This code can hardly work. I guess the original version was intent to build MD5 hashes and these hashes are typically numbers.

The SHA-256 hash would be by far to large for a number therefore such hashes are typically Strings (like a guid).

```
    public static String buildSHA256Hash(String content) {

     if (content == null) {
      return null;
     }
     // calculate hash
     MessageDigest messageDigest;
  try {
   messageDigest = MessageDigest.getInstance("SHA-256");
  } catch (NoSuchAlgorithmException e) {
   throw new RuntimeException("Digest Algorithm SHA-256 could not be found in this environment.", e);
  }

  final byte[] result = messageDigest.digest(content.getBytes(Charset.forName("UTF-8")));

  // convert hash to requested encoding
  return Base64.encodeToBase64String(result);
    }
```

And here the Base64 class. This is a Talend Routine you should create:

```
package routines;

import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.io.Writer;

public final class Base64 {

    static private final int    BASELENGTH           = 255;
    static private final int    LOOKUPLENGTH         = 64;
    static public final int     TWENTYFOURBITGROUP   = 24;
    static private final int    EIGHTBIT             = 8;
    static private final int    SIXTEENBIT           = 16;
    static private final int    SIGN                 = -128;
    static private final byte   PAD                  = (byte) '=';
    private static final byte[] base64Alphabet       = new byte[BASELENGTH];
    private static final byte[] lookUpBase64Alphabet = new byte[LOOKUPLENGTH];
    static private final int    TOKENS_PER_LINE      = 32;
    static private final String CR                   = "\n";

    static {
        for (int i = 0; i < BASELENGTH; i++) {
            base64Alphabet[i] = -1;
        }
        for (int i = 'Z'; i >= 'A'; i--) {
            base64Alphabet[i] = (byte) (i - 'A');
        }
        for (int i = 'z'; i >= 'a'; i--) {
            base64Alphabet[i] = (byte) ((i - 'a') + 26);
        }
```

```
        for (int i = '9'; i >= '0'; i--) {
            base64Alphabet[i] = (byte) ((i - '0') + 52);
        }
        base64Alphabet['+'] = 62;
        base64Alphabet['/'] = 63;
        for (int i = 0; i <= 25; i++) {
            lookUpBase64Alphabet[i] = (byte) ('A' + i);
        }
        for (int i = 26, j = 0; i <= 51; i++, j++) {
            lookUpBase64Alphabet[i] = (byte) ('a' + j);
        }
        for (int i = 52, j = 0; i <= 61; i++, j++) {
            lookUpBase64Alphabet[i] = (byte) ('0' + j);
        }
        lookUpBase64Alphabet[62] = (byte) '+';
        lookUpBase64Alphabet[63] = (byte) '/';
    }

    /**
     * Checks if the given string is a base64 encoded string
     * @param base64String
     * @return true if it is most likely base64 encoded
     *
* {Category} Base64
*
* {param} string(base64String) string: String.
*
* {example} isBase64(base64String) #
     */
    public static boolean isBase64(String base64String) {
        return isArrayByteBase64(base64String.getBytes());
    }

    private static boolean isBase64(byte octect) {
        return ((octect == PAD) || (base64Alphabet[octect] != -1));
    }

    /**
     * Checks if the given string is a base64 encoded array
     * @param arrayOctect
     * @return true if it is most likely base64 encoded
     *
* {Category} Base64
*
* {param} byte(arrayOctect) byte[]: String.
*
* {example} isArrayByteBase64(arrayOctect) #
     */
    public static boolean isArrayByteBase64(byte[] arrayOctect) {
        if (arrayOctect.length == 0) {
            return true;
        }
        for (int i = 0; i < arrayOctect.length; i++) {
            if (!Base64.isBase64(arrayOctect[i])) {
                return false;
            }
```

```
    }
    return true;
}

public static byte[] encode(byte[] binaryData) {
    return encode(binaryData, binaryData.length);
}

/**
 * Encodes hex octects into Base64.
 *
 * @param binaryData Array containing binary data to encode.
 * @return Base64-encoded data.
