mathematicians Ron Rivest, Adi Shamir and Leonard Adleman developed a system which changed the way the world thought about cryptography. The algorithm they developed has became known as RSA, and it is the first example of a widely deployed public key crypto-system (Bar-Yosef 2012). Figure 12.1 below demonstrates the features of a public key crypto-system. Essentially, public key crypto-systems allow mathematical guarantees as to what entity can open a message and which entity a message has come from.

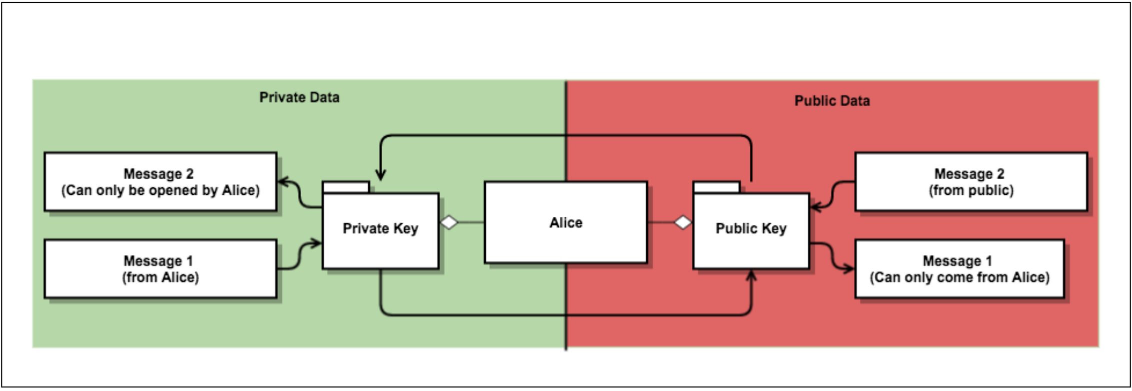


Figure 12.1 Public key crypto-system

In short a public key crypto-system such as RSA allows for entities to securely exchange symmetric keys anonymously and allows for digital signatures to mathematically establish identity. Public key cryptography is ultimately the technology that underpins modern website security, and is a central technology in the TrusNet project. It can also be controversial, because although it ultimately protects our modern financial system, it stands in direct confrontation with the often-touted idea of a "back-door".

12.4.4 Clipper Chip

Frequently, we here calls for "back-doors" to be placed in crypto-systems. The most recent example of this, as mentioned earlier, is the conflict between Apple and the FBI. In a broader sense, the argument made for inserting "back-doors" into crypto-systems is that crypto-systems without a "back-door" are effectively immune to legal tools such as search warrants. Without taking a position on the ethically of "back-doors," it is important to note that from a technical perspective, they are essentially impossible to build.

In 1993, the United States developed the now infamous "Clipper Chip," which could theoretically allow law enforcement agency to unlock a device with a master key. In order to prevent use, keys were split in half with different government agencies keeping the two halves, in theory only combining them when a system needed to be justifiably unlocked. The Clipper Chip was based in hardware, not software and was at first thought to be tamper-proof.

Clearly, however, recent events clearly show that the Clipper Chip was never widely implemented. Shortly after its release, a researcher at Bell Labs not only manged to disable the back-door on the chip, but was actually able to extract the master key from a chip (Blaze 1994). Furthermore, in analyzing the failure of the Clipper Chip, the researcher who broke the chip essentially concluded that the issue was not merely with the implementation of the Clipper Chip, it was rather with the idea that any system could simultaneously provide adequate data-security and allow a "back-door" for law enforcement.

12.4.5 International Laws

The laws surrounding strong cryptographic algorithms like AES and RSA are inconsistent across the globe. According to the laws of many nations, the United States included, cryptography is considered to be 'arms' and is, at least officially, regulated as such (Koop 1996). Even so, the laws regulating cryptography in the United States are murky. In some cases, source code was found to be First Amendment protected speech, making any domestic restrictions or even export controls on such code unconstitutional. In other cases, however, export restrictions were held up (Koop 2013).

In the United States, there is a movement to require technology companies to be able to unlock any encrypted device. Two senators are currently writing a bill that would ban effectively strong encryption (Staff 2016a). If this ban were to pass, technology such as TrusNet would become illegal, as there is no way for a company to decrypt a message not intended for them.

Some nations hardly regulate digital cryptography at all. These include some of the places one might expect such as Denmark, Japan, and Sweden as well as a number of nations with a long history of attacks on basic Human Rights such as Syria and Venezuela (Koop 2013). Many of the nations which have not heavily regulated cryptography seem to be nations with either young or non-existent high-tech sectors.

The last group is the nations which heavily restrict the use of digital cryptography. In many cases, the nations which heavily restrict cryptography are the same totalitarian states which regularly crack-down on freedom of speech, place such as Russia, China and Egypt. But frequently, states such as France, Italy and even the European Union as a whole put heavy and circuitous regulations on the use of cryptography for the purposes of national defense (Koop 2013). In short, International Laws regarding cryptography have not been well defined, and this is becoming an increasingly large source of conflict in international business, as many Facebook's WhatsApp has shown in its interactions abroad (Lomas 2016).

12.5 Ethical Framing

TrusNet exists in a fundamentally gray ethical area, as we cannot control who uses it or for what purpose. Our target goal includes a system with both encryption and authentication. It is by design that only a specific node can ever decrypt a message. Even if a node is compromised, it is unable to decrypt the messages intended for other nodes. If a malicious group used our protocol to transmit messages, there is no "back-door" system to allow authorities in, nor is there the possibility to create such a system. One of the realities we have to accept in creating TrusNet is the possibility of its harmful use.

Another feature is the authentication we are using. While a message can not be decrypted, the meta-data can be accurately recorded. This means that any node on the network can look at a message and determine its sender, receiver, size, and time of transmission. TrusNet has no concept of anonymity. If the identity of one node is known, an observer can use this information to make connections about users on the network. In much the same way that authorities are unable to read messages, they can monitor their transmission. If authorities apprehend on malicious user, they can know both how many other potentially malicious users exist and whom those individuals were communicating with.

12.5.1 Utilitarianism

Ultimately, it is probably utilitarianism which best frames the ethics of this project. Even many of the other ethical systems will end up defaulting to the same reasoning used by utilitarianism. Without a doubt, as reviewed in this analysis, great harm can come through strong digital encryption. Such codes have protected terrorists, murderers and sexual abusers. For decades, in fact, utilitarian analysis would likely come down against the strong cryptography used in the TrusNet project.

As the Internet of Things expands, however, strong cryptography begins to have more benefits than it does downsides. Nothing illustrates this better than the recent Jeep Cherokee hack, where researchers remotely shut off a Jeep as it was driving down the high-way (GreenBerg 2015). This, along with the VTech hack mentioned earlier (Risen 2015) demonstrates that data encryption is no longer simply a matter of privacy, failure to secure data can put lives at risk. As of yet, no one is known to have been killed by a compromised "smart" device, but most experts agree that it is only a matter of time. In-order to protect the public as a whole, utilitarianism calls for the protection of digital communication to prevent these kinds of physical attacks through software, even at the cost of protecting criminals.

12.5.2 IEEE

Oddly, IEEE as an organization has not formally taken a stance on strong cryptography. However, the first point of IEEE's code of ethics states that the member of IEEE promise "to accept responsibility in making decisions consistent with the safety, health, and welfare of the public, and to disclose promptly factors that might endanger the public or the environment" (IEEE 2016). The difficulty with strong cryptography, as has been discussed, is both using and not using such systems has negative effects of the public. It is at this point where IEEE's ethical stance begins to merge with that of Utilitarianism (12.5.1), weighing the pros and cons of strong digital encryption.

12.5.3 Kantian Ethics

Immanuel Kant describes a duty based ethical system, where the morality of actions is based on intent rather than the effect of the action (Kemerling 2011). At first, Kantian ethics appears to look kindly upon those providing strong crypto-systems with the intent that such tools be used benevolently. Under Kant, there would be no immoral action on the part of the cryptographer if the system was used maliciously. This line of reasoning is actually a large part of the moral rationale for TrusNet: the system is not designed or intended to be used a chat protocol.

Kant's arguments, however, begins to have issues when we consider cases like the Clipper Chip (12.4.4) or the Jeep hack (GreenBerg 2015). In both these cases, the intention of the designers was that the design systems would either provide security or at the very least be tamper proof. Ultimately, however, the intentions stopped mattering when catastrophic and dangerous weaknesses were uncovered in both systems. Regardless of the intent of the systems, both could potentially put lives in jeopardy, so it is difficult to subscribe to the idea that both systems were ethical given their capacity for destruction.

12.5.4 Moral Relativism

Frankly, moral relativism offers a poor framework to analyze advanced encryption systems. The central issue with moral relativism in cryptography is that despite efforts by many entities to the contrary, the Internet is a global system spanning major and diverse cultures around the world. But, moral relativism would hold that it is the norms of a specific culture which define the morality of strong cryptography within those cultures. This reasoning very quickly breaks down.

First, cultural norms surrounding data security only begin to emerge as the Internet permeates a cultures, which is why many developing nations simply have no cultural norms regarding encryption (Koop 2013). Even once norms are established, they frequently change as technology evolves, and the United States is a perfect example of this.

Second, cultural norms may be uninformed of the technical reality. The Clipper Chip (12.4.4) is a perfect example of this. Many politician and Americans today will frequently tout the possibility of constructing "back-doors" into encryption products, despite the fact that experts in the field nearly universally denounce these a technical impossibility. The intention of the norm demanding a "back-door" is to allow Law Enforcement to access encrypted data in specific circumstances, but the reality is that such a norm eliminates encrypted data entirely. Here, we see moral relativism fall apart, betraying the very intention of the culture's moral norm.

Lastly, the Internet is far too international for moral relativism to yield any meaningful answers. If one ascribes to the morality of the culture one operates in, what happens when operation occurs across cultural norms? Consider a software team in the United States developing an application intended for use in Italy with data servers in the Netherlands. Is the data encrypted according the cultural norms of Italy, the Netherlands or the United States? On the one hand, Italians will be the primary audience, so perhaps it is their norm that should be respected. On the other hand, the Dutch are actually storing the data, so surely their culture should determine what techniques are used to encrypt data on disk. But then again, according to moral relativism, one cannot ask the United States developers to disregard their own cultural norms in the development of the system. A strict adherence to moral relativism would leave such international software products impossible to develop, resulting in a more stratified and disjointed world.

12.5.5 Catholic Social Teaching

Unsurprisingly, the Catholic Church rarely inserts itself into international technical debates. Given this, the Catholic Church has never officially taken a stance on strong encryption. But the fundamental underpinnings of Catholic Moral teachings are relevant to TrusNet and any strong cryptographic system, namely the common good and human freedom. Strong cryptographic systems relate to the common good because they help stabilize the emergent Internet of Things. In short, the common good ascribes to the same reasoning used in Utilitarianism (12.5.1).

Human freedom, however, introduces a new ethical dimension of strong cryptography yet to be mentioned. While strong cryptography can be used to hide deplorable criminal behavior, such as Child Pornography or terrorism, it can also be used to hide behavior which has been unjustifiably outlawed. The Tor network, which allows users to hide the source and destination of their web traffic, is an excellent example of this juxtaposition (Kiosowski 2014). Tor is often used for criminal activity, and this is usually why it makes the news. But Tor is also used to subvert draconian censorship laws in nations such as China and Russia. Tor, and strong cryptography as a whole, are used by journalists, diplomats and Human Rights activists around the world. In short, cryptography is a central and indispensable tool in breaking down and exposing government oppression.

12.6 Conclusion

Ultimately, our network is another option for others to use and build upon. While TrusNet encryption is almost impossible to break, several critical design choices prevent TrusNet from being used for large scale communication. The overall computational cost of the network scales poorly, so above a certain number of nodes, our network is simply inefficient to the point of uselessness. We feel that while no system is perfect, we have developed a secure method of communication with structural choices which help alleviate some of the problems with malicious use of advanced encryption.

Chapter 13

Test Plan

13.1 Unit Testing

Unit tests are designed to verify the functionality of specific code modules. For our project, we will use two types of unit tests: informal and automated.

13.1.1 Informal Unit Testing

Before integrating new code, developers are expected to confirm the functionality of their additions through informal unit testing. Additionally, if additions are substantial, developers will utilize git's branching ability to protect functioning code from errors introduced by untested code. Informal unit tests are never considered adequate, code must (at a minimum) be integrated into the automated unit testing framework.

13.1.2 Automated Unit Testing

TrusNet contains an Automated Unit Testing framework, designed to run a battery of tests as defined by library dependencies. Each library contains a library testing class, which can be added to test battery for any executable. Each executable artifact, with the exception of the Unit Test executable itself, must be able to define a Unit Test battery for itself. The precise mechanics of the automated Unit Testing framework are explained in the documentation for the UnitTest library.

13.2 Cross-Platform Testing

The TrusNet project must be able to run on multiple platforms. For the demonstration project, there are two classes of platforms to be tested. Some platforms must display the TrusNet UI, these platforms are:

- Windows
- OSX
- Linux

The other class of platforms need only run the headless release of TrusNet. These platforms are:

- Raspian
- Intel Edison

Cross platform tests were preformed by running the Automated Unit Tests on each of the target platforms frequently. Additionally, end-to-end tests were preformed on diverse platforms.

13.3 Security Testing

Security testing involves attempting to sabotage the TrusNet system during runtime. Because of the strong mathematical backing of public-key cryptography, some security testing is unnecessary as the algorithms used have been vigorously proven. Despite this guaranteed security, toward the conclusion of the TrusNet project, we allowed our peers to attempt to compromise the security of the TrusNet system in a variety of attacks, including:

- Man-in-the-middle
- Denial of Service
- Eavesdropping

13.4 End-to-End Testing

Throughout development, the system was frequently tested end-to-end. End-to-end tests involved attempting to use the system to create and use a mesh network. These tests, in addition to testing end-to-end functionality, also tested cross platform functionality.

13.5 User Testing

Toward the conclusion of Senior Design development, non-technical individuals tested the TrusNet system. In particular, these testers used the demonstration application User Interface and reported on the usability of the system.

Chapter 14

Test Results

14.1 Unit Testing Results

Our extensive unit testing allowed us to catch numerous problems far before they would have appeared in general operation. The unit tests were built during our standard build, so each call to build the project rebuilt the unit test suite. Thanks to this fact, we constantly ran the unit tests.

14.2 Speed Testing Results

Table 14.1 notes the performance of several critical steps in our security system. Notably, the initial key creation take excessive amounts of time, especially on the Intel Edison. However, the actual message passing (in the final column) is all under one thousandth of a second, even with 512 bit RSA authentication on the Intel Edison. Consult 14.1 for a graphical representation of one of these time profiling tests.

Table 14.1 Time Profiling

Test	i5 3.4 GHz	i5 1.6 GHz	Xeon 3.7 GHz	Edison
Key Generation				
RSA 128 bit	0.1861	0.9811	0.4207	4.7658
RSA 256 bit	3.8253	9.2788	3.4820	75.6635
RSA 512 bit	18.3665	90.6345	36.9524	675.8570
Basic Message Passing				
RSA 128 bit	0.0209	0.1448	0.0364	0.5063
RSA 256 bit	0.1094	0.4304	0.2449	3.6867
RSA 512 bit	0.6373	2.8642	1.7653	24.3530
Secure Gateway Connection				
RSA 128 bit	0.0454	0.1448	0.0761	1.1115
RSA 256 bit	0.3960	0.8666	0.4751	7.5162
RSA 512 bit	1.2737	5.9611	3.5429	49.8386
Gateway Message Passing				
RSA 128 bit	2.504e-05	8.038e-05	5.282e-05	8.745e-04
RSA 256 bit	2.526e-05	8.028e-05	5.234e-05	8.837e-04
RSA 512 bit	2.536e-05	7.978e-05	5.476e-05	8.818e-04

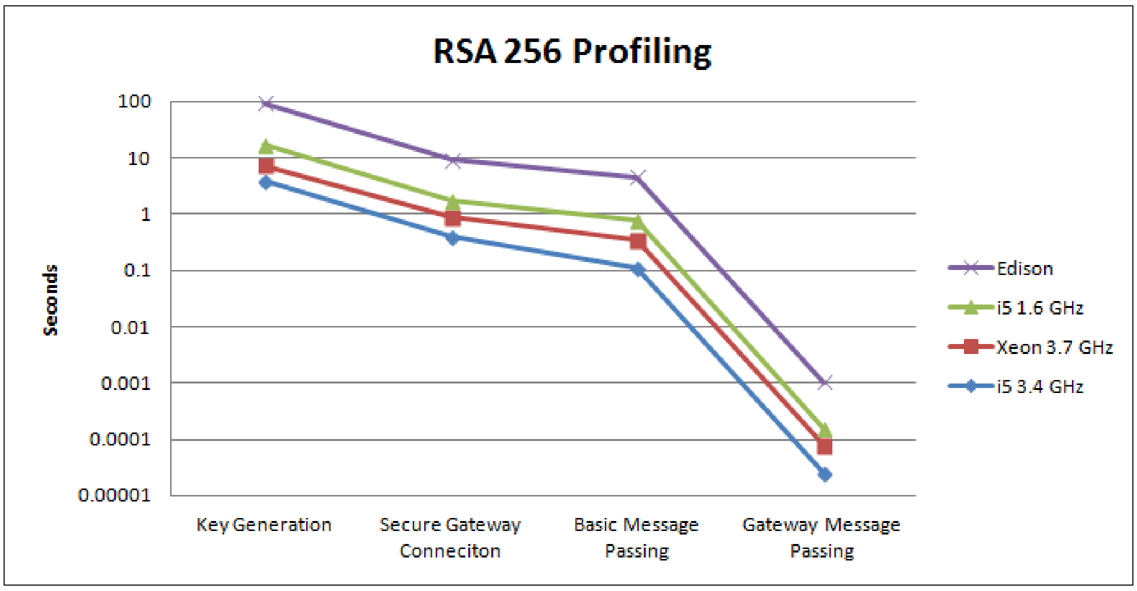


Figure 14.1 256 bit RSA time profiling

14.3 Demonstration Testing Results

The demonstration unit contained its own unit tests, which were tested in the macro unit testing commands whenever the test suite was run on an Intel Edison platform. Additionally, we also had a dedicated embedded system test which ran various sweeps across motors, tested motor enables and disables, and confirmed the accurate reading of the joystick unit. These tests required human input and verification, but allowed us to tune and test our demonstration system before the TrusNet protocol was fully operational.

Chapter 15

User Manual

The user manual provides a description of how to use each of the three deliverables of the TrusNet project. For those interested in using the library, consult 15.3 to see how to download, compile and link the source code. Very detailed documentation is provided in the repository containing the source-code of the library.

15.1 Headless Application

The headless application is designed for use with the UI application. However, before operation, several steps are required to ensure that every node is prepared to demonstrate.

15.1.1 Unit Testing

When adding a node, it is important to make sure that all the necessary libraries and subsystems are installed on a given node. Furthermore, we wish to ensure that all the functions used in the demonstration and TrusNet protocol are operational. In order to ensure proper functionality, running the UnitTestExe executable is the first step when working with any node, regardless of type. If any tests fail, they will be noted in the terminal output or any file you wish to pipe into. If small changes have been made to a node after the initial unit test, calling "UnitTestExe fast" from your terminal client. This skips the RSA generation and verification tests. Once the unit tests are successful, you can be sure that a given platform can run TrusNet.

15.1.2 Edison Setup

For the Intel Edisons specifically, several additional steps are required for demonstration. Because each Edison can be a joystick, robot, or standard node, you must configure each Edison for its respective role. The configuration file is located at Remote_Files/<Device Name>/Remote_Controller. Here, you can change the "No Hardware" line to "joystick" or "bot" for those respective nodes. You should also make a note of the IP address for each node. This will assist you in setting up the network as well as allow you to remotely connect to the Edisons.

15.1.3 Network Setup

Once all nodes have been setup, unit tests run, and Edisons configured, you must setup the network. Launch BaseForm on a computer and use the UI application to add all the nodes to that computer via the IP form element of the UI application. Then boot all the nodes one at a time until all are online running either RemoteMain or BaseForm. Thanks to the IP sharing, each node will get a copy of all the IPs from the inital node, then each will verify the existence of all other nodes. The network is now ready to be demonstrated.

15.1.4 Demonstration

For demonstration purposes, robots bind to the first joystick they encounter. We recommend that you launch your joystick node first, then launch your robot node, then launch any other nodes you wish to have in the network. This will ensure that the joystick and robot node connect to each other first and allow for optimal demonstration.

15.2 UI Application

The UI application is a multi-user graphical interface which visualizes the intrinsics of TrusNet. This application is designed to be used on multiple different operating systems.

15.2.1 Getting Started

On both Windows and Unix systems, the BaseForm application is accessed by clicking on the executable. Even on Mac, where such behavior usually opens an application the user’s root, will localize the application to the directory it is contained in. BaseForm will save its files in a directory called "WifiRC_Base."

On Windows, a number of .dll files are needed to run BaseForm. The list is:

- freeglut.dll
- msvcp110.dll
- msvcr110.dll

These files should have been included in your distribution of the UI application.

15.2.2 Logging In

Upon starting the application, Figure 15.1 is an example of the form a user will encounter. If a user is returning to this machine, the user logs in. If, however, a user is new, this user must create a new user as shown in Figure 15.2. Note that this particular form asks a prospective user for a password twice, and Figure 15.3 shows an example of what happens if that check should fail. After choosing a user-name, password and initial public key size, keys are generated and a waiting form is displayed until generation is complete, as shown in Figure 15.4.

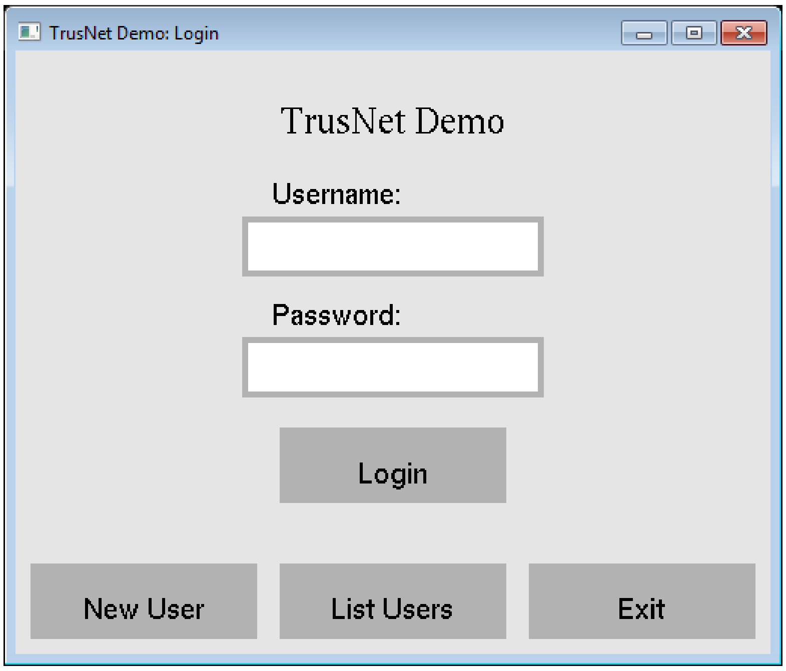
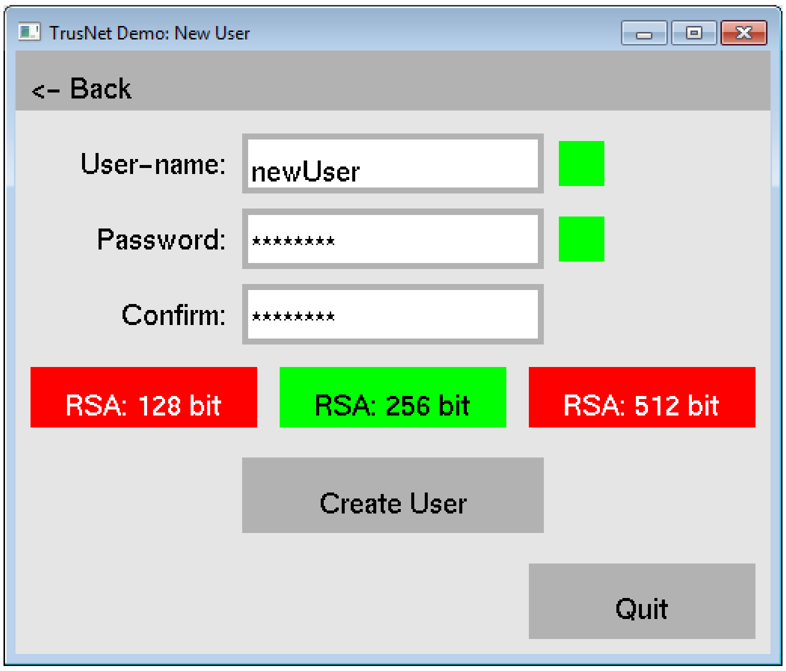


Figure 15.1 Basic login form


TrusNet Demo: New User

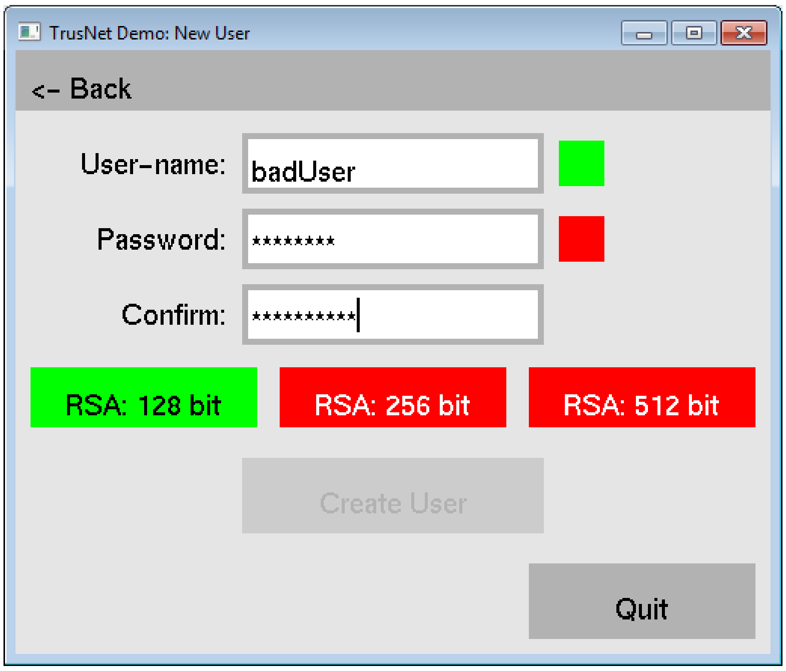
<- Back

User-name: newUser 🟩

Password: ******** 🟩

Confirm: ********

Figure 15.2 Building a new user


TrusNet Demo: New User

<- Back

User-name: badUser 🟩

Password: ******** 🟥

Confirm: **********

Figure 15.3 New user with mis-matching passwords

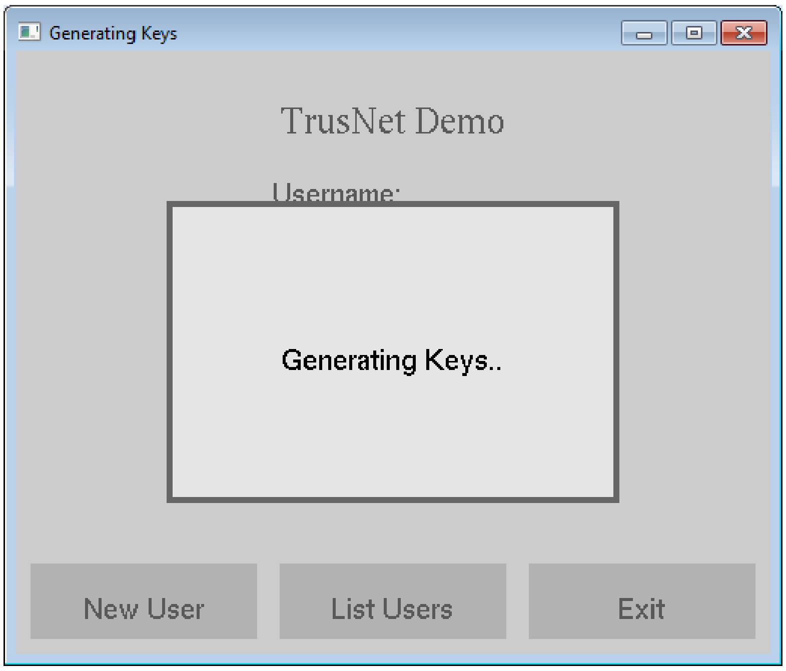
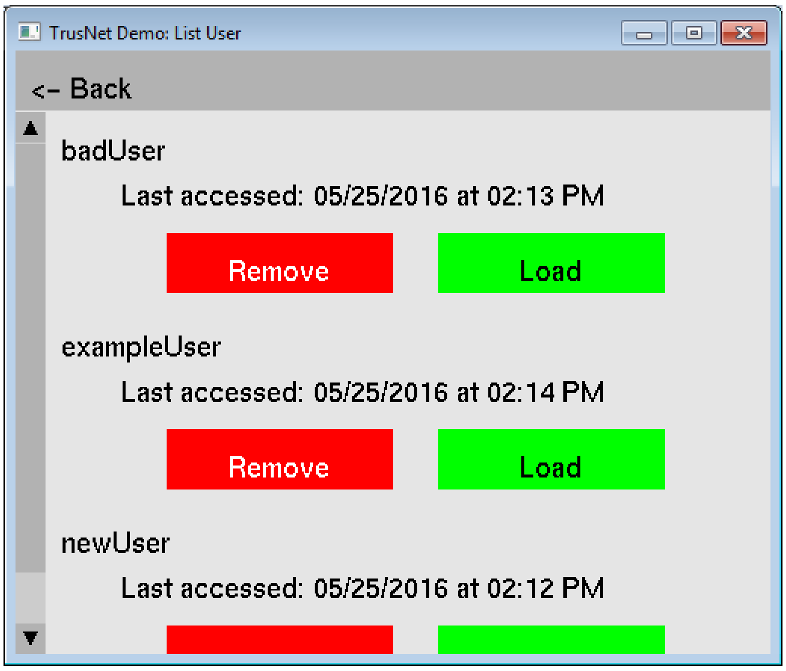


Figure 15.4 Generating new RSA keys

To check if a user is saved on this machine, enter into the "List Users" form. Figure 15.5 shows an example of what this form may look like. Note that users can also be deleted from a machine through this form. Lastly, if a password is entered incorrectly, a pop-up like the one in Figure 15.6 will be displayed.


