XYO Network Interaction Protocol v1.11

XYO

December, 2018

1 Overview

The XYO Interaction Protocol determines how devices can interact with each other though XYO. The protocol is broken into three layers, Transport, Exchange, and Data.

2 Transport Layer

The transport layer determines how devices speak to each other via Bluetooth, TCP/IP, or other transport. All these will have a similar structures, but will have transport specific differences.

2.1 Bluetooth (BLE)

The Bluetooth transfer layer essentially abstracts GATT (Generic Attributes) characteristics to an input and output stream. Both client and server can send send any data to each other.

2.1.1 Primary Service

This service contains 2 characteristics, a read characteristic and a write characteristic. The XYO Primary Service UUID indicates that a device is capable of XYO functionality. The UUIDs for the service are listed below.

XYO Primary Service UUID: d684352e-df36-484e-bc98-2d5398c5593e
– **Write Characteristic UUID:** 727a3639-0eb4-4525-b1bc-7fa456490b2d
– **Read Characteristic UUID:** d96b6ad7-cbcb-4979-bf06-a1051edaecb4

2.1.2 Sending Data

Transferring data between two XYO BLE devices consists of reading and writing to characteristics. All data is chunked into individual packets that can be reconstructed by the client or the server. When the connection is established, the client must subscribe to the read characteristic, so it can be notified when to read the servers data.

Client Sending to Server

After the two parties have established a connection. A client client can send information to the server by writing the respected bytes to the write characteristic.

Server Sending to Client

After the two parties have established a connection. A server can send to the client by sending a BLE notification to the client to read from its read characteristic.

2.1.3 Advertising and Device Discovery

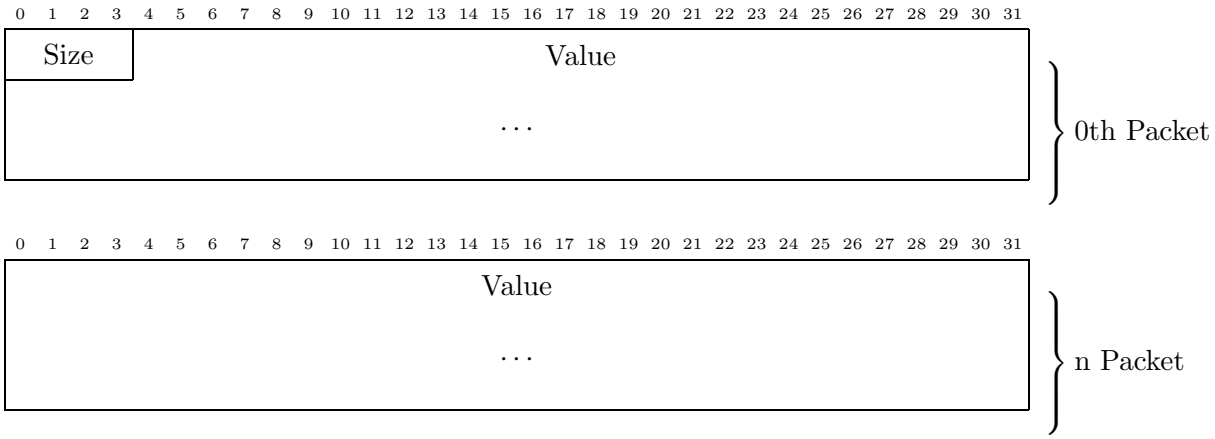
XYO Enabled BLE devices should be advertising the XYO Primary Service UUID to identify them as available for a connection session. If the device is not available for a connection, it should not be advertising the primary service. This means unless a device can support mutable XYO connections at once, it should stop advertising after a connection has been established.

Many modern BLE devices rotate MAC addresses in order to combat unwanted tracking of the device. Due to this, it is hard to identify a device sully off of its mac address over a long period of time. To solve this, a node can optionally advertise any series of random bytes as the manufacturer id.

2.1.4 Chunking Data

When sending any data between two devices, it is necessary to chunk all of the data wishing to be sent into smaller segments so large amounts of data can be transacted.

To chunk data, a device must prepend a 4 byte unsigned integer (Big Endian) with the value of the size of the entire data wishing to be sent, in bytes. This size includes itself. After the size has been prepended, the device may chunk it up into as many portions as wanted, as long as the largest packet is smaller than or equal to the MTU of the connection.



We can infer that the packet is done being sent once the number of value bytes is equal to the $size - 4$.

