Encoder/Decoder Algorithm for SerialNumber Tag 9F1E

Due to the limitation of 8 characters maximum for Tag 9F1E, the Encoder/Decoder algorithm aims to compress the 10-byte Augusta device serial number into an 8-byte string.

The algorithm uses a reference table to take advantage of the unused last bit in the printable characters ASCII range.

Please refer to the Appendix for the reference class.

APPENDIX: 6-bit ASCII character EncoderDecoder Class

public static class EncoderDecoder

{

static ReferenceTable referenceTable = ReferenceTableValue.Create();

public static string Decode(byte[] encode)

{

string binary = "";

foreach (var item in encode)

{

binary += Convert.ToString(item, 2).PadLeft(8, '0');

}

while (binary.Length % 6 != 0)

{

binary = "0" + binary;

}

var sixPadBitbinaryArray = Enumerable.Range(0, binary.Length / 6).

Select(pos => binary.Substring(pos \* 6, 6)).ToArray();

StringBuilder result = new StringBuilder();

int count = 0;

foreach (var item in sixPadBitbinaryArray)

{

string element = Convert.ToInt32(item, 2).ToString("X").PadLeft(2, '0');

if (element == "00" && count == 0)

{

count++;

continue;

}

count++;

result.Append(referenceTable.GetChar(element[0].ToString(), element[1].ToString()));

}

return result.ToString();

}

public static byte[] Encode(string input)

{

if (!referenceTable.IsValidString(input))

{

throw new Exception("invalid string. use char from table" + input);

}

string eightPadBitbinary = "";

foreach (var item in input.ToCharArray())

{

eightPadBitbinary += Hex2binaryWithSixPadding(referenceTable[item]);

}

while (eightPadBitbinary.Length % 8 != 0)

{

eightPadBitbinary = "0" + eightPadBitbinary;

}

var eightPadBitbinaryArray = Enumerable.Range(0, eightPadBitbinary.Length / 8).

Select(pos => Convert.ToByte(eightPadBitbinary.Substring(pos \* 8, 8), 2)).ToArray();

return eightPadBitbinaryArray;

}

static string Hex2binaryWithSixPadding(string hexvalue)

{

var hexToBin = String.Join(String.Empty, hexvalue.Select(c => Convert.ToString(Convert.ToUInt32(c.ToString(), 16), 2).PadLeft(4, '0')));

string result = hexToBin;

while (result.Length > 6 && result.StartsWith("0"))

result = hexToBin.TrimStart(new char[] { '0' }); ;

if (result.Length > 6)

throw new Exception("hex to bin length error HexValue = " + hexvalue);

else

result = result.PadLeft(6, '0');

return result;

}

}

public class ReferenceTableValue

{

static List<string> list = new List<string>();

private ReferenceTableValue()

{

}

public string GetAllChar()

{

return string.Join("", list);

}

public static ReferenceTable Create()

{

ReferenceTable t = new ReferenceTable();

t.ReferenceTableValues = new List<ReferenceTableValue>();

list.Add(":1234567890=≠≤![");

list.Add(" /STUVWXYZ],(→≡~");

list.Add("-JKLMNOPQR%$\*↑↓>");

list.Add("+ABCDEFGHI<.)≥?;");

for (int row = 0; row <= 3; row++)

{

for (int col = 0; col <= 15; col++)

{

char cc = list[row].Substring(col, 1).ToCharArray()[0];

ReferenceTableValue rf = new ReferenceTableValue(String.Format("{0:X}", row), String.Format("{0:X}", col), cc);

t.ReferenceTableValues.Add(rf);

}

}

return t;

}

public ReferenceTableValue(string x, string y, char charector)

{

this.X = x;

this.Y = y;

this.Charector = charector;

}

public string X { get; set; }

public string Y { get; set; }

public char Charector { get; set; }

}

public class ReferenceTable

{

public List<ReferenceTableValue> ReferenceTableValues { get; set; }

public bool IsValidString(string input)

{

if (string.IsNullOrWhiteSpace(input) || ReferenceTableValues == null || ReferenceTableValues.Count == 0)

{

return false;

}

var refval = ReferenceTableValues[0];

string allStringChar = refval.GetAllChar();

foreach (var item in input.ToCharArray())

{

if (!allStringChar.Contains(item))

{

return false;

}

}

return true;

}

public string this[char val]

{

get

{

if (ReferenceTableValues == null)

{

return null;

}

foreach (var item in ReferenceTableValues)

{

if ((int)item.Charector == (int)val)

{

return item.X + item.Y;

}

}

throw new Exception(string.Format("Value not found for char [{0}]", val.ToString()));

}

}

public char GetChar(string \_x, string \_y)

{

if (ReferenceTableValues == null)

{

return '\0';

}

foreach (var item in ReferenceTableValues)

{

if (item.X == \_x && item.Y == \_y)

{

return item.Charector;

}

}

throw new Exception(string.Format("value not found."));

}

}