BME Data Feed Compression Guidelines

Document Name:	BME Data Feed Compression Guidelines
Version:	1.0
Related to:	BME Data Feed Release 5.5
Last update:	21-Oct-09

REVISION HISTORY

This section refers to the major changes of this version in comparison to the previous version.

Version	Date	Comments
1.0	21/10/2009	First cut

TABLE OF CONTENTS

1 Introduction

The BME Data Feed system establishes the possibility to receive an individually compressed BME feed connection. This is done to utilize line capacity better. By using compression it is possible to send significantly more data in a line. Data sent from customer to BME is always uncompressed.

2 Activation of compression per client request

Activation is done by sending a new, additional FID-only field in the Login Request from a user. Providing this FID indicates that the user is capable of processing compressed and uncompressed data i.e. to decompress compressed data. As this is a FID-only field the existence itself is sufficient.

Layout:

	Field Name	Value
*	**Message ID Folder**	
*	MESSAGE_ID	MID_CONNECTIVITY_REQUEST
*	REQUEST_ID	
*	**Connectivity Request Folder**	
*	CONNECTIVITY_REQUEST_SUBJECT	LOGIN
*	USER_ID	
*	PASSWORD	
	OUTBOUND_MESSAGE_KEY	
	MESSAGE_OFFSET	
	RECOVERY_DATE	
	RECOVERY_TIME	
	ENABLE_COMPRESSION	**Allows BME Data Feed to send compressed data**

Field	Data Type	Description	FID	**16bit FID**
ENABLE_COMPRESSION	**N/A (Empty)**	**Allows BME Data Feed to send compressed data**	**799**	**031F**

3 Activation of compression by BME Data Feed system

If ENABLE_COMPRESSION is missing within LOGIN request data will always be sent uncompressed to a user. If ENABLE_COMPRESSION has been sent BMEDF decides based on overall load and data amount, whether data shall be compressed or not. Compressed data can follow uncompressed data and vice versa. In case of compression one or more BMEDF messages from the original BMEDF-format are taken to one single block of data. This block is subjected to compression. This refers to the General Message Structure (cf. chapter 5.1 BME Data Feed Interface Specification). The result has to be processed like a Message Content Segment. A Message Length Segment is provided in front of the Content Segment. In this Message Length Segment the CF Bit is set. This indicates that the Message content segment contains compressed data.

4 Decompression

A user that has activated compression has to inspect the CF-Bit when parsing the Message Length Segments of a BMEDF Message. If this Bit is not set it is a single, not compressed BMEDF message. If the CF-Bit is set, the message content segment is a block of compressed data that has to be expanded before further processing. Therefore the chosen decompression method has to be implemented. This results in a serial of one or more BMEDF messages each consisting of Message Length Segment und Message Content Segment. The CF-Bit of these messages is not set.

5 Compression method

Compression method deflate/inflate algorithm of zlib-implementation version 1.2.2 is used. A free implementation as C-Source is given at http://www.gzip.org/zlib/.

An example implementation for decompression as C-Source implementation in the area of BMEDF is given in the next chapter.