2.2 TCP/IP

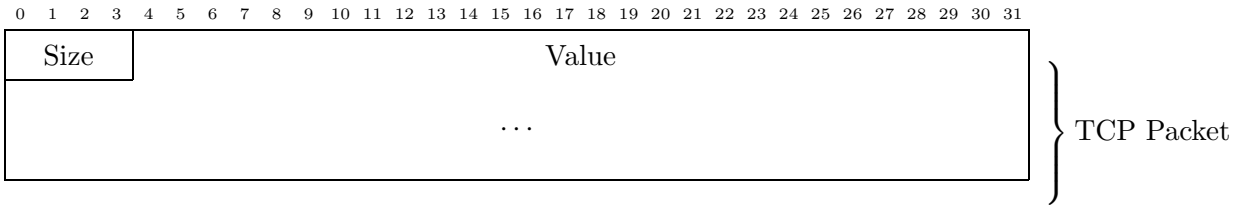
The TCP/IP network is a simple way to transact information between devices.

2.2.1 Connection Session

During a TCP connection between two devices, a socket is only established once. After the connection session is over, the socket is closed by either party.

2.2.2 Transacting Data

To send data between nodes, a 4 byte unsigned integer (Big Endian) must be sent through the socket (size includes itself), then the value wishing to be sent. We can infer that the packet is done being sent once the number of value bytes is equal to the $size - 4$.



3 Exchange Layer

The exchange layer determines the packet sequencing expectations for an interaction between devices. This is consistent regardless of transport.

3.1 Bound Witness Sequence

For clarity, we will refer to the parties as 0, 1, etc... Bound witness follow the idea of a fetter and a witness. A fetter is an object that contains all signed metadata with the public keys of the device. A witness is an object that contains all unsigned metadata along with the signature of the bound witness. Parties in a bound witness follow a first in, lat out basis. The signature signs all of the fetters contacted together.

Single Party Bound Witness

Step	Party	Description
1	0	Fetter
1	0	Witness

Two Party Bound Witness

Step	Party	Description
1	0	Fetter
2	1	Fetter
3	1	Witness
4	0	Witness

Three Party Bound Witness

Step	Party	Description
1	0	Fetter
2	1	Fetter
3	2	Fetter
4	2	Witness
5	1	Witness
6	0	Witness

4 Data Layer

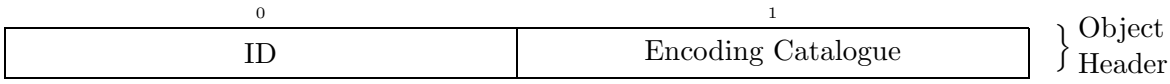
The data layer specifies how data blocks are interpreted by devices. All numbers are passed in Big Endian. There is no padding between items being passed. Everything that is encoded is a superset of an *Object*. An object is defined by having an object header that contains information about how to unpack that object. This information includes how to read the size of the object, if the object is Iterable (typed and untyped), and the ID of the object.

5 Object Structure

Name	**Description**
Object Header	Info about the object and size
Size	Size of the object
Payload	The raw contents of the object.

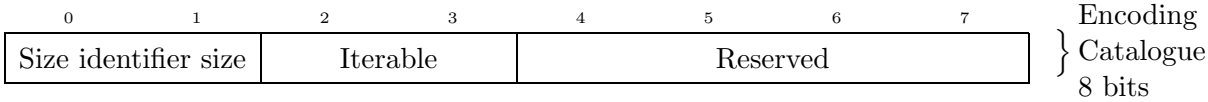
6 Object Header

The object header is prepended to every object to obtain information about the object. This is broken into two primary sections, the encoding catalogue and ID.



6.1 Encoding catalogue (1 Byte)

The encoding catalogue gives information about the size of the size and if the object is Iterable (typed and untyped).



6.1.1 Size identifier size (2 bits)

These two bits are used to determine how many bytes to read to obtain the size of the object after the header of the object. The states of the Size identifier size flags are listed below. Note: *All sizes include themes, are big endian, and are unsigned.*

Flag	Name
00b	1 Byte Size.
01b	2 Byte Size.
10b	4 Byte Size.
11b	8 Byte Size.

6.1.2 Iterable (2 bits)

This flag is the 3-4th most significant bits. The flags follow the table below:

Flag	Name
00b	Not Iterable.
01b	Associative Array.
10b	Untyped Iterable.
11b	Typed Iterable.

6.2 ID (1 Byte)

Used as a ID for the value/payload. This value is appended to the Encoding catalogue.