TrusNet Demo: List User

<- Back

badUser

Last accessed: 05/25/2016 at 02:13 PM

Remove | Load

exampleUser

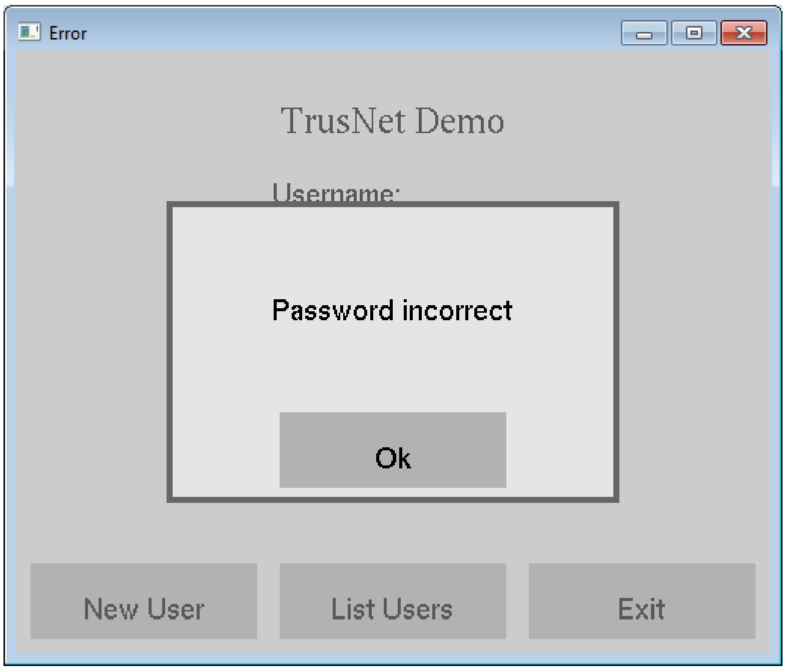
Last accessed: 05/25/2016 at 02:14 PM

Remove | Load

newUser

Last accessed: 05/25/2016 at 02:12 PM

Figure 15.5 List of known users


Error

TrusNet Demo

Username:

Password incorrect

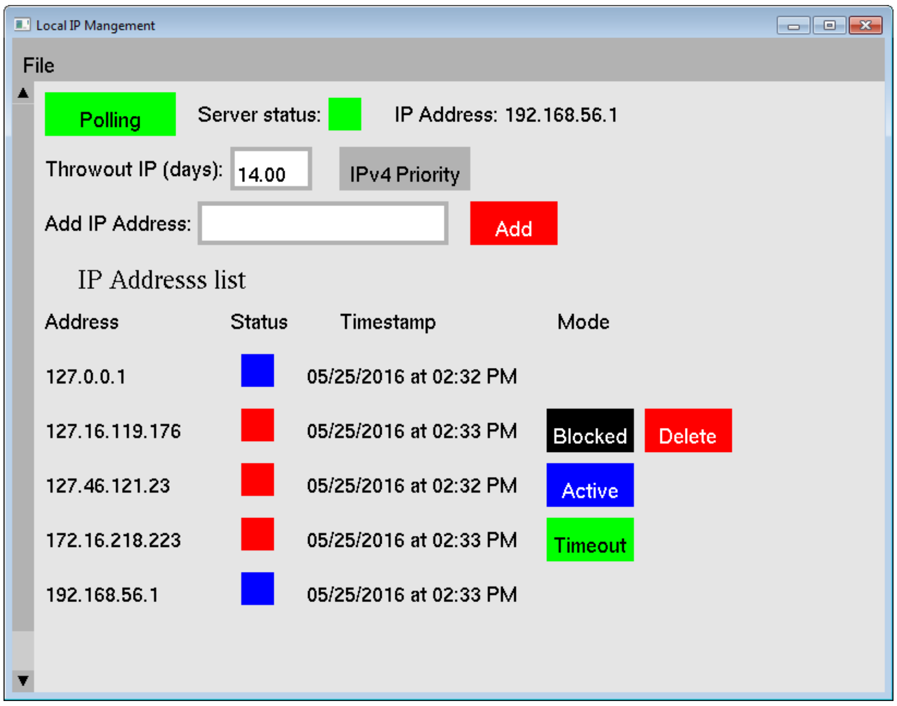
Ok

New User | List Users | Exit

Figure 15.6 Password error pop-up

15.2.3 IP Address Management

Figure 15.7 demonstrates what the IP address management form looks like. This form allows for adding both IPv4 and IPv6 addresses, and is managing both an IPv4 and IPv6 server. Additionally, IP Addresses can be blocked, constantly polling or polling until some time-out value is reached. IP Addresses also are paired with an indicator, which is either blue, red, yellow or green. Blue indicates the IP address is referencing this machine, red indicates no connection has been established, yellow indicates a connection is being authenticated and green indicates a connection is both secure and authenticated.



Server status: 🟩 IP Address: 192.168.56.1

Throwout IP (days): 14.00

Add IP Address:

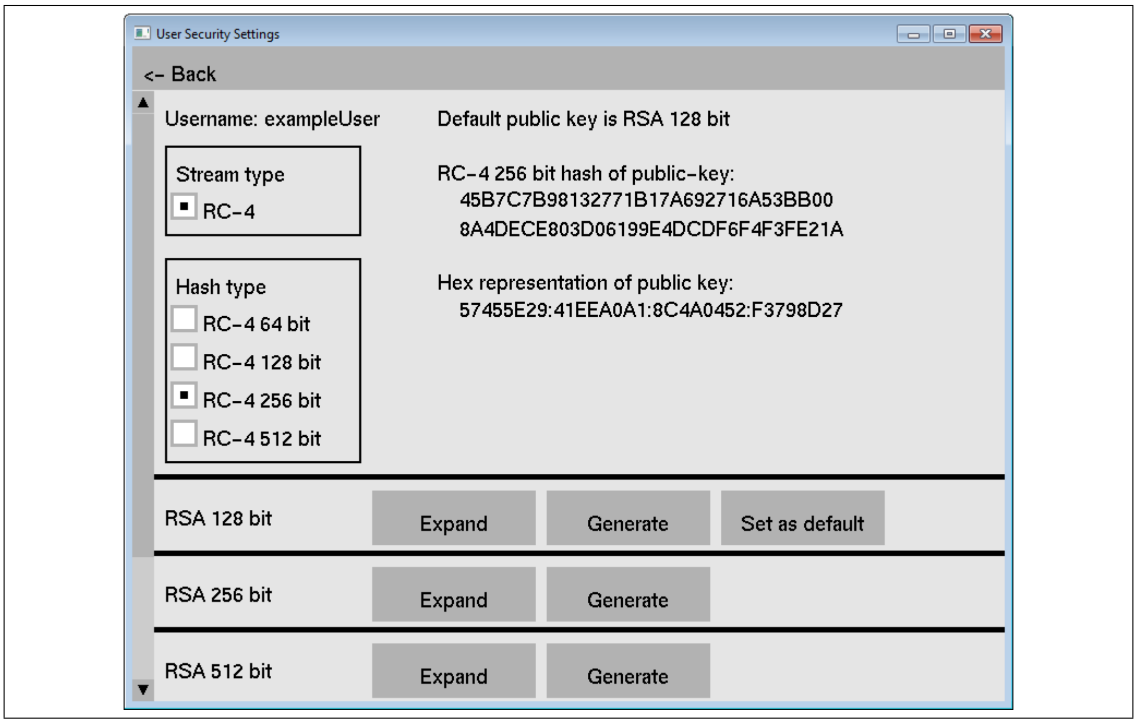
IP Addresss list

Address	Status	Timestamp	Mode	
127.0.0.1	🟦	05/25/2016 at 02:32 PM		
127.16.119.176	🟥	05/25/2016 at 02:33 PM	Blocked	Delete
127.46.121.23	🟥	05/25/2016 at 02:32 PM	Active	
172.16.218.223	🟥	05/25/2016 at 02:33 PM	Timeout	
192.168.56.1	🟦	05/25/2016 at 02:33 PM		

Figure 15.7 Managing IP addresses

15.2.4 Key Management

Figure 15.8 shows an example of management of cryptographic algorithms and keys. Through this interface, users can change what their preferred algorithms are and regenerate public keys.


User Security Settings

<- Back

Username: exampleUser

Default public key is RSA 128 bit

Stream type
- [✓ RC-4]

RC-4 256 bit hash of public-key:
45B7C7B98132771B17A692716A53BB00
8A4DECE803D06199E4DCDF6F4F3FE21A

Hash type
- RC-4 64 bit
- RC-4 128 bit
- [✓ RC-4 256 bit]
- RC-4 512 bit

Hex representation of public key:
57455E29:41EEA0A1:8C4A0452:F3798D27

RSA 128 bit — Expand — Generate — Set as default

RSA 256 bit — Expand — Generate

RSA 512 bit — Expand — Generate

Figure 15.8 Cryptographic management

15.2.5 Node Interaction

In the main form, users can interact with connected nodes. Figure 15.9 gives an example of the list of connected node. Similar to the IP Address management in section 15.2.3, red indicates no connection, yellow indicates an active authentication process and green indicates a connection is secure and authenticated. Figure 15.10 shows an example of the node interaction menu when a node is unconnected. Figure 15.11 demonstrates the interface when a node is a joystick, and constantly broadcasting messages indicating the position of the joystick. Lastly, Figure 15.12 shows the interface when a form is both sending commands to and receiving feedback from a remote node running under the robot configuration.

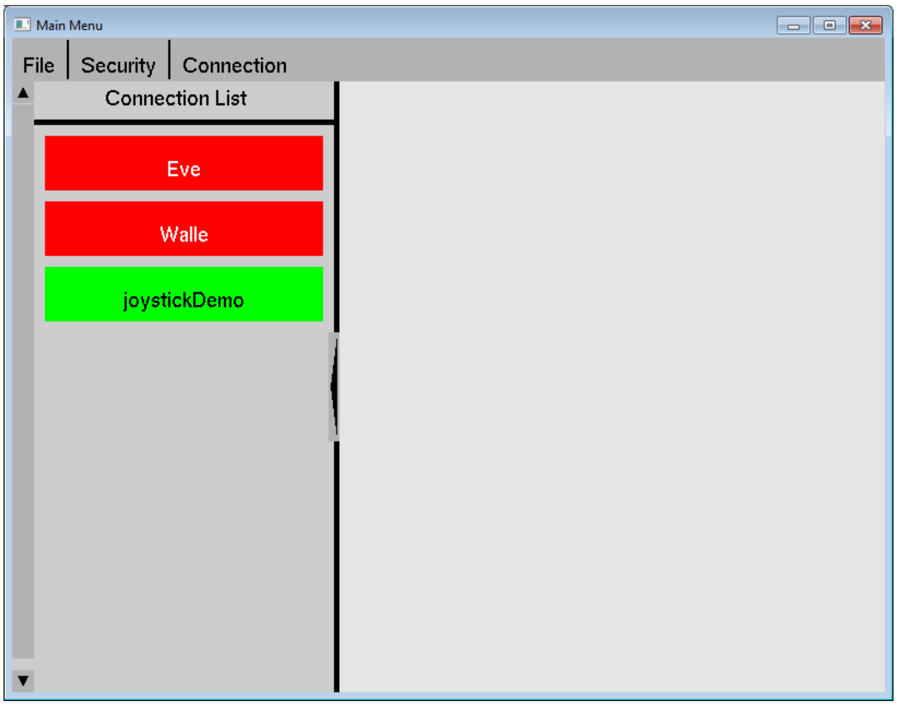
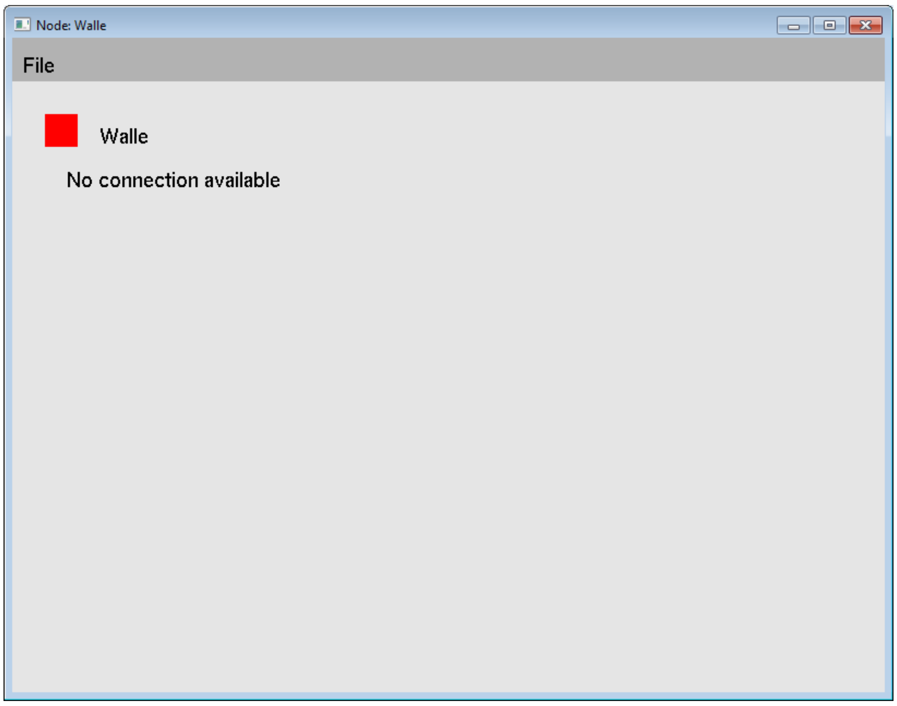


Figure 15.9 Connection list



Walle

No connection available

Figure 15.10 Node interface: no connection

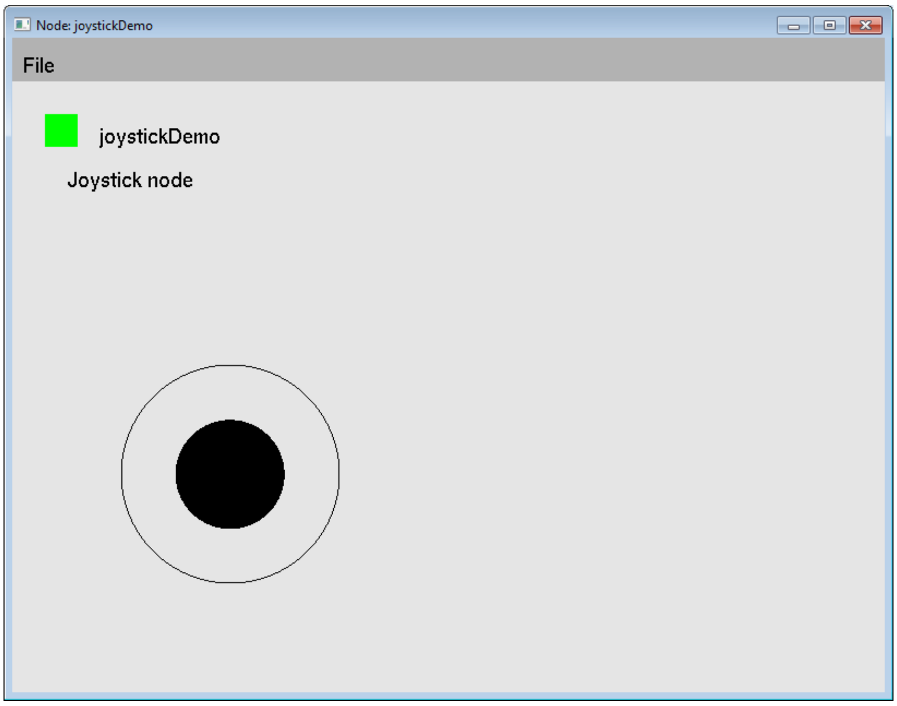


Figure 15.11 Node interface: joystick

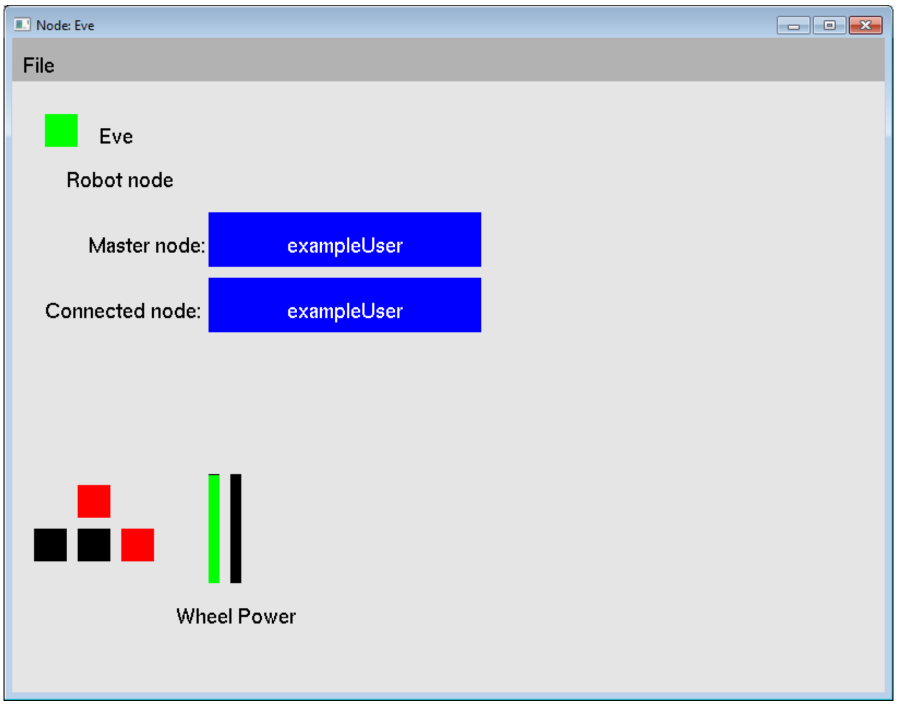


Figure 15.12 Node interface: robot

15.3 Library Usage

Building the source code for this project and linking this project to outside projects requires a number of outside software tools. This manual also provides a description of the scripts involved, what they do and how to make any needed modifications.

15.3.1 Required Tools

In-order to properly utilize the scripts provided by the TrusNet architecture, users first need to download the tools below:

- CMake
- git
- Doxygen (optional)

All three of these tools are available for both Windows and Unix systems. Note that these tools must be installed on the command line. If these tools are improperly installed, the scripts will alert the user. Note that for Unix systems, bash must be installed, and for Windows systems, batch files need to be executable. Scripts are provided for both Unix and Windows systems.

15.3.2 Repositories

The TrusNet project is primarily stored on github. The primary repository is the WifiRC_Builder repository, who's address is:

```
https://github.com/JonWBedard/WifiRC_Builder
```

This repository contains a script to pull the correct version of all repositories required to build the TrusNet project. To pull the most recent version of the project on a Unix system, run:

```
$ git clone https://github.com/JonWBedard/WifiRC_Builder
$ cd WifiRC_Builder
$ git checkout June2016\_Final
$ ./pullRepos.bash
```

Note that the final version of the project as of June 2016 is on the "June2016_Final" branch. The details of the master branch will change as development updates occur, and this document become out of date.

15.3.3 IDE Support

CMake support multiple different IDEs, although our scripts only support five build types. Our scripts search for a set of installed IDEs, a list of which is below. If none of these IDEs is found, a simple makefile is used.

- Visual Studios (10-14)
- XCode
- Eclipse
- Code Blocks

Note that if you do not like the behavior of the automatic IDE selection scripts, you can modify the scripts in Datastructures/Windows/genIDE.bat for Windows or Datastructures/Unix/genIDE.bash for Unix systems. If you intend on including the library in your own project, consult the scripts in WifiRC_Builder to see how CMake is called from scripts.

15.3.4 Building and Compiling

After all repositories are installed, run buildWindowsWifiRC.bat if on a Windows system or buildUnixWifiRC.bash if on a Unix system. This script will create a new folder on the same level as "WifiRC_Builder" called "build," and place the build folder "WifiRC" and "WifiRC_Headless" in the build folder.

In order to link either TrusNet or CryptoGateway to your own project through CMake, use the following structure:

```
INCLUDE(.../CryptoGateway/CMakeLists.txt)
INCLUDE(.../TrusNet/CMakeLists.txt)
...
TARGET_LINK_LIBRARIES( EXE\_NAME
                ${OS_LIBS}
                ${EXE_LIBS}
                ...
        )
        ...
```

The two macros linked to are a list of libraries TrusNet and CryptoGateway depend on, including TrusNet and CryptoGateway. Consult the detailed documentation contained in the TrusNet and CryptoGateway repositories for usage of the actual libraries.

Chapter 16

Documentation

The appendix contains basic documentation for each of the libraries and modules included in the project. These parts describe the intended use of each library or module, its dependencies and the classes inside of it. Note that more detailed documentation is available on-line. For brevity, however, this documentation has been left out. The libraries and modules used in this project are as follows.

- Datastructures
- UnitTest
- osMechanics
- CryptoGateway
- TrusNet
- EdisonHAL
- WifiRC
- RemoteMain
- glGraphics
- CryptoLogin
- BaseForm

Figure 16.1 visualizes the dependencies between the libraries. As over viewed in the User Manual, the WifiRC_Builder repository should be used to pull all of the dependent repositories when one intends on building any one of the deliverables. Figure 16.1 does not include testing libraries or testing executables, although some are built for this project. These testing assets are as follows:

- osMechanicsTest
- CryptoGatewayTest
- EdisonHalTest
- glGraphicsTest
- UnitTestExe (executable)
- HardwareTest (executable)
- SpeedProfiling (executable)

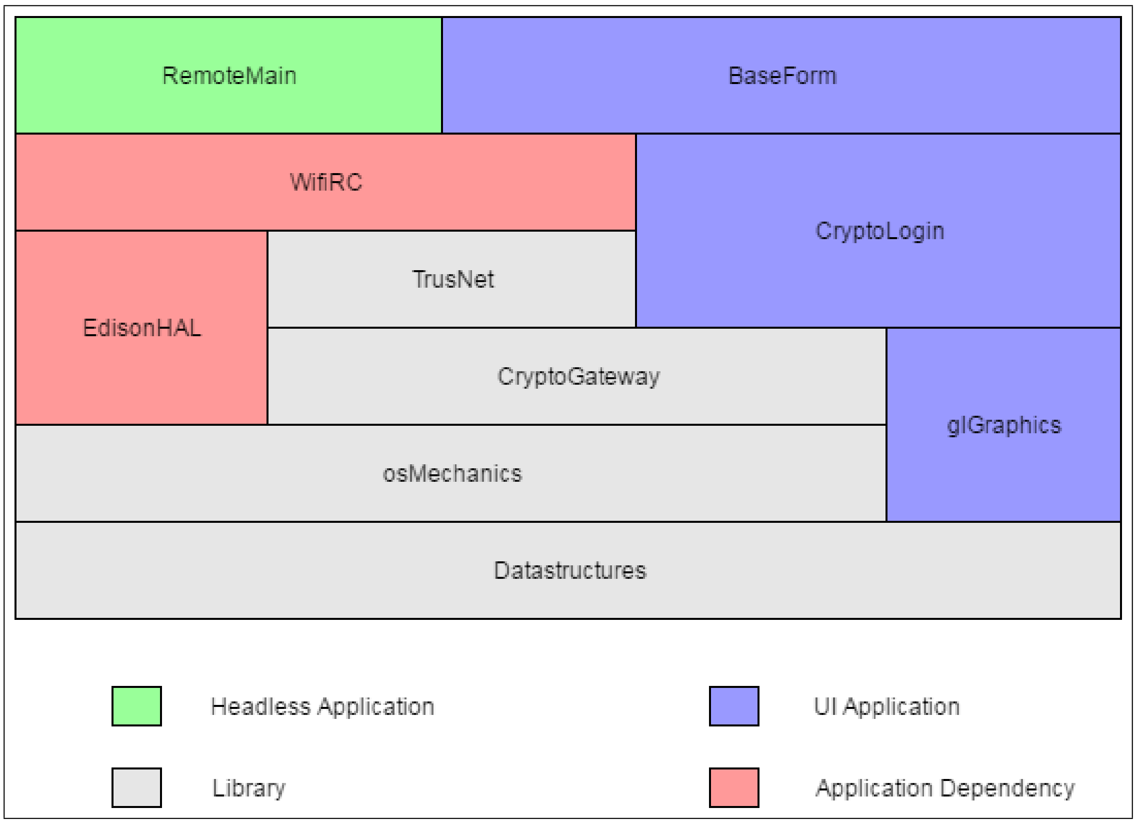


Figure 16.1 Library dependencies

Bibliography

Bar-Yosef, N. (2012). Understanding public key cryptography and the history of rsa.

Blaze, M. (1994). Protocol failure in the escrowed encryption standard.

Bormann, C. (2015). Constrained application protocol.

Donovan, P. (2012). The flaw in the jn-25 series of ciphers, ii. *Cryptologia*, 36.

Gillogly, J. (1995). Ciphertext-only cryptanalysis of enigma. *Cryptologia*, 19(4).

GreenBerg, A. (2015). Hackers remotely kill a jeep on the highway - with me in it.

IEEE (2016). Code of ethics ieee.

Kemerling, G. (2011). Kant the moral order.

Kiosowski, T. (2014). What is tor and should i use it?

Koop, B. (1996). Encryption law–ii: A survey of cryptography laws and regulations'. *Computer Law and Security Reporter*, 12:349–355.

Koop, B. (2013). Crypto law survey.

Lomas, N. (2016). Whatsapp completes end-to-end encryption rollout.

Niem, T. C. (11-04-2002). Bluetooth and its inherent security issues. Technical report, SANS GIAC Security Essentials Certification.

Patel, H. (2015). *Improving ZigBee Device Network Authentication Using Ensemble Decision Tree Classifiers With Radio Frequency Distinct Native Attribute Fingerprinting*, volume 64. IEEE.

Prados, J. (1996). Battle of midway.

Risen, T. (2015). Vtech hack shows kids at risk with wifi toys.

Selberg, A. (1949). *An Elementary Proof of the Prime-Number Theorem*, volume 50 of *Second*. Annals of Mathematics.

Selent, D. (2010). Advanced encryption standard. *Rivier Academic Journal*, 6.

Smid, M. and Branstad, D. (1988). The data encryption standard: Past and future. *Proceedings of the IEEE*, 76.

Staff, D. D. (2016a). Senate bill effectivly bans strong encryption.

Staff, N. Y. T. (2016b). Breaking down apple's iphone fight with the u.s. government.

