Inter-process Protocol v1.0 Specification

# Abstract

The Inter-Process Protocol (I2P) is a mid-level communications protocol and set of data structures for remote procedure calls and inter-process communication that mashes together elements of Open Sound Control (OSC), MIDI, ASCII C0 Control Codes, LISP, and AI philosophy. The protocol facilitates procedure calls with multiple parameters and return values, data serialization/marshalling, message passing, remote process oversight, and interface querying.

I2P is currently being developed in the I2P Wiki, and will get copied into this document at a later date. Currently, I copied the I2P 3.11 spec table of contents, and I’m replacing it with the I2P spec because I know it’s an acceptable ISO format. I’m also looking at the Lightweight M2M specification for inspiration.

# Author

Copyright 2017 Chal Jamison McCollough.

# License

Copyright 2016 (C) [Cale McCollough](mailto:cale.mccollough@gmail.com)

Use of this document is subject to the terms and conditions that can be found at http://calemccollough.github.io/i2puseragreement.html.

Table of Contents

[Abstract 1](#_Toc471680043)

[Author 1](#_Toc471680044)

[Introduction 4](#_Toc471680045)

[Organization of I2P 4](#_Toc471680046)

[Terminology 4](#_Toc471680047)

[I2P Script 4](#_Toc471680048)

[Data Types 4](#_Toc471680049)

[List of Types 5](#_Toc471680050)

[Boolean Types 5](#_Toc471680051)

[Integers 6](#_Toc471680052)

[Signed and Unsigned Varints 6](#_Toc471680053)

[Floating-point Numbers 6](#_Toc471680054)

[Timestamps 6](#_Toc471680055)

[Arrays 6](#_Toc471680056)

[Strings 7](#_Toc471680057)

[Hierarchical Data Types 7](#_Toc471680058)

[Escape Sequences 7](#_Toc471680059)

[Units and Books 8](#_Toc471680060)

[Operational Behavior 9](#_Toc471680061)

[Unsigned Integer Size 9](#_Toc471680062)

[64-bit Memory Alignment 9](#_Toc471680063)

[Common Portal Types 9](#_Toc471680064)

[Member Headers 9](#_Toc471680065)

[Object Stack 10](#_Toc471680066)

[Device Controls 10](#_Toc471680067)

[DC1 Root 10](#_Toc471680068)

[DC2 Door 10](#_Toc471680069)

[DC3 XOFF 10](#_Toc471680070)

[DC4 Return Values 10](#_Toc471680071)

[Commands 10](#_Toc471680072)

[Daemon Operations 10](#_Toc471680073)

[I2P Program Operations 11](#_Toc471680074)

[Procedure Calls 11](#_Toc471680075)

[Return Values 12](#_Toc471680076)

[Point-to-Point Connections 12](#_Toc471680077)

[Closed Door Requests 12](#_Toc471680078)

[UART MAB Interrupt 12](#_Toc471680079)

[SPI Interrupt 12](#_Toc471680080)

[Network Connections 12](#_Toc471680081)

[Quality of Service for Procedure Call Return Values 12](#_Toc471680082)

[Message delivery retry 12](#_Toc471680083)

[Message Receipt 13](#_Toc471680084)

[Message Ordering 13](#_Toc471680085)

[Topic Names and Topic Filters 13](#_Toc471680086)

[Topic wildcards 13](#_Toc471680087)

[Topics beginning with $ 13](#_Toc471680088)

[Topic semantic and usage 13](#_Toc471680089)

[Handling errors 13](#_Toc471680090)

[Security 13](#_Toc471680091)

[Currently Supported Communication Protocols 13](#_Toc471680092)

[UDP 13](#_Toc471680093)

[Inter-Process 13](#_Toc471680094)

[I2C, Serial, SPI 13](#_Toc471680095)

[DTLS on Embedded Systems 13](#_Toc471680096)

[Authentication of Daemons by Agents 13](#_Toc471680097)

[Authentication of Agents by Daemons 14](#_Toc471680098)

[Integrity of Escape Sequences 14](#_Toc471680099)

