PUBLICATION OF MITCH DATA TO IDP

MARKET DATA

August 2016

CONTENTS

1 DOCUMENT HISTORY

Revision History			
Name	**Designation**	**Signature**	**Date**
Daniel Nieuwoudt	Senior Developer		2016-08-18
Hafeni Heita	Intermediate Developer		2017-08-24
Hafeni Heita	Intermediate Developer		2018-09-26

2 SCOPE OF THIS DOCUMENT

The following document describes the format in which Vonnegut provides the Mitch UDP Public Data Extract. The nature of this document is technical, and is intended for a technical audience, as it discusses the envelope structure of the binary file provided.

3 FORMAT OF THE FILE

The file can be provided to clients in one of two formats:

3.1 Uncompressed

The file contains no compression algorithm and the envelope should be processed directly from the file stream being read. The file name will have a **.bin** file extension.

3.2 Compressed

The file has been compressed using the standard .NET GZIP algorithm subsequent to the file being exported. The file needs to be decompressed before or during the processing of the envelope. The file name will contain a **.gz** extension.

4 ENVELOPE STRUCTURE

Field	Type	Description
HEADER		
Start Record Indicator	Byte	Value will always be 255
Length	UShort	2 Bytes indicating the length of the record (Little Endian)
CRC	Byte	CRC of the two length bytes
RECORD STRUCTURE		
Id	Long	The sequential ID of the message when written out (Little Endian)
Message Type	String	The message type of the MITCH Message. Allows users of the file to pre-filter messages without the need to decode the message to determine the message type. N.B. This is a Pascal encoded string with an unsigned short length (Little Endian) indicator preceding the string.
Data	Byte[]	The raw MITCH message that can be decoded using the MITCH specification
NB: [1] The "Data" byte array contains the original MITCH Message. This needs to be decoded according to the Market Data Gateway (MITCH - UDP) v3.02 specification. The specification can be found at		

https://www.jse.co.za/services/itac

5 GLOSSARY OF TERMS

Term	Description
CRC	A cyclic redundancy check (CRC) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data
IDP	Information Dissemination Portal
ITCH	ITCH is a direct data-feed protocol such as TCP (Transmission Control Protocol) or UDP (User Datagram Protocol)
MITCH	Millennium ITCH
GZIP	Standard file compression algorithm used by the .NET framework. Reduces the size of a file

6 APPENDIX

To aid in reading our custom binary format, we have provided some sample C# code to aid in the Level 2 MITCH Tick Data extraction process. Note: This code assumes that the file has been decompressed already.

```
public class MitchReaderSample
    {
        const byte StartOfMessageIndicator = 255;
        /// <summary>
        /// StartCapture: Reads the MITCH extraction data provided as a FileInfo object
        /// </summary>
        /// <param name="deviceFile"></param>
        public static void StartCaptureSample(FileInfo deviceFile)
        {
            var messageCount = 0;

            using (var fileStream = new FileStream(deviceFile.FullName, FileMode.Open,
FileAccess.Read, FileShare.Read))
            using (var binaryReader = new BinaryReader(fileStream))
            {
                do
                {
                    try
                    {
                        var messageWithHeader = GetMessageWithHeader(binaryReader);
                        if (messageWithHeader.HasValue)
                        {
                            var messageDataBuilder = new StringBuilder();
                            Array.ForEach(messageWithHeader.Value.Data, x =>
messageDataBuilder.Append(x));


                            Console.WriteLine("Message #   : {0}", ++messageCount);
                            Console.WriteLine("MessageType : {0}",
messageWithHeader.Value.MessageType);
                            Console.WriteLine("Message Data: {0}",
messageDataBuilder.ToString());
                        }
                    }
                    catch (EndOfStreamException)
                    {
                        break;
                    }
                } while (true);
            }
        }
```

```
  /// <summary>
        ///  Returns the Constructed Message or null if the start of message inidicator
is not read.
        /// </summary>
        /// <param name="binaryReader"></param>
        /// <returns></returns>
        public static ExtractionMessage? GetMessageWithHeader(BinaryReader binaryReader)
        {
            if (binaryReader.ReadByte() == StartOfMessageIndicator)
            {
                return ExtractionMessage.BuildExtractionMessage(binaryReader);
            }
            return null;
        }
```

```
public static byte[] CorrectEndianness(this byte[] value, Endianness endianness)

{

  switch (endianness)

  {

    case Endianness.LittleEndian:

         if (!BitConverter.IsLittleEndian) Array.Reverse(value);

         break;

    case Endianness.BigEndian:

         if (BitConverter.IsLittleEndian) Array.Reverse(value);

         break;

    case Endianness.Default:

          break;

           }

           return value;

        }
```

```
public struct ExtractionMessage
  {
    public ushort Length;

    public byte Crc;

    public long Id;

    public string MessageType;

    public byte[] Data;

    /// <summary>
    /// Constructs a ExtractionMessage structure from a list of bytes
    /// </summary>
    /// <param name="binaryReader"></param>
    /// <returns></returns>
    public static ExtractionMessage BuildExtractionMessage(BinaryReader binaryReader)
      {
        const int bytesInLong = 8;
        const int bytesInShort = 2;
        const int bytesInMessageType = 4;
        const int bytesInUnitHeaderLength = 4;

        var lengthBuffer = binaryReader
                           .ReadBytes(bytesInShort)
                           .CorrectEndianness(Endianness.LittleEndian);

        var shortBuffer = BitConverter.ToUInt16(lengthBuffer, 0);

        var payloadLength = shortBuffer - bytesInLong
                          - bytesInMessageType - bytesInUnitHeaderLength;

        var crc = binaryReader.ReadByte();

        var longBuffer = binaryReader
                         .ReadBytes(bytesInLong)
                         .CorrectEndianness(Endianness.LittleEndian);

        var id = BitConverter.ToInt64(longBuffer, 0);

        lengthBuffer = binaryReader
                       .ReadBytes(bytesInShort)
                       .CorrectEndianness(Endianness.LittleEndian);

        var stringLength = BitConverter.ToUInt16(lengthBuffer, 0); //Pascal string

        var messageType = Encoding
                          .ASCII
                          .GetString(binaryReader
                                     .ReadBytes(bytesInMessageType - stringLength));

        lengthBuffer = binaryReader
                       .ReadBytes(bytesInUnitHeaderLength)
                       .CorrectEndianness(Endianness.LittleEndian);

        var unitHeaderLength = BitConverter.ToUInt32(lengthBuffer, 0); //ignored
```

```
            var data = binaryReader.ReadBytes(payloadLength);
            return new ExtractionMessage()
              {
                Length = length,
                Crc = crc,
                Id = id,
                MessageType = messageType,
                Data = data
               };
            }
        }
    }
```

Disclaimer: To the extent allowed by law, JSE Limited ("**JSE**") does not (expressly, tacitly or impliedly) guarantee or warrant the sequence, accuracy, completeness, availability, reliability or any other aspect of any of the data used in this Level 2 MITCH Tick Data, ("**Data**"), or that the Data is up to date. In addition, the sample code provided herin is provided "as is" and any express or implied warranties, including the implied warranties of merchantability and fitness for a particular purpose are disclaimed. To the extent allowed by law, neither the JSE nor any of its directors, officers, employees, contractors, agents or representatives are liable in any way to the user or to any other natural or juristic person ("**Person**") for any loss or damage as a result of any format, delay, inaccuracy, error or omission in relation to any Data or the sample code and any actions taken or not taken by or on behalf of any Person in reliance of any Data or the use of the sample code. The JSE is entitled to terminate the provision of the Data and the sample code at any time, without notice and without liability to any Person.