6 Example

6.1 Source Code

Example.c

```
/*
 * example.c    to give an example of decompressing BMEDF output feed
 *              it reads raw BMEDF data from stdin
 *              and prints some length information
 */

#include <stdio.h>
#include <stdlib.h>
#include <memory.h>

#include "zlib.h"

// procedure to get number of Mlbs and
// message length from Mlb header
static void processMlbHeader(u_char* pcMsg, size_t* puNoMlbs, size_t* puMsgLen) {

    // how many Mlbs do we have?
    size_t uNoMlbs = ((pcMsg[0] >> 5) & 0x03) + 1;

    // calculated length of messsage
    size_t uMsgLen;

    switch (uNoMlbs) {

        case 1:
            uMsgLen = (pcMsg[0] & 0x1f);
            break;
        case 2:
            uMsgLen = ((pcMsg[0] & 0x1f) << 8) + pcMsg[1];
            break;
        case 3:
            uMsgLen = ((((pcMsg[0] & 0x1f) << 8) + pcMsg[1]) << 8) + pcMsg[2];
            break;

        default:
            fprintf(stderr,
                "unsupported number of Mlbs, 1. Mlb 0x%02x\n", pcMsg[0]);
            exit(1);
    }

    // message length = 0?
    if (!uMsgLen) {
        fprintf(stderr, "unexpected message length=0\n");
        exit(1);
    }

    printf("processMlbHeader: NoMlbs=%u MsgLen=%u\n", uNoMlbs, uMsgLen);

    *puNoMlbs = uNoMlbs;
    *puMsgLen = uMsgLen;
}
```

```
typedef int bool;   // to be compatible

// maximum possible message length including Mlb header
// (compressed or uncompressed)
#define MAX_MSG_LEN (2*1024*1024+2)

// procedure to process a single BMEDF message
// pcMsg    points to the identification folder FID
// uMsgLen  is the length of the message without Mlb header
static void processBMEMessage(u_char* pcMsg, size_t uMsgLen) {

    // now do what ever you have to do!

    printf("processBMEMessage: MsgLen %u\n", uMsgLen);
}

// read in BME Outbound Feed, compressed or not,
// and separate it into single messages

int main(int argc, char* argv[]) {

    z_stream    oStream; // used by decompression

    u_char      acMsg[MAX_MSG_LEN]; // input buffer
    size_t      uMsgLen;            // message length
    size_t      uNoMlbs;            // number of Mlbs (length of Mlb header)

    // initialize decompression
    memset(&oStream, 0, sizeof(oStream));

    if (inflateInit2(&oStream, 15) != Z_OK) {
        fprintf(stderr, "inflateInit(&oStream, 15) failed\n");
        exit(1);
    }

    /* read at minimum 3 bytes */
    while (fread(acMsg, 3, 1, stdin) == 1) {

        bool bIsNotCompressed = !(acMsg[0] & 0x80); // is compression bit reset?

        printf("data is %scompressed ", bIsNotCompressed ? "un" : "");

        processMlbHeader(acMsg, &uNoMlbs, &uMsgLen);

        // adjust input buffer and length, since we already read 3 bytes
        // and read the rest of the message into the input buffer
        if (fread(acMsg+3, uMsgLen + uNoMlbs - 3, 1, stdin) != 1) {
            perror("fread(stdin)");
            exit(1);
        }

        if (bIsNotCompressed) {

            processBMEMessage(acMsg+uNoMlbs, uMsgLen);

        } else { // input is compressed

            // buffer for uncompressed messages
```

```
static u_char   acMsgUncompr[MAX_MSG_LEN];

int     z_ret;  // returned value from decompression routines

// where the compressed data comes from
oStream.next_in     = acMsg+uNoMlbs;
oStream.avail_in    = uMsgLen;

// where the decompressed data goes to
oStream.next_out    = acMsgUncompr;
oStream.avail_out   = MAX_MSG_LEN;

z_ret = inflate(&oStream, Z_SYNC_FLUSH);

if (z_ret == Z_STREAM_END) {
    if ((z_ret = inflateReset(&oStream)) != Z_OK) {
        fprintf(stderr,
            "inflateReset(&oStream) = %d failed, error=%s\n",
            z_ret,  oStream.msg);
        exit(1);
    }
} else
if (z_ret != Z_OK) {
    fprintf(stderr,
        "inflate(&oStream, Z_SYNC_FLUSH) = %d failed, error=%s\n",
        z_ret, oStream.msg);
    exit(1);
}

// some input left over
if (oStream.avail_in) {
    fprintf(stderr,
        "inflate(&oStream, Z_SYNC_FLUSH) avail_in=%d != 0\n",
        oStream.avail_in);
    exit(1);
}

{
    // pointer to next uncompressed message
    u_char* pcMsg       = acMsgUncompr;

    // length of uncompressed data block
    size_t uMsgLenUncompr = oStream.next_out - acMsgUncompr;

    printf("uncompressed length=%u\n", uMsgLenUncompr);

    // for each message in uncompressed buffer
    // - get NoMlbs and MsgLen
    // - process message

    for (; uMsgLenUncompr > 3;
            pcMsg += uMsgLen, uMsgLenUncompr -= uMsgLen) {

        if (pcMsg[0] & 0x80) {
            fprintf(stderr,
                "unsupported double compression, 1. Mlb=0x%02x\n",
                pcMsg[0]);
            exit(1);
```

```
                    }

                    processMlbHeader(pcMsg, &uNoMlbs, &uMsgLen);

                    // skip Mlb header
                    pcMsg               += uNoMlbs;
                    uMsgLenUncompr  -= uNoMlbs;

                    if (uMsgLen > uMsgLenUncompr) {
                        fprintf(stderr,
                            "unexpected uncompressed message length %u > %u\n",
                            uMsgLen, uMsgLenUncompr);
                        exit(1);
                    }

                    processBMEMessage(pcMsg, uMsgLen);
                }

                if (uMsgLenUncompr) {
                    fprintf(stderr,
                        "unexpected leftover length %u\n",
                        uMsgLenUncompr);
                    exit(1);
                }
            }

        } // input was compressed
    } // while more input

    // EOF or ERROR from stdin?

    return 0;
}
```

6.2 Makefile

```
# makefile for SUN Solaris
# the used zlib library version 1.2.2 is installed to $(HOME)/local

INCLUDE       = -I$(HOME)/local/include
LDFLAGS       = -L$(HOME)/local/lib -lz

GNU := CC           = gcc
GNU := CFLAGS = $(INCLUDE) -g -pedantic -Wall -Wstrict-prototypes -O3 -std=c99

SOLARIS := CC       = cc
SOLARIS := CFLAGS   = $(INCLUDE) -g -fast -xcg89

all:   example

GNU:   all

SOLARIS:      all

example:      example.c
       $(CC) -o example example.c $(CFLAGS) $(LDFLAGS)
```

```
clean:
	rm -f example
```

Appendix A: References

1995-2004 Jean-loup Gailly and Mark Adler zlib version 1.2.2, October 3rd, 2004

Appendix B: BME Data Feed Documents

- BME Data Feed Interface Specifications
- BME Data Feed Fields and Products
- BME Data Feed Fields and Products Guidelines