Buffer And InputStream / OutputStream Explanation

We will do an small Hello World sample with the example of `Buffer`

Buffer.h

Initialization of Buffer.

A Buffer requires a struct and an array to allocate memory.

```
#define MY_BUFFER_LENGTH   20

static char myBufferArray[MY_BUFFER_LENGTH];

static Buffer myBuffer;
```

To initialize the buffer

```
initBuffer(&myBuffer, &myBufferArray, MY_BUFFER_LENGTH, "TEST", "TEST2");
```

Using the outputStream to fill the buffer

To append to the buffer a String, we will use the `outputStream` of the buffer. An output stream is an abstraction to append char to several objects like UART, I2C, Buffer …

You can access to the output Stream of the buffer by using the following function :

```
OutputStream* getOutputStream(Buffer* buffer)
```

OutputStream.h

`OutputStream` has several function but one is very important for us

```
void writeChar (OutputStream* outputStream, char c);
```

To append « Hello World », you can do :

```
OutputStream* outputStream = getOutputStream(buffer) ;

outputStream->writeChar(outputStream, 'H') ;

outputStream->writeChar(outputStream, 'e') ;

outputStream->writeChar(outputStream, 'l') ;

…
```

PrintWriter .h

But using it is a little bit difficult especially if you want to write integer, decimal …, that's why we can use the PrintWriter.h

Examples of function :

```
void appendString(OutputStream* stream, const char* s);
```

So to use it, we can do :

```
appendString(outputStream, « HelloWorld ») ;
```

What does it does ??

```
void appendString(OutputStream* outputStream, const char* s) {
    while (*s != '\0') {
        append(outputStream, *s++);
    }
}
```

We see that we do not write directly on a physical object like an UART or an memory object like Buffer, but in the abstraction of outputStream.

What appends to « HelloWorld » ?

'H' ➔ *(PrintWriter.h)* append(outputStream, 'H') ➔ *(OutputStream.h)* outputStream->append(outputStream, 'H') ➔ *(Buffer)* buffer->bufferWriteChar (buffer, 'H')

And this for all characters.

Why do we do all this things to write a char into a buffer ?????

If the outputStream corresponds to an UART1, the following process will happen :

'H' ➔ *(PrintWriter.h)* append(outputStream, 'H') ➔ *(OutputStream.h)* outputStream->append(outputStream, 'H') ➔ *(UART)* uart->putc1 ('H')

If we want to write a String a UART, this the same interface. We have no duplication for all « print » functions.

Let's see several `PrintWriter.h` function

```
/**
 * Send an signed int into hexadecimal value (4 chars)
 * @param value the value which must be sent
 */
void appendHex4(OutputStream* outputStream, signed int value);
/**
 * Sends the decimal value of a long.
 * @param value the value to send
```

```
 * @return the number of characters sent
 */
int appendDec(OutputStream* stream, signed long value);
/**
 * Sends the decimal value of a float.
 * @param value the value to send
 */
int appendDecf(OutputStream* stream, float value);
```

Buffer read

Buffer stores internally the value of « HelloWorld » into an array of char. To get this characters, we will use an `InputStream`, which is the mirror function of `OutputStream`.

You can access to the input Stream of the buffer by using the following function :

```
InputStream* getInputStream(Buffer* buffer)
```

```
InputStream* inputStream =  getInputStream(&buffer)
```

InputStream.h

2 functions are specially important in inputStream :

```
/**
* Function which is able to get a character from the stream.
*/
char readChar (InputStream* inputStream);


/**
* Function which is able to return if there is character to read.
*/
BOOL availableData(InputStream* inputStream);
```

So to write the content of the buffer into the UART, we can do like this :

```
InputStream* inputStream = getInputStream(&buffer) ;
while (inputStream->availableData(inputStream)) {
      char c = inputStream->readChar(inputStream);
      putc1(c) ; // Sample
}
```

This is interesting, but we can do better using the outputStream of a serial Stream and use `IOUtils.h`

IOUtils.h

```
unsigned int copyInputToOutputStream(InputStream* inputStream, OutputStream*
outputStream, filterCharFunction* filterChar, int dataCountToCopy)
```

```
copyInputToOutputStream(inputStream, serialOutputStream, NULL, COPY_ALL);
```

Will copy all data from the buffer (through his inputStream), to the serialOutputStream (which will do the putc ….)

You can filtered data, but here NULL for no filter, and COPY_ALL to handle all-datas.