[Detecting Compromise of Daemons and Agents 14](#_Toc471680100)

[Detecting Abnormal Behaviors 14](#_Toc471680101)

[Security proUnits 14](#_Toc471680102)

[Conformance Targets 14](#_Toc471680103)

[I2P Daemon 14](#_Toc471680104)

[I2P Agent 14](#_Toc471680105)

[Inter-process LISP 14](#_Toc471680106)

[# ASCII 32-63 LISP Commands Table 15](#_Toc471680107)

[Operators 15](#_Toc471680108)

[Comments 15](#_Toc471680109)

[Atomic Sequences 16](#_Toc471680110)

[LISP Sequences 16](#_Toc471680111)

# Introduction

## Organization of I2P

I2P is organized into three sections: Data Types, Operation Behavior, and Inter-process LISP. Inter-process LISP is an example of an I2P program, and defines it’s operational behavior.

## Terminology

**I2C** – Inter IC bus is a serial bus protocol like SPI and UART (@wiki I2C Bus).

**SPI** – SPI is a serial port bus like I2C and UART (@wiki SPI Bus).

**UART** or **Serial Port** – A standard universal asynchronous receiver/transmitter serial port bus like I2C and SPI (@wiki UART)

**Inter-process Communication** – Taken from Wikipedia: Inter-process Communication - In computer science, inter-process communication or inter-process communication (IPC) refers specifically to the mechanisms an operating system provides to allow processes it manages to share data. Typically, applications can use IPC, categorized as clients and servers, where the client requests data and the server responds to client requests. Many applications are both clients and servers, as commonly seen in distributed computing. Methods for achieving IPC are divided into categories which vary based on software requirements, such as performance and modularity requirements, and system circumstances, such as network bandwidth and latency. (@wiki Inter-process),

**I2P Client** – A background process, app or state-based machine, that is not presumed to be controlled by a Server. A Daemon is assumed to be dumb, but might be intelligent.

**Agent** – Taken from Wikipedia: Intelligent Agent – “In artificial intelligence, an intelligent agent (IA) is an autonomous entity which observes through sensors and acts upon an environment using actuators (i.e. it is an agent) and directs its activity towards achieving goals (i.e. it is "rational", as defined in economics). Intelligent agents may also learn or use knowledge to achieve their goals. They may be very simple or very complex: a reflex machine such as a thermostat is an intelligent agent.”

**I2P Agent** – A type of Agent that is also a Daemon. An I2P Agent

**I2P Entity** – An I2P entity refers to either an I2P Agent or Daemon.

**UDP** – Universal Datagram Packet.

**I2P Portal** –A communication link that connects two processes/devices.

## I2P Script

Most of the examples in this document are written in I2P Script, which is a human-readable version of I2P. I2P script gets translated directly into byte encoded I2P. Numbers can be transferred either in decimal, binary, true/false, or in hexadecimal format. Please read the Data Types section below if you are not familiar with the script as the section is both a Data Types and I2P Script tutorial.

# Data Types

All data in I2P is represented by 28 primitive types and 4 hierarchical types bellow. Most types are just strings, integers, floating point numbers, and arrays. Varints, or variable length integers, are also just integers. I2P adds to that key-value Dictionaries which we call Books, an optional hash table of contents, and escape sequence; which are basically an atomic backspace-terminated string of subroutines.

### List of Types

| ID | Type | Alt Name | Width | Description |

|:--:|:----:|:---------:|:------:|:------------|

| 0 | NIL | null/void | 0 | Nil/null/void type.|

| 1 | SOH | Address | <=N | An address of an I2P device or function stored as a string.|