Appendix A

Gant Charts

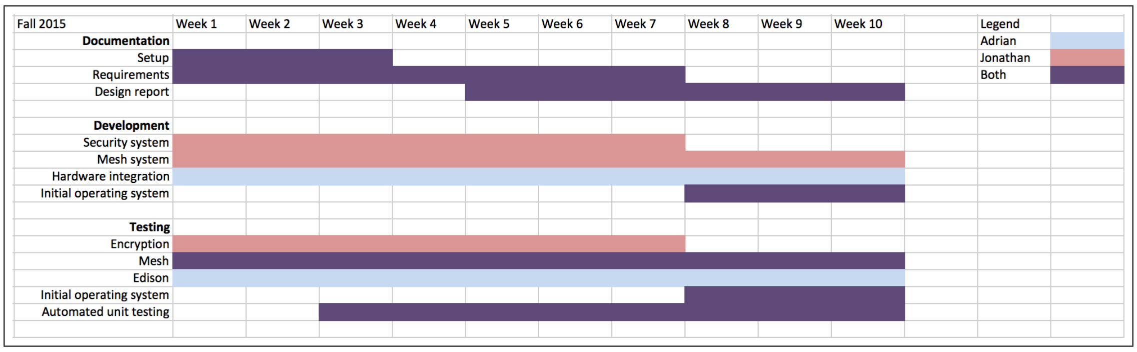


Figure A.1 Fall 2015 Gantt chart

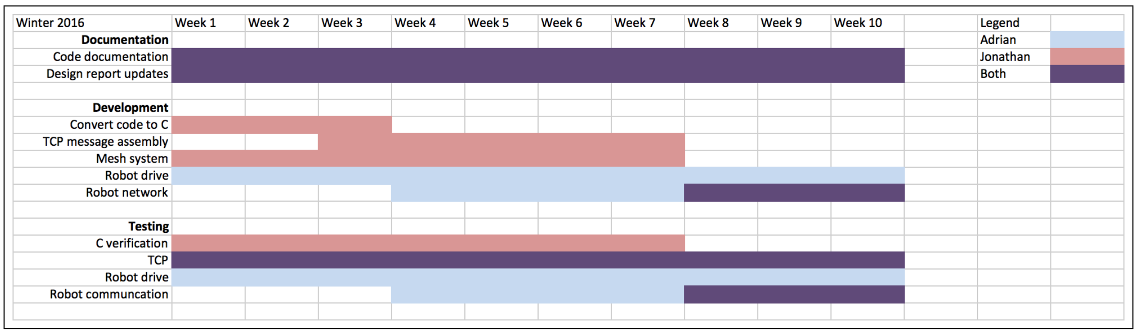
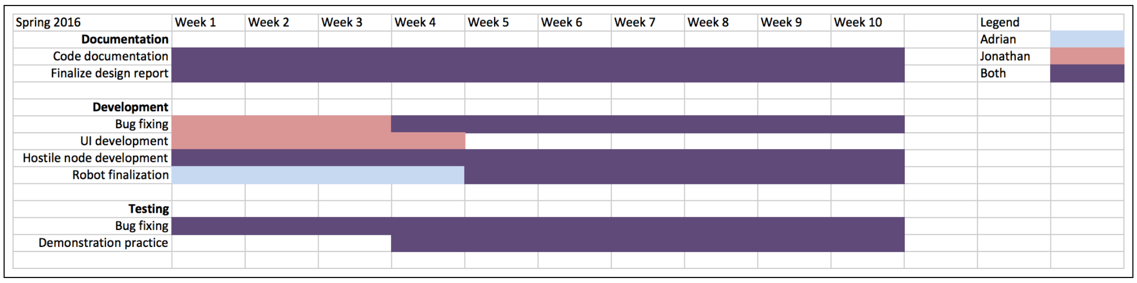


Figure A.2 Winter 2016 Gantt chart



Spring 2016	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10		Legend	
Documentation												Adrian	🟦
Code documentation	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪		Jonathan	🟥
Finalize design report	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪		Both	🟪
Development													
Bug fixing	🟥	🟥	🟥	🟪	🟪	🟪	🟪	🟪	🟪	🟪			
UI development	🟥	🟥	🟥	🟥									
Hostile node development	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪			
Robot finalization	🟦	🟦	🟦	🟦	🟪	🟪	🟪	🟪	🟪	🟪			
Testing													
Bug fixing	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪			
Demonstration practice				🟪	🟪	🟪	🟪	🟪	🟪	🟪			

Figure A.3 Spring 2016 Gantt chart

Appendix B

Risk Tables

B.1 Management Risks

Risks in this category impact the ability of team members to complete their tasks, or work adequately together. Management risks are not technical in nature, and are shown in Table B.1.

B.2 Design Risks

Design risks are risks which effect the central design premises used in the project. Risks in this section have some of the highest impacts because these risks have the potential to undermine the paradigm which TrusNet was conceived in. These risks are outlined in Table B.2.

B.3 Demonstration Risks

This category of risks is entirely technical in nature. These risks effect only the ability to demonstrate the system, not a failure of the core functionality of the system. Table B.3 outlines these risks.

B.4 Security Risks

Since the core functionality of the TrusNet system is security, risks in this category effect the core functionality of the system. While primarily technical in nature, these risks can also be legal as well and are outlined in Table B.4.

Table B.1 Management Risks

Risk	Consequences	Probability	Severity	Impact	Mitigation Strategy
Illness	Some members will be unable to work for a time	0.7	5	3.5	To avoid problems brought on by illness, we have a system of repositories, allowing us to work individually. Additionally, we have regular theory crafting meeting to ensure we each understand each module, allowing for us to temporarily take over for one another.
Job demands	Both of us have jobs, so pressing business requirements may slow progress	0.1	8	0.8	We are staying on a tight schedule to ensure that we have plenty of extra time later in the year to finish work delayed by job requirements.

Table B.2 Design Risks

Risk	Consequences	Probability	Severity	Impact	Mitigation Strategy
Mesh Problem	Building a constantly updating and intelligent network is no simple task. Even within a single medium, we are likely to have issues connecting and communicating efficiently.	0.8	0.6	4.8	We are separating the individual communication and mesh elements of our network. Even with a poor mesh, our security and communication systems still provide useful advancements for other mesh networks.
Porting Difficulty	We may need to put the code on yet unknown hardware.	0.2	7	1.4	The entire TrusNet API is built on core libraries and C++. Given that almost every piece of hardware runs C++ code, we should be able to add new platforms with little effort.

Table B.3 Demonstration Risks

Risk	Consequences	Probability	Severity	Impact	Mitigation Strategy
Platform Variance	This network is designed to run on a variety of platforms. Each will have its own issues independent of the other platforms	0.8	5	4	Many of the things we have done for porting apply here. Additionally, we are making use of CMake to standardize compilation across systems.
Edison Failure	One or more of our Edison boards may become inoperable.	0.3	8	2.4	In order to prevent damage to the Edisons, we keep them stored in packaging. We also have six boards, but our project only needs 3 to demonstrate its functionality.
Wifi Problem	The wifi communication methods we are currently using may have unseen problems.	0.1	5	0.5	Wifi is well understood, so unknown problems are unlikely. In case of wifi problems, we can use other technologies to communicate, such as Bluetooth.
Robot Failure	We have access to two simple omnibots from the robotics systems lab. While reliable, these robots are not always perfect and may have issues.	0.1	5	0.5	One of the omnibots is kept as a demonstration unit while the other is undergoing a rebuild. Once complete, the rebuilt robot will have almost total hardware redundancy.

Table B.4 Security Risks

Risk	Consequences	Probability	Severity	Impact	Mitigation Strategy
Security Failure	Encryption is always advancing. Our security methods may become broken during development.	0.1	9	0.9	We have built our own key generation algorithms to ensure that outside access is minimal. We have also abstracted the security so that we can change algorithms without breaking our overall security theory.
Legal problems	Various governments have a history of outlawing encryption or other wise rendering this technology illegal.	0.02	10	0.2	The likelihood of our technology being outlawed in the United States is fairly low. Should a single encryption method be banned, we can change certain elements to ensure the overall project can continue.

Part I

Datastructures Library

Appendix C

Introduction

The Datastructures library contains a series of utility classes and template classes used for the organization and management of data. Most notably, this library allow dynamic memory management through the smart_ptr class and provides a flexible runtime data container in the ads (Abstract Data Structure) template and its children.

C.1 Unit Testing

The testing of the Datastructures library is contained within the UnitTest library. Since the UnitTest library uses the functionality of the Datastructures library, the Datastructures library cannot be dependent on the UnitTest library as the UnitTest library is already dependent on the Datastructures library

C.2 Namespace os

Datastructures extends the os namespace. The os namespace is designed for tools, algorithms and data-structures used in programs of all types. Structures in this library do not implement operating system specific interfaces such as sockets and file I/O. The osMechanics library also extends the os namespace.

Appendix D

File Index

D.1 File List

Here is a list of all files with brief descriptions:

Appendix E

File Documentation

E.1 Datastructures.h File Reference

Master Datastructures header file.

E.1.1 Detailed Description

Master Datastructures header file.

Author

Jonathan Bedard

Date

2/14/2016

Bug No known bugs.

All of the headers in the Datastructures library are held in this file. When using the Datastructures library, it is expected that this header is included instead of the individual required headers.

E.2 abstractSorting.h File Reference

Template for sorting arrays.

Namespaces

- **os**

Functions

- template<class dataType >

  int **os::defaultCompareSort** (const dataType &v1, const dataType &v2)

  Basic compare.

- template<class dataType >

  int **os::pointerCompareSort** (smart_ptr< dataType > ptr1, smart_ptr< dataType > ptr2)

  Raw pointer compare.

- template<class dataType >

  void **os::quicksort** (dataType ∗arr, unsigned int length, int(∗sort_comparison)(const dataType &, const dataType &)=&defaultCompareSort)

  Template quick-sort.

- template<class dataType >

  void **os::pointerQuicksort** (smart_ptr< smart_ptr< dataType > > arr, unsigned int length, int(∗sort_comparison)(smart_ptr< dataType >, smart_ptr< dataType >)=&pointerCompare↩ Sort)

  Template for quick-sort, pointer version.

E.2.1 Detailed Description

Template for sorting arrays.

Author

Jonathan Bedard

Date

2/15/2016

Bug No known bugs.

This file contains a template class definition of an AVL tree and its nodes. This tree has insertion, search and deletion of O(log(n)) where n is the number of nodes in the tree. This tree is thread safe.

E.3 ads.h File Reference

Abstract datastructure interface.

Classes

- class **os::ptrComp**
  Pointer compare interface.
- class **os::adnode**< **dataType** >
  Abstract data-node.
- class **os::ads**< **dataType** >
  Abstract datastructure.

Namespaces

- **os**

E.3.1 Detailed Description

Abstract datastructure interface.
Author

Jonathan Bedard

Date

5/9/2016

Bug No known bugs.

This file contains definitions of a set of class interfaces used by abstract datastructures and classes interfacing with abstract datastructures.

E.4 asyncAVL.h File Reference

Asynchronous AVL tree.

Classes

- class **os::asyncAVLNode**< **dataType** >
  Node for usage in an asynchronous AVL tree.
- class **os::asyncAVLTree**< **dataType** >
  Asynchronous balanced binary search tree.

Namespaces

- **os**

E.4.1 Detailed Description

Asynchronous AVL tree.

Author

Jonathan Bedard

Date

5/9/2016

Bug No known bugs.

This file contains a template class definition of an AVL tree and its nodes. This tree has insertion, search and deletion of O(log(n)) where n is the number of nodes in the tree. This tree is thread safe.

E.5 AVL.h File Reference

AVL tree.

Classes

- class **os::AVLNode**< **dataType** >
  Node for usage in an AVL tree.
- class **os::AVLTree**< **dataType** >
  Balanced binary search tree.

Namespaces

- **os**

E.5.1 Detailed Description

AVL tree.

Author

Jonathan Bedard

Date

2/12/2016

Bug No known bugs.

This file contains a template class definition of an AVL tree and its nodes. This tree has insertion, search and deletion of O(log(n)) where n is the number of nodes in the tree. This tree is not thread safe.

E.6 eventDriver.h File Reference

Event sender and receiver.

Classes

- class **os::eventSender**< **receiverType** >
  Class which enables event sending.
- class **os::eventReceiver**< **senderType** >
  Class which enables event receiving.

Namespaces

- **os**

Variables

- std::recursive_mutex ∗ **os::eventLock**
  Event processing mutex.

E.6.1 Detailed Description

Event sender and receiver.

Author

Jonathan Bedard

Date

5/9/2016

Bug No known bugs.

Both **os::eventReceiver** (p. **??**) and **os::eventSender** (p. **??**) are experimental classes and have not been tested or utilized.

E.7 eventDriver.cpp File Reference

Event driver implementation.

E.7.1 Detailed Description

Event driver implementation.

Author

Jonathan Bedard

Date

2/28/2016

Bug No known bugs.

This file implements **os::eventLock** (p. **??**) for **os::eventSender** (p. **??**) and **os::eventReceiver** (p. **??**). These are experimental class and not yet used or tested

E.8 list.h File Reference

Doubly Linked List.

Classes

- class **os::unsortedListNode**< **dataType** >
  Node for usage in a linked list.
- class **os::unsortedList**< **dataType** >
  Unsorted linked list.

Namespaces

- **os**

E.8.1 Detailed Description

Doubly Linked List.

Author

Jonathan Bedard

Date

2/1/2016

Bug No known bugs.

This file contains a template class definition of a linked list and its nodes. This list has insertion, find and delete of O(n). The linked list provided is doubly linked, allowing for forward and backward traversal. This list is not thread safe.

E.9 matrix.h File Reference

Matrix templates.

Classes

- class **os::matrix**< **dataType** >
  Raw matrix.
- class **os::indirectMatrix**< **dataType** >
  Indirect matrix.

Namespaces

- **os**

Functions

- template<class dataType >

  bool **os::compareSize** (const matrix< dataType > &m1, const matrix< dataType > &m2)

  Compares the size of two matrices.
- template<class dataType >

  bool **os::compareSize** (const indirectMatrix< dataType > &m1, const matrix< dataType > &m2)

  Compares the size of two matrices.
- template<class dataType >

  bool **os::compareSize** (const matrix< dataType > &m1, const indirectMatrix< dataType > &m2)

  Compares the size of two matrices.
- template<class dataType >

  bool **os::compareSize** (const indirectMatrix< dataType > &m1, const indirectMatrix< dataType > &m2)

  Compares the size of two matrices.
- template<class dataType >

  bool **os::testCross** (const matrix< dataType > &m1, const matrix< dataType > &m2)

  Tests if the cross-product is a legal operation.
- template<class dataType >

  bool **os::testCross** (const indirectMatrix< dataType > &m1, const matrix< dataType > &m2)

  Tests if the cross-product is a legal operation.
- template<class dataType >

  bool **os::testCross** (const matrix< dataType > &m1, const indirectMatrix< dataType > &m2)

  Tests if the cross-product is a legal operation.
- template<class dataType >

  bool **os::testCross** (const indirectMatrix< dataType > &m1, const indirectMatrix< dataType > &m2)

Tests if the cross-product is a legal operation.

- template<class dataType >

  bool **operator==** (const **os::matrix**< dataType > &m1, const **os::matrix**< dataType > &m2)

  Test for equality.

- template<class dataType >

  bool **operator==** (const **os::indirectMatrix**< dataType > &m1, const **os::matrix**< dataType > &m2)

  Test for equality.

- template<class dataType >

  bool **operator==** (const **os::matrix**< dataType > &m1, const **os::indirectMatrix**< dataType > &m2)

  Test for equality.

- template<class dataType >

  bool **operator==** (const **os::indirectMatrix**< dataType > &m1, const **os::indirectMatrix**< data↩Type > &m2)

  Test for equality.

- template<class dataType >

  bool **operator!=** (const **os::matrix**< dataType > &m1, const **os::matrix**< dataType > &m2)

  Test for inequality.

- template<class dataType >

  bool **operator!=** (const **os::indirectMatrix**< dataType > &m1, const **os::matrix**< dataType > &m2)

  Test for inequality.

- template<class dataType >

  bool **operator!=** (const **os::matrix**< dataType > &m1, const **os::indirectMatrix**< dataType > &m2)

  Test for inequality.

- template<class dataType >

  bool **operator!=** (const **os::indirectMatrix**< dataType > &m1, const **os::indirectMatrix**< data↩Type > &m2)

  Test for inequality.

- template<class dataType >

  os::matrix< dataType > **operator+** (const **os::matrix**< dataType > &m1, const **os::matrix**< dataType > &m2)

  Addition.

- template<class dataType >

  os::matrix< dataType > **operator+** (const **os::indirectMatrix**< dataType > &m1, const **os**↩ **::matrix**< dataType > &m2)

  Addition.

- template<class dataType >

  os::matrix< dataType > **operator+** (const **os::matrix**< dataType > &m1, const **os::indirect**↩ **Matrix**< dataType > &m2)

  Addition.

- template<class dataType >

  os::indirectMatrix< dataType > **operator+** (const **os::indirectMatrix**< dataType > &m1, const **os::indirectMatrix**< dataType > &m2)

  Addition.

- template<class dataType >

  os::matrix< dataType > **operator-** (const **os::matrix**< dataType > &m1, const **os::matrix**< dataType > &m2)

  Subtraction.

- template<class dataType >

  os::matrix< dataType > **operator-** (const **os::indirectMatrix**< dataType > &m1, const **os**↩ **::matrix**< dataType > &m2)

  Subtraction.

- template<class dataType >

  os::matrix< dataType > **operator-** (const **os::matrix**< dataType > &m1, const **os::indirect**↩ **Matrix**< dataType > &m2)

  Subtraction.

- template<class dataType >

  os::indirectMatrix< dataType > **operator-** (const **os::indirectMatrix**< dataType > &m1, const **os::indirectMatrix**< dataType > &m2)

Subtraction.

- template<class dataType >

  os::matrix< dataType > **operator*** (const **os::matrix**< dataType > &m1, const **os::matrix**< dataType > &m2)

  Cross-product.

- template<class dataType >

  os::matrix< dataType > **operator*** (const **os::indirectMatrix**< dataType > &m1, const **os**↩ **::matrix**< dataType > &m2)

  Cross-product.

- template<class dataType >

  os::matrix< dataType > **operator*** (const **os::matrix**< dataType > &m1, const **os::indirect**↩ **Matrix**< dataType > &m2)

  Cross-product.

- template<class dataType >

  os::indirectMatrix< dataType > **operator*** (const **os::indirectMatrix**< dataType > &m1, const **os::indirectMatrix**< dataType > &m2)

  Cross-product.

- template<class dataType >

  os::matrix< dataType > **operator*** (const dataType &d1, const **os::matrix**< dataType > &m1)

  Scalar multiplication.

- template<class dataType >

  os::matrix< dataType > **operator*** (const **os::matrix**< dataType > &m1, const dataType &d1)

  Scalar multiplication.

- template<class dataType >

  os::matrix< dataType > **operator/** (const **os::matrix**< dataType > &m1, const dataType &d1)

  Scalar division.

- template<class dataType >

  os::indirectMatrix< dataType > **operator*** (const dataType &d1, const **os::indirectMatrix**< dataType > &m1)

  Scalar multiplication.

- template<class dataType >

  os::indirectMatrix< dataType > **operator**∗ (const **os::indirectMatrix**< dataType > &m1, const dataType &d1)

  Scalar multiplication.
- template<class dataType >

  os::indirectMatrix< dataType > **operator**/ (const **os::indirectMatrix**< dataType > &m1, const dataType &d1)

  Scalar division.
- template<class dataType >

  std::ostream & **operator**<< (std::ostream &os, const **os::matrix**< dataType > &dt)

  Prints out a matrix.
- template<class dataType >

  std::ostream & **operator**<< (std::ostream &os, const **os::indirectMatrix**< dataType > &dt)

  Prints out a matrix.

E.9.1 Detailed Description

Matrix templates.

Author

Jonathan Bedard

Date

2/2/2016

Bug No known bugs.

This file contains two template class definitions for matrices. One of these is an "indirect" matrix, meaning that the is an array of pointers, and the other is a direct matrix, meaning the matrix is an array of values.

E.9.2 Function Documentation

template<class dataType > bool operator!= (const **os::matrix**< dataType > & m1, const **os::matrix**< dataType > & m2)

Test for inequality.

Calls '==' and then inverts the result. Depends on the '!=' operator of dataType.

Parameters

`in`	*m1*	Raw matrix reference
`in`	*m2*	Raw matrix reference

Returns

False if exactly equivalent

template<class dataType > bool operator!= (const **os::indirectMatrix**< dataType > & m1, const **os::matrix**< dataType > & m2)

Test for inequality.

Calls '==' and then inverts the result. Depends on the '!=' operator of dataType.

Parameters

`in`	*m1*	Indirect matrix reference
`in`	*m2*	Raw matrix reference

Returns

False if exactly equivalent

template<class dataType > bool operator!= (const **os::matrix**< dataType > & m1, const **os::indirectMatrix**< dataType > & m2)

Test for inequality.

Calls '==' and then inverts the result. Depends on the '!=' operator of dataType.

Parameters

`in`	*m1*	Raw matrix reference
`in`	*m2*	Indirect matrix reference

Returns

False if exactly equivalent

template<class dataType > bool operator!= (const **os::indirectMatrix**< dataType > & m1, const **os::indirectMatrix**< dataType > & m2)

Test for inequality.

Calls '==' and then inverts the result. Depends on the '!=' operator of dataType.

Parameters

`in`	*m1*	Indirect matrix reference
`in`	*m2*	Indirect matrix reference

Returns

False if exactly equivalent

template<class dataType > **os::matrix**<dataType> operator∗ (const **os::matrix**< dataType > & m1, const **os::matrix**< dataType > & m2)

Cross-product.

Preforms the cross-product. The cross- product is undefined if the width of m1 does not equal the height of m2. If the cross-product is undefined, a matrix of size (0,0) will be returned. Depends on the '∗' and '+=' operator of the dataType.

Parameters

`in`	*m1*	Raw matrix reference
`in`	*m2*	Raw matrix reference

Returns

m1 x m2 (raw matrix)

template<class dataType > **os::matrix**<dataType> operator* (const **os::indirectMatrix**< dataType > & m1, const **os::matrix**< dataType > & m2)

Cross-product.

Preforms the cross-product. The cross- product is undefined if the width of m1 does not equal the height of m2. If the cross-product is undefined, a matrix of size (0,0) will be returned. Depends on the '*' and '+=' operator of the dataType.

Parameters

`in`	*m1*	Indirect matrix reference
`in`	*m2*	Raw matrix reference

Returns

m1 x m2 (raw matrix)

template<class dataType > **os::matrix**<dataType> operator* (const **os::matrix**< dataType > & m1, const **os::indirectMatrix**< dataType > & m2)

Cross-product.

Preforms the cross-product. The cross- product is undefined if the width of m1 does not equal the height of m2. If the cross-product is undefined, a matrix of size (0,0) will be returned. Depends on the '*' and '+=' operator of the dataType.

Parameters

`in`	*m1*	Raw matrix reference
`in`	*m2*	Indirect matrix reference

Returns

m1 x m2 (raw matrix)

template<class dataType > **os::indirectMatrix**<dataType> operator* (const **os::indirectMatrix**< dataType > & m1, const **os::indirectMatrix**< dataType > & m2)

Cross-product.

Preforms the cross-product. The cross- product is undefined if the width of m1 does not equal the height of m2. If the cross-product is undefined, a matrix of size (0,0) will be returned. Depends on the '*' and '+=' operator of the dataType.

Parameters

`in`	*m1*	Indirect matrix reference
`in`	*m2*	Indirect matrix reference

Returns

m1 x m2 (indirect matrix)

template<class dataType > **os::matrix**<dataType> operator* (const dataType & d1, const **os::matrix**< dataType > & m1)

Scalar multiplication.

Multiplies a matrix by a constant. This function depends on the '*' operator of the dataType.

Parameters

`in`	*d1*	Scalar data type
`in`	*m1*	Raw matrix reference

Returns

d1 ∗ m1 (raw matrix)

template<class dataType > **os::matrix**<dataType> operator∗ (const **os::matrix**< dataType > & m1, const dataType & d1)

Scalar multiplication.

Multiplies a matrix by a constant. This function depends on the '∗' operator of the dataType.

Parameters

in	*m1*	Raw matrix reference
in	*d1*	Scalar data type

Returns

d1 ∗ m1 (raw matrix)

template<class dataType > **os::indirectMatrix**<dataType> operator∗ (const dataType & d1, const **os::indirectMatrix**< dataType > & m1)

Scalar multiplication.

Multiplies an indirect matrix by a constant. This function depends on the '∗' operator of the data↩

Type.

Parameters

in	*d1*	Scalar data type
in	*m1*	Indirect matrix reference

Returns

d1 ∗ m1 (indirect matrix)

template<class dataType > **os::indirectMatrix**<dataType> operator∗ (const **os::indirectMatrix**< dataType > & m1, const dataType & d1)

Scalar multiplication.

Multiplies an indirect matrix by a constant. This function depends on the '$*$' operator of the data↩
Type.

Parameters

`in`	*m1*	Indirect matrix reference
`in`	*d1*	Scalar data type

Returns

d1 $*$ m1 (indirect matrix)

template<class dataType > **os::matrix**<dataType> operator+ (const **os::matrix**< dataType > & m1, const **os::matrix**< dataType > & m2)

Addition.

Preforms matrix addition. Matrix addition is undefined if the two matrices are of different size. If the operation is undefined, a matrix of size (0,0) will be returned. Depends on the '+' operator of dataType.

Parameters

`in`	*m1*	Raw matrix reference
`in`	*m2*	Raw matrix reference

Returns

m1 + m2 (raw matrix)

template<class dataType > **os::matrix**<dataType> operator+ (const **os::indirectMatrix**< dataType > & m1, const **os::matrix**< dataType > & m2)

Addition.

Preforms matrix addition. Matrix addition is undefined if the two matrices are of different size. If the operation is undefined, a matrix of size (0,0) will be returned. Depends on the '+' operator of dataType.

Parameters

`in`	*m1*	Indirect matrix reference
`in`	*m2*	Raw matrix reference

Returns

m1 + m2 (raw matrix)

template<class dataType > **os::matrix**<dataType> operator+ (const **os::matrix**< dataType > & m1, const **os::indirectMatrix**< dataType > & m2)

Addition.

Preforms matrix addition. Matrix addition is undefined if the two matrices are of different size. If the operation is undefined, a matrix of size (0,0) will be returned. Depends on the '+' operator of dataType.

Parameters

`in`	*m1*	Raw matrix reference
`in`	*m2*	Indirect matrix reference

Returns

m1 + m2 (raw matrix)

template<class dataType > **os::indirectMatrix**<dataType> operator+ (const **os::indirectMatrix**< dataType > & m1, const **os::indirectMatrix**< dataType > & m2)

Addition.

Preforms matrix addition. Matrix addition is undefined if the two matrices are of different size. If the operation is undefined, a matrix of size (0,0) will be returned. Depends on the '+' operator of dataType.

Parameters

`in`	*m1*	Indirect matrix reference

Parameters

`in`	*m2*	Indirect matrix reference

Returns

m1 + m2 (indirect matrix)

template<class dataType > **os::matrix**<dataType> operator- (const **os::matrix**< dataType > & m1, const **os::matrix**< dataType > & m2)

Subtraction.

Preforms matrix subtraction. Matrix subtraction is undefined if the two matrices are of different size. If the operation is undefined, a matrix of size (0,0) will be returned. Depends on the '-' operator of dataType.

Parameters

`in`	*m1*	Raw matrix reference
`in`	*m2*	Raw matrix reference

Returns

m1 - m2 (raw matrix)

template<class dataType > **os::matrix**<dataType> operator- (const **os::indirectMatrix**< dataType > & m1, const **os::matrix**< dataType > & m2)

Subtraction.

Preforms matrix subtraction. Matrix subtraction is undefined if the two matrices are of different size. If the operation is undefined, a matrix of size (0,0) will be returned. Depends on the '-' operator of dataType.

Parameters

`in`	*m1*	Indirect matrix reference
`in`	*m2*	Raw matrix reference

Returns

m1 - m2 (raw matrix)

template<class dataType > **os::matrix**<dataType> operator- (const **os::matrix**< dataType > & m1, const **os::indirectMatrix**< dataType > & m2)

Subtraction.

Preforms matrix subtraction. Matrix subtraction is undefined if the two matrices are of different size. If the operation is undefined, a matrix of size (0,0) will be returned. Depends on the '-' operator of dataType.

Parameters

`in`	*m1*	Raw matrix reference
`in`	*m2*	Indirect matrix reference

Returns

m1 - m2 (raw matrix)

template<class dataType > **os::indirectMatrix**<dataType> operator- (const **os::indirectMatrix**< dataType > & m1, const **os::indirectMatrix**< dataType > & m2)

Subtraction.

Preforms matrix subtraction. Matrix subtraction is undefined if the two matrices are of different size. If the operation is undefined, a matrix of size (0,0) will be returned. Depends on the '-' operator of dataType.

Parameters

`in`	*m1*	Indirect matrix reference
`in`	*m2*	Indirect matrix reference

Returns

m1 - m2 (indirect matrix)

template<class dataType > **os::matrix**<dataType> operator/ (const **os::matrix**< dataType > & m1, const dataType & d1)

Scalar division.

