# HexEncoder

## From Crypto++ Wiki

The **HexEncoder** converts bytes to base 16 encoded data. Since the **HexEncoder** inherits from BufferedTransformation, the filter can participate in pipelining. The partner decoder is a HexDecoder. The class documentation is located at HexEncoder Class Reference (http://www.cryptopp.com/docs/ref/class_hex_encoder.html) .

## Contents

## Construction

```
HexEncoder(BufferedTransformation *attachment=NULL,
           bool uppercase=true,
           int outputGroupSize=0,
           const std::string &separator=":",
           const std::string &terminator="")
```

`attachment` is a BufferedTransformation, such as another filter or sink.

`uppercase` is an output formatting option and determines if output is uppercase or lowercase.

`outputGroupSize` is an output formatting option and determines the number of hexadecimal digit groups. For example, if `outputGroupSize = 4`, then an output string is formatted as "FFEE:DDCC:BBAA:9988:7766:5544:3322:1100".

`separator` is a string used as a delimiter. The default is a colon, and a space (with a grouping of 4) will format the string as "FFEE DDCC BBAA 9988 7766 5544 3322 1100". Encoding a Binary String for C Output shows a slightly more interesting use of the delimiter.

`terminator` adds a terminator to the output string. If `outputGroupSize = 0`, then a terminator of 'h' could be used to signify a hexadecimal string: "FFEEDDCCBBAA998877665544332211h".

## Sample Programs

## Encoding a Binary String (Non-Filter)

The following demonstrates decoding a string using `Put` and `Get`.

```
byte decoded[] = { 0xFF, 0xEE, 0xDD, 0xCC, 0xBB, 0xAA, 0x99, 0x88, 0x77, 0x66, 0x55, 0x44, 0x33, 0x22, 0x11, 0x0
string encoded;

HexEncoder encoder;
encoder.Put(decoded, sizeof(decoded));
encoder.MessageEnd();

word64 size = encoder.MaxRetrievable();
if(size)
{
    encoded.resize(size);
    encoder.Get((byte*)encoded.data(), encoded.size());
}

cout << encoded << endl;
```

A run of the above program produces the following output.

```
$ ./cryptopp-test.exe
FFEEDDCCBBAA99887766554433221100
```

## Encoding a Binary String (Filter)

Encoding a String (Non-Filter) performed a Put/Get sequence to transform the data. Crypto++ offers filters, which can simplify the process as shown below by taking advantage of Crypto++'s pipeline design.

```
byte decoded[] = { 0xFF, 0xEE, 0xDD, 0xCC, 0xBB, 0xAA, 0x99, 0x88, 0x77, 0x66, 0x55, 0x44, 0x33, 0x22, 0x11, 0x0
string encoded;

StringSource ss(decoded, sizeof(decoded), true,
    new HexEncoder(
        new StringSink(encoded)
    ) // HexEncoder
); // StringSource

cout << encoded << endl;
```

As with the previous example, a run produces the following output.

```
$ ./cryptopp-test.exe
FFEEDDCCBBAA99887766554433221100
```

## Encoding a Binary String for C Output

The following produces C array style output. Notice the the *encoded* string is "0x", and the separator is specified as ", 0x". The separator ensures all element (except the first) have a "0x". To embue the first element we a "0x", we simply set the output string *encoded* to "0x".

```
byte decoded[] = { 0xFF, 0xEE, 0xDD, 0xCC, 0xBB, 0xAA, 0x99, 0x88, 0x77, 0x66, 0x55, 0x44, 0x33, 0x22, 0x11, 0x0
```

```
string encoded = "0x";

StringSource ss(decoded, sizeof(decoded), true,
    new HexEncoder(
        new StringSink(encoded),
        true,   // uppercase
        2,      // grouping
        ", 0x"  // separator
    ) // HexDecoder
); // StringSource

cout << encoded << endl;
```

Output is as follows.

```
$ ./cryptopp-test.exe
0xFF, 0xEE, 0xDD, 0xCC, 0xBB, 0xAA, 0x99, 0x88, 0x77, 0x66, 0x55, 0x44, 0x33, 0x22, 0x11, 0x00
```