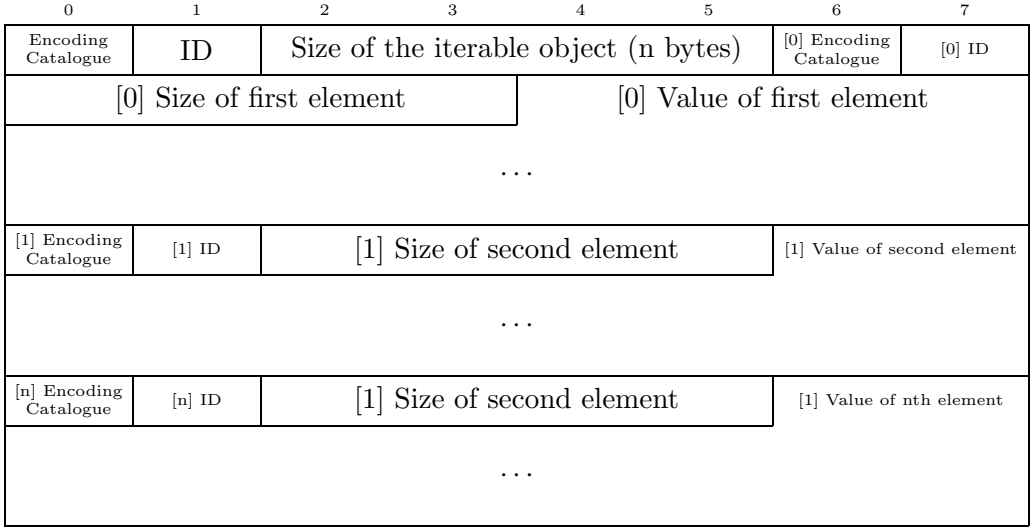
7 Iterable Objects

Iterable Objects are objects that contain a set of child objects. If an object is iterable, the proper flag will be set in the object header. There are two types of iterable objects: typed iterable objects, and untyped Iterable Objects. If an iterable is typed, the proper flag will be set in the object header.

7.1 Untyped Iterable Objects

Untyped Iterable Objects are objects that contain a set of objects that have more than one type of header. Untyped iterable objects are created by concatenating many objects together do not that share one header.

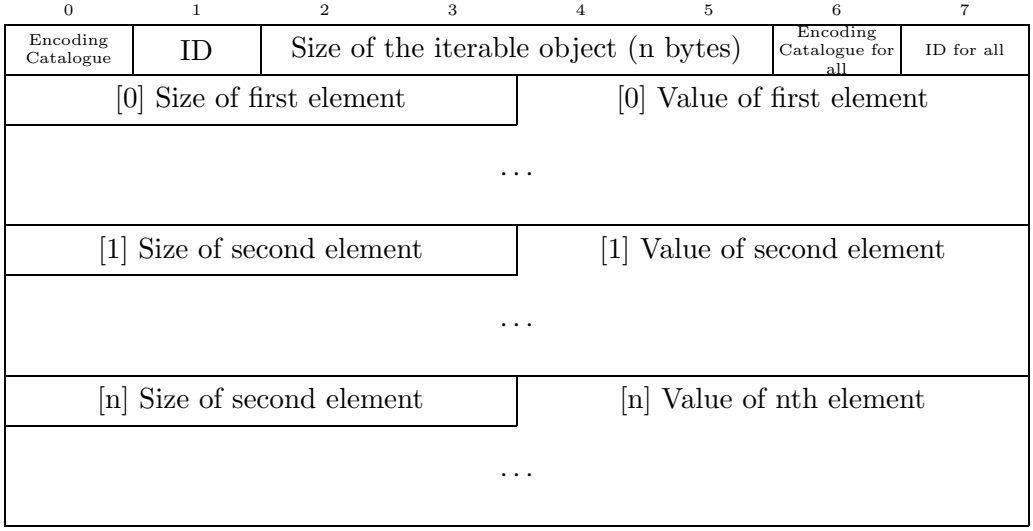
7.1.1 Example Structure



7.2 Typed Iterable Objects

Typed Iterable Objects are objects that contain a set of objects that all share the same header. Typed iterable objects are created by concatenating many objects without their header together and prepending it with the shared header.

