From Array to Real

Here is the breakdown of the code that I used for converting an Array of two 16-bit values to a Real value.

Here's a breakdown of what each part of the code does:

1. Create a buffer

```
var buf = new ArrayBuffer(4); //0,1,2,3
```

This line creates a new ArrayBuffer of 4 bytes. An ArrayBuffer is a raw binary buffer; it doesn't have a format until you view it through a view, like DataView or typed arrays.

2. Create a data view of it:

```
var view = new DataView(buf);
```

A DataView is created for the ArrayBuffer. The DataView provides a low-level interface for reading and writing multiple number types in the buffer, without having to care about the platform's endianness (byte order).

3. Write integers to it:1

```
view.setUint16(0, msg.payload[1]);
view.setUint16(2, msg.payload[0]);
```

These lines write two 16-bit unsigned integers (Uint16) to the buffer. The first integer is written at byte position 0, and the second at byte position 2. The values for these integers are taken from msg.payload[1] and msg.payload[0].

This is swapping the order of two 16-bit parts of a 32-bit data structure.

4. Read the buffer as a float:

```
var num = view.getFloat32(0);
```

Here, the code is reading the data in the buffer as a 32-bit floating point number (Float32), starting from byte position 0. This operation effectively treats the two combined 16-bit integers (now swapped) as a single 32-bit float.

5. Format and return the message:

```
msg.topic = "Analog Input 1";
msg.payload = Number(num.toFixed(2));
```

```
return msg;
```

The msg object is being modified by setting msg.topic to a string ("Analog Input 1") and updating msg.payload to be the 32-bit float read from the buffer, converted to a number in JavaScript (which is inherently a double-precision 64-bit number), and formatted to two decimal places. Finally, the modified msg object is returned.
This code is likely used in a context where binary data from sensors or similar devices is being processed and converted into a readable format for further use, such as displaying in a dashboard or logging.

From Real to Array

Here is the breakdown of the code that I used for converting a Real value to Array to two 16-bit values.

Here's a breakdown of what each part of the code does:

1. Create a Buffer and a DataView:

```
// Step 1: Create a buffer and a data view
var buf = new ArrayBuffer(4);
var view = new DataView(buf);
```

First, you create an ArrayBuffer of 4 bytes (to store a 32-bit float) and a DataView for this buffer.

2. Write the Floating-Point Number:

```
// Step 2: Write the floating-point number as a 32-bit float
var floatNumber = msg.payload;
view.setFloat32(0, floatNumber);
```

Use the DataView to write the floating-point number as a 32-bit float.

3. Read the Buffer as Two 16-bit Integers with byte swap:

```
// Step 3: Read the buffer as two 16-bit integers with byte swap
var int16_1 = view.getUint16(2);
var int16_2 = view.getUint16(0);
```

Then, read two 16-bit integers from the DataView with byte swap.

4. Send the result

```
// The result is an array of two 16-bit integers
msg.payload = [int16_1, int16_2];
```

```
return msg;
```

The result array will contain two 16-bit integers that represent the original 32-bit float.

This method is useful when you need to send floating-point data in a format that requires 16-bit integer segments, which is common in certain communication protocols in automation and IIoT systems.