| 2 | STX | String | <=N | A UTF-8/ASCII string.|

| 3 | SI1 | int8\_t | -1 | An 8-bit signed integer.|

| 4 | UI1 | uint8\_t | 1 | An 8-bit unsigned integer.|

| 5 | BOL | bool | -1 | A non-zero false boolean variable.|

| 6 | SI2 | int16\_t | -2 | A 16-bit signed varint.|

| 7 | UI2 | uint16\_t | 2 | A 16-bit unsigned integer.|

| 8 | HLF | half | 2 | A 16-bit floating-point number.|

| 9 | SI4 | int32\_t | -4 | A 32-bit signed varint.|

| 10 | UI4 | uint32\_t | 4 | A 32-bit unsigned integer.|

| 11 | FLT | float | 4 | A 32-bit floating-point number.|

| 12 | TMS | Time\_s | -4 | A 32-bit second since epoch timestamp.|

| 13 | SI8 | int64\_t | -8 | A 64-bit signed integer.|

| 14 | UI8 | uint64\_t | 8 | A 64-bit unsigned integer.|

| 15 | DBL | double | 8 | A 64-bit floating-point number.|

| 16 | TMU | Time\_us | -8 | A 64-bit microsecond since epoch timestamp.|

| 17 | SV2 | int16\_t | -1 - 3 | A 32-bit signed varint.|

| 18 | SV4 | int32\_t | -1 - 5 | A 32-bit signed varint.|

| 19 | SV8 | int64\_t | -1 - 9 | A 64-bit signed varint.|

| 20 | UV2 | uint16\_t | 1 - 3 | A 32-bit unsigned varint.|

| 21 | UV4 | uint32\_t | 1 - 5 | A 32-bit unsigned varint.|

| 22 | UV8 | uint64\_t | 1 - 9 | A 64-bit unsigned varint.|

| 23 | AR1 | Array-1 | N | An array of up to 2^8-1 like primitive types 2-14.|

| 24 | AR2 | Array-2 | N | An array of up to 2^16-1 like primitive types 2-14.|

| 25 | AR4 | Array-4 | N | An array of up to 2^32-1 like primitive types 2-14.|

| 26 | AR8 | Array-8 | N | An array of up to 2^64-1 like primitive types 2-14.|

| 27 | ESC | Sequence | ? | An atomic escape sequence of I2P procedure calls.|

| 28 | BK8 | Book-8 | N | A book with up to 2^30-1 members and less than 2^64 bytes data.|

| 29 | BK4 | Book-4 | N | A book with up to 2^14-1 members and less than 2^32 bytes data.|

| 30 | BK2 | Book-2 | N | A book with up to 2^14-1 members and less than 2^16 bytes data.|

| 31 | US | Unit | N | A unit separator for splitting data up into transmission blocks.|

#### List of Types Key

| Width | Description |

|:-----:|:-----------:|

| -X | Signed integer type.|

| ? | Type of unknown size.|

| N | Has pre-specified buffer of size N bytes.|

| <=N | Has pre-specified buffer of size N bytes but can use less than that.|

## Boolean Types

There are two types of Boolean variables, and BOL and ACK. A BOL is a traditional non-zero true Boolean true variable. For speed reasons, ASCII 'T' and 'F' are used in I2P Script.

BOL 0

BOL 1

BOL -127

BOL T

BOL F

## Integers

I2P supports both traditional 8, 16, 32, and 64-bit signed integers and unsigned integers, and 16, 32, and 64 bit floating-point numbers, as well as MSB-encoded 1-to-9-byte length variable signed and unsigned integers called varints. MSb variant encoding uses the MSb of each byte to determine if another byte is to be loaded. This allows values less than 128 to be sent using only one byte, 14-bit values in two bytes, 21-bit values in three bytes and so on.

## Signed and Unsigned Varints

Variants are MSb variant encoded Integers. Both Signed and Unsigned Varints must use the most significant bit asserted is used to marks if another byte is loaded. This allows for single byte varints to be ASCII characters.

UV2 128; = 0b0000\_0001\_1000\_0000

UV2 255; = 0b0000\_0001\_1111\_1111

SV2 -64; = 0b0011\_1111

## Floating-point Numbers

I2P supports, 16, 32, and 64 bit floating-point numbers, also called half, float, and double precision float-point numbers. Floating-point numbers do not lend themselves very well to varint compression, so using a 16-bit floating-point number can dramatically improve performance when low-precision is needed.

HLF 0.0 ; Use a HLF to save memory and bandwidth!