 */
private static byte[] encode(byte[] binaryData, int length) {
    final int lengthDataBits = (length << 3);
    final int fewerThan24bits = lengthDataBits % TWENTYFOURBITGROUP;
    final int numberTriplets = lengthDataBits / TWENTYFOURBITGROUP;
    byte[] encodedData = null;
    if (fewerThan24bits != 0) {
        encodedData = new byte[((numberTriplets + 1) << 2)];
    } else {
        encodedData = new byte[(numberTriplets << 2)];
    }
    byte k = 0;
    byte l = 0;
    byte b1 = 0;
    byte b2 = 0;
    byte b3 = 0;
    int encodedIndex = 0;
    int dataIndex = 0;
    int i = 0;
    for (i = 0; i < numberTriplets; i++) {
        dataIndex = i * 3;
        b1 = binaryData[dataIndex];
        b2 = binaryData[dataIndex + 1];
        b3 = binaryData[dataIndex + 2];
        l = (byte) (b2 & 0x0f);
        k = (byte) (b1 & 0x03);
        encodedIndex = (i << 2);
        final byte val1 = (((b1 & SIGN) == 0) ? ((byte) ((b1 >> 2))) : ((byte) ((b1 >> 2) ^ 0xc0)));
        final byte val2 = (((b2 & SIGN) == 0) ? ((byte) ((b2 >> 4))) : ((byte) ((b2 >> 4) ^ 0xf0)));
        final byte val3 = (((b3 & SIGN) == 0) ? ((byte) ((b3 >> 6))) : ((byte) ((b3 >> 6) ^ 0xfc)));
        encodedData[encodedIndex] = lookUpBase64Alphabet[val1];
        encodedData[encodedIndex + 1] = lookUpBase64Alphabet[(val2 | (k << 4))];
        encodedData[encodedIndex + 2] = lookUpBase64Alphabet[((l << 2) | val3)];
        encodedData[encodedIndex + 3] = lookUpBase64Alphabet[(b3 & 0x3f)];
    }
    dataIndex = i * 3;
    encodedIndex = (i << 2);
    if (fewerThan24bits == EIGHTBIT) {
        b1 = binaryData[dataIndex];
        k = (byte) (b1 & 0x03);
        final byte val1 = (((b1 & SIGN) == 0) ? ((byte) ((b1 >> 2))) : ((byte) ((b1 >> 2) ^ 0xc0)));
        encodedData[encodedIndex] = lookUpBase64Alphabet[val1];
        encodedData[encodedIndex + 1] = lookUpBase64Alphabet[k << 4];
```

```
            encodedData[encodedIndex + 2] = PAD;
            encodedData[encodedIndex + 3] = PAD;
        } else if (fewerThan24bits == SIXTEENBIT) {
            b1 = binaryData[dataIndex];
            b2 = binaryData[dataIndex + 1];
            l = (byte) (b2 & 0x0f);
            k = (byte) (b1 & 0x03);
            final byte val1 = (((b1 & SIGN) == 0) ? ((byte) ((b1 >> 2))) : ((byte) ((b1 >> 2) ^ 0xc0)));
            final byte val2 = (((b2 & SIGN) == 0) ? ((byte) ((b2 >> 4))) : ((byte) ((b2 >> 4) ^ 0xf0)));
            encodedData[encodedIndex] = lookUpBase64Alphabet[val1];
            encodedData[encodedIndex + 1] = lookUpBase64Alphabet[(val2 | (k << 4))];
            encodedData[encodedIndex + 2] = lookUpBase64Alphabet[l << 2];
            encodedData[encodedIndex + 3] = PAD;
        }
        return encodedData;
    }

    /**
     * Decodes Base64 data into octects
     *
     * @param binaryData Byte array containing Base64 data
     * @return Array containing decoded data.
     */
    public static byte[] decode(byte[] base64Data) {
        // handle the edge case, so we don't have to worry about it later
        if (base64Data.length == 0) {
            return new byte[0];
        }
        final int numberQuadruple = (base64Data.length >> 2);
        byte[] decodedData = null;
        byte b1 = 0;
        byte b2 = 0;
        byte b3 = 0;
        byte b4 = 0;
        byte marker0 = 0;
        byte marker1 = 0;
        int encodedIndex = 0;
        int dataIndex = 0;
        int lastData = base64Data.length;
        // ignore the '=' padding
        while (base64Data[lastData - 1] == PAD) {
            if (--lastData == 0) {
                return new byte[0];
            }
        }
        decodedData = new byte[lastData - numberQuadruple];
        for (int i = 0; i < numberQuadruple; i++) {
            dataIndex = (i << 2);
            marker0 = base64Data[dataIndex + 2];
            marker1 = base64Data[dataIndex + 3];
            b1 = base64Alphabet[base64Data[dataIndex]];
            b2 = base64Alphabet[base64Data[dataIndex + 1]];
            if ((marker0 != PAD) && (marker1 != PAD)) {
                b3 = base64Alphabet[marker0];
                b4 = base64Alphabet[marker1];
                decodedData[encodedIndex] = (byte) (b1 << 2 | b2 >> 4);
```

```
                decodedData[encodedIndex + 1] = (byte) (((b2 & 0xf) << 4) | ((b3 >> 2) & 0xf));
                decodedData[encodedIndex + 2] = (byte) (b3 << 6 | b4);
            } else if (marker0 == PAD) {
                decodedData[encodedIndex] = (byte) (b1 << 2 | b2 >> 4);
            } else if (marker1 == PAD) {
                b3 = base64Alphabet[marker0];
                decodedData[encodedIndex] = (byte) (b1 << 2 | b2 >> 4);
                decodedData[encodedIndex + 1] = (byte) (((b2 & 0xf) << 4) | ((b3 >> 2) & 0xf));
            }
            encodedIndex += 3;
        }
        return decodedData;
    }

    /**
     * creates a BASE64 coded String from original byte data
     * @param data uncoded raw data
     * @return String in BASE64 coded
     *
* {Category} Base64
*
* {param} byte[](data) data: String.