Divides a matrix by a constant. This function depends on the '/' operator of the dataType. No zero check, as the dataType is not defined.

Parameters

`in`	*m1*	Raw matrix reference
`in`	*d1*	Scalar data type

Returns

m1/d (raw matrix)

template<class dataType > **os::indirectMatrix**<dataType> operator/ (const **os::indirectMatrix**< dataType > & m1, const dataType & d1)

Scalar division.

Divides an indirect matrix by a constant. This function depends on the '/' operator of the dataType. No zero check, as the dataType is not defined.

Parameters

`in`	*m1*	Raw matrix reference
`in`	*d1*	Scalar data type

Returns

m1/d (raw matrix)

template<class dataType > std::ostream& operator<< (std::ostream & os, const **os::matrix**< dataType > & dt)

Prints out a matrix.

Prints out the entire matrix in the provided output stream. This matrix will be printed out in text form and requires the dataType of the matrix to define an ostream operator.

Parameters

	[in/out]	os std::ostream reference
`in`	*dt*	Raw matrix reference

Returns

std::ostream os

template<class dataType > std::ostream& operator<< (std::ostream & os, const **os::indirectMatrix**< dataType > & dt)

Prints out a matrix.

Prints out the entire matrix in the provided output stream. This matrix will be printed out in text form and requires the dataType of the matrix to define an ostream operator.

Parameters

	[in/out]	os std::ostream reference
`in`	*dt*	Indirect matrix reference

Returns

std::ostream os

template<class dataType > bool operator== (const **os::matrix**< dataType > & m1, const **os::matrix**< dataType > & m2)

Test for equality.

Tests the two matrices for equal size and then tests each matrix element for equality as well. This function is dependent on the '!=' definition of the dataType.

Parameters

in	*m1*	Raw matrix reference
in	*m2*	Raw matrix reference

Returns

True if exactly equivalent

template<class dataType > bool operator== (const **os::indirectMatrix**< dataType > & m1, const **os::matrix**< dataType > & m2)

Test for equality.

Tests the two matrices for equal size and then tests each matrix element for equality as well. This function is dependent on the '!=' definition of the dataType.

Parameters

in	*m1*	Indirect matrix reference
in	*m2*	Raw matrix reference

Returns

True if exactly equivalent

template<class dataType > bool operator== (const **os::matrix**< dataType > & m1, const **os::indirectMatrix**< dataType > & m2)

Test for equality.

Tests the two matrices for equal size and then tests each matrix element for equality as well. This function is dependent on the '!=' definition of the dataType.

Parameters

`in`	*m1*	Raw matrix reference
`in`	*m2*	Indirect matrix reference

Returns

True if exactly equivalent

template<class dataType > bool operator== (const **os::indirectMatrix**< dataType > & m1, const **os::indirectMatrix**< dataType > & m2)

Test for equality.

Tests the two matrices for equal size and then tests each matrix element for equality as well. This function is dependent on the '!=' definition of the dataType.

Parameters

`in`	*m1*	Indirect matrix reference
`in`	*m2*	Indirect matrix reference

Returns

True if exactly equivalent

E.10 osLogger.h File Reference

Logging for os namespace.

Namespaces

- **os**

Functions

- std::ostream & **os::osout_func** ()
  Standard out object for os namespace.
- std::ostream & **os::oserr_func** ()
  Standard error object for os namespace.

Variables

- smart_ptr< std::ostream > **os::osout_ptr**
  Standard out pointer for os namespace.
- smart_ptr< std::ostream > **os::oserr_ptr**
  Standard error pointer for os namespace.

E.10.1 Detailed Description

Logging for os namespace.

Jonathan Bedard

Date

1/30/2016

Bug No known bugs.

This file contains declarations which are used for logging within the os namespace.

E.11 osLogger.cpp File Reference

Logging for os namespace, implementation.

E.11.1 Detailed Description

Logging for os namespace, implementation.

Jonathan Bedard

Date

2/15/2016

Bug No known bugs.

This file contains global functions and variables used for logging in the os namespace.

E.12 osVectors.h File Reference

Vector templates.

Classes

- class **os::vector2d**< **dataType** >
  2-dimensional vector
- class **os::vector3d**< **dataType** >
  3-dimensional vector

Namespaces

- **os**

Typedefs

- typedef vector2d< int8_t > **os::vector2d_8**
  8 bit 2-d vector
- typedef vector2d< uint8_t > **os::vector2d_u8**
  unsigned 8 bit 2-d vector
- typedef vector2d< int16_t > **os::vector2d_16**

16 bit 2-d vector

- typedef vector2d< uint16_t > **os::vector2d_u16**
  unsigned 16 bit 2-d vector
- typedef vector2d< int32_t > **os::vector2d_32**
  32 bit 2-d vector
- typedef vector2d< uint32_t > **os::vector2d_u32**
  unsigned 32 bit 2-d vector
- typedef vector2d< int64_t > **os::vector2d_64**
  64 bit 2-d vector
- typedef vector2d< uint64_t > **os::vector2d_u64**
  unsigned 64 bit 2-d vector
- typedef vector2d< float > **os::vector2d_f**
  float 2-d vector
- typedef vector2d< double > **os::vector2d_d**
  double 2-d vector
- typedef vector3d< int8_t > **os::vector3d_8**
  8 bit 3-d vector
- typedef vector3d< uint8_t > **os::vector3d_u8**
  unsigned 8 bit 3-d vector
- typedef vector3d< int16_t > **os::vector3d_16**
  16 bit 3-d vector
- typedef vector3d< uint16_t > **os::vector3d_u16**
  unsigned 16 bit 3-d vector
- typedef vector3d< int32_t > **os::vector3d_32**
  32 bit 3-d vector
- typedef vector3d< uint32_t > **os::vector3d_u32**
  unsigned 32 bit 3-d vector
- typedef vector3d< int64_t > **os::vector3d_64**
  64 bit 3-d vector
- typedef vector3d< uint64_t > **os::vector3d_u64**
  unsigned 64 bit 3-d vector
- typedef vector3d< float > **os::vector3d_f**
  float 3-d vector
- typedef vector3d< double > **os::vector3d_d**
  double 3-d vector

E.12.1 Detailed Description

Vector templates.

Author

Jonathan Bedard

Date

3/12/2016

Bug No known bugs.

This file contains two template classes defining vector objects. Vectors can, in a broad sense, be used for any class which defines general mathematical operations. This particular file offers vector type definitions for all of the basic integer and floating point types.

E.13 set.h File Reference

Smart Set.

Classes

- class **os::smartSet**< **dataType** >
  Smart set abstract data-structures.

Namespaces

- **os**

Enumerations

- enum **os::setTypes** { **os::def_set** =0, **os::small_set**, **os::sorted_set** }
  Index of abstract data-structures.

E.13.1 Detailed Description

Smart Set.

Author

Jonathan Bedard

Date

2/12/2016

Bug No known bugs.

This file contains a template class defining a "smart set." A smart set wraps other forms of abstract data structures, allowing applications to define abstract data-structures by numbered indexes.

E.14 smartPointer.h File Reference

Template declaration of **os::smart_ptr** (p. **??**).

Classes

- class **os::smart_ptr**< **dataType** >
  Reference counted pointer.

Namespaces

- **os**

Typedefs

- typedef void(∗ **os::void_rec**) (void ∗)
  Deletion function typedef.

Enumerations

- enum **os::smart_pointer_type** {
  os::null_type =0, **os::raw_type**, **os::shared_type**, **os::shared_type_array**,
  os::shared_type_dynamic_delete }
  *Enumeration for types of **os::smart_ptr** (p. **??**).*

Functions

- template<class targ , class src >

  smart_ptr< targ > **os::cast** (const **os::smart_ptr**< src > &conv)

  *__os::smart_ptr__ (p. **??**) cast function*

- template<class dataType >

  bool **operator==** (const **os::smart_ptr**< dataType > &c1, const **os::smart_ptr**< dataType > &c2)

- template<class dataType >

  bool **operator==** (const **os::smart_ptr**< dataType > &c1, const dataType ∗c2)

- template<class dataType >

  bool **operator==** (const dataType ∗c1, const **os::smart_ptr**< dataType > &c2)

- template<class dataType >

  bool **operator==** (const **os::smart_ptr**< dataType > &c1, const void ∗c2)

- template<class dataType >

  bool **operator==** (const void ∗c1, const **os::smart_ptr**< dataType > &c2)

- template<class dataType >

  bool **operator==** (const **os::smart_ptr**< dataType > &c1, const int c2)

- template<class dataType >

  bool **operator==** (const int c1, const **os::smart_ptr**< dataType > &c2)

- template<class dataType >

  bool **operator==** (const **os::smart_ptr**< dataType > &c1, const long c2)

- template<class dataType >

  bool **operator==** (const long c1, const **os::smart_ptr**< dataType > &c2)

- template<class dataType >

  bool **operator==** (const **os::smart_ptr**< dataType > &c1, const unsigned long c2)

- template<class dataType >

  bool **operator==** (const unsigned long c1, const **os::smart_ptr**< dataType > &c2)

- template<class dataType >

  bool **operator!=** (const **os::smart_ptr**< dataType > &c1, const **os::smart_ptr**< dataType >

&c2)

- template<class dataType >

  bool **operator!=** (const **os::smart_ptr**< dataType > &c1, const dataType ∗c2)
- template<class dataType >

  bool **operator!=** (const dataType ∗c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >

  bool **operator!=** (const **os::smart_ptr**< dataType > &c1, const void ∗c2)
- template<class dataType >

  bool **operator!=** (const void ∗c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >

  bool **operator!=** (const **os::smart_ptr**< dataType > &c1, const int c2)
- template<class dataType >

  bool **operator!=** (const int c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >

  bool **operator!=** (const **os::smart_ptr**< dataType > &c1, const long c2)
- template<class dataType >

  bool **operator!=** (const long c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >

  bool **operator!=** (const **os::smart_ptr**< dataType > &c1, const unsigned long c2)
- template<class dataType >

  bool **operator!=** (const unsigned long c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >

  bool **operator**< (const **os::smart_ptr**< dataType > &c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >

  bool **operator**< (const **os::smart_ptr**< dataType > &c1, const dataType ∗c2)
- template<class dataType >

  bool **operator**< (const dataType ∗c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >

  bool **operator**< (const **os::smart_ptr**< dataType > &c1, const void ∗c2)

- template<class dataType >

  bool **operator<** (const void ∗c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >

  bool **operator<** (const **os::smart_ptr**< dataType > &c1, const int c2)
- template<class dataType >

  bool **operator<** (const int c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >

  bool **operator<** (const **os::smart_ptr**< dataType > &c1, const long c2)
- template<class dataType >

  bool **operator<** (const long c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >

  bool **operator<** (const **os::smart_ptr**< dataType > &c1, const unsigned long c2)
- template<class dataType >

  bool **operator<** (const unsigned long c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >

  bool **operator<=** (const **os::smart_ptr**< dataType > &c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >

  bool **operator<=** (const **os::smart_ptr**< dataType > &c1, const dataType ∗c2)
- template<class dataType >

  bool **operator<=** (const dataType ∗c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >

  bool **operator<=** (const **os::smart_ptr**< dataType > &c1, const void ∗c2)
- template<class dataType >

  bool **operator<=** (const void ∗c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >

  bool **operator<=** (const **os::smart_ptr**< dataType > &c1, const int c2)
- template<class dataType >

  bool **operator<=** (const int c1, const **os::smart_ptr**< dataType > &c2)

- template<class dataType >

  bool **operator<=** (const **os::smart_ptr**< dataType > &c1, const long c2)
- template<class dataType >

  bool **operator<=** (const long c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >

  bool **operator<=** (const **os::smart_ptr**< dataType > &c1, const unsigned long c2)
- template<class dataType >

  bool **operator<=** (const unsigned long c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >

  bool **operator>** (const **os::smart_ptr**< dataType > &c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >

  bool **operator>** (const **os::smart_ptr**< dataType > &c1, const dataType ∗&c2)
- template<class dataType >

  bool **operator>** (const dataType ∗&c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >

  bool **operator>** (const **os::smart_ptr**< dataType > &c1, const void ∗c2)
- template<class dataType >

  bool **operator>** (const void ∗c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >

  bool **operator>** (const **os::smart_ptr**< dataType > &c1, const int c2)
- template<class dataType >

  bool **operator>** (const int c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >

  bool **operator>** (const **os::smart_ptr**< dataType > &c1, const long c2)
- template<class dataType >

  bool **operator>** (const long c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >

  bool **operator>** (const **os::smart_ptr**< dataType > &c1, const unsigned long c2)

- template<class dataType >

  bool **operator>** (const unsigned long c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >

  bool **operator>=** (const **os::smart_ptr**< dataType > &c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >

  bool **operator>=** (const **os::smart_ptr**< dataType > &c1, const dataType ∗&c2)
- template<class dataType >

  bool **operator>=** (const dataType ∗&c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >

  bool **operator>=** (const **os::smart_ptr**< dataType > &c1, const void ∗c2)
- template<class dataType >

  bool **operator>=** (const void ∗c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >

  bool **operator>=** (const **os::smart_ptr**< dataType > &c1, const int c2)
- template<class dataType >

  bool **operator>=** (const int c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >

  bool **operator>=** (const **os::smart_ptr**< dataType > &c1, const long c2)
- template<class dataType >

  bool **operator>=** (const long c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >

  bool **operator>=** (const **os::smart_ptr**< dataType > &c1, const unsigned long c2)
- template<class dataType >

  bool **operator>=** (const unsigned long c1, const **os::smart_ptr**< dataType > &c2)

E.14.1 Detailed Description

Template declaration of **os::smart_ptr** (p. **??**).

Author

Jonathan Bedard

Date

4/18/2016

Bug No known bugs.

This file contains a template declaration of **os::smart_ptr** (p. **??**) and supporting constants and functions. Note that because **os::smart_ptr** (p. **??**) is a template class, the implimentation of **os↩ ::smart_ptr** (p. **??**) occurs here as well.

E.14.2 Function Documentation

template<class dataType > bool operator!= (const **os::smart_ptr**< dataType > & c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator!= (const **os::smart_ptr**< dataType > & c1, const dataType ∗ c2) [inline]

template<class dataType > bool operator!= (const dataType ∗ c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator!= (const **os::smart_ptr**< dataType > & c1, const void ∗ c2) [inline]

template<class dataType > bool operator!= (const void ∗ c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator!= (const **os::smart_ptr**< dataType > & c1, const int c2) [inline]

template<class dataType > bool operator!= (const int c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator!= (const **os::smart_ptr**< dataType > & c1, const long c2) [inline]

template<class dataType > bool operator!= (const long c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator!= (const **os::smart_ptr**< dataType > & c1, const unsigned long c2) [inline]

template<class dataType > bool operator!= (const unsigned long c1, const **os::smart_ptr**< dataType > & c2) [inline] template<class dataType > bool operator< (const **os::smart_ptr**< dataType > & c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator< (const **os::smart_ptr**< dataType > & c1, const dataType ∗ c2) [inline]

template<class dataType > bool operator< (const dataType ∗ c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator< (const **os::smart_ptr**< dataType > & c1, const void ∗ c2) [inline]

template<class dataType > bool operator< (const void ∗ c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator< (const **os::smart_ptr**< dataType > & c1, const int c2) [inline]

template<class dataType > bool operator< (const int c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator< (const **os::smart_ptr**< dataType > & c1, const long c2) [inline]

template<class dataType > bool operator< (const long c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator< (const **os::smart_ptr**< dataType > & c1, const unsigned long c2) [inline]

template<class dataType > bool operator< (const unsigned long c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator<= (const **os::smart_ptr**< dataType > & c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator<= (const **os::smart_ptr**< dataType > & c1, const dataType ∗ c2) [inline]

template<class dataType > bool operator<= (const dataType ∗ c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator<= (const **os::smart_ptr**< dataType > & c1, const void ∗ c2) [inline]

template<class dataType > bool operator<= (const void ∗ c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator<= (const **os::smart_ptr**< dataType > & c1, const int c2) [inline]

template<class dataType > bool operator<= (const int c1, const **os::smart_ptr**< dataType > & c2) [inline] template<class dataType > bool operator<= (const **os::smart_ptr**< dataType > & c1, const long c2) `[inline]`

template<class dataType > bool operator<= (const long c1, const **os::smart_ptr**< dataType > & c2) `[inline]`

template<class dataType > bool operator<= (const **os::smart_ptr**< dataType > & c1, const unsigned long c2) `[inline]`

template<class dataType > bool operator<= (const unsigned long c1, const **os::smart_ptr**< dataType > & c2) `[inline]`

template<class dataType > bool operator== (const **os::smart_ptr**< dataType > & c1, const **os::smart_ptr**< dataType > & c2) `[inline]`

template<class dataType > bool operator== (const **os::smart_ptr**< dataType > & c1, const dataType ∗ c2) `[inline]`

template<class dataType > bool operator== (const dataType ∗ c1, const **os::smart_ptr**< dataType > & c2) `[inline]`

template<class dataType > bool operator== (const **os::smart_ptr**< dataType > & c1, const void ∗ c2) `[inline]`

template<class dataType > bool operator== (const void ∗ c1, const **os::smart_ptr**< dataType > & c2) `[inline]`

template<class dataType > bool operator== (const **os::smart_ptr**< dataType > & c1, const int c2) `[inline]`

template<class dataType > bool operator== (const int c1, const **os::smart_ptr**< dataType > & c2) `[inline]`

template<class dataType > bool operator== (const **os::smart_ptr**< dataType > & c1, const long c2) `[inline]`

template<class dataType > bool operator== (const long c1, const **os::smart_ptr**< dataType > & c2) `[inline]`

template<class dataType > bool operator== (const **os::smart_ptr**< dataType > & c1, const unsigned long c2) `[inline]`

template<class dataType > bool operator== (const unsigned long c1, const **os::smart_ptr**< dataType > & c2) `[inline]`

template<class dataType > bool operator> (const **os::smart_ptr**< dataType > & c1, const **os::smart_ptr**< dataType > & c2) `[inline]`

template<class dataType > bool operator> (const **os::smart_ptr**< dataType > & c1, const dataType ∗& c2) `[inline]`

template<class dataType > bool operator> (const dataType ∗& c1, const **os::smart_ptr**< dataType > & c2) `[inline]` template<class dataType > bool operator> (const **os::smart_ptr**< dataType > & c1, const void ∗ c2) [inline]

template<class dataType > bool operator> (const void ∗ c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator> (const **os::smart_ptr**< dataType > & c1, const int c2) [inline]

template<class dataType > bool operator> (const int c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator> (const **os::smart_ptr**< dataType > & c1, const long c2) [inline]

template<class dataType > bool operator> (const long c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator> (const **os::smart_ptr**< dataType > & c1, const unsigned long c2) [inline]

template<class dataType > bool operator> (const unsigned long c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator>= (const **os::smart_ptr**< dataType > & c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator>= (const **os::smart_ptr**< dataType > & c1, const dataType ∗& c2) [inline]

template<class dataType > bool operator>= (const dataType ∗& c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator>= (const **os::smart_ptr**< dataType > & c1, const void ∗ c2) [inline]

template<class dataType > bool operator>= (const void ∗ c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator>= (const **os::smart_ptr**< dataType > & c1, const int c2) [inline]

template<class dataType > bool operator>= (const int c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator>= (const **os::smart_ptr**< dataType > & c1, const long c2) [inline]

template<class dataType > bool operator>= (const long c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator>= (const **os::smart_ptr**< dataType > & c1, const unsigned long c2) [inline] template<class dataType > bool operator>= (const unsigned long c1, const **os::smart_ptr**< dataType > & c2) `[inline]`

E.15 staticConstantPrinter.h File Reference

Constant printing support.

Classes

- class **os::constantPrinter**
  Prints constant arrays to files.

Namespaces

- **os**

E.15.1 Detailed Description

Constant printing support.

Author

Jonathan Bedard

Date

1/31/2016

Bug No known bugs.

This file contains a class which helps facilitate printing massive tables of constants. It outputs .h and .cpp files with configured arrays of constants.

E.16 staticConstantPrinter.cpp File Reference

Constant printing support, implementation.

E.16.1 Detailed Description

Constant printing support, implementation.

Author

Jonathan Bedard

Date

4/618/2016

Bug No known bugs.

This file implements **os::constantPrinter** (p. **??**). Consult **staticConstantPrinter.h** (p. 134) for detailed documentation.

Part II

Unit Test Library

Appendix F

Introduction

The UnitTest library contains classes which preform automated unit tests while a project is under development. Utilizing C++ exceptions, the UnitTest library separates its test battery into libraries tested, suites in libraries and tests in suites. The UnitTest library iterates through instantiated libraries running every test suite in the library.

F.1 Namespace test

The test namespace is designed to hold all of the classes and functions related to unit testing. Classes and functions in the test namespace should not be included in the final release application. It is expected that libraries add to this namespace and place their own testing assets here. Note that the test namespace uses elements from the os namespace, all of these elements are defined in the Datastructures library.

F.2 Datastructures Testing

The Datastructures library is rigorously unit tested by the UnitTest library, and the Datastructures unit tests are automatically included in any system unit test unless specifically removed. The Datastructures UnitTests are particularly important because the Datastructures library serves as a base for memory management and data organization. These tests fall broadly into two categories: deterministic and random.

Deterministic tests preform the exact same test every iteration. Deterministic tests are used to ensure that specific functions and operators are returning expected data. Deterministic tests don't merely identify the existence of an error, but usually identify the precise nature of the error as well.

Random tests use a random number generator to preform a unique test with every iteration. This allows unit tests to, over time, catch edge cases with complex data structures. In contrast to deterministic tests, random testing will usually not identify the precise nature of the error.

Note that as a general rule, the implementation of tests is not documented. The location of test suites is documented, through both .h and .cpp files, but the classes and functions which make up these tests are not included.

Appendix G

File Index

G.1 File List

Here is a list of all files with brief descriptions:

Appendix H

File Documentation

H.1 DatastructuresTest.h File Reference

Datastructures library test.

H.1.1 Detailed Description

Datastructures library test.

Author

Jonathan Bedard

Date

2/4/2016

Bug No known bugs.

Contains the declaration of the Datastructures library test. Note that this library test is automatically added to all Unit Test executables.

H.2 DatastructuresTest.cpp File Reference

Datastructures library test implementation.

H.2.1 Detailed Description

Datastructures library test implementation.

Author

Jonathan Bedard

Date

4/18/2016

Bug No known bugs.

Implements the Datastructures library test. These tests are designed to guarantee the functionality of each of the elements in the Datastructures library.

H.3 masterTestHolder.h File Reference

Library tests, masterTestHolder singleton.

Classes

- class **test::libraryTests**
  Library test group.
- class **test::masterTestHolder**
  Unit Test singleton.

Namespaces

- **test**

H.3.1 Detailed Description

Library tests, masterTestHolder singleton.

Jonathan Bedard

Date

4/11/2016

Bug No known bugs.

This file contains declarations for the library test base class and **test::masterTestHolder** (p. **??**) singleton class. This file represents the top level of the Unit Test driver classes.

H.4 masterTestHolder.cpp File Reference

Library tests, masterTestHolder singleton implementations.

H.4.1 Detailed Description

Library tests, masterTestHolder singleton implementations.

Jonathan Bedard

Date

4/11/2016

Bug No known bugs.

This file contains implementations for the library test base class and **test::masterTestHolder** (p. **??**) singleton class. Consult **masterTestHolder.h** (p. 142) for details.

H.5 singleTest.h File Reference

Single test class.

Classes

- class **test::singleTest**
  Single unit test class.
- class **test::singleFunctionTest**
  Single unit test from function.

Namespaces

- **test**

Typedefs

- typedef void($*$ **test::testFunction**) ()
  Typedef for single test function.

H.5.1 Detailed Description

Single test class.

Jonathan Bedard

Date

2/6/2016

Bug No known bugs.

This file contains declarations for a single unit test. Unit tests can be defined as separate class or a simple test function.

H.6 singleTest.cpp File Reference

Single test class implementation.

H.6.1 Detailed Description

Single test class implementation.

Jonathan Bedard

Date

2/6/2016

Bug No known bugs.

This file contains implementation for a single unit test. Consult singeTest.h for details.

H.7 TestSuite.h File Reference

Single test class.

Classes

- class **test::testSuite**

Namespaces

- **test**

H.7.1 Detailed Description

Single test class.

Jonathan Bedard

Date

4/11/2016

Bug No known bugs.

This file contains declarations for a test suite. Test suites contain lists of unit tests.

H.8 TestSuite.cpp File Reference

Single test class.

H.8.1 Detailed Description

Single test class.

Jonathan Bedard

Date

2/12/2016

Bug No known bugs.

This file contains declarations for a test suite. Consult **testSuite.h** (p. 144) for details.

H.9 UnitTest.h File Reference

Unit Test header file.

Namespaces

- **test**

Functions

- void **test::startTests** ()

  Print out header for Unit Tests.
- void **test::endTestsError** (os::smart_ptr< std::exception > except)

  End tests in error.
- void **test::endTestsSuccess** ()

  End tests successfully.
- void **test::testInit** (int argc=0, char ∗∗argv=NULL)

  Test initialization.

H.9.1 Detailed Description

Unit Test header file.

Author

Jonathan Bedard

Date

4/2/2016

Bug No known bugs.

Packages all headers required for the UnitTest library and declares a number of global test functions used for initializing and ending a Unit Test battery.

H.10 UnitTest.cpp File Reference

Unit Test logging and global functions.

H.10.1 Detailed Description

Unit Test logging and global functions.

Author

Jonathan Bedard

Date

2/4/2016

Bug No known bugs.

Implements logging in the test namespace. Implements a number of global test functions used for initializing and ending a Unit Test battery.

H.11 UnitTestLog.h File Reference

Namespaces

- **test**

Functions

- std::ostream & **test::testout_func** ()
  Standard out object for test namespace.
- std::ostream & **test::testerr_func** ()
  Standard error object for test namespace.

Variables

- os::smart_ptr< std::ostream > **test::testout_ptr**
  Standard out pointer for test namespace.
- os::smart_ptr< std::ostream > **test::testerr_ptr**
  Standard error pointer for test namespace.

H.12 UnitTestExceptions.h File Reference

Common exceptions thrown by unit tests.

Classes

- class **test::generalTestException**
  Base class for test exceptions.
- class **test::unknownException**

Unknown exception class.

- class **test::nullFunctionException**

  NULL function exception class.

Namespaces

- **test**

H.12.1 Detailed Description

Common exceptions thrown by unit tests.

Jonathan Bedard

Date

2/19/2016

Bug No known bugs.

This file contains a number of common test exceptions used by unit tests. All of these classes extend std::exception.

Part III

osMechanics Library

Appendix I

Introduction

The osMechanics library contains classes which are general tools for navigating file systems, thread management and logging. Some classes, particularly those dealing with threading, sockets and file access, differ from operating system to operating system. CMake should handle all operating system variances.

I.1 Namespace

osMechanics extends the os namespace. The os namespace is designed for tools, algorithms and data-structures used in programs of all types. Note that the Datastructures library also uses the os namespace.

Appendix J

File Index

J.1 File List

Here is a list of all files with brief descriptions:

Appendix K

File Documentation

K.1 logger.cpp File Reference

logger implementation file

Functions

- static void **loggerSavingThread** (void ∗ptr, smart_ptr< **threadHolder** > th)

Variables

- smart_ptr< **Log** > **_single_log**
- static bool **singleton_bool** = false

K.1.1 Detailed Description

logger implementation file

Jonathan Bedard

Date

4/23/2015

Bug No known bugs.

The implementation of our logging systems are in this file. The logger records various timing, operation, and debug information and places it in various files so that we can better analyze our own system's performance.

K.1.2 Function Documentation

static void loggerSavingThread (void ∗ ptr, smart_ptr< **threadHolder** > th) `[static]`

K.1.3 Variable Documentation

smart_ptr<**Log**> _single_log

bool singleton_bool = false `[static]`

K.2 logger.h File Reference

logger header file

Classes

- class **os::logStatusHolder**
- class **os::logStatusListener**
- struct **os::logLine**
- class **os::LogStreamListener**
- class **os::LineLogger**
- class **os::LogSaver**
- class **os::LineSaver**
- class **os::LineSaverListener**
- class **os::Log**
- class **os::LogDirectedStream**

Namespaces

- **os**

Variables

- **logStatusHolder os::logStatus**
- **Log** & **os::logger** =∗Log::singleton()

K.2.1 Detailed Description

logger header file

Jonathan Bedard

Date

4/23/2016

Bug No known bugs.

All of the headers in the Datastructures library are held in this file. When using the Datastructures library, it is expected that this header is included instead of the individual required headers.