FLT 0.1 ; Wastes a lot of space!

DBL -0.1 ; Wastes a whole lot of space!

## Timestamps

There are two types of timestamps, a 31-bit Unix timestamp, and 63-bit microsecond timestamp in C++11 chrono format. Human-readable dates are in the following formats:

TMS YYYY-MM-DD@HH:MM:ss

TMU YYYY-MM-DD@HH:MM:ss:uu

; Date ---v v—-Minutes

TMS "2016-07-13@15:39:23"

; Hours--^ ^--- seconds

; Date ---v H--v v-—Min. v-- microseconds

TMU "2016-07-13@15:39:23:999"

; Hours--^ ^--- seconds

## Arrays

Arrays can only be created out of like primitive types, and cannot contain strings or hierarchical types. To create an array of strings, use a book. There are 4 types of arrays for 8, 16, 32, and 64 bit sizes. The size of the array is determined by the number of elements times the size of each element.

#### Array Example

AR1 3 UI1 ; Creates an array of 3 unsigned bytes.

1,

2,

3, ; Commas are allowed on the final element.

AR2 2 HLF ;

0.1, ; A half is perfect for low-precision numbers like 0.0 and 0.1.

0.0

#### Example of Array with Errors

AR1 ; Too many members to fit in a byte!

1,024, ; 1024 is too big for UI1.

2, 3, 5,

AR2 2 HLF ; Previous array does not contain 256 items!

0.1,

0.0,

0.3 ; Too many members!

AR1 1 RS ; Cannot contain Hierarchical data types!

## Strings

I2P supports UTF-8/ASCII/c-style strings. Strings are denoted by “double quotes”. (@see Headers). An I2P implementation may enforce strict Unicode compliance.

STX "Example"

## Hierarchical Data Types

Hierarchical data types (HDT) have one or more levels of hierarchy. There are hierarchical types, an Escape Sequence (ESC), three Unit types, and four Record types. Hierarchical levels are terminated using either a NIL, BS, CR, or 0. For an example, see the Escape Sequence Example below.

### Escape Sequences

An Escape Sequence in I2P is a backspace-terminated finite expression of atomic object stack operations. Escape Sequences are composed of sequences Address(s) that address the next object on the stack, and arguments for parameters called by the procedure calls. For example, if we started at index 3, then we could end up at any index above 3, or at 3, but below 3. Expressions can be terminated by either popping the first object off the stack using the ASCII BS (Backspace) command.

The expression can be terminated at any point above in the stack using the ASCII CR (Carriage Return) command. When the CR command is used, the stack location is preserved at the stack index where the CR command was received. For example, you push object 2 onto the stack, then object 3, then pop, push object 4, then CR, and the object stack index will remain at 4. Expression can be nested as follows:

#### Escape Sequence Example

; Hierarchy Description:

; Let ‘A’ be a device named “A” that contains a single sub-device at index ‘B’ named “dev”.

; Let ‘B’ be a sub-device of “A” in index ‘B’ that has a function ‘foo’ at index ‘D’ that takes a UI1 and returns NIL.

DC1/A/B/D 1/CR ; A pushes the device at index ‘A’ onto the stack and CR pops both A and dev off the stack.

“A”/dev/BS ; “A” with quotes is the string “A”. BS pops “A” off the stack.

ESC/A/BS ; This is a nested escape sequence.

ESC/“A”/“foo” 1/CR ; Quotes around foo unnecessary but legal.

A/foo 1/FF ; FF closes all the ESC.

FF FF ; Please note there is one FF per ESC.

0xXXXXXXXX ; followed by a 32-bit hash to verify data integrity.

## Books and Folders

Books and Folders are like Python key-value dictionaries composed of the 32 I2P types. Book and folders are hierarchical and may contain subfolders and “sub-books”. Records and Units are both types of books, one and having a hash table and the other perspective other does not.

For situations where an application requires the fastest push-back performance possible, meaning adding to the end of the data buffer, a Unit works best because there is no hash table. For applications that require fastest search through large Books, a Record works best.