*
* {example} encodeToBase64String(data) #
     */
    public static String encodeToBase64String(byte[] data) {
        return toString(encode(data), false);
    }

    /**
     * creates a BASE64 coded String from original string
     * @param data clear text
     * @return String in BASE64 coded
     * @throws UnsupportedEncodingException
     *
* {Category} Base64
*
* {param} String(data) data: String.
*
* {example} encodeToBase64String(data) #
     */
    public static String encodeToBase64String(String data) throws UnsupportedEncodingException {
        return toString(encode(data.getBytes("UTF-8")), false);
    }

    /**
     * retrieves the original data from in BASE64 coded String
     * @param base64String String contains in BASE64 coded data
     * @return byte array with raw data
     */
    public static byte[] decodeFromBase64String(String base64String) {
        return decode(toBase64Binary(base64String));
    }

    /**
     * creates a String from in base64 coded data
```

```
     * @param base64Data base64 coded data
     * @return String
     */
    public static String toString(byte[] base64Data) {
        return toString(base64Data, true);
    }

    /**
     * transform array into String containing hex-tokens
     * @param base64Data BASE64-Daten
     * @param breakLines true=add line breaks at pos 33
     * @return string-presentations
     */
    public static String toString(byte[] base64Data, boolean breakLines) {
        final StringBuffer sb = new StringBuffer(base64Data.length);
        int tokenCount = 0;
        int tokenPerLine = 0;
        for (int i = 0; i < base64Data.length; i++) {
            sb.append((char) base64Data[i]);
            tokenPerLine++;
            tokenCount++;
            if ((breakLines && (tokenPerLine == TOKENS_PER_LINE)) && (tokenCount < base64Data.length)) {
                tokenPerLine = 0;
                sb.append(CR);
            }
        }
        return sb.toString();
    }

    /**
     * write base64 array in a writer, without memory consumption
     * @param base64Data
     * @param out Writer for the output
     */
    public static void write(byte[] base64Data, Writer out) throws IOException {
        int tokenCount = 0;
        int tokenPerLine = 0;
        for (int i = 0; i < base64Data.length; i++) {
            out.write((char) base64Data[i]);
            tokenPerLine++;
            tokenCount++;
            if ((tokenPerLine == TOKENS_PER_LINE) && (tokenCount < base64Data.length)) {
                tokenPerLine = 0;
                out.write(CR);
                out.flush();
            }
        }
    }

    /**
     * convert string-presentation into byte64-array
     * @param base64String
     * @return byte-array containing base64-bytes
     */
    public static byte[] toBase64Binary(String base64String) {
        // Anzahl der auszuwertenden Zeichen im String z?hlen
```

```
        final int length = base64String.length();
        int count = 0;
        char c;
        for (int i = 0; i < length; i++) {
            c = base64String.charAt(i);
            if (!Character.isWhitespace(c)) {
                count++;
            }
        }
        final byte[] array = new byte[count];
        count = 0;
        for (int i = 0; i < length; i++) {
            c = base64String.charAt(i);
            if (!Character.isWhitespace(c)) {
                array[count] = (byte) c;
                count++;
            }
        }
        return array;
    }

    /**
     * Converts a given Base64 encoded string into clear text
     * @param base64EncodedString
     * @return the clear text
     *
* {Category} Base64
*
    * {param} String(base64EncodedString) String: String.
    *
    * {example} getText(base64EncodedString) #
     */
    public static String getText(String base64EncodedString) {
        return new String(decode(toBase64Binary(base64EncodedString)));
    }

}
```