K.3 multiLock.cpp File Reference

multiLock implementation file

K.3.1 Detailed Description

multiLock implementation file

Jonathan Bedard

Date

9/29/2015

Bug No known bugs.

This is the implementation of our multiLock. It is platform agnostic.

K.4 multiLock.h File Reference

multiLock header file

Classes

- class **os::multiLock**
  os::multilock class definition Defines the os::multilock class. This class has 4 variables and 8 methods

Namespaces

- **os**

K.4.1 Detailed Description

multiLock header file

Jonathan Bedard

Date

1/30/2016

Bug No known bugs.

This is the multilock header we are using. It has reading and writing locks, allowing multiple users to read, but only one to write at any given time.

K.5 osFunctions.cpp File Reference

osFunctions implementation file

K.5.1 Detailed Description

osFunctions implementation file

Jonathan Bedard

Date

5/20/2016

Bug No known bugs.

This is the implementation of the osFunctions that do not care about operating system. This is mostly converting bit structures between different hardware platforms.

K.6 osFunctions.h File Reference

osFunctions header file

Namespaces

- **os**

Functions

- uint16_t **os::to_comp_mode** (uint16_t i)

  Changes bit order for compatibility Depending on the system at hand, bits may be in several different orders. This function swaps to compatibility mode.

- uint16_t **os::from_comp_mode** (uint16_t i)

  Changes bit order for compatibility Depending on the system at hand, bits may be in several different orders. This function swaps from compatibility mode.

- uint32_t **os::to_comp_mode** (uint32_t i)

  Changes bit order for compatibility Depending on the system at hand, bits may be in several different orders. This function swaps to compatibility mode.

- uint32_t **os::from_comp_mode** (uint32_t i)

  Changes bit order for compatibility Depending on the system at hand, bits may be in several different orders. This function swaps from compatibility mode.

- uint64_t **os::to_comp_mode** (uint64_t i)

  Changes bit order for compatibility Depending on the system at hand, bits may be in several different orders. This function swaps to compatibility mode.

- uint64_t **os::from_comp_mode** (uint64_t i)

  Changes bit order for compatibility Depending on the system at hand, bits may be in several different orders. This function swaps from compatibility mode.

- uint64_t **os::getTimestamp** ()

  Gets a timestamp Generates a time stamp from the time function.

- bool **os::testCreateFolder** (std::string n)

  Test if a folder exists Checks if a given folder exists. If it does not exist, this function will create said folder.

- std::string **os::convertTimestamp** (uint64_t stamp)

  Type conversion on timestamp Converts the timestamp from an integer into a string.

K.6.1 Detailed Description

osFunctions header file

Jonathan Bedard

Date

5/20/2016

Bug No known bugs.

This is the definitions for some of our compatibility functions.

K.7 osMechanics.h File Reference

osMechanics header file

K.7.1 Detailed Description

osMechanics header file

Jonathan Bedard

Date

2/24/2015

Bug No known bugs.

This file includes all of our headers, so that other libraries can easily include the osMechanics library with one include.

K.8 osMechanicsTest.cpp File Reference

Test implimentaiton for osMechanics.

K.8.1 Detailed Description

Test implimentaiton for osMechanics.

Author

Adrian Bedard

Date

4/12/2016

Bug No known bugs.

Binds all osMechanics test suites. These suites test the basic funcitonality of the osMechanics library. Projects which utilize osMechanics are suggested to bind the osMechanics library tests to their own test suite.

K.9 osMechanicsTest.h File Reference

osMechanics tests

K.9.1 Detailed Description

osMechanics tests

Jonathan Bedard

Date

4/11/2016

Bug No known bugs.

This is the test suite for the osMechanics library.

K.10 osThreads.cpp File Reference

threads implementation file

Functions

- void **temp_thread_call** (void $*$ptr_array, bool typ, std::string thread_info)
- void **wait_for_threads** ()

Variables

- static **spinLock globalThreadLock**
- static **threadTracker** $*$ **static_ref** = NULL

K.10.1 Detailed Description

threads implementation file

Jonathan Bedard

Date

4/18/2016

Bug No known bugs.

This is the implementation of our multi threading system.

K.10.2 Function Documentation

void temp_thread_call (void ∗ ptr_array, bool typ, std::string thread_info)

void wait_for_threads ( )

K.10.3 Variable Documentation

spinLock globalThreadLock [static]

threadTracker∗ static_ref = NULL [static]

K.11 osThreads.h File Reference

osThreads header file

Classes

- class **os::threadHolder**
- class **os::threadTracker**

  Monitors a range of threads This class holds a range of threadHolders. This includes both active and expired threads, ensuring the ability to operate on many threads in mass.

Namespaces

- **os**

Functions

- smart_ptr< std::thread > **os::spawnThread** (void(∗func)(void ∗), void ∗ptr, std::string thread↩_info="")

- smart_ptr< std::thread > **os::spawnThread** (void(∗func)(void ∗, smart_ptr< **threadHolder** >), void ∗ptr, std::string thread_info="")

K.11.1 Detailed Description

osThreads header file

Jonathan Bedard

Date

4/13/2016

Bug No known bugs.

This is the osThreads header we are using. This header allows us to use multithreading with our own types, pointers, and management

K.12 safeQueue.h File Reference

safe queue header file

Classes

- class **os::safeQueue**< **dataType** >
  This is the ***safeQueue*** *(p.* **??***) class The* ***safeQueue*** *(p.* **??***) class is thread safe. It is a template class.*

Namespaces

- **os**

K.12.1 Detailed Description

safe queue header file

Jonathan Bedard

Date

11/9/2015

Bug No known bugs.

This is a thread safe queue, so we can multi thread safely.

K.13 savableClass.cpp File Reference

Implementation of the generalized savable class.

K.13.1 Detailed Description

Implementation of the generalized savable class.

Author

Jonathan Bedard

Date

4/12/2016

Bug None

Provides an implementation of the savable class, used to tie together multiple classes which need to be saved as a group.

K.14 savableClass.h File Reference

Defines a set of classes facilitating saving.

Classes

- class **os::savable**
  Basic saving class.
- class **os::savingGroup**
  Group of saving classes.

Namespaces

- **os**

K.14.1 Detailed Description

Defines a set of classes facilitating saving.

Author

Jonathan Bedard

Date

4/12/2016

Bug None

Provides a definition of user which has a user-name, password and associated bank of public keys.

K.15 Serial.h File Reference

determines which serial methods are needed.

K.15.1 Detailed Description

determines which serial methods are needed.

Jonathan Bedard

Date

5/20/2016

Bug No known bugs.

This header determines if we are on a windows or unix system, then includes a different header for each.

K.16 serialThread.cpp File Reference

serialThread implementation file

Functions

- static void **serialSearch** (void ∗ptr, smart_ptr< **threadHolder** > th)

K.16.1 Detailed Description

serialThread implementation file

Jonathan Bedard

Date

11/1/2015

Bug No known bugs.

These implementations allow us to create threads for monitoring serial communication

K.16.2 Function Documentation

static void serialSearch (void ∗ ptr, smart_ptr< **threadHolder** > th) `[static]`

K.17 serialThread.h File Reference

serial thread header file

Classes

- class **os::serialThread**

  Serial (p. **??**) *communication thread The is a serial class that runs as a thread. Thanks to this fact, we can run multiple serial communication threads as well as run a primary set of threads at once.*

Namespaces

- **os**

K.17.1 Detailed Description

serial thread header file

Jonathan Bedard

Date

11/9/2015

Bug No known bugs.

This is a serial thread class. This class allows us to monitor multiple ports effectively simultaneously.

K.18 socketFrame.cpp File Reference

socketFrame implementation file

Functions

- void **close_all_sockets** ()

Variables

- static smart_ptr< **socketTracker** > **st_instance** = NULL

K.18.1 Detailed Description

socketFrame implementation file

Jonathan Bedard

Date

2/12/2016

Bug No known bugs.

This is the implementation of our socket user, UDP socket, and socket tracker. Socket communication is important for us, and this allows us to safely have reliable sockets.

K.18.2 Function Documentation

void close_all_sockets ( )

K.18.3 Variable Documentation

smart_ptr<**socketTracker**> st_instance = NULL `[static]`

K.19 socketFrame.h File Reference

socket frame header file

Classes

- class **os::socketUser**

  Socket user class This class allows us to manage sockets.
- class **os::UDPSocket**

  ***UDPSocket** (p. **??**) class A class for UDPSockets, which in turn allows us to multi thread the packet send/receive functionality.*
- class **os::socketTracker**

  ***socketTracker** (p. **??**) class Tracks all currently active sockets.*

Namespaces

- **os**

K.19.1 Detailed Description

socket frame header file

Jonathan Bedard

Date

4/12/2016

Bug No known bugs.

Generalized socket class.

K.20 spinLock.cpp File Reference

spinLock file

K.20.1 Detailed Description

spinLock file

Jonathan Bedard

Date

5/20/2016

Bug No known bugs.

This file includes different implementations of our spin lock depending on the operating system.

K.21 spinLock.h File Reference

spinLock file

K.21.1 Detailed Description

spinLock file

Jonathan Bedard

Date

5/20/2016

Bug No known bugs.

This header includes different versions of the spin lock header depending on the operating system.

K.22 threadDistribution.cpp File Reference

thread distribution implementation file

Functions

- static void **executor_thread_starter** (void ∗ptr, smart_ptr< **threadHolder** > th)

K.22.1 Detailed Description

thread distribution implementation file

Jonathan Bedard

Date

4/18/2015

Bug No known bugs.

These methods determine which thread will operate next.

K.22.2 Function Documentation

static void executor_thread_starter (void ∗ ptr, smart_ptr< **threadHolder** > th) `[static]`

K.23 threadDistribution.h File Reference

thread distribution header file

Classes

- class **os::threadActor**

  threadActor (p. **??**) *class This class holds information for determining which thread goes at a give time.*

- class **os::threadDistributor**

  Distributes threads This class allows us to determine which thread should execute at any given time.

- class **os::executorThread**

  executorThread (p. **??**) *class This class holds a thread which has multiple steps.*

- class **os::singleAction**

  single action class This class is for a thread with only one action.

Namespaces

- **os**

Functions

- float **os::getSysTime** ()

  gets time Gets the current system time.

K.23.1 Detailed Description

thread distribution header file

Jonathan Bedard

Date

4/18/2015

Bug No known bugs.

This the thread distribution system.

K.24 unix_osFunctions.cpp File Reference

os functions implementation file

Functions

- static void **receiveThreadServerIPV4** (void ∗ptr, smart_ptr< **threadHolder** > th)
- static void **receiveThreadServerIPV6** (void ∗ptr, smart_ptr< **threadHolder** > th)

Variables

- static os::smart_ptr< **threadDistributor** > **ipthread** = NULL
- static std::string **local_path** = ""

K.24.1 Detailed Description

os functions implementation file

Jonathan Bedard

Date

5/20/2016

Bug No known bugs.

This is the implementation of the UNIX specific functions.

K.24.2 Function Documentation

static void receiveThreadServerIPV4 (void ∗ ptr, smart_ptr< **threadHolder** > th) `[static]`

static void receiveThreadServerIPV6 (void ∗ ptr, smart_ptr< **threadHolder** > th) `[static]`

K.24.3 Variable Documentation

os::smart_ptr<**threadDistributor**> ipthread = NULL `[static]`

std::string local_path = "" `[static]`

K.25 unix_osFunctions.h File Reference

os functions header file

Classes

- class **os::IPAddress**

  os::IPAddress (p. **??**) *class definition This is an IP Address class It has 2 variables and 10 methods*

- class **os::myIPAddress**

  Holds a node's own IP address Every node needs it's own IP address. This class holds that value, as well as provide several functions for determining priorities.

- class **os::UDPPacket**
- class **os::UDPClient**
- struct **os::UDPAVLNode**
- class **os::UDPServer**

Namespaces

- **os**

Functions

- void **os::sleep** (int32_t x)

  Sleep the thread for a certain amount of time This is a simple sleep function, it takes in a length of time to sleep and return nothing.

- void **os::startInternet** (bool multiThread=true)

  Activates Internet Spawns an IP thread distributor, if one does not currently exist.

- void **os::closeInternet** ()

  Deactivates Internet deletes the IP thread distributor and sets the thread pointer to null.

- smart_ptr< **threadDistributor** > **os::internetThreads** ()

  Return IP thread distributor Gives the ipthread distributor to the caller.
- int32_t **os::cp_clock_gettime** (int32_t X, struct timeval ∗tv)

  Gets time Returns the current time to the caller. This is designed to work across a range of platforms and format the time to a high precision.
- void **os::strcpy_s** (char ∗output, int32_t inlen, const char ∗input)

  String copier Safely calls string copy.
- bool **os::is_directory** (std::string file)

  Determines if a file is a directory Checks if a given file is a directory.
- bool **os::check_exists** (std::string name)

  Checks if a given file exists Takes a file and checks if it exists. A directory is considered existing.
- smart_ptr< std::string > **os::list_files** (std::string directory, uint32_t &len)

  Return contents of directory Creates an array of strings of all the names inside a given directory. This is not recursive.
- std::string **os::extract_name** (std::string full_path)

  Extracts a given file Extracts a file or directory.
- void **os::delete_file** (std::string path)

  Deletes a file Deletes the file or directory at the given path. This is a recursive delete.
- void **os::setLocalPath** (int argc, char ∗∗argv)

  Sets local path Sets the local path given the received arguments.
- std::string **os::getLocalPath** ()

  Returns local path.
- static int32_t **os::fopen_s** (FILE ∗∗fp, const char ∗file_name, const char ∗typ)

  fopen_s for windows This is a file open function for windows so that we can more efficiently write multi platform code.

Variables

- const uint32_t **CLOCK_REALTIME** =0
- const uint32_t **CLOCK_MONOTONIC** =1
- const uint32_t **os::BUFLEN** =512
- const std::string **os::DEFAULT_IP** ="127.0.0.1"
- const uint32_t **os::MY_MESSAGE_NOTIFICATION** =1048

K.25.1 Detailed Description

os functions header file

Jonathan Bedard

Date

5/20/2016

Bug No known bugs.

This is the file which contains the declarations for the OS unique functions.

K.25.2 Variable Documentation

const uint32_t CLOCK_MONOTONIC =1

const uint32_t CLOCK_REALTIME =0

K.26 unix_Serial.h File Reference

Serial header file.

Classes

- class **os::Serial**

  *This is the **Serial** (*p. **??***) class. **Serial** (*p. **??***) objects allow us to abstract out most of the platform irregularities across multiple systems.*

Namespaces

- **os**

Variables

- const uint32_t **os::ARDUINO_WAIT_TIME** =2000

K.26.1 Detailed Description

Serial header file.

Jonathan Bedard

Date

5/20/2016

Bug No known bugs.

This is the Serial thread. It allows us to establish serial communication across a range of systems. There are multiple versions of this header and C file. Which version is used is determined by the current platform. This is the UNIX version.

K.27 unix_spinLock.cpp File Reference

spinLock implementation file

K.27.1 Detailed Description

spinLock implementation file

Jonathan Bedard

Date

5/20/2016

Bug No known bugs.

This is the UNIX implementation of our spin lock.

K.28 unix_spinLock.h File Reference

spin lock header file

Classes

- class **os::spinLock**

Namespaces

- **os**

K.28.1 Detailed Description

spin lock header file

Jonathan Bedard

Date

5/20/2016

Bug No known bugs.

This is the spinLock that we use to safely multi thread.

K.29 USBAccess.cpp File Reference

USBAccess implementation file.

K.29.1 Detailed Description

USBAccess implementation file.

Jonathan Bedard

Date

11/3/2015

Bug No known bugs.

These are simple USB methods. They are unused in our larger project.

K.30 USBAccess.h File Reference

USBAccess header file.

Classes

- class **os::USBNode**
  This class stores the location of a USB device.
- class **os::USBFile**

Namespaces

- **os**

K.30.1 Detailed Description

USBAccess header file.

Jonathan Bedard

Date

6/21/2015

Bug No known bugs.

This is a pair of simple classes for working with USB devices.

K.31 win_osFunctions.cpp File Reference

os functions implementation file

K.31.1 Detailed Description

os functions implementation file

Jonathan Bedard

Date

5/20/2016

Bug No known bugs.

This is the implementation of the windows specific functions.

K.32 win_osFunctions.h File Reference

os functions header file

K.32.1 Detailed Description

os functions header file

Jonathan Bedard

Date

5/20/2016

Bug No known bugs.

This is the file which contains the declarations for the OS unique functions.

K.33 win_Serial.h File Reference

Serial header file.

K.33.1 Detailed Description

Serial header file.

Jonathan Bedard

Date

5/20/2016

Bug No known bugs.

This is the Serial thread. It allows us to establish serial communication across a range of systems. There are multiple versions of this header and C file. Which version is used is determined by the current platform. This is the windows version.

K.34 win_spinLock.cpp File Reference

spinLock implementation file

K.34.1 Detailed Description

spinLock implementation file

Jonathan Bedard

Date

5/20/2016

Bug No known bugs.

This is the windows implementation of our spin lock.

K.35 win_spinLock.h File Reference

spin lock header file

K.35.1 Detailed Description

spin lock header file

Jonathan Bedard

Date

5/20/2016

Bug No known bugs.

This is the spinLock that we use to safely multi thread.

K.36 XMLParser.cpp File Reference

XML parser implementation file.

K.36.1 Detailed Description

XML parser implementation file.

Jonathan Bedard

Date

2/7/2015

Bug No known bugs.

Our XML parse is implemented in this file. We have several functions that allow us to easily convert XML data from file to program and vice versa.

K.37 XMLParser.h File Reference

XML Parser header file.

Classes

- class **os::XML_Node**

  XML Node class The core node of our XML parsing.

Namespaces

- **os**
- **os::xml**

Typedefs

- typedef smart_ptr< XML_Node > **os::smartXMLNode**
- typedef smart_ptr< unsortedList< XML_Node > > **os::smartXMLNodeList**

Functions

- bool **os::XML_Output** (std::string path, **smartXMLNode** head)

  outputs tree Outputs an XML tree into a file.
- **smartXMLNode os::XML_Input** (std::string path)

  imports tree Imports an XML tree from a file.
- void **os::xml::insertTabs** (std::ofstream &f, int32_t x)

  adds tabs Adds tabs.
- void **os::xml::writeNode** (std::ofstream &f, smartXMLNode node, int32_t depth)

  writes nodes Writes all the nodes to a file. This function runs recursively.
- std::vector< std::string > **os::xml::readTillTag** (std::ifstream &f)

  reads until next tag Reads a file until the next tag is found.
- std::string **os::xml::readThroughTag** (std::ifstream &f)

  reads through the next tag Reads a file until a tag is found, including that tag.
- **smartXMLNode os::xml::parseNode** (std::ifstream &f)

  parses a node Pulls a node from a file and returns it.
- bool **os::xml::compareTrees** (**smartXMLNode** n1, **smartXMLNode** n2)

  compares trees Determines if two nodes are equivalent.

K.37.1 Detailed Description

XML Parser header file.

Jonathan Bedard

Date

2/7/2015

Bug No known bugs.

This is our XML Parser, so we can standardize use across systems.

K.38 XMLTest.cpp File Reference

XML tests.

K.38.1 Detailed Description

XML tests.

Jonathan Bedard

Date

2/29/2016

Bug No known bugs.

These are the tests for our XML classes.

K.39 XMLTest.h File Reference

SML test header file.

K.39.1 Detailed Description

SML test header file.

Jonathan Bedard

Date

4/12/2016

Bug No known bugs.

This is the test suite for the XML tests.

Part IV

CryptoGateway Library

Appendix L

Introduction

The CryptoGateway library contains classes which handle cryptography. CryptoGateway is designed as an open source library, so much of the cryptography within the library is relatively simple. Crypto-Gateway is not meant to define cryptography to be used widely, rather, it is meant to provide a series of generalized hooks and interfaces which can be extended to various cryptographic algorithms.

L.1 Namespace

CryptoGateway uses the crypto namespace. The crypto namespace is designed for class, functions and constants related to cryptography. CrytpoGateway depends on many of the tools defined in the os namespace. Additionally, the crypto namespace contains a series of nested namespaces which help to disambiguate constants.

Appendix M

File Index

M.1 File List

Here is a list of all files with brief descriptions:

Appendix N

File Documentation

N.1 binaryEncryption.cpp File Reference

Implementation of binary encryption files.

N.1.1 Detailed Description

Implementation of binary encryption files.

Author

Jonathan Bedard

Date

4/18/2016

Bug None

Implements the binary encryption files. Consult **binaryEncryption.h** (p. 189) for details on using these classes.

N.2 binaryEncryption.h File Reference

Definition of binary encryption files.

Classes

- class **crypto::binaryEncryptor**

Encrypted binary file output.

- class **crypto::binaryDecryptor**
  Encrypted binary file output.

Namespaces

- **crypto**

N.2.1 Detailed Description

Definition of binary encryption files.

Author

Jonathan Bedard

Date

3/7/2016

Bug None

Provides an interface to dump and retrieve data from an encrypted binary file without concern as to the encryption algorithm used.

N.3 c_BaseTen.c File Reference

Implementation of base-10 algorithms.

N.3.1 Detailed Description

Implementation of base-10 algorithms.

Author

Jonathan Bedard

Date

2/12/2016

Bug No known bugs.

This file implements all of the basic functionality of a base-10 integer. All integer operations, both basic and otherwise, are implemented in this file.

N.4 c_BaseTen.h File Reference

Base-10 number functions.

Functions

- struct **numberType** ∗ **buildBaseTenType** ()
  Construct a base-10 number.
- int **base10Addition** (const uint32_t ∗src1, const uint32_t ∗src2, uint32_t ∗dest, uint16_t length)
  Base-10 addition.
- int **base10Subtraction** (const uint32_t ∗src1, const uint32_t ∗src2, uint32_t ∗dest, uint16_t length)
  Base-10 subtraction.
- int **base10Multiplication** (const uint32_t ∗src1, const uint32_t ∗src2, uint32_t ∗dest, uint16_t length)
  Base-10 multiplication.
- int **base10Division** (const uint32_t ∗src1, const uint32_t ∗src2, uint32_t ∗dest, uint16_t length)
  Base-10 division.
- int **base10Modulo** (const uint32_t ∗src1, const uint32_t ∗src2, uint32_t ∗dest, uint16_t length)
  Base-10 modulo.
- int **base10Exponentiation** (const uint32_t ∗src1, const uint32_t ∗src2, uint32_t ∗dest, uint16↩ _t length)
  Base-10 exponentiation.
- int **base10ModuloExponentiation** (const uint32_t ∗src1, const uint32_t ∗src2, const uint32_t ∗src3, uint32_t ∗dest, uint16_t length)
- int **base10GCD** (const uint32_t ∗src1, const uint32_t ∗src2, uint32_t ∗dest, uint16_t length)

- int **base10ModInverse** (const uint32_t ∗src1, const uint32_t ∗src2, uint32_t ∗dest, uint16_t length)
- int **primeTest** (const uint32_t ∗src1, uint16_t test_iteration, uint16_t length)

N.4.1 Detailed Description

Base-10 number functions.

Author

Jonathan Bedard

Date

2/12/2016

Bug No known bugs.

Contains functions which define a base-10 integer. There functions are bound to a number type.

N.4.2 Function Documentation

int base10Addition (const uint32_t ∗ src1, const uint32_t ∗ src2, uint32_t ∗ dest, uint16_t length)

Base-10 addition.

This function takes in two arrays which represent base-10 numbers, preforms src1+src2 on the pair and then output the result to dest. Note that all three arrays must be the same size.

Parameters

`in`	*src1*	Argument 1
`in`	*src2*	Argument 2
`out`	*dest*	Output
`in`	*length*	Number of uint32_t in the arrays

Returns

1 if success, 0 if failed

int base10Division (const uint32_t ∗ src1, const uint32_t ∗ src2, uint32_t ∗ dest, uint16_t length)

Base-10 division.

This function takes in two arrays which represent base-10 numbers, preforms src1/src2 on the pair and then output the result to dest. Note that all three arrays must be the same size.

Parameters

in	*src1*	Argument 1
in	*src2*	Argument 2
out	*dest*	Output
in	*length*	Number of uint32_t in the arrays

Returns

1 if success, 0 if failed

int base10Exponentiation (const uint32_t ∗ src1, const uint32_t ∗ src2, uint32_t ∗ dest, uint16_t length)

Base-10 exponentiation.

This function takes in two arrays which represent base-10 numbers, preforms src1+src2 on the pair and then output the result to dest. Note that all three arrays must be the same size.

Parameters

in	*src1*	Argument 1
in	*src2*	Argument 2
out	*dest*	Output
in	*length*	Number of uint32_t in the arrays

Returns

1 if success, 0 if failed

int base10GCD (const uint32_t ∗ src1, const uint32_t ∗ src2, uint32_t ∗ dest, uint16_t length)

int base10ModInverse (const uint32_t ∗ src1, const uint32_t ∗ src2, uint32_t ∗ dest, uint16_t length)

int base10Modulo (const uint32_t ∗ src1, const uint32_t ∗ src2, uint32_t ∗ dest, uint16_t length)

Base-10 modulo.

This function takes in two arrays which represent base-10 numbers, preforms src1src2 on the pair and then output the result to dest. Note that all three arrays must be the same size.

Parameters

`in`	*src1*	Argument 1
`in`	*src2*	Argument 2
`out`	*dest*	Output
`in`	*length*	Number of uint32_t in the arrays

Returns

1 if success, 0 if failed

int base10ModuloExponentiation (const uint32_t ∗ src1, const uint32_t ∗ src2, const uint32_t ∗ src3, uint32_t ∗ dest, uint16_t length)

int base10Multiplication (const uint32_t ∗ src1, const uint32_t ∗ src2, uint32_t ∗ dest, uint16_t length)

Base-10 multiplication.

This function takes in two arrays which represent base-10 numbers, preforms src1∗src2 on the pair and then output the result to dest. Note that all three arrays must be the same size.

Parameters

`in`	*src1*	Argument 1
`in`	*src2*	Argument 2

Parameters

`out`	*dest*	Output
`in`	*length*	Number of uint32_t in the arrays

Returns

1 if success, 0 if failed

int base10Subtraction (const uint32_t ∗ src1, const uint32_t ∗ src2, uint32_t ∗ dest, uint16_t length)

Base-10 subtraction.

This function takes in two arrays which represent base-10 numbers, preforms src1-src2 on the pair and then output the result to dest. Note that all three arrays must be the same size.

Parameters

`in`	*src1*	Argument 1
`in`	*src2*	Argument 2
`out`	*dest*	Output
`in`	*length*	Number of uint32_t in the arrays

Returns

1 if success, 0 if failed

struct **numberType**∗ buildBaseTenType ( )

Construct a base-10 number.

This function will return a **numberType** (p. **??**) pointer defining the function pointers for a base-10 number. Note that the resulting pointer points to a structure which is static to the **c_BaseTen.c** (p. 190) file.

Returns

Pointer to **numberType** (p. **??**) of type base-10

int primeTest (const uint32_t ∗ src1, uint16_t test_iteration, uint16_t length)

N.5 c_cryptoTesting.cpp File Reference

Implementation for C file testing.

N.5.1 Detailed Description

Implementation for C file testing.

Author

Jonathan Bedard

Date

2/12/2016

Bug No known bugs.

This file implements test suites which are testing raw C code. This file currently tests the Base-↩ Ten suite.

N.6 c_cryptoTesting.h File Reference

Header for C file testing.

N.6.1 Detailed Description

Header for C file testing.

Author

Jonathan Bedard

Date

2/12/2016

Bug No known bugs.

This header is meant for the test suites which are testing raw C code. This header currently contains the Base-Ten suite.

N.7 c_numberDefinitions.c File Reference

Implementation of basic number.

N.7.1 Detailed Description

Implementation of basic number.

Author

Jonathan Bedard

Date

2/12/2016

Bug No known bugs.

Most numerical operations must be defined by the specific number type, but a select few are generally applicable across all number types, these are implemented here.

N.8 c_numberDefinitions.h File Reference

Basic number declarations.

Classes

- struct **numberType**
  Number type function structure.

Typedefs

- typedef int(∗ **operatorFunction**) (const uint32_t ∗, const uint32_t ∗, uint32_t ∗, uint16_t)
  Operator function typedef.
- typedef int(∗ **tripleCalculation**) (const uint32_t ∗, const uint32_t ∗, const uint32_t ∗, uint32_t ∗, uint16_t)
  Triple operator function typedef.
- typedef int(∗ **shiftFunction**) (const uint32_t ∗, uint16_t, uint32_t ∗, uint16_t)
  Shift operator function typedef.
- typedef int(∗ **compareFunction**) (const uint32_t ∗, const uint32_t ∗, uint16_t)
  Comparison function typedef.