There are 3 types of Units, US2, US4, and US8, and 3 types of records RS2, RS4, and RS8. All Units and records have the same interface but use 16, 32, and/or 64-bit data types. This is to is to reduce the used memory and bandwidth. Books can easily be converted to and from the 3 book types. Books cannot store more data than the numbers of bits can. Below is a list of the books and their memory profiles.

### Book Memory Overhead

| #Bits | Max # Members | Max Header Size | Max Data Size | Overhead Per Object |

|:-----:|:-------------:|:---------------:|:-------------:|:-------------------:|

| 16 | 255 | 2^16 | 2^16 | 6 + 3 per index + buffer.|

| 32 | 2^16 - 1 | 2^32 | 2^32 | 16 + 4 per index + buffer.|

| 64 | 2^25 – 1 | 2^32 | 2^64 | 24 + 16 per index + buffer.|

\* All sizes listed in bytes.

#### Unit and Record Example

BK2 5 ; Preallocate a buffer of size 5.

"Item 1" UI1 23

"Item 2" STR "Hello world!" ; String will be stored with the default buffer size.

"Item 3" 32 STR "Hello world!" ; Optional buffer size comes before the type.

"Item 4" 6 ESC 1 2 3 0 0 0 ; It’s faster to parse script with a preallocated buffer size.

“Item 5” BK2 ; There is a lot of extra copying elsewise.

"Item 1" UI4 2,747,572 ; Commas are allowed in numbers in I2P.

"Item 2" 23 STR "Hello world!"

NIL

; No need for a NIL to close this books of pre-specified length.

#### Example of Books with Errors

BK2 2

"Item 1" 256 STR Error!\0 ; Book2 can only store up to 255 members!

"Item 2" BK4 ; Cannot put larger book in smaller book!

NIL

; ...

"Item 65" BOL 1 ; Cannot possibly have more than 2^N / 4 - 1 items!

; Did not close book with NIL, BS, CR, or 0!

# Operational Behavior

The I2P Communication Protocol is based off ASCII C0 Codes applied to the Chinese room. I2P was initially conceived to run over a serial connection, and the design philosophy of I2P is to be more human-readable when viewed with a standard serial port GUI program such as Real-Term, and to allow users to control devices using a computer keyboard by typing in real-time.

## Memory Usage Levels

To save memory and simplify development, I2P must allow for 4 different memory usage levels (MUL) based upon the size of the unsigned integer used for the buffer. The sizes are 8, 16, 32, and 64-bit labeled MUL 1-4 respectively. MUL(s) are allows bound by CPU word size.

### Memory Usage Level 1

MUL 1 uses an 8-bit unsigned integer for the buffer, must only allow for an 8-bit Book-1. Use MUL1 for systems with approximately 2KB or less memory.

### Memory Usage Level 2

MUL 2 uses a 16-bit unsigned integer for the buffer, and must only allow 8 and 16-bit Books and Books. Use MUL2 for systems with more than approximately 2KB RAM.

### Memory Usage Level 3

MUL 3 uses a 32-bit unsigned integer for the buffer, and must only allow 8, 16, and 32-bit **Books**. Use MUL3 for systems with 64KB or more RAM.

### Memory Usage Level 4

MUL 4 uses a 64-bit unsigned integer for the buffer, and must allow for all Book and Book sizes. Use MUL4 to work with more than 4GB of data, and/or to work with **Books** with lots of members.

## Member Headers

I2P Headers are similar scanf and printf strings, only they do not mix a string with the types to parse. I2P headers must be a packed array of unsigned integers that use a single integer to represent the type, 0-31. I2P implementations must scan the Rx data, and may scan Tx data. For Rx headers, types with specified buffer sizes such as Groups, Books, etc, the size of the buffer must be specified in the header after the type. Tx Headers must use an 8-bit unsigned integer and must not contain the size of the prespecified buffer size types. This design choice is strictly for convenience in coding.