## Attaching a BufferedTransformation

Sometimes its advantageous to attach (or change an attached) BufferedTransformation on the fly. The code below attaches a StringSink at runtime.

```
byte decoded[] = { 0xFF, 0xEE, 0xDD, 0xCC, 0xBB, 0xAA, 0x99, 0x88, 0x77, 0x66, 0x55, 0x44, 0x33, 0x22, 0x11, 0x0
string encoded;

HexEncoder encoder;

encoder.Attach( new StringSink( encoded ) );
encoder.Put( decoded.data(), sizeof(decoded) );
encoder.MessageEnd();

// encoded holds the encoded string
```

## Scripting and Strings

On occasion, the mailing list will receive questions on cross-validation. For example, see *AES CTR Chiper. Different output between PHP-mcrypt and Crypto++ (http://groups.google.com/group/cryptopp-users/browse_thread/thread/73765be8f6334bbb)* . In the question, PHP-mcrypt strings are used as follows:

```
$key = "1234567890123456789012345678901234567890123456789012345678901234";
$key = pack("H".strlen($key), $key);
$iv = "1111111111222222222233333333334444444444555555555566666666667777";
$iv = pack("H".strlen($iv), $iv);
```

One of the easiest ways to avoid typos is via Copy/Paste and a HexDecoder:

```
string encodedKey = "1234567890123456789012345678901234567890123456789012345678901234";
string encodedIv = "1111111111222222222233333333334444444444555555555566666666667777";
string key, iv;

StringSource ssk(encodedKey, true /*pump all*/,
    new HexDecoder(
        new StringSink(key)
    ) // HexDecoder
); // StringSource

StringSource ssv(encodedIv, true /*pump all*/,
    new HexDecoder(
        new StringSink(iv)
    ) // HexDecoder
); // StringSource
```

After running the above code, *key* and *iv* are hexadecimal (i.e., binary) strings rather than printable (i.e., ASCII) strings. Below, GDB displays the binary digits in octal, so 0x12 = \022, 0x34 = \064, 0x56 = V (printable), 0x78 = x (printable), and 0x90 = \220.

```
Breakpoint 1, main (argc=1, argv=0x7fffffffe398) at cryptopp-test.cpp:38
(gdb) p encodedKey
$1 = {static npos = 18446744073709551615,
  _M_dataplus = {<std::allocator<char>> = {<__gnu_cxx::new_allocator<char>> = {<No data fields>}, <No data field
    _M_p = 0x614078 "1234567890123456789012345678901234567890123456789012345678901234"}}
(gdb) p key
$2 = {static npos = 18446744073709551615,
  _M_dataplus = {<std::allocator<char>> = {<__gnu_cxx::new_allocator<char>> = {<No data fields>}, <No data field
    _M_p = 0x6142a8 "\022\064Vx\220\022\064Vx\220\022\064Vx\220\022\064Vx\220\022\064Vx\220\022\064Vx\220\022\06
```

Finally, the strings *key* and *iv* can be used with Encryption or Decryption objects as follows.

```
CTR_Mode< AES >::Encryption enc;
enc.SetKeyWithIV(key.data(), key.size(), iv.data());
```

## Missing Data

Its not uncommon to send data through a pipeline and then have nothing in the sink. This is usually due to the compiler matching the wrong function. For example:

```
string source = "ABCD...WXYZ", destination;
StringSink ss(source,
    new HexEncoder(
        new StringSink(destination)
    ) // HexEncoder
); // StringSink
```

After the above code executes, `destination` will likely be empty because the compiler coerces the `HexEncoder` to a `bool` (the `pumpAll` parameter), which leaves the `StringSource`'s attached transformation `NULL`. The compiler will do so without warning, even with `-Wall -Wextra -Wconversion`. To resolve the issue, explicitly specify the **`pumpAll`** parameter:

```
string source = "ABCD...WXYZ", destination;
StringSink ss(source, true /*pumpAll*/
    new HexEncoder(
        new StringSink(destination)
    ) // HexEncoder
); // StringSink
```

Retrieved from "http://www.cryptopp.com/wiki/HexEncoder"
Categories: Sample | Filter

- This page was last modified on 18 January 2014, at 05:23.