Functions

- struct **numberType** ∗ **buildNullNumberType** ()
  Construct a NULL number.
- int **standardCompare** (const uint32_t ∗src1, const uint32_t ∗src2, uint16_t length)
  Standard comparision.
- int **standardRightShift** (const uint32_t ∗src1, uint16_t src2, uint32_t ∗dest, uint16_t length)
  Right shift.
- int **standardLeftShift** (const uint32_t ∗src1, uint16_t src2, uint32_t ∗dest, uint16_t length)
  Left shift.

N.8.1 Detailed Description

Basic number declarations.

Author

Jonathan Bedard

Date

2/12/2016

Bug No known bugs.

Contains function typedefs used for various number operations and defines a few nearly universal numerical functions.

N.8.2 Typedef Documentation

typedef int(∗ compareFunction) (const uint32_t ∗, const uint32_t ∗, uint16_t)

Comparison function typedef.

This function typedef defines a function which takes in two arrays which represent numbers and then compares them.

Parameters

`in`	*uint32↩ _t∗*	Argument 1
`in`	*uint32↩ _t∗*	Argument 2
`in`	*uint16↩ _t*	size

Returns

-1 if 1<2, 0 if 1==2, 1 if 1>2

typedef int(∗ operatorFunction) (const uint32_t ∗, const uint32_t ∗, uint32_t ∗, uint16_t)

Operator function typedef.

This function typedef defines a function which takes in two arrays which represent numbers, preform some operation on the pair and then output the result to a third array.

Parameters

`in`	*uint32↩ _t∗*	Argument 1
`in`	*uint32↩ _t∗*	Argument 2
`out`	*uint32↩ _t∗*	Output

Parameters

in	*uint16↩ _t*	size

Returns

1 if success, 0 if failed

typedef int(∗ shiftFunction) (const uint32_t ∗, uint16_t, uint32_t ∗, uint16_t)

Shift operator function typedef.

This function typedef defines a function which takes in an array representing a number, shifts it the provided number of bits and outputs the result into the second array.

Parameters

in	*uint32↩ _t∗*	Argument 1
in	*uint16↩ _t*	Bits to shift
out	*uint32↩ _t∗*	Output
in	*uint16↩ _t*	size

Returns

1 if success, 0 if failed

typedef int(∗ tripleCalculation) (const uint32_t ∗, const uint32_t ∗, const uint32_t ∗, uint32_t ∗, uint16_t)

Triple operator function typedef.

This function typedef defines a function which takes in three arrays which represent numbers, preform some operation on the triple and then output the result to a fourth array.

Parameters

in	*uint32↩ _t∗*	Argument 1
in	*uint32↩ _t∗*	Argument 2
in	*uint32↩ _t∗*	Argument 3
out	*uint32↩ _t∗*	Output
in	*uint16↩ _t*	size

Returns

1 if success, 0 if failed

N.8.3 Function Documentation

struct **numberType**∗ buildNullNumberType ( )

Construct a NULL number.

This function will return a **numberType** (p. **??**) pointer defining the function pointers for a NULL number. Note that the resulting pointer points to a structure which is static to the **c_number↩ Definitions.c** (p. 197) file.

Returns

Pointer to **numberType** (p. **??**) of type NULL

int standardCompare (const uint32_t ∗ src1, const uint32_t ∗ src2, uint16_t length)

Standard comparision.

This function takes in two arrays which represent numbers and then compares them.

Parameters

in	*src1*	Argument 1
in	*src2*	Argument 2
in	*length*	Number of uint32_t in the arrays

Returns

-1 if 1<2, 0 if 1==2, 1 if 1>2

int standardLeftShift (const uint32_t ∗ src1, uint16_t src2, uint32_t ∗ dest, uint16_t length)

Left shift.

Shifts the bits in src1 in the left direction src2 number of bits. Output the result in dest. Note that dest and src1 should be the same size.

Parameters

in	*src1*	Argument 1
in	*src2*	Bits to shift
out	*dest*	Output
in	*length*	Number of uint32_t in the arrays

Returns

1 if success, 0 if failed

int standardRightShift (const uint32_t ∗ src1, uint16_t src2, uint32_t ∗ dest, uint16_t length)

Right shift.

Shifts the bits in src1 in the right direction src2 number of bits. Output the result in dest. Note that dest and src1 should be the same size.

Parameters

in	*src1*	Argument 1

Parameters

`in`	*src2*	Bits to shift
`out`	*dest*	Output
`in`	*length*	Number of uint32_t in the arrays

Returns

1 if success, 0 if failed

N.9 cryptoCConstants.h File Reference

Extern declarations of C constants.

Variables

- const int **crypto_numbertype_default**
  Default number ID.
- const int **crypto_numbertype_base10**
  Base-10 number ID.
- const char $*$ **crypto_numbername_default**
  Default number marker.
- const char $*$ **crypto_numbername_base10**
  Base-10 number marker.

N.9.1 Detailed Description

Extern declarations of C constants.

Author

Jonathan Bedard

Date

2/12/2016

Bug No known bugs.

Declares a number of constants needed by both the C numerical algorithms and by C++ number classes.

N.9.2 Variable Documentation

const char∗ crypto_numbername_base10

Base-10 number marker.

This constant is "Base 10 Type". It represents a number of type base-10, or standard integer.

const char∗ crypto_numbername_default

Default number marker.

This constant is "NULL Type". It represents an untyped number.

const int crypto_numbertype_base10

Base-10 number ID.

This constant is 1. It represents a number of type base-10, or standard integer.

const int crypto_numbertype_default

Default number ID.

This constant is 0. It represents an untyped number.

N.10 cryptoCHeaders.h File Reference

Collected headers for C source code.

N.10.1 Detailed Description

Collected headers for C source code.

Author

Jonathan Bedard

Date

2/20/2016

Bug None

N.11 cryptoConstants.cpp File Reference

Implementation of CryptoGateway constants.

N.11.1 Detailed Description

Implementation of CryptoGateway constants.

Author

Jonathan Bedard

Date

3/19/2016

Bug None

Binds all of the scoped constants used by CryptoGateway. The nested namespaces ensure that there is no ambiguity as to the purpose and nature of the constants.

N.12 cryptoConstants.h File Reference

Extern definitions of CryptoGateway constants.

N.12.1 Detailed Description

Extern definitions of CryptoGateway constants.

Author

Jonathan Bedard

Date

3/19/2016

Bug None

Consult **cryptoConstants.cpp** (p. 205) for details. This file merely defines extern references to the global constants in **cryptoConstants.cpp** (p. 205).

N.13 cryptoCSource.cpp File Reference

Implementation of all C code.

N.13.1 Detailed Description

Implementation of all C code.

Author

Jonathan Bedard

Date

2/13/2016

Bug No known bugs.

This file includes all of the .c files needed for this library. It allows the CMake scripts for this project to be entirely C++ while still includeing raw C code.

N.14 cryptoError.cpp File Reference

Implementation of error sender and listener.

N.14.1 Detailed Description

Implementation of error sender and listener.

Author

Jonathan Bedard

Date

4/16/2016

Bug None

Implements the error sender and listeners. These classes allow for managing the throwing of **crypto::errorPointer** (p. **??**). Consult **cryptoError.h** (p. 207) for details.

N.15 cryptoError.h File Reference

Declaration of cryptographic errors.

Classes

- class **crypto::error**
  Sortable exception.
- class **crypto::passwordSmallError**
  Symmetric key too small.
- class **crypto::passwordLargeError**
  Symmetric key too big.
- class **crypto::bufferSmallError**
  Buffer too small.
- class **crypto::bufferLargeError**
  Buffer too large.
- class **crypto::insertionFailed**
  ADS Insertion Failed.
- class **crypto::customError**
  *Custom **crypto::error** (p. **??**).*

- class **crypto::fileOpenError**

  File open error.
- class **crypto::fileFormatError**

  File format error.
- class **crypto::illegalAlgorithmBind**

  Algorithm bound failure.
- class **crypto::hashCompareError**

  Hash mis-match.
- class **crypto::hashGenerationError**

  Hash generation error.
- class **crypto::actionOnFileError**

  File error.
- class **crypto::actionOnFileClosed**

  File closed error.
- class **crypto::publicKeySizeWrong**

  Public-key size error.
- class **crypto::keyMissing**

  Key missing error.
- class **crypto::NULLPublicKey**

  NULL public-key error.
- class **crypto::NULLDataError**

  NULL data error.
- class **crypto::NULLMaster**

  NULL master error.
- class **crypto::masterMismatch**

  Master mis-match.
- class **crypto::unknownErrorType**

  Unknown error.
- class **crypto::stringTooLarge**

  String size error.
- class **crypto::errorListener**

  ***crypto::error** (p. **??**) listener*
- class **crypto::errorSender**

  *Sends **crypto::error** (p. **??**).*

Namespaces

- **crypto**

Typedefs

- typedef os::smart_ptr< error > **crypto::errorPointer**
  Smart pointer to ***crypto::error*** *(p.* **??***).*

N.15.1 Detailed Description

Declaration of cryptographic errors.

Author

Jonathan Bedard

Date

4/1/2016

Bug None

Declares a number of errors for the CryptoGateway package. Also declares two classes to manage the sending and listening for the throwing of **crypto::errorPointer** (p. **??**).

N.16 cryptoFileTest.cpp File Reference

Implementation for cryptographic file testing.

N.16.1 Detailed Description

Implementation for cryptographic file testing.

Author

Jonathan Bedard

Date

4/18/2016

Bug No known bugs.

This file implements a series of tests designed to confirm the stability of cryptographic save file and load file functions.

N.17 cryptoFileTest.h File Reference

Header for cryptographic file testing.

N.17.1 Detailed Description

Header for cryptographic file testing.

Author

Jonathan Bedard

Date

3/5/2016

Bug No known bugs.

This contains a number of test suites and supporting classes which are designed to test the functionality of saving and loading cryptographic files, both binary and EXML.

N.18 CryptoGateway.h File Reference

Global include file.

Namespaces

- **crypto**

Variables

- bool **crypto::global_logging**
  Deprecated logging flag.

N.18.1 Detailed Description

Global include file.

Author

Jonathan Bedard

Date

4/16/2016

Bug None

This file contains all of the headers in the CryptoGateway library. Project which depend on the CryptoGateway library need only include this file.

N.19 cryptoHash.cpp File Reference

Implementation of crypto hashing.

N.19.1 Detailed Description

Implementation of crypto hashing.

Implementation of RC4 hash.

Author

Jonathan Bedard

Date

2/23/2016

Bug None

Implements basic hashing frameworks and the XOR hash. Note that the XOR hash is not cryptographically secure. Consult **cryptoHash.h** (p. 212) for details.

Author

Jonathan Bedard

Date

2/23/2016

Bug None

Implements the RC-4 hash algorithm. The RC-4 hashing algorithm is likely secure, but not proven secure. Consult the **RC4_Hash.h** (p. 229) for details.

N.20 cryptoHash.h File Reference

Declaration of crypto hashing.

Classes

- class **crypto::hash**
  Base hash class.
- class **crypto::xorHash**
  XOR hash class.

Namespaces

- **crypto**

Functions

- std::ostream & **crypto::operator**<< (std::ostream &os, const hash &num)
  Output stream operator.
- std::istream & **crypto::operator**>> (std::istream &is, hash &num)
  Input stream operator.
- template<class hashClass >

  hashClass **crypto::hashData** (uint16_t hashType, const unsigned char ∗data, uint32_t length)
  Hashes data with the specified algorithm.

N.20.1 Detailed Description

Declaration of crypto hashing.

Implementation of RC4 hash.

Author

Jonathan Bedard

Date

2/23/2016

Bug None

Declares base cryptographic hashing class and functions. All hash algorithms should extend this hash class.

Author

Jonathan Bedard

Date

2/23/2016

Bug None

Declares the RC-4 hash algorithm. The RC-4 hashing algorithm is likely secure, but not proven secure.

N.21 cryptoLogging.cpp File Reference

Logging for crypto namespace, implementation.

N.21.1 Detailed Description

Logging for crypto namespace, implementation.

Jonathan Bedard

Date

2/23/2016

Bug No known bugs.

This file contains global functions and variables used for logging in the crypto namespace.

N.22 cryptoLogging.h File Reference

Logging for crypto namespace.

Namespaces

- **crypto**

Functions

- std::ostream & **crypto::cryptoout_func** ()
  Standard out object for crypto namespace.
- std::ostream & **crypto::cryptoerr_func** ()
  Standard error object for crypto namespace.

Variables

- os::smart_ptr< std::ostream > **crypto::cryptoout_ptr**
  Standard out pointer for crypto namespace.
- os::smart_ptr< std::ostream > **crypto::cryptoerr_ptr**
  Standard error pointer for crypto namespace.

N.22.1 Detailed Description

Logging for crypto namespace.

Jonathan Bedard

Date

2/23/2016

Bug No known bugs.

This file contains declarations which are used for logging within the crypto namespace.

N.23 cryptoNumber.cpp File Reference

Implements basic number types.

N.23.1 Detailed Description

Implements basic number types.

Author

Jonathan Bedard

Date

4/3/2016

Bug No known bugs.

Implements basic large numbers and the more specific large integer. Consult **cryptoNumber.h** (p. 215) for details.

N.24 cryptoNumber.h File Reference

Defines basic number types.

Classes

- class **crypto::number**
  Basic number definition.
- class **crypto::integer**
  Integer number definition.

Namespaces

- **crypto**

Functions

- std::ostream & **crypto::operator**<< (std::ostream &os, const number &num)
  Output stream operator.
- std::istream & **crypto::operator**>> (std::istream &is, number &num)
  Input stream operator.

N.24.1 Detailed Description

Defines basic number types.

Author

Jonathan Bedard

Date

3/2/2016

Bug No known bugs.

Contains declarations of large numbers for usage inside the CryptoGateway. The two numbers defined in this file are the general structure for large numbers and a basic integer.

N.25 cryptoNumberTest.cpp File Reference

Testing **crypto::number** (p. **??**) and **crypto::integer** (p. **??**).

N.25.1 Detailed Description

Testing **crypto::number** (p. **??**) and **crypto::integer** (p. **??**).

Author

Jonathan Bedard

Date

4/18/2016

Bug No known bugs.

This file has a series of tests which confirm the functionality of **crypto::integer** (p. **??**) and it's base class, **crypto::number** (p. **??**).

N.26 cryptoPublicKey.cpp File Reference

Generalized and RSA public key implementation.

N.26.1 Detailed Description

Generalized and RSA public key implementation.

Author

Jonathan Bedard

Date

5/5/2016

Bug No known bugs.

Contains implementation of the generalized public key and the RSA public key. Consult **crypto↩ PublicKey.h** (p. 217) for details.

N.27 cryptoPublicKey.h File Reference

Generalized and RSA public keys.

Classes

- class **crypto::keyChangeReceiver**

    Interface for receiving key changes.

- class **crypto::keyChangeSender**

    Interface inherited by ***publicKey*** *(p.* **??***).*

- class **crypto::publicKey**

    Base public-key class.

- class **crypto::publicRSA**

    RSA public-key encryption.

- class **crypto::RSAKeyGenerator**

    Helper key generation class.

Namespaces

- **crypto**

N.27.1 Detailed Description

Generalized and RSA public keys.

Author

Jonathan Bedard

Date

5/9/2016

Bug No known bugs.

Contains declarations of the generalized public key and the RSA public key. These classes can both encrypt and decrypt public keys.

N.28 cryptoTest.cpp File Reference

CryptoGateway library test constructor.

N.28.1 Detailed Description

CryptoGateway library test constructor.

Author

Jonathan Bedard

Date

4/7/2016

Bug No known bugs.

Binds all test suites for the test::CryptoGatewayLibraryTest. This library test is called "Crypto↩ Gateway."

N.29 cryptoTest.h File Reference

CryptoGateway library test header.

N.29.1 Detailed Description

CryptoGateway library test header.

Author

Jonathan Bedard

Date

4/2/2016

Bug No known bugs.

Contains declarations need to bind the CryptoGateway test library to the unit test driver.

N.30 gateway.cpp File Reference

Implements the gateway.

N.30.1 Detailed Description

Implements the gateway.

Author

Jonathan Bedard

Date

5/9/2016

Bug No known bugs.

Implements the gateway defined in **gateway.h** (p. 220). Consult **gateway.h** (p. 220) for details.

N.31 gateway.h File Reference

Defines the gateway.

Classes

- class **crypto::gatewaySettings**
  Holds settings for gateway encryption.
- class **crypto::gateway**
  Security gateway.

Namespaces

- **crypto**

N.31.1 Detailed Description

Defines the gateway.

Author

Jonathan Bedard

Date

5/9/2016

Bug No known bugs.

This file contains the declaration for the gateway and the gateway settings. This header file is the culmination of the CryptoGateway library.

Note that due to development constraints, the gatewaySettings class is being pushed out in a frame-work form and is intended to contain a large set of algorithm definitions as well as an algorithm use agreement protocol.

N.32 gatewayTest.cpp File Reference

Implementation for end-to-end gateway testing.

N.32.1 Detailed Description

Implementation for end-to-end gateway testing.

Author

Jonathan Bedard

Date

4/26/2016

Bug No known bugs.

This file contains implementation of the key bank tests and the end-to-end gateway tests. These tests are not exhaustive, they test basic functionality of both structures.

N.33 gatewayTest.h File Reference

Header for end-to-end gateway testing.

N.33.1 Detailed Description

Header for end-to-end gateway testing.

Author

Jonathan Bedard

Date

3/20/2016

Bug No known bugs.

This header contains declarations of the key bank tests and the end-to-end gateway tests. These tests are not exhaustive, they test basic functionality of both structures.

N.34 hashTest.cpp File Reference

Implementation for hash tests.

N.34.1 Detailed Description

Implementation for hash tests.

Author

Jonathan Bedard

Date

4/18/2016

Bug No known bugs.

This file contains algorithm-specific cryptographic hash testing. These tests confirm that the respective hash algorithms are outputting their expected value.

N.35 hashTest.h File Reference

Header for hash testing.

N.35.1 Detailed Description

Header for hash testing.

Author

Jonathan Bedard

Date

4/18/2016

Bug No known bugs.

This file contains a number of template classes used to confirm the functionality of cryptographic hash algorithms.

N.36 hexConversion.cpp File Reference

Hex conversion implementation.

N.36.1 Detailed Description

Hex conversion implementation.

Author

Jonathan Bedard

Date

3/16/2016

Bug No known bugs.

Implements the set of hex conversion functions. Consult **hexConversion.h** (p. 223) for details.

N.37 hexConversion.h File Reference

Hex conversion header.

Namespaces

- **crypto**

Functions

- bool **crypto::isHexCharacter** (char c)
  Check the character type.
- std::string **crypto::toHex** (unsigned char i)
  Converts an 8 bit integer to a hex string.
- std::string **crypto::toHex** (uint32_t i)
  Converts an 32 bit integer to a hex string.
- unsigned char **crypto::fromHex8** (const std::string &str)
  Converts a hex string to an 8 bit integer.
- uint32_t **crypto::fromHex32** (const std::string &str)
  Converts a hex string to an 32 bit integer.

N.37.1 Detailed Description

Hex conversion header.

Author

Jonathan Bedard

Date

3/16/2016

Bug No known bugs.

Contains a set of functions to convert integers and characters from a hex string and converts hex strings to integers and characters.

N.38 keyBank.cpp File Reference

Implimentation for the AVL tree based key bank.

N.38.1 Detailed Description

Implimentation for the AVL tree based key bank.

Author

Jonathan Bedard

Date

4/19/2016

Bug No known bugs.

This file contians the implimentation for the **crypto::avlKeyBank** (p. **??**) and supporting classes. Consult **keyBank.h** (p. 225) for details.

N.39 keyBank.h File Reference

Header for the AVL tree based key bank.

Classes

- class **crypto::nodeGroup**
  Node group.
- class **crypto::nodeNameReference**
  Name storage node.
- class **crypto::nodeKeyReference**
  Key storage node.
- class **crypto::keyBank**
  Key bank interface.
- class **crypto::avlKeyBank**
  AVL key back.

Namespaces

- **crypto**

N.39.1 Detailed Description

Header for the AVL tree based key bank.

Author

Jonathan Bedard

Date

4/19/2016

Bug No known bugs.

This file contians declarations for the **crypto::avlKeyBank** (p. **??**) and supporting classes. Note that the key-bank may later be implimented with more advanced datastructures.

N.40 message.cpp File Reference

Crypto-Gateway message implementation.

N.40.1 Detailed Description

Crypto-Gateway message implementation.

Author

Jonathan Bedard

Date

4/16/2016

Bug No known bugs.

Implements the message used by the crypto-gateway to pass encrypted data between machines.

N.41 message.h File Reference

Crypto-Gateway message.

Classes

- class **crypto::message**
  Crypto-Gateway message.

Namespaces

- **crypto**

N.41.1 Detailed Description

Crypto-Gateway message.

Author

Jonathan Bedard

Date

4/16/2016

Bug No known bugs.

The message declared in this file acts as a message for the Crypto-Gateway. These messages are intended to be converted to machine-to-machine communication.

N.42 publicKeyPackage.cpp File Reference

Implementation of public key bank.

N.42.1 Detailed Description

Implementation of public key bank.

Author

Jonathan Bedard

Date

5/19/2016

Bug None

Implements a bank of public key types to be accessed at run-time. Essentially acts as a meta-object access bank.

N.43 publicKeyPackage.h File Reference

Declaration of public key bank.

Classes

- class **crypto::publicKeyPackageFrame**
- class **crypto::publicKeyPackage**< **pkType** >
- class **crypto::publicKeyTypeBank**

Namespaces

- **crypto**

N.43.1 Detailed Description

Declaration of public key bank.

Author

Jonathan Bedard

Date

5/19/2016

Bug None

Declares a bank of public keys as well as supporting classes. Acts as a meta-object construct for public-key algorithms.

N.44 publicKeyTest.h File Reference

Public Key tests.

N.44.1 Detailed Description

Public Key tests.

Author

Jonathan Bedard

Date

4/18/2016

Bug No known bugs.

Since the public key tests are defined by very simple tests, the template testing classes contained in this file are also defined in this file. There is no .cpp file paired with this particular header.

N.45 RC4_Hash.cpp File Reference

N.46 RC4_Hash.h File Reference

Classes

- class **crypto::rc4Hash**
  RC-4 hash class.

Namespaces

- **crypto**

N.47 staticTestKeys.cpp File Reference

Auto-generated.

N.47.1 Detailed Description

Auto-generated.

Author

None

Bug None

N.48 staticTestKeys.h File Reference

Auto-generated.

N.48.1 Detailed Description

Auto-generated.

Author

None

Bug None

N.49 streamCipher.cpp File Reference

N.50 streamCipher.h File Reference

Classes

- class **crypto::streamCipher**
- class **crypto::RCFour**
- class **crypto::streamPacket**
- class **crypto::streamEncrypter**
- class **crypto::streamDecrypter**

Namespaces

- **crypto**

Variables

- bool **global_logging**

N.50.1 Variable Documentation

bool global_logging

N.51 streamPackage.cpp File Reference

Implementation of streaming bank.

N.51.1 Detailed Description

Implementation of streaming bank.

Author

    Jonathan Bedard

Date

    5/19/2016

Bug None

Implements a a bank of stream ciphers and hash algorithms to be accessed at run-time. Essentially acts as a meta-object access bank.

N.52 streamPackage.h File Reference

Declaration of streaming bank.

Classes

- class **crypto::streamPackageFrame**
- class **crypto::streamPackage**< **streamType, hashType** >
- class **crypto::streamPackageTypeBank**

Namespaces

- **crypto**

N.52.1 Detailed Description

Declaration of streaming bank.

Author

Jonathan Bedard

Date

5/19/2016

Bug None

Declares a bank of stream ciphers and hash algorithms along with supporting classes. Acts as a meta-object construct for public-key algorithms.

N.53 streamTest.cpp File Reference

Implementation for stream tests.

N.53.1 Detailed Description

Implementation for stream tests.

Author

Jonathan Bedard

Date

4/18/2016

Bug No known bugs.

This file contains algorithm-specific cryptographic stream testing. These tests confirm that the respective stream algorithms are outputting their expected value.

N.54 streamTest.h File Reference

Header for stream testing.

N.54.1 Detailed Description

Header for stream testing.

Author

Jonathan Bedard

Date

4/18/2016

Bug No known bugs.

This file contains a number of template classes used to confirm the functionality of cryptographic stream objects.

N.55 testKeyGeneration.cpp File Reference

N.56 testKeyGeneration.h File Reference

Implementation of test key binding.

N.56.1 Detailed Description

Implementation of test key binding.

Binds generated testing keys.

Author

Jonathan Bedard

Date

4/18/2016

Bug No known bugs.

Implements the binding of the static test keys to arrays in memory. Consult **testKeyGeneration.h** (p. 233) for details.

Author

Jonathan Bedard

Date

2/12/2016

Bug No known bugs.

Provides access to the keys generated and stored in **staticTestKeys.h** (p. 230) and **staticTest←Keys.cpp** (p. 229). These keys are always copied into a raw array of uint32_t.

N.57 user.cpp File Reference

Implementation of the CryptoGateway user.

N.57.1 Detailed Description

Implementation of the CryptoGateway user.

Author

Jonathan Bedard

Date

4/26/2016

Bug None

Provides an implementation of user which has a user-name, password and associated bank of public keys. Consult **user.h** (p. 235) for details.

N.58 user.h File Reference

Definition of the CryptoGateway user.

Classes

- class **crypto::user**
  Primary user class.

Namespaces

- **crypto**

N.58.1 Detailed Description

Definition of the CryptoGateway user.

Author

Jonathan Bedard

Date

4/26/2016

Bug None

Provides a definition of user which has a user-name, password and associated bank of public keys.

N.59 XMLEncryption.cpp File Reference

Implementation of RC-4.

N.59.1 Detailed Description

Implementation of RC-4.

Implements encrypted XML functions.

Author

Jonathan Bedard

Date

5/19/2016

Bug None

Implements the RC-4 stream cipher and more generally, a framework for all stream ciphers to use.

Author

Jonathan Bedard

Date

5/19/2016

Bug None

Implements functions to save and load XML trees in files locked with both a password and with public keys.

N.60 XMLEncryption.h File Reference

Defines basic stream ciphers.

Namespaces

- **crypto**

Functions

- bool **crypto::EXML_Output** (std::string path, os::smartXMLNode head, unsigned char ∗sym↩ Key, unsigned int passwordLength, os::smart_ptr< streamPackageFrame > spf=NULL)
- bool **crypto::EXML_Output** (std::string path, os::smartXMLNode head, std::string password, os::smart_ptr< streamPackageFrame > spf=NULL)

- bool **crypto::EXML_Output** (std::string path, os::smartXMLNode head, os::smart_ptr< public↩ Key > pbk, unsigned int lockType=file::PRIVATE_UNLOCK, os::smart_ptr< streamPackage↩ Frame > spf=NULL)
- bool **crypto::EXML_Output** (std::string path, os::smartXMLNode head, os::smart_ptr< number > publicKey, unsigned int pkAlgo, unsigned int pkSize, os::smart_ptr< streamPackage↩ Frame > spf=NULL)
- os::smartXMLNode **crypto::EXML_Input** (std::string path, unsigned char ∗symKey, unsigned int passwordLength)
- os::smartXMLNode **crypto::EXML_Input** (std::string path, std::string password)
- os::smartXMLNode **crypto::EXML_Input** (std::string path, os::smart_ptr< publicKey > pbk, os::smart_ptr< keyBank > kyBank, os::smart_ptr< nodeGroup > &author)
- os::smartXMLNode **crypto::EXML_Input** (std::string path, os::smart_ptr< publicKey > pbk)
- os::smartXMLNode **crypto::EXML_Input** (std::string path, os::smart_ptr< keyBank > kyBank)
- os::smartXMLNode **crypto::EXML_Input** (std::string path, os::smart_ptr< keyBank > kyBank, os::smart_ptr< nodeGroup > &author)

N.60.1 Detailed Description

Defines basic stream ciphers.

Provides structure to encrypt an XML save file.

Author

Jonathan Bedard

Date

5/19/2016

Bug None

Defines some basic stream ciphers and stream cipher tools for basic encryption.

Author

Jonathan Bedard

Date

5/19/2016

Bug None

Provides functions to save and load XML trees in encrypted files.

Part V

TrusNet Library

Appendix O

Introduction

The TrusNet library contains classes which handle message and node management. The TrusNet library sends and receives messages through the CryptoGateway library. TrusNet also manages connections between nodes, exposing these connections to be used elsewhere. The library provides a few network hooks and means of managing those network types.

O.1 Namespace

TrusNet uses the tnet namespace. The tnet namespace is designed for class, functions and constants related to node management and high-level message passing. TrusNet depends on many of the tools defined in the os and crypto namespaces.