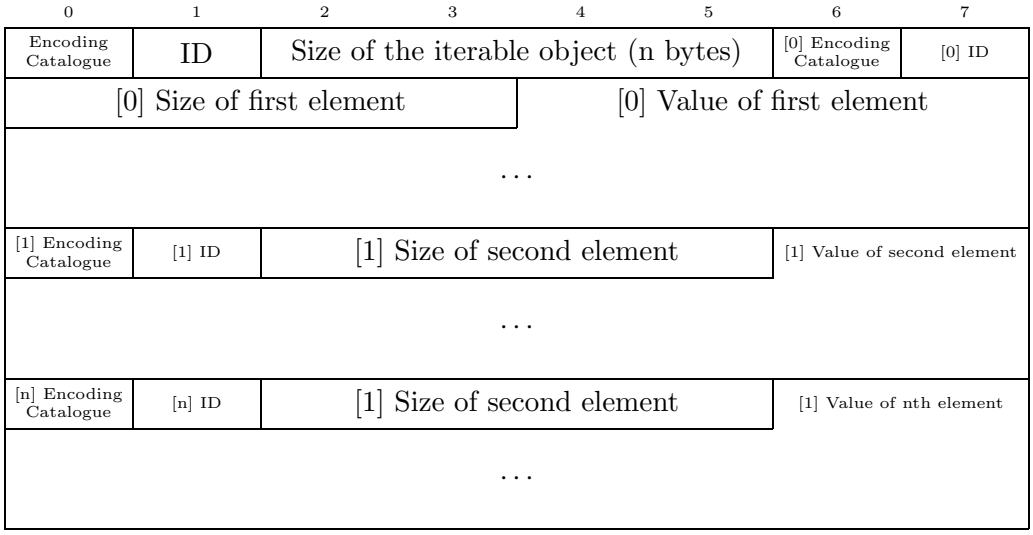
7.2.1 Example Structure



7.3 Associative Array

Associative Arrays are just like untyped arrays with the condition that every elements id in the array must be unique, similar to an "object" in JSON.

7.3.1 Example Structure



7.4 First Class Objects

Array
Id: 1
Iterable: True
Descreption: A general array type.

Bound Witness
Id: 2
Iterable: True
Descreption: An XYO Bound Witness

Origin Index
Id: 3
Iterable: False
Descreption: The index (height) of the current bound witness in an origin chain.

Next Public Key
Id: 4
Iterable: True
Descreption: The next key to use in an origin chain.

Bridge Block Set
Id: 5
Iterable: True
Descreption: An array of blocks. Used when bridging.

Bridge Hash Set
Id: 6
Iterable: True
Descreption: An array of hashes of blocks. Used when bridging.

Previous Hash
Id: 8
Iterable: True
Descreption: The previous Bound Witness hash.

ECDSA Signature with Sha256 on Secp256k1
Id: 9

Iterable: False
Descreption: ECDSA Signature with Sha256 on the curve: Secp256k1.

RSA Signature
Id: 10
Iterable: False
Descreption: The modulus of the RSA signature.

Stub Signature
Id: 11
Iterable: False
Descreption: A signature used for development.

EC Secp256k1 Uncompressed Public Key
Id: 12
Iterable: False
Descreption: EC Uncompressed Public key on the curve Secp256k1.

RSA Public Key
Id: 13
Iterable: False
Descreption: RSA Public Key.

Stub Public Key
Id: 14
Iterable: False
Descreption: A stub public key used for development.

Stub Hash
Id: 15
Iterable: False
Descreption: A stub hash used for development.

SHA 256
Id: 16
Iterable: False
Descreption: A SHA 256 Hash.

SHA 3
Id: 17

Iterable: False
Descreption: A SHA 3 Hash.

GPS
Id: 18
Iterable: Yes
Descreption: GPS Location.

RSSI
Id: 19
Iterable: No
Descreption: RSSI of the Bluetooth Connection.

Unix Time
Id: 20
Iterable: No
Descreption: The Unix time of the bound witness in milliseconds.

Fetter
Id: 21
Iterable: Yes
Descreption: A bound witness fetter.

Fetter Set
Id: 22
Iterable: Yes
Descreption: A set of fetters.

Witness
Id: 23
Iterable: Yes
Descreption: A bound witness witness.

Witness Set
Id: 24
Iterable: Yes
Descreption: A set of witnesses.

Key Set
Id: 25

Iterable: Yes
Descreption: A set of public keys.

Signature Set
Id: 26
Iterable: Yes
Descreption: A set of signatures.

Bound Witness Fragment
Id: 27
Iterable: Yes
Descreption: A witnesses and fetters.

Latitude
Id: 28
Iterable: No
Descreption: Latitude.

Longitude
Id: 29
Iterable: No
Descreption: Longitude.