### Rx Header Format

{ NumParameters, Param1, (Optional Param1 Buffer Size),..., ParamN, (Optional ParamN Buffer Size) }

### Tx header Format

{ NumParameters, Param1,..., ParamN }

#### C++ Header Example

static const unsigned int rxHeaderWithoutStrings[] = { 2, UI1, SI2 },

rxHeaderWithStrings[] = { 2, UI1, STX, 32 }; //< We need to specify the size to scan it!

static const unsigned char txHeaderWithStrings[] = { 2, UI1, STR }; //< We don’t scan tx output.

#### Example C++ Implementation Example

#include <ChineseRoom.hpp>

using namespace \_;

struct Member  
{

const char\* name; //< The name of the member.

const int\* rxHeader, //< The rx header.

\*txHeader; //< The tx header.

const char\* description; //< The description of the header.  
};

const Member\* CppHeaderExample (Terminal\* io)

{

void\* params[2];

static const int rxHeader = { 2, UI4, STR, 32 },

txHeader[] = { 2, UI4, STR };

static const Member m = { “functionName”, rxHeader, txHeader, “Description” };

if (io == nullptr) return m;

uint32\_t inputA,

inputB,

outputA = 1,

outputB = 2;

if (io->read (m.rxHeader, args (params, &inputA, &inputB))) return readError();

/// Do some function logic here.

return io->write (m.txHeader, args (params, &outputA, &output));

}

## Portals

A portal is a connection that connects two rooms. The following is a short list of some of the protocols used in this spec:

1. UART
2. SPI
3. BLE
4. TCP/IP
5. UDP
6. Inter-process (Same system)

## Devices and Device Stacks

To control devices in I2P, each Terminal must have its own stack of devices of some arbitrary size. Each device must have up to 192 members, referenced using a single byte using indexes 64-255. Each member must be either a device, or a procedure call. If the member is a device, then it must be pushed onto the top of the device stack, else it must be treated like a procedure call. The device stack must use the backspace command (BS) to pop devices off the stack.

## Daemon Procedures

The Inter-process protocol relies on the first 32 ASCII characters. Printable ASCII C0 codes characters, VT, HT, CR, ESC, etc. must be treated as whitespace for human readable formatting.

### Daemon Operations

| Id | ASCII | Return | Param | Description |

|:---:|:-----:|:------:|:-----:|:------------|

| 0 | NUL | NIL | NIL | Pops a device off the stack.|

| 1 | SOH | FS1 | FS1 | Undefined.|

| 2 | STX | NIL | FS1 | Logs a messages to the current log.|

| 3 | ETX | NIL | NIL | Starts a new log on the selected device.|

| 4 | EOT | NIL | NIL | Disconnect from the Portal.|

| 5 | ENQ | NIL | STR | Sends an enquiry string to the requesting device.|

| 6 | AWK | NIL | NIL | Acknowledges system is responding correctly.|

| 7 | BEL | FS1 | NIL | Request from an Agent to open the Door.|

| 8 | BS | NIL | NIL | Pops a device off the stack.|

| 9 | HT | NIL | NIL | Whitespace.|

| 10 | LF | NIL | NIL | Whitespace.|

| 11 | VT | NIL | NIL | Whitespace.|

| 12 | FF | NIL | NIL | Whitespace.|

| 13 | CR | NIL | NIL | Whitespace.|

| 14 | SO | NIL | NIL | Undefined.|

| 15 | SI | NIL | NIL | Undefined.|

| 16 | DLE | NIL | NIL | Undefined.|

| 17 | DC1 | NIL | NIL | Pushes DC1 onto the device stack.|

| 18 | DC2 | NIL | NIL | Pushes DC2 onto the device stack.|

| 19 | DC3 | NIL | NIL | Pushes DC3 onto the device stack.|

| 20 | DC4 | NIL | NIL | Pushes DC4 onto the device stack.|

| 21 | NAK | NIL | NIL | Undefined.|

| 22 | SYN | NIL | NIL | Pings the system and/or responds to a ping.|

| 23 | ETB | NIL | NIL | Undefined.|

| 24 | CAN | NIL | NIL | Cancels the current ESC.|

| 25 | EM | NIL | NIL | Signals the input buffer has run out.|

| 26 | SUB | NIL | NIL | Reports that the previous message was an error message.|