Appendix P

File Index

P.1 File List

Here is a list of all files with brief descriptions:

Appendix Q

File Documentation

Q.1 Connection.cpp File Reference

Implementation of Connection and ConnectionModule.

Q.1.1 Detailed Description

Implementation of Connection and ConnectionModule.

Jonathan Bedard

Date

5/23/2016

Bug No known bugs.

Consult **Connection.h** (p. 243) for details. Note that each module must be independently authenticated, a Connection is merely a collection of ConnectionModules.

Q.2 Connection.h File Reference

Definition of Connection and ConnectionModule.

Classes

- class **tnet::Connection**
- class **tnet::ConnectionModule**

Namespaces

- **tnet**

Variables

- const int **tnet::MAX_CONNECTION_TIMEOUT** =180
- const int **tnet::CONNECTION_TIMEOUT** =30
- const unsigned int **tnet::PEER** =0
- const unsigned int **tnet::BLOCKED** =1
- const unsigned int **tnet::ONLOOKER** =2
- const unsigned int **tnet::PERMISSION_MIN** =PEER
- const unsigned int **tnet::PERMISSION_MAX** =ONLOOKER
- const unsigned int **tnet::PING_MESSAGE** =1
- const unsigned int **tnet::CRYPTO_MESSAGE** =2
- const unsigned int **tnet::MULTICAST_MESSAGE** =3

Q.2.1 Detailed Description

Definition of Connection and ConnectionModule.

Jonathan Bedard

Date

5/23/2016

Bug No known bugs.

Connection and ConnectionModule are secure abstractions of the communication protocols available in TrusNet.

Q.3 LocalIPManager.cpp File Reference

Implementation of LocalIPManager and IPModule.

Q.3.1 Detailed Description

Implementation of LocalIPManager and IPModule.

Jonathan Bedard

Date

5/23/2016

Bug No known bugs.

Consult **LocalIPManager.h** (p. 245) for details. Note that the LocalIPManager will attempt to instantiate both at IPv4 and IPv6 server.

Q.4 LocalIPManager.h File Reference

Definition of LocalIPManager and IPModule.

Classes

- class **tnet::IPModuleListener**
- class **tnet::LocalIPManager**
- class **tnet::IPModule**

Namespaces

- **tnet**

Variables

- const std::string **IPADDRESSFILE** ="IP_Address_Bank"
- const std::string **IDFILE** ="ID_Info_File"
- const unsigned int **IPV4_PORT_DEFAULT** =2690
- const unsigned int **IPV6_PORT_DEFAULT** =2691
- const unsigned int **IP_BLOCKED** =0
- const unsigned int **IP_ACTIVE** =1
- const unsigned int **IP_TIMEOUT** =2

- const std::string **IP_STRING_BLOCKED** ="blocked"
- const std::string **IP_STRING_ACTIVE** ="active"
- const std::string **IP_STRING_TIMEOUT** ="timeout"
- const unsigned int **TIMEOUT_MINIMUM** =1
- const unsigned int **TIMEOUT_DEFAULT** =1209600

Q.4.1 Detailed Description

Definition of LocalIPManager and IPModule.

Jonathan Bedard

Date

5/23/2016

Bug No known bugs.

LocalIPManager and IPModule comprise the IP hook for TrusNet.

Q.4.2 Variable Documentation

const std::string IDFILE ="ID_Info_File"

const unsigned int IP_ACTIVE =1

const unsigned int IP_BLOCKED =0

const std::string IP_STRING_ACTIVE ="active"

const std::string IP_STRING_BLOCKED ="blocked"

const std::string IP_STRING_TIMEOUT ="timeout"

const unsigned int IP_TIMEOUT =2

const std::string IPADDRESSFILE ="IP_Address_Bank"

const unsigned int IPV4_PORT_DEFAULT =2690

const unsigned int IPV6_PORT_DEFAULT =2691

const unsigned int TIMEOUT_DEFAULT =1209600

const unsigned int TIMEOUT_MINIMUM =1

Q.5 MessageTypes.cpp File Reference

Implementation MessageType and MessageRegistry.

Q.5.1 Detailed Description

Implementation MessageType and MessageRegistry.

Jonathan Bedard

Date

5/23/2016

Bug No known bugs.

Consult **MessageTypes.h** (p. 247) for details. Note that the MessageRegistry is a singleton class.

Q.6 MessageTypes.h File Reference

Defines MessageType and MessageRegistry.

Classes

- class **tnet::MessageType**
- class **tnet::Blocked**
- class **tnet::PingMessage**
- class **tnet::ForwardMessage**
- class **tnet::KeyExchangeMessage**
- class **tnet::SigningMessage**
- class **tnet::GatewayExchangeMessage**
- class **tnet::ConfirmErrorMessage**
- class **tnet::BasicErrorMessage**
- class **tnet::TimeoutErrorMessage**
- class **tnet::PermenantErrorMessage**
- class **tnet::NetworkMessage**
- class **tnet::IPAddressMessage**
- class **tnet::LogMessage**
- class **tnet::MessageRegistry**

Namespaces

- **tnet**

Functions

- void **tnet::register_message_types** ()
- os::smart_ptr< MessageRegistry > **tnet::getRegistry** ()
- os::smart_ptr< NetworkMessage > **tnet::NetworkMessageType** ()
- os::smart_ptr< LogMessage > **tnet::LogMessageType** ()

Variables

- const unsigned int **tnet::NUM_MESSAGE_TYPES** =256
- NetworkMessage **tnet::netMess**
- IPAddressMessage **tnet::ipAddressMess**
- LogMessage **tnet::logMes**

Q.6.1 Detailed Description

Defines MessageType and MessageRegistry.

Jonathan Bedard

Date

5/23/2016

Bug No known bugs.

Defines the MessageType base class and the MessageRegistry used to keep track of all of the MessageTypes.

Q.7 NetworkManager.cpp File Reference

Implements class used in network hooks.

Q.7.1 Detailed Description

Implements class used in network hooks.

Jonathan Bedard

Date

5/23/2016

Bug No known bugs.

Consult **NetworkManager.h** (p. 249) for details. This file implements the various classes used in network management.

Q.8 NetworkManager.h File Reference

Defines class used in network hooks.

Classes

- class **tnet::NetworkPoller**
- class **tnet::ConnectionListener**
- class **tnet::NetworkManager**
- class **tnet::NetworkFramework**

Namespaces

- **tnet**

Variables

- const std::string **tnet::ID_INFO_FILE** ="ID_Info_File"
- const std::string **tnet::CONNECTION_PATH** ="Connections"
- const unsigned int **tnet::MAX_NETWORKS** =1
- const unsigned int **tnet::VERSION_1** =1
- const unsigned int **tnet::VERSION_2** =2
- const unsigned int **tnet::CURRENT_VERSION** =VERSION_2

- const unsigned int **tnet::LOCAL_IP_INDEX** =0
- const unsigned int **tnet::MASTER** =0
- const unsigned int **tnet::INDEPENDENT** =1
- const unsigned int **tnet::SERVANT** =2
- const unsigned int **tnet::SLAVE** =3
- const unsigned int **tnet::PRIVILEGE_MIN** =MASTER
- const unsigned int **tnet::PRIVILEGE_MAX** =SLAVE

Q.8.1 Detailed Description

Defines class used in network hooks.

Jonathan Bedard

Date

5/23/2016

Bug No known bugs.

The classes defined in this file are used by network protocol hooks within TrusNet. These base classes allowing things such as polling, event listening and basic connection management.

Q.9 NetworkMaster.cpp File Reference

Implements a NetworkManager driver.

Q.9.1 Detailed Description

Implements a NetworkManager driver.

Jonathan Bedard

Date

5/23/2016

Bug No known bugs.

Consult **NetworkMaster.h** (p. 251) for details. Note that the NetworkMaster class is meant to be extended so that messages can be processed in a meaningful way.

Q.10 NetworkMaster.h File Reference

Defines a NetworkManager driver.

Classes

- class **tnet::NetworkMaster**

Namespaces

- **tnet**

Q.10.1 Detailed Description

Defines a NetworkManager driver.

Jonathan Bedard

Date

5/23/2016

Bug No known bugs.

The class defined in this class provides a class to driver the NetworkManager and receive events from the NetworkManager when messages arrive.

Q.11 TrusNet.h File Reference

Global include file.

Q.11.1 Detailed Description

Global include file.

Author

Jonathan Bedard

Date

5/23/2016

Bug None

This file contains all of the headers in the TrusNet library. Project which depend on the TrusNet library need only include this file.

Q.12 TrusNetComplete.h File Reference

Q.13 TrusNetFunctions.cpp File Reference

Logging for tnet namespace, implementation.

Q.13.1 Detailed Description

Logging for tnet namespace, implementation.

Jonathan Bedard

Date

5/23/2016

Bug No known bugs.

This file contains global functions and variables used for logging in the tnet namespace.

Q.14 TrusNetFunctions.h File Reference

Logging for tnet namespace.

Namespaces

- **tnet**

Functions

- std::ostream & **tnet::tnetout_func** ()
- std::ostream & **tnet::tneterr_func** ()

Variables

- os::smart_ptr< std::ostream > **tnet::tnetout_ptr**
- os::smart_ptr< std::ostream > **tnet::tneterr_ptr**

Q.14.1 Detailed Description

Logging for tnet namespace.

Jonathan Bedard

Date

5/23/2016

Bug No known bugs.

This file contains declarations which are used for logging within the tnet namespace.

Q.15 trusNetPacket.cpp File Reference

Implementation MessageType and MessageRegistry.

Q.15.1 Detailed Description

Implementation MessageType and MessageRegistry.

Jonathan Bedard

Date

5/23/2016

Bug No known bugs.

Consult **trusNetPacket.h** (p. 253) for details. This file implements the details of TrusNet packet management.

Q.16 trusNetPacket.h File Reference

Declaration of various packet classes.

Classes

- class **tnet::byte_array**
- class **tnet::trusPacketAssemblerData**
- class **tnet::TrusPacket**
- class **tnet::trusPacketAssembler**

Namespaces

- **tnet**

Typedefs

- typedef os::smart_ptr< byte_array > **tnet::byte_array_ptr**
- typedef os::smart_ptr< std::vector< os::smart_ptr< crypto::message > > > **tnet::interior↩ MessageList**
- typedef os::smart_ptr< TrusPacket > **tnet::smartTrusPacket**

Enumerations

- enum **tnet::PacketTypes** { **tnet::pck_ERROR**, **tnet::pck_UDP**, **tnet::pck_CHAINED** }

Variables

- const unsigned int **tnet::MAX_TRUS_UDP_SIZE** =470

Q.16.1 Detailed Description

Declaration of various packet classes.

Jonathan Bedard

Date

5/23/2016

Bug No known bugs.

The classes in the module are designed to abstract data packet separation in TrusNet.

Q.17 trusPermissions.cpp File Reference

Future implementation of permission management.

Q.17.1 Detailed Description

Future implementation of permission management.

Jonathan Bedard

Date

5/23/2016

Bug No known bugs.

Current, permissions are managed in the modules utilizing TrusNet. This will change in the future.

Q.18 trusPermissions.h File Reference

Future declaration of permission management.

Namespaces

- **tnet**

Q.18.1 Detailed Description

Future declaration of permission management.

Jonathan Bedard

Date

5/23/2016

Bug No known bugs.

Current, permissions are managed in the modules utilizing TrusNet. This will change in the future.

Part VI

EdisonHAL Library

Appendix R

Introduction

The EdisonHAL library provides a basic wrapper around the hardware IO pins of the Intel Edison platform. In order to allow for easy testing on different platforms, the EdisonHAL library fakes I/O pin functionality when not running on an Intel Edison.

R.1 Unit Testing

Currently, the EdisonHAL library does not define any Unit Tests to be included in an application's battery of unit tests.

R.2 Namespace edison

EdisonHAL establishes the edison namespace. The edison namespace is designed for function calls and basic algorithms relevant to the physical I/O pins on the Intel Edison.

Appendix S

File Index

S.1 File List

Here is a list of all files with brief descriptions:

Appendix T

File Documentation

T.1 EdisonHal.h File Reference

This is a top level header that can be included into other libraries.

Namespaces

- **edison**

Functions

- std::ostream & **edison::edout_func** ()
  Standard out object for edison namespace.
- std::ostream & **edison::ederr_func** ()
  Standard error object for edison namespace.

Variables

- os::smart_ptr< std::ostream > **edison::edout_ptr**
  Standard out pointer for edison namespace.
- os::smart_ptr< std::ostream > **edison::ederr_ptr**
  Standard error pointer for edison namespace.

T.1.1 Detailed Description

This is a top level header that can be included into other libraries.

Author

Jonathan Bedard

Date

5/2/2016

Bug No known bugs.

This header adds the streams for errors and includes the other necessary headers for the hardware header.

T.2 EdisonHalLogging.cpp File Reference

Namespaces

- **edison**

Functions

- std::ostream & **edison::edout_func** ()
  Standard out object for edison namespace.
- std::ostream & **edison::ederr_func** ()
  Standard error object for edison namespace.

T.3 EdisonHalTest.cpp File Reference

Variables

- int **r**

T.3.1 Variable Documentation

int r

T.4 EdisonHalTest.h File Reference

Namespaces

- **edison**

T.5 hardwareEntry.cpp File Reference

Functions

- int **main** (int argc, char ∗∗argv)

T.5.1 Function Documentation

int main (int argc, char ∗∗ argv)

T.6 helloWorldMraa.cpp File Reference

Functions

- int **hello** ()

T.6.1 Function Documentation

int hello ( )

T.7 helloWorldMraa.h File Reference

Functions

- int **hello** ()

T.7.1 Function Documentation

int hello ( )

T.8 motorController.cpp File Reference

Functions

- void **sig_handler** (int signo)
- int **blink** ()
- int **sweep** ()
- int **read** ()

Variables

- int **running** = 0
- static int **iopin**

T.8.1 Function Documentation

int blink ( )

int read ( )

void sig_handler (int signo)

int sweep ( )

[Interesting]

T.8.2 Variable Documentation

int iopin `[static]`

int running = 0

T.9 motorController.h File Reference

Classes

- class **edison::motorController**
- class **edison::joyStick**
- class **edison::robot**

Namespaces

- **edison**

Functions

- int **edison::blink** ()
- int **edison::sweep** ()
- int **edison::read** ()

Part VII

WifiRC Library

Appendix U

Introduction

The WifiRC library defines a set of controllers which utilize TrusNet in-order to demonstrate the capabilities of the system. While there are two main controller types defined, Remote and Base, there are 3 different configurations of the Remote controller type.

U.1 Namespace wrc

The wrc namespace is designed to define the set of controllers and helper classes for these controllers. The wrc namespace does not define the network interfaces used or hardware interfaces, it merely acts as a manager between the network interface and local hardware or UI interface.

Appendix V

File Index

V.1 File List

Here is a list of all files with brief descriptions:

Appendix W

File Documentation

W.1 BaseController.cpp File Reference

Implements the base controller class.

W.1.1 Detailed Description

Implements the base controller class.

Jonathan Bedard

Date

5/23/2016

Bug No known bugs.

The base class is used by the user-interface to connect and manage connections with remote nodes.

W.2 Controller.cpp File Reference

Implements the basic controller class.

W.2.1 Detailed Description

Implements the basic controller class.

Jonathan Bedard

Date

5/23/2016

Bug No known bugs.

The controller class is inherited by the remote and base. It provides a basic interface for user management, messaging and file saving.

W.3 Controller.h File Reference

Definition of various controller types.

Classes

- class **wrc::Controller**
- class **wrc::RemoteController**
- class **wrc::BaseController**

Namespaces

- **wrc**

Variables

- const unsigned int **wrc::IDENTIFIER** =1
- const unsigned int **wrc::AUTO_SELECT** =0
- const unsigned int **wrc::PREFFERED_AUTO** =1
- const unsigned int **wrc::LIST_SELECTION** =2
- const unsigned int **wrc::PREFFERED_LIST** =3
- const unsigned int **wrc::ONE_SELECTION** =4
- const unsigned int **wrc::DEFAULT_CHOICE** =0
- const unsigned int **wrc::LIST_CHOICE** =1
- const unsigned int **wrc::PREFERED_CHOICE** =2
- const unsigned int **wrc::BLOCKED_CHOICE** =3

- const std::string **wrc::AUTO_SELECT_STRING** ="Auto-Select"
- const std::string **wrc::PREFFERED_AUTO_STRING** ="Prefered_Auto"
- const std::string **wrc::LIST_SELECTION_STRING** ="List_Selection"
- const std::string **wrc::PREFFERED_LIST_STRING** ="Prefered_List"
- const std::string **wrc::ONE_SELECTION_STRING** ="One_Selection"
- const std::string **wrc::DEFAULT_CHOICE_STRING** ="Default"
- const std::string **wrc::LIST_CHOICE_STRING** ="List"
- const std::string **wrc::PREFERED_CHOICE_STRING** ="Prefered"
- const std::string **wrc::BLOCKED_CHOICE_STRING** ="Blocked"
- const std::string **wrc::AUTO_SELECT_UI** ="Auto-Select"
- const std::string **wrc::PREFFERED_AUTO_UI** ="Prefered Auto"
- const std::string **wrc::LIST_SELECTION_UI** ="List Selection"
- const std::string **wrc::PREFFERED_LIST_UI** ="Prefered List"
- const std::string **wrc::ONE_SELECTION_UI** ="One Selection"
- const std::string **wrc::DEFAULT_CHOICE_UI** ="Unselected"
- const std::string **wrc::LIST_CHOICE_UI** ="Selected"
- const std::string **wrc::PREFERED_CHOICE_UI** ="Primary"
- const std::string **wrc::BLOCKED_CHOICE_UI** ="Blocked"

W.3.1 Detailed Description

Definition of various controller types.

Jonathan Bedard

Date

5/23/2016

Bug No known bugs.

The remote and base controllers are defined in this class. Note that both forms of controller inherit from a shared controller class.

W.4 controllerMessageType.cpp File Reference

Implements the message types for WifiRC.

W.4.1 Detailed Description

Implements the message types for WifiRC.

Jonathan Bedard

Date

5/23/2016

Bug No known bugs.

W.5 controllerMessageType.h File Reference

Defines the message types for WifiRC.

Classes

- class **wrc::MotorControlMessage**
- class **wrc::ConnectionListMessage**

Namespaces

- **wrc**

Variables

- MotorControlMessage **wrc::motorControlMessageType**
- ConnectionListMessage **wrc::connectionListMessage**

W.5.1 Detailed Description

Defines the message types for WifiRC.

Jonathan Bedard

Date

5/23/2016

Bug No known bugs.

This file defines the MotorControlMessage and the ConnectionListMessage, two message types used by WifiRC to communicate between two nodes.

W.6 Converter.h File Reference

Defines a set of control basics.

Classes

- class **wrc::GenControls**
- class **wrc::RCControls**

Namespaces

- **wrc**

W.6.1 Detailed Description

Defines a set of control basics.

Jonathan Bedard

Date

5/23/2016

Bug No known bugs.

This file defines a general control framework and a control framework used for controlling an RC car. Currently, the WifiRC library does not use these interfaces.

W.7 GenControls.cpp File Reference

Defines the general control framework.

W.7.1 Detailed Description

Defines the general control framework.

Jonathan Bedard

Date

5/23/2016

Bug No known bugs.

Note that this particular framework is currently not used in the WifiRC library.

W.8 RCControls.cpp File Reference

Defines the control framework for an RC car.

W.8.1 Detailed Description

Defines the control framework for an RC car.

Jonathan Bedard

Date

5/23/2016

Bug No known bugs.

Note that this particular framework is currently not used in the WifiRC library.

W.9 RemoteController.cpp File Reference

Implements the remote controller class.

W.9.1 Detailed Description

Implements the remote controller class.

Jonathan Bedard

Date

5/23/2016

Bug No known bugs.

The remote class is used by headless applications to run or simulate IOT devices.

W.10 siblingStruct.cpp File Reference

Implementation of the siblingStruct class.

W.10.1 Detailed Description

Implementation of the siblingStruct class.

Jonathan Bedard

Date

5/23/2016

Bug No known bugs.

This file implements the siblingStruct class used to hold the state of a remote node.

W.11 siblingStruct.h File Reference

Definition of the siblingStruct class.

Classes

- struct **wrc::siblingData**

Namespaces

- **wrc**

W.11.1 Detailed Description

Definition of the siblingStruct class.

Jonathan Bedard

Date

5/23/2016

Bug No known bugs.

This file defines the siblingStruct class used to hold the state of a remote node, particularly the node type, connection list and data intrinsics.

W.12 WifiRC.h File Reference

Global include file.

W.12.1 Detailed Description

Global include file.

Author

Jonathan Bedard

Date

5/23/2016

Bug None

This file contains all of the headers in the WifiRC library. Project which depend on the WifiRC library need only include this file.

W.13 WifiRCComplete.h File Reference

W.14 wifiRCLogging.cpp File Reference

Logging for wrc namespace, implementation.

W.14.1 Detailed Description

Logging for wrc namespace, implementation.

Jonathan Bedard

Date

5/23/2016

Bug No known bugs.

This file contains global functions and variables used for logging in the wrc namespace.

W.15 wifiRCLogging.h File Reference

Logging for wrc namespace.

Namespaces

- **wrc**

Functions

- std::ostream & **wrc::wrcout_func** ()
- std::ostream & **wrc::wrcerr_func** ()

Variables

- os::smart_ptr< std::ostream > **wrc::wrcout_ptr**
- os::smart_ptr< std::ostream > **wrc::wrcerr_ptr**

W.15.1 Detailed Description

Logging for wrc namespace.

Jonathan Bedard

Date

5/23/2016

Bug No known bugs.

This file contains declarations which are used for logging within the wrc namespace.

Part VIII

glGraphics Library

Appendix X

Introduction

The glGraphics library contains cross-platform graphics tools. glGraphics visualizes through openGL, allowing graphics to appear consistent across all systems. Furthermore, glGraphics provides an interface to preform unit tests on the interfaces constructed through the library.

X.1 Unit Testing

glGraphics defines a number of testing primitives and, most notably, a headless UI driver, to allow for the creation of headless UI unit tests to confirm the functionality of certain graphical components. It is expected that libraries and executables which utilize this library will also use it's unit testing architecture.

X.2 Namespace

glGraphics uses the gl namespace. The gl namespace is designed for class, functions and constants related to visualization through openGL. glGraphics depends on many of the tools defined in the os namespace.

Appendix Y

File Index

Y.1 File List

Here is a list of all files with brief descriptions:

Appendix Z

File Documentation

Z.1 freeglut.h File Reference

Implements the UI testing framework.

Z.1.1 Detailed Description

Implements the UI testing framework.

Author

Jonathan Bedard

Date

5/20/2016

Bug None

Conditional freeglut include for Windows machines.

Z.2 freeglut_ext.h File Reference

Implements the UI testing framework.

Z.2.1 Detailed Description

Implements the UI testing framework.

Author

Jonathan Bedard

Date

5/20/2016

Bug None

Conditional freeglut_ext include for Windows machines.

Z.3 freeglut_std.h File Reference

Implements the UI testing framework.

Z.3.1 Detailed Description

Implements the UI testing framework.

Author

Jonathan Bedard

Date

5/20/2016

Bug None

Conditional freeglut_std include for Windows machines.

Z.4 glCheckbox.cpp File Reference

Implements a checkbox and checkboxGroup.

Z.4.1 Detailed Description

Implements a checkbox and checkboxGroup.

Author

Jonathan Bedard

Date

5/23/2016

Bug None

Consult **glCheckbox.h** (p. 285) for details. Note that the graphics class implemented here all require a frame in their constructor.

Z.5 glCheckbox.h File Reference

Defines a checkbox and checkboxGroup.

Classes

- class **gl::checkbox**
- class **gl::checkboxGroup**

Namespaces

- **gl**

Enumerations

- enum **gl::checkboxGroupType** { **gl::checkbox_noRestriction** =0, **gl::checkbox_single**, **gl↩ ::checkbox_chooseOne**, **gl::checkbox_chooseSome** }

Z.5.1 Detailed Description

Defines a checkbox and checkboxGroup.

Author

Jonathan Bedard

Date

5/23/2016

Bug None

This file defines a checkbox and a checkboxGroup. Both these classes are meant to be visualized.

Z.6 glColors.cpp File Reference

Implements the color list.

Z.6.1 Detailed Description

Implements the color list.

Author

Jonathan Bedard

Date

5/23/2016

Bug None

Consult **glColors.h** (p. 286) for details. Implements each of the defined colors.

Z.7 glColors.h File Reference

Defines a number of colors.

Namespaces

- **gl**
- **gl::col**

Variables

- const color **gl::col::red**
- const color **gl::col::yellow**
- const color **gl::col::green**
- const color **gl::col::blue**
- const color **gl::col::black**
- const color **gl::col::brown**
- const color **gl::col::white**
- const color **gl::col::clear**
- const color **gl::col::clickedBlue**
- const color **gl::col::textboxBlue**
- const color **gl::col::darkGray**
- const color **gl::col::gray**
- const color **gl::col::inactiveGray**
- const color **gl::col::lightGray**
- const color **gl::col::overlayGray**
- const color **gl::col::overlayBlack**
- const color **gl::col::darkGreen**

Z.7.1 Detailed Description

Defines a number of colors.
Author

Jonathan Bedard

Date

5/23/2016

Bug None

Each of the colors defined in this file is held in a doubly nested namespace. The colors defined in this file are immutable.

Z.8 glContainers.cpp File Reference

Implements various graphics containers.

Z.8.1 Detailed Description

Implements various graphics containers.

Author

Jonathan Bedard

Date

5/23/2016

Bug None

Consult **glContainers.h** (p. 288) for details. The containers defined include, most notably, a box, drop-down menu and scroll area.

Z.9 glContainers.h File Reference

Defines a number graphics containers.

Classes

- class **gl::box**
- class **gl::wrappableBox**
- class **gl::fileBar**
- class **gl::barGroup**
- class **gl::scrollbar**
- class **gl::scrollArea**
- class **gl::entireFormScroll**

Namespaces

- **gl**

Enumerations

- enum **gl::scrollbarDir** { **gl::scrollbar_vertical** =0, **gl::scrollbar_horizontal** }

Z.9.1 Detailed Description

Defines a number graphics containers.

Author

Jonathan Bedard

Date

5/23/2016

Bug None

The classes defined in this file extend the **gl::frame** (p. **??**). Each of these containers has certain unique qualities.

Z.10 glForm.cpp File Reference

Implements various forms and drivers.

Z.10.1 Detailed Description

Implements various forms and drivers.

Author

Jonathan Bedard

Date

5/24/2016

Bug None

Consult **glForm.h** (p. 290) for details. Note that each driver only draws a single form, but these forms are defined in a tree hierarchy.

Z.11 glForm.h File Reference

Defines forms and UI drivers.

Classes

- class **gl::form**
- class **gl::form3d**
- class **gl::baseUIDriver**
- class **gl::UIDriver**
- class **gl::testingDriver**

Namespaces

- **gl**

Enumerations

- enum **gl::resizePolicyEnum** {
  gl::resize_none =0, **gl::resize_master**, **gl::resize_minimum**, **gl::resize_maximum**,
  gl::resize_bounded, **gl::resize_lock**, **gl::resize_custom** }

Variables

- const int **gl::form_width_minimum** =116
- const int **gl::form_height_minimum** =0

Z.11.1 Detailed Description

Defines forms and UI drivers.

Author

Jonathan Bedard

Date

5/24/2016

Bug None

Defines both a 3-d and 2-d form, as well as standard and testing drivers for displaying and testing form hierarchies.

Z.12 glFrame.cpp File Reference

Implements the graphics element and frame.

Z.12.1 Detailed Description

Implements the graphics element and frame.
Author

Jonathan Bedard

Date

5/24/2016

Bug None

Consult **glFrame.h** (p. 291) for details. Many of the functions defined here are extended in the many classes which inherit from these base classes.

Z.13 glFrame.h File Reference

Defines the graphics element and frame.

Classes

- struct **gl::color**
- class **gl::clickedListener**
- class **gl::pressedListener**

- class **gl::depressedListener**
- class **gl::enterListener**
- class **gl::resizedListener**
- class **gl::clickedFunction**
- class **gl::clickedFunctionVoid**
- class **gl::pressedFunction**
- class **gl::pressedFunctionVoid**
- class **gl::depressedFunction**
- class **gl::depressedFunctionVoid**
- class **gl::enterFunction**
- class **gl::enterFunctionVoid**
- class **gl::resizeFunction**
- class **gl::resizeFunctionVoid**
- class **gl::element**
- class **gl::frame**

Namespaces

- **gl**

Typedefs

- typedef void(∗ **gl::elementHandler**) (os::smart_ptr< element > elm)
- typedef void(∗ **gl::elementHandler_void**) (os::smart_ptr< element > elm, void ∗vptr)

Z.13.1 Detailed Description

Defines the graphics element and frame.