| 27 | ESC | NIL | ESC | Starts an interrupt ESC.|

| 28 | FS | NIL | NIL | Pushes the filesystem onto the stack if supported.|

| 29 | GS | NIL | NIL | Undefined.|

| 30 | RS | NIL | NIL | Undefined.|

| 31 | US | NIL | NIL | Pushes the return value Book onto the stack.|

## Device Controls

In the root scope of every I2P Room must be four devices labeled DC1, DC2, DC3, and DC4, corresponding to C0 codes 17-20.

### DC1 - Root

DC1 must be a device with Root scope of the Door. This is analogous to the Each door may have a different root device, allowing for different privilege levels.

### DC2 - Self

DC2 must be a device with the scope of the Door leading to the origin system of the Terminal. This room may have multiple terminals that lead to it, so the default index must be index 64 or ASCII ‘@’.

### DC3 - XOFF

DC3 must be a device that does nothing by default. XOFF may be overridden by the user for application specific purposes.

### DC4 - User

## DC4 must be a device that does nothing by default. DC4 may be overridden by the user for application specific purposes.

## Procedure Calls

Procedure calls must be able to take multiple parameters, and return multiple return values. Procedure calls must be performed by selecting a device that has up to 192 members by issuing a single byte function call index, followed by the parameters for that function in the 64-bit memory aligned section described above. A single procedure call cannot be processed by itself, but must be contained in an ESC. A single function call looks like this in I2P script:

; in C++, the Member header looks like this:

; static const int rxHeader { ] = {1, UI4, UI8 },

; txHeader[] = { NILL, NILL };

; static const Member\* m = { “exampleFunction”,

; rxHeader,

; txHeader,  
; “Description” };

64 1 2 CR ; Index 64 is the first user-definable function index.

This example calls the member at index 64 and passes it a UI4 with a value of 1 and UI8 with value of 2, then the function call is not processed until the CR byte is received, or the procedure call gets canceled upon timeout. The above example required a total of 14 bytes to perform, with 4 wasted bytes after the UI4.

### Return Values

Return values must be performed using a Book that is pushed onto the device stack using the ASCII US command 31; also called the US Book. When a procedure call with a return value is called, an sub-Book is created in the US Book, and the address of this newly created Book passed into the calling function in the form of an ESC.

The size of the Book used for the US Book will inherently cap the you a cap on size of the return value data size of the data limits of the Book size used; 8, 16, 32, or 64-bit values. It is up to the implementation to ensure that you can fit the amount of return data into your buffer!

#### C++11 Return Value Example

static const Member m64 = { “exampleWithReturnTypes”,

Rx<0>::Header,

Tx<2, UI1, UI1> };

// I2P Call: 64

// Local system creates entry into US with return address of { 31, 1

### ESC Interrupt Procedures

Interrupt procedures may be initiated using the ESC command. Due to the nature of real-time systems, interrupt procedures must not take any parameters. To regain control an I2P system, implementations must be able to stream a series of ESC bytes to the target system.

## P2P Connections

Point-to-Point (P2P) Connections are connections from one process or machine to another without any network in between. SPI and UART are primary examples. In these cases, synchronization is different, and often dependent on the specific UART hardware. Due to I2P’s escape sequences, I2P can be streamed serially without the need for any special hardware to start the sequence or control traffic. Often is the case with devices with non-ideal electronic components, the connection will sometimes cut out, and junk data will get streamed to the target. In this situation, streaming a stream of NIL commands will. Sometimes a target will be expecting much more data streamed to it, so to snap the device out of it, an interrupt method is required. There is a different interrupt protocol for each communication protocol.

## Network Connections

I2P is a serial protocol developed to work well on both UART and UPD devices, as well as a variety of other transport protocols

## Hand Shake Procedure

1. Connector broadcasts a stream of one or more BEL bytes.
2. Connectee responds with a stream of one or more BEL byte.
3. Connector sends and ENQ byte followed by the EnqueryHeader.
4. Connectee responds with EnquireyHeader.
5. Connector and Connectee both entre the connected.

## Initial Connection

## Closed Door Requests

Closed door requests are requests that will require a reconnection to accomplish. Example updating firmware on a microcontroller where the system needs to reboot.

### UART MAB Interrupt

For UART, I2P uses the same interrupt system as the DMX512 lighting protocol.

### SPI Interrupt

A Chip Select pin is used to interrupt SPI devices.

## Procedure Call Quality of Service

For all remote procedure calls that return NIL, there is no checking to see if a function call made. Quality of Service (QoS) is performed using return parameters. Each function call with at least one return parameter gets a 16-bit function call index attached to it, which is then used to throw an error if the return arguments were not received.

## Message delivery retry

Message deliver retry is performed using a special Book that

## Message Receipt

## Message Ordering

Message ordering is performed using a Book.

## Observed Devices

Some devices may be observed by a list of entities. When a device is observed, any time any of the states are changed, the entity will report it’s changes to the broker.

## Handling errors

Errors must be handled before calling automated procedure calls!

# Security

I2P is a mid-level protocol that runs on most communication protocols, and security is handled differently for different protocols.

## Supported Communication Protocols

### I2C, Serial, SPI

Devices connected to I2C, Serial, and SPI do not require additional security measures. These are hard-wired devices so they are assumed to be slave-master relationship.

### UDP

User Datagram Protocol is designed to run over less than idea network conditions, and may have dropped packets. UDP packets have packet size limitations of up to 64KB.

### Inter-Process

Inter-process communication is assumed to be a secure connection without data loss, but this behavior is not guaranteed.

## DTLS on Embedded Systems

## Authentication of Daemons by Agents

## Authentication of Agents by Daemons

## Integrity of Escape Sequences

ESC must be checked for integrity before calling the procedure call. A basic NIL-NIL procedure call would require the bytes

## Detecting Compromise of Daemons and Agents

Compromised daemons and agents can be detected by monitoring .

## Detecting Abnormal Behaviors

Abnormal behavior can be recognized by the following features:

1. Invalid procedure calls.
2. Timing out during procedure calls.
3. Not closing ESC.
4. Losing low-level connection link.
5. Malformed UTF-8 strings.

It is out of the scope of I2P to detect variables that are out of range for parameter inputs in procedure calls.

## Security Profiles

### Slave Profile

A slave device assumes that there is no

# Conformance Targets

## I2P Client

A target is certified to be an I2P Client if the requirements for Section one and two are meet. Clients must implement functions 0 through 31.

## I2P Server

A target is certified as an I2P Server if the requirements of Section One, Two, and Three are meet. Server must implement functions 32 through 63.

# Appendix A: ASCII C0 Codes

| ID | Code | Name |

|:---:|:-----:|:------:|

| 0 | NUL | NIL |

| 1 | SOH | Start of header |

| 2 | STX | Start of text |

| 3 | ETX | End of text |

| 4 | EOT | End of transmission |

| 5 | ENQ | Enquiry |

| 6 | AWK | Acknowledgement |

| 7 | BEL | Bell |

| 8 | BS | Backspace |

| 9 | HT | Horizontal tab |

| 10 | LF | Line feed |

| 11 | VT | Vertical tab |

| 12 | FF | Form feed |

| 13 | CR | Carriage return |

| 14 | SO | Shift out |

| 15 | SI | Shift in |

| 16 | DLE | Data link escape |

| 17 | DC1 | DC1 |

| 18 | DC2 | DC2 |

| 19 | DC3 | DC3 |

| 20 | DC4 | DC4 |

| 21 | NAK | Negative acknowledge |

| 22 | SYN | Synchronous idle |

| 23 | ETB | End of transmission block |

| 24 | CAN | Cancel |

| 25 | EM | End of medium |

| 26 | SUB | Substitute |

| 27 | ESC | Escape |

| 28 | FS | Book separator |

| 29 | GS | Group separator |

| 30 | RS | Record separator |

| 31 | US | Book separator |