Author

Jonathan Bedard

Date

5/24/2016

Bug None

The classes defined in this header are the basic classes all of the graphics library builds off of. These include a basic event framework, the base element class and the frame class, which defines an element which holds other elements.

Z.14 glInput.cpp File Reference

Implements the graphics input classes.

Z.14.1 Detailed Description

Implements the graphics input classes.

Author

Jonathan Bedard

Date

5/24/2016

Bug None

Consult **glInput.h** (p. 293) for details. Along with class definitions, a number of global constants are implemented in this file.

Z.15 glInput.h File Reference

Defines a number of input classes.

Classes

- class **gl::key**
- class **gl::mouseListener**

- class **gl::keyboardListener**
- class **gl::globalMouseListener**
- class **gl::globalKeyboardListener**
- class **gl::mouse**
- class **gl::keyboard**

Namespaces

- **gl**

Enumerations

- enum **gl::elementDepth** { **gl::defaultDepth** =0, **gl::bottomDepth**, **gl::middleDepth**, **gl::top↩ Depth** }
- enum **gl::keyType** { **gl::key_standard** =0, **gl::key_special** =1 }

Variables

- const key **gl::ESCAPE**
- const key **gl::ENTER**
- const key **gl::DELETE_KEY**
- const key **gl::BACKSPACE**
- const key **gl::TAB**
- const key **gl::SPACE**
- const key **gl::ARROWUP**
- const key **gl::ARROWDOWN**
- const key **gl::ARROWLEFT**
- const key **gl::ARROWRIGHT**
- const int **gl::MOUSE_DOWN** =0
- const int **gl::MOUSE_UP** =1
- const int **gl::MOUSE_RIGHT_BUTTON** =2
- const int **gl::MOUSE_SCROLLBUTTON** =1

- const int **gl::MOUSE_LEFT_BUTTON** =0
- const int **gl::MOUSE_SCROLLUP** =3
- const int **gl::MOUSE_SCROLLDOWN** =4

Z.15.1 Detailed Description

Defines a number of input classes.
Author

Jonathan Bedard

Date

5/25/2016

Bug None

The classes and objects defined in this file are used for gathering user input, either from the mouse or keyboard.

Z.16 glLabel.cpp File Reference

Implements text visualization tools.

Z.16.1 Detailed Description

Implements text visualization tools.
Author

Jonathan Bedard

Date

5/25/2016

Bug None

Consult **glLabel.h** (p. 296) for details. Because of the limitations of openGL, there are only a few fonts available.

Z.17 glLabel.h File Reference

Defines text visualization classes.

Classes

- class **gl::font**
- class **gl::TimesRomanStroke**
- class **gl::TimesMonoRomanStroke**
- class **gl::TimesRoman**
- class **gl::Helvetica**
- class **gl::label**
- class **gl::button**
- class **gl::arrowButton**
- class **gl::activeDisplayButton**
- class **gl::activeDisplayArrowButton**

Namespaces

- **gl**

Enumerations

- enum **gl::lateralTextLayout** { **gl::layout_right** =0, **gl::layout_center**, **gl::layout_left** }
- enum **gl::verticalTextLayout** { **gl::layout_top** =0, **gl::layout_middle**, **gl::layout_bottom** }
- enum **gl::arrowType** { **gl::arrow_up** =0, **gl::arrow_right**, **gl::arrow_down**, **gl::arrow_left** }

Functions

- os::smart_ptr< std::string > **gl::displayAssembleString** (std::string str, int &length)
- void **gl::drawText** (double x, double y, std::string str, const color &c, const font ∗_font, lateral←
TextLayout layout=layout_left)
- void **gl::drawText** (double x, double y, const char ∗str, const color &c, const font ∗_font, lateral←
TextLayout layout=layout_left)

- void **gl::drawText** (double x, double y, os::smart_ptr< std::string > strArr, int length, const color &c, const font ∗_font, lateralTextLayout layout)
- double **gl::textHeight** (os::smart_ptr< std::string > strArr, int length, const font ∗_font)
- double **gl::textHeight** (std::string str, const font ∗_font)
- double **gl::textWidth** (os::smart_ptr< std::string > strArr, int length, const font ∗_font)
- double **gl::textWidth** (std::string str, const font ∗_font)
- std::string **gl::textChop** (std::string str, double length_bound, const font ∗_font)

Z.17.1 Detailed Description

Defines text visualization classes.

Author

Jonathan Bedard

Date

5/25/2016

Bug None

The classes and functions defined in this file allow for rendering text in a form. This file also defines a number of basic buttons.

Z.18 glLibrary.h File Reference

Unified graphics library header.

Z.18.1 Detailed Description

Unified graphics library header.

Author

Jonathan Bedard

Date

5/25/2016

Bug None

Includes all of the required graphics files for the glGraphics library.

Z.19 glLogging.cpp File Reference

Logging for gl namespace, implementation.

Z.19.1 Detailed Description

Logging for gl namespace, implementation.

Jonathan Bedard

Date

2/15/2016

Bug No known bugs.

This file contains global functions and variables used for logging in the gl namespace.

Z.20 glLogging.h File Reference

Logging for gl namespace.

Namespaces

- **gl**

Functions

- std::ostream & **gl::glout_func** ()
- std::ostream & **gl::glerr_func** ()

Variables

- os::smart_ptr< std::ostream > **gl::glout_ptr**
- os::smart_ptr< std::ostream > **gl::glerr_ptr**

Z.20.1 Detailed Description

Logging for gl namespace.

Jonathan Bedard

Date

1/30/2016

Bug No known bugs.

This file contains declarations which are used for logging within the gl namespace.

Z.21 glOSFunctions.cpp File Reference

Includes the correct glOSFuncitons cpp file.

Z.21.1 Detailed Description

Includes the correct glOSFuncitons cpp file.

Author

Jonathan Bedard

Date

5/20/2016

Bug None

Z.22 glOSFunctions.h File Reference

Includes the correct glOSFuncitons header.

Namespaces

- **gl**

Variables

- const double **gl::PI** =3.14159265
- const double **gl::DEG_RAD** =(PI/180)
- const double **gl::RAD_DEG** =(180/PI)

Z.22.1 Detailed Description

Includes the correct glOSFuncitons header.

Author

Jonathan Bedard

Date

5/20/2016

Bug None

Z.23 glPopUp.cpp File Reference

Implements a checkbox and checkboxGroup.

Z.23.1 Detailed Description

Implements a checkbox and checkboxGroup.

Author

Jonathan Bedard

Date

5/25/2016

Bug None

Consult **glPopUp.h** (p. 301) for details. This file also implements a basic navigation form, which allows for users to go "back" to a previous form.

Z.24 glPopUp.h File Reference

Defines a number of form types.

Classes

- class **gl::popUp**
- class **gl::singleButtonPopUp**
- class **gl::navForm**

Namespaces

- **gl**

Z.24.1 Detailed Description

Defines a number of form types.

Author

Jonathan Bedard

Date

5/25/2016

Bug None

The classes defined here extend the form class. Most notably, this file defines a "pop-up" class which allows for a temporary form on-top of the primary form.

Z.25 glTest.cpp File Reference

Defines glGraphics test suite.

Z.25.1 Detailed Description

Defines glGraphics test suite.

Author

Jonathan Bedard

Date

5/19/2016

Bug None

Defines the test suite for the glGraphics library.

Z.26 glTest.h File Reference

Testing suite for the glGraphics library.

Z.26.1 Detailed Description

Testing suite for the glGraphics library.

Author

Jonathan Bedard

Date

5/19/2016

Bug None

Declares the test suite for the glGraphics library. This suite can be added to the test battery for an application

Z.27 glTestingFrame.cpp File Reference

Implements the UI testing framework.

Z.27.1 Detailed Description

Implements the UI testing framework.

Author

Jonathan Bedard

Date

5/19/2016

Bug None

Implements tools used to preform unit tests on the graphics framework.

Z.28 glTestingFrame.h File Reference

Defines the UI testing framework.

Classes

- class **test::singleUITest**
- class **test::singleUIFunctionTest**
- class **test::UITestSuite**
- class **test::resizedTestListener**
- class **test::clickedTestListener**
- class **test::pressedTestListener**
- class **test::depressedTestListener**
- class **test::enterTestListener**
- class **test::testForm**
- class **test::textboxForm**
- class **test::checkboxForm**

Namespaces

- **test**
- **test::macro**

Functions

- void **test::setUpGraphicsTest** ()
- void **test::teardownGraphicsTest** ()
- os::smart_ptr< **gl::testingDriver** > **test::getTestDriver** () throw (os::smart_ptr<std::exception>)
- void **test::macro::moveMouseTo** (int xPos, int yPos)
- void **test::macro::moveMouseTo** (const **gl::element** &elm)
- void **test::macro::moveMouseTo** (std::string str)
- void **test::macro::mousePress** ()
- void **test::macro::mouseRelease** ()
- void **test::macro::mouseClick** ()
- void **test::macro::mouseClick** (**gl::element** &elm)
- void **test::macro::mouseClick** (std::string str)
- void **test::macro::keyboardClick** (const **gl::key** &_key)
- void **test::macro::keyboardDown** (const **gl::key** &_key)
- void **test::macro::keyboardUp** (const **gl::key** &_key)
- void **test::macro::keyboardType** (const std::string &str)
- void **test::macro::clickFileBar** (os::smart_ptr< **gl::barGroup** > target)
- os::smart_ptr< os::unsortedList< **gl::element** > > **test::macro::searchUIBy** (std::string str)

Z.28.1 Detailed Description

Defines the UI testing framework.

Author

Jonathan Bedard

Date

5/19/2016

Bug None

Defines tools used to preform unit tests on the graphics framework.

Z.29 glTextbox.cpp File Reference

Implements a text-box.

Z.29.1 Detailed Description

Implements a text-box.

Author

Jonathan Bedard

Date

5/25/2016

Bug None

Consult **glTextbox.h** (p. 305) for details. The textbox implemented here does not currently support copying and pasting.

Z.30 glTextbox.h File Reference

Defines a text-box.

Classes

- class **gl::textbox**

Namespaces

- **gl**

Z.30.1 Detailed Description

Defines a text-box.

Author

Jonathan Bedard

Date

5/25/2016

Bug None

The text-box class allows a user to enter a string into the element and for this string to be accessed for some other use

Z.31 glut.h File Reference

Windows glut header.

Z.31.1 Detailed Description

Windows glut header.

Author

None

Date

5/20/2016

Bug None

Z.32 glut_w.h File Reference

Z.33 image_DXT.cpp File Reference

simple DXT compression / decompression code

Z.33.1 Detailed Description

simple DXT compression / decompression code

Author

Jonathan Dummer

Date

7/31/2007

Bug None

Modified for usage in glGraphics by Jonathan Bedard.

Z.34 image_DXT.h File Reference

simple DXT compression / decompression code

Z.34.1 Detailed Description

simple DXT compression / decompression code

Author

Jonathan Dummer

Date

7/31/2007

Bug None

Modified for usage in glGraphics by Jonathan Bedard.

Z.35 image_helper.cpp File Reference

Image helper functions.

Z.35.1 Detailed Description

Image helper functions.

Author

Jonathan Dummer

Date

7/31/2007

Bug None

Modified for usage in glGraphics by Jonathan Bedard.

Z.36 image_helper.h File Reference

Image helper functions.

Z.36.1 Detailed Description

Image helper functions.

Author

Jonathan Dummer

Date

7/31/2007

Bug None

Modified for usage in glGraphics by Jonathan Bedard.

Z.37 osGraphics.h File Reference

OS specific openGL functions.

Z.37.1 Detailed Description

OS specific openGL functions.

Author

Jonathan Bedard

Date

5/20/2016

Bug None

Includes a the correct osGraphics header for a specific operating system.

Z.38 SOIL.cpp File Reference

Simple OpenGL Image Library.

Z.38.1 Detailed Description

Simple OpenGL Image Library.

Author

Jonathan Dummer

Date

7/26/2007

Bug None

Public Domain using Sean Barret's stb_image as a base

```
Thanks to:
          Sean Barret - for the awesome stb_image
          Dan Venkitachalam - for finding some non-compliant DDS files, and patching some explicit casts
          everybody at gamedev.net
```

Edited by Jonathan Bedard for compatibility with glGraphics C++ library

Z.39 SOIL.h File Reference

Simple OpenGL Image Library.

Z.39.1 Detailed Description

Simple OpenGL Image Library.

Author

Jonathan Dummer

Date

7/26/2007

Bug None

A tiny c library for uploading images as textures into OpenGL. Also saving and loading of images is supported.

Edited by Jonathan Bedard for compatibility with glGraphics C++ library

Z.40 stb_image_aug.cpp File Reference

JPEG/PNG reader implementation.

Z.40.1 Detailed Description

JPEG/PNG reader implementation.

Author

Jonathan Dummer

Date

7/26/2007

Bug None

Edited by Jonathan Bedard for compatibility with glGraphics C++ library

Z.41 stb_image_aug.h File Reference

JPEG/PNG reader header.

Z.41.1 Detailed Description

JPEG/PNG reader header.

Author

Jonathan Dummer

Date

7/26/2007

Bug None

Edited by Jonathan Bedard for compatibility with glGraphics C++ library

Z.42 stbi_DDS_aug.h File Reference

DDS loading support.

Z.42.1 Detailed Description

DDS loading support.

DDS file support.

Author

Jonathan Dummer

Date

7/26/2007

Bug None

Edited by Jonathan Bedard for compatibility with glGraphics C++ library

Author

Jonathan Dummer

Date

7/26/2007

Bug None

Edited by Jonathan Bedard for compatibility with glGraphics C++ library

Z.43 stbi_DDS_aug_c.h File Reference

Z.44 textureManager.cpp File Reference

Implements the texture manager.

Z.44.1 Detailed Description

Implements the texture manager.

Author

Jonathan Bedard

Date

5/25/2016

Bug None

Consult **textureManager.h** (p. 312) for details. The textureManager is a singleton class, and can only be instantiated once.

Z.45 textureManager.h File Reference

Defines a texture-manager.

Classes

- struct **gl::texture_data**
- class **gl::textureManager**
- class **gl::imageElement**

Namespaces

- **gl**

Functions

- GLuint **gl::load_texture** (std::string file_name)
- os::smart_ptr< textureManager > **gl::globalTextureManager** ()
- void **gl::deleteTextures** ()

Z.45.1 Detailed Description

Defines a texture-manager.

Author

Jonathan Bedard

Date

5/25/2016

Bug None

The texture manager allows for global management of graphics textures used in a graphics program.

Z.46 unix_glOSFunctions.cpp File Reference

Unix specific graphics functions.

Z.46.1 Detailed Description

Unix specific graphics functions.

Author

Jonathan Bedard

Date

5/20/2016

Bug None

Implements a number of graphics functions uniquely for Unix.

Z.47 unix_glOSFunctions.h File Reference

Unix specific graphics functions.

Namespaces

- **gl**

Functions

- void **gl::glSetSource** (char ∗source_string)
- std::string **gl::glGetSource** ()
- std::string **gl::glGetExecutable** ()
- int **gl::glGetSourceDepth** ()
- bool **gl::glIsBase** (std::string loc)
- void **gl::glTestCreateFolder** (std::string n)
- bool **gl::gl_is_directory** (std::string file)
- std::string ∗ **gl::gl_list_files** (std::string directory, int ∗len)
- std::string **gl::gl_extract_name** (std::string full_path)
- void **gl::gl_delete_file** (std::string path)
- FILE ∗ **gl::fopen_s** (FILE ∗∗f, const char ∗name, const char ∗read_type)

Variables

- const std::string **gl::DEFAULT_LOC** ="/home"

Z.47.1 Detailed Description

Unix specific graphics functions.

Author

Jonathan Bedard

Date

5/20/2016

Bug None

Defines a number of graphics functions uniquely for Unix.

Z.48 unix_osGraphics.h File Reference

Unix specific openGL functions.

Functions

- static void **glutLeaveMainLoop** ()
- static int **glutStrokeHeight** (const void ∗ptr)
- static int **glutBitmapHeight** (const void ∗ptr)

Z.48.1 Detailed Description

Unix specific openGL functions.

Author

Jonathan Bedard

Date

5/20/2016

Bug None

Includes a number of headers specific to certain operating systems needed for openGL.

Z.48.2 Function Documentation

static int glutBitmapHeight (const void * ptr) `[static]`

static void glutLeaveMainLoop ( ) `[static]`

static int glutStrokeHeight (const void * ptr) `[static]`

Z.49 win_freeglut.h File Reference

Windows freeglut header.

Z.49.1 Detailed Description

Windows freeglut header.

Author

Jonathan Bedard

Date

5/20/2016

Bug None

Z.50 win_freeglut_ext.h File Reference

Windows freeglut_ext header.

Z.50.1 Detailed Description

Windows freeglut_ext header.

Author

Jonathan Bedard

Date

5/20/2016

Bug None

Z.51 win_freeglut_std.h File Reference

Windows freeglut_std header.

Z.51.1 Detailed Description

Windows freeglut_std header.

Author

Jonathan Bedard

Date

5/20/2016

Bug None

Z.52 win_glOSFunctions.cpp File Reference

Windows specific graphics functions.

Z.52.1 Detailed Description

Windows specific graphics functions.

Author

Jonathan Bedard

Date

5/20/2016

Bug None

Implements a number of graphics functions uniquely for Windows.

Z.53 win_glOSFunctions.h File Reference

Windows specific graphics functions.

Z.53.1 Detailed Description

Windows specific graphics functions.

Author

Jonathan Bedard

Date

5/20/2016

Bug None

Defines a number of graphics functions uniquely for Windows.

Z.54 win_osGraphics.h File Reference

Windows specific openGL functions.

Z.54.1 Detailed Description

Windows specific openGL functions.

Author

Jonathan Bedard

Date

5/25/2016

Bug None

Includes a number of headers specific to certain operating systems needed for openGL.

Part IX

CryptoLogin Library

Appendix AA

Introduction

The CryptoLogin library contains a series of forms and support class used to open, manage and edit users. These forms utilize the Datastructures, osMechanics, CryptoGateway and glGraphics libraries.

AA.1 Unit Testing

Currently, the CryptoLogin library is not tested. In the future, the CryptoLogin will leverage the testing functionality of the glGraphics library to preform basic testing on the forms through the headless testing interface in glGraphics.

AA.2 Namespace login

This namespace contains forms used for both logging in and visualizing the basics of a user. Note that the initial entry form is a template class so that the login namespace can open any form when logging in. It is expected that the subsequent form will provide opportunities for the user to open some of the user editing forms provided by the login namespace.

Appendix AB

File Index

AB.1 File List

Here is a list of all files with brief descriptions:

Appendix AC

File Documentation

AC.1 createUserForm.cpp File Reference

Implements the user creation form.

AC.1.1 Detailed Description

Implements the user creation form.

Implements key management form.

Author

Jonathan Bedard

Date

4/21/2016

Bug None

Implements the user creation form. Consult **createUserForm.h** (p. 325) for details.

Author

Jonathan Bedard

Date

5/5/2016

Bug None

Implements the key management form, which can be accessed from the secure side side of the gateway

AC.2 createUserForm.h File Reference

Classes

- class **login::createUser**

Namespaces

- **login**

AC.3 cryptoLogin.h File Reference

All login header files.

AC.3.1 Detailed Description

All login header files.

Author

Jonathan Bedard

Date

4/12/2016

Bug None

Includes all login header files to be used outside the library.

AC.4 cryptoLoginLog.h File Reference

Logging for login namespace.

Namespaces

- **login**

Functions

- std::ostream & **login::loginout_func** ()
  Standard out object for login namespace.
- std::ostream & **login::loginerr_func** ()
  Standard error object for login namespace.

Variables

- os::smart_ptr< std::ostream > **login::loginout_ptr**
  Standard out pointer for login namespace.
- os::smart_ptr< std::ostream > **login::loginerr_ptr**
  Standard error pointer for login namespace.

AC.4.1 Detailed Description

Logging for login namespace.

Author

Jonathan Bedard

Date

4/12/2016

Bug None

This file contains declarations which are used for logging within the login namespace.

AC.5 keyGenerationPopUp.cpp File Reference

Key generation pop-up.

AC.5.1 Detailed Description

Key generation pop-up.

Author

Jonathan Bedard

Date

5/8/2016

Bug None

Implements the key-generation pop-up which provides a UI while public keys are being generated.

AC.6 keyGenerationPopUp.h File Reference

Various crypto-graphic pop-ups.

Classes

- class **login::pulblicKeyPopUp**
  Used when generating keys.
- class **login::userLoadingPopUp**
  Used when loading the user.

Namespaces

- **login**

AC.6.1 Detailed Description

Various crypto-graphic pop-ups.

Author

Jonathan Bedard

Date

4/21/2016

Bug None

Provides two pop-up forms used by the CryptoLogin library. One is used when generating public keys and another is used when loading user data.

AC.7 listUsersForm.cpp File Reference

AC.8 listUsersForm.h File Reference

Classes

- class **login::userFrame**

  Defines a user display frame This frame displays basic user data before a user is logged in. This data is defined from the login meta-data.

- class **login::listUsers**

  *List-user form A navigation form listing all users associated with a particular **loginMetaData** (p. **??**) class.*

Namespaces

- **login**

AC.9 loginMain.cpp File Reference

Miscellaneous implementation for login namespace.

AC.9.1 Detailed Description

Miscellaneous implementation for login namespace.

Author

Jonathan Bedard

Date

3/7/2016

Bug None

Implements a number of functions and objects used in the login namespaces. In particular, this file implements the logging structures for the login namespace.

AC.10 loginMain.h File Reference

Entry login form.

Classes

- class **login::mainLogin**< **nextForm** >
  Login form.

Namespaces

- **login**

AC.10.1 Detailed Description

Entry login form.

Author

Jonathan Bedard

Date

4/20/2016

Bug None

Provides a template class for applications which would like to include a password to log in. Will allow the user to create new users and define public keys for those users as well.

AC.11 loginMetaData.cpp File Reference

AC.12 loginMetaData.h File Reference

Impliments login-form meta-data.

Classes

- struct **login::userNode**
  User node.
- class **login::loginMetaData**
  Login meta-data class.

Namespaces

- **login**

Variables

- const char $*$ **login::META_FILE**
  Meta-data file name.
- const char $*$ **login::USERS_FOLDER**
  Folder holding user data.

AC.12.1 Detailed Description

Impliments login-form meta-data.

Contains meta-data for login form.

Author

Jonathan Bedard

Date

4/18/2016

Bug None

Impliments the login meta-data class. Consult **loginMetaData.h** (p. 330) for details.

Author

Jonathan Bedard

Date

4/13/2016

Bug None

Provides meta-data to the login form. This class has default states, so if a custom meta-data class is not passed to the login form, a default one will be created.

AC.13 manageKeysForm.cpp File Reference

AC.14 manageKeysForm.h File Reference

Classes

- class **login::publicKeyTypeFrame**
- class **login::userSettingsForm**
- class **login::keyBankForm**

Namespaces

- **login**

Part X

RemoteMain Executable

Appendix AD

Introduction

The RemoteMain executable runs a headless controller to demonstrate TrusNet's functionality. As an executable, RemoteMain also defines a testing battery. In addition, RemoteMain defines the SpeedProfiling executable used to test performance within the TrusNet library.

AD.1 Namespace

RemoteMain does not define a namespace. Rather, RemoteMain essentially acts as a wrapper around WifiRC's RemoteController class.

Appendix AE

File Index

AE.1 File List

Here is a list of all files with brief descriptions:

Appendix AF

File Documentation

AF.1 RemoteMain.cpp File Reference

Entry point to headless application.

AF.1.1 Detailed Description

Entry point to headless application.

Author

Jonathan Bedard

Date

5/11/2016

Bug None

Starts the TrustNet headless demonstration application. Randomly creates a user if one does not already exist.

AF.2 remoteMainTestInit.cpp File Reference

Test Initialization for RemoteMain.

AF.2.1 Detailed Description

Test Initialization for RemoteMain.

Author

Jonathan Bedard

Date

4/26/2016

Bug No known bugs.

Binds library tests for Remote Main. Binds the osMechanics and CryptoGateway library tests.

AF.3 speedProfiling.cpp File Reference

Entry point to speed profiling.

AF.3.1 Detailed Description

Entry point to speed profiling.

Author

Jonathan Bedard

Date

4/27/2016

Bug None

Speed profiling function. This preforms basic speed tests on different platforms and records the results.

Part XI

BaseForm Executable

Appendix AG

Introduction

This executable is designed to visualize the TrusNet library. It depends on a number of other libraries, including TrusNet, CryptoLogin and glGraphics. This executable uses the graphics framework of glGraphics to visualize the TrusNet library. As an executable, BaseForm also defines a testing battery.

AG.1 Namespace base

Establishes the base namespace, which is designed to be used for functions and classes unique to the visualizing of the TrusNet library. Note that this particular namespace is not designed to be extended outside of this project.

Appendix AH

File Index

AH.1 File List

Here is a list of all files with brief descriptions:

Appendix AI

File Documentation

AI.1 appEntry.cpp File Reference

Entry point to TrusNet demo.

AI.1.1 Detailed Description

Entry point to TrusNet demo.

Author

Jonathan Bedard

Date

4/16/2016

Bug None

Starts the TrustNet demonstration application. Uses the login form to intialize a user.

AI.2 baseFormTestInit.cpp File Reference

AI.3 fileDropDownForm.cpp File Reference

Implementation of file navigation form for TrusNet demo.

AI.3.1 Detailed Description

Implementation of file navigation form for TrusNet demo.

Author

Jonathan Bedard

Date

5/21/2016

Bug None

Implements the form structure which allows for intuitive navigation between forms within the application.

AI.4 fileDropDownForm.h File Reference

File navigation form for TrusNet demo.

Classes

- class **base::fileDropDownForm**
  File navigation form for TrusNet demo Declaration of the navigation form used by the TrusNet demo.
- struct **base::connectionButton**
  Connection button Button with a reference to a connection allowing the opening of a new form.
- class **base::sideUserList**
  Side user list Lists all users currently associated with a particular node.

Namespaces

- **base**

AI.4.1 Detailed Description

File navigation form for TrusNet demo.

Author

Jonathan Bedard

Date

5/21/2016

Bug None

Declares a form with a drop-down allowing for exiting, logging out and going back.

AI.5 IPAddressForm.cpp File Reference

Implementation for the IP-Address form.

AI.5.1 Detailed Description

Implementation for the IP-Address form.

Author

Jonathan Bedard

Date

5/21/2016

Bug None

Implements the IP address form. This form allows for insertion and management of IP addresses.

Consult **IPAddressForm.h** (p. 343) for details.

AI.6 IPAddressForm.h File Reference

IP Address form declaration.

Classes

- class **base::IPAddressFrame**
- class **base::IPAddressForm**

Namespaces

- **base**

AI.6.1 Detailed Description

IP Address form declaration.

Author

Jonathan Bedard

Date

5/21/2016

Bug None

The IP Address form visualizes the IPv4 and IPv6 addresses used by TrusNet to communicate over the Internet protocol. Can add, remove and manage IP addresses.

AI.7 networkMapForm.cpp File Reference

Implements the network-map form.

AI.7.1 Detailed Description

Implements the network-map form.

Author

Jonathan Bedard

Date

5/21/2016

Bug None

Currently, the network-map form essentially just wraps the file drop-down form, declaring a unique set of menu items at the top level.

AI.8 networkMapForm.h File Reference

Defines the network-map form.

Classes

- class **base::networkMapForm**

Namespaces

- **base**

AI.8.1 Detailed Description

Defines the network-map form.

Author

Jonathan Bedard

Date

5/21/2016

Bug None

The network-map form is designed to be the entry point to the BaseForm applicaiton after login has occurred.

AI.9 nodeControlForm.cpp File Reference

Implementation of node control form assets.

AI.9.1 Detailed Description

Implementation of node control form assets.

Author

Jonathan Bedard

Date

5/8/2016

Bug None

The node control form allows direct control of node intrinsics.

AI.10 nodeControlForm.h File Reference

Node control UI assets.

Classes

- class **base::nodeControlForm**

Namespaces

- **base**

AI.10.1 Detailed Description

Node control UI assets.

Author

Jonathan Bedard

Date

5/8/2016

Bug None

UI elements meant to control intrinsics of a remote node.

AI.11 programTools.cpp File Reference

Implimentation of program tools.

AI.11.1 Detailed Description

Implimentation of program tools.

Author

Jonathan Bedard

Date

4/16/2016

Bug None

Program tools are basic functions used accross this demo application.

AI.12 programTools.h File Reference

Program tools header.

Namespaces

- **tools**

Functions

- void **tools::setupLogging** ()
  Set-up logging functions.

AI.12.1 Detailed Description

Program tools header.

Author

Jonathan Bedard

Date

4/16/2016

Bug None

Function declarations of all program tools and headers of of all libraries used in this application.