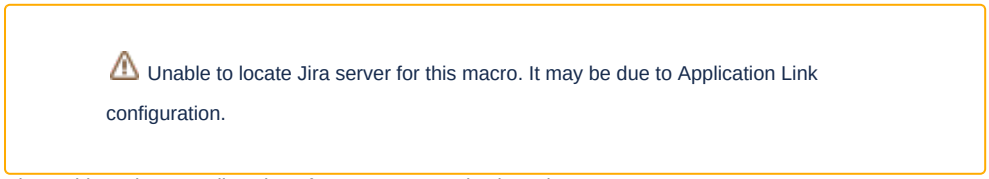
Buffer Allocations: How big is too big, how small is too small?

In examining 

I've found that under-allocation may be problematic, over-allocation of resources may also have issues.

In the original under-allocation definition of `InStreamFromByteChannel`, the `bb` buffer would only read a maximum of 4 times the `sizeHint` the class received as a parameter, which defaults to half a mebibyte. Because there is no code to determine if more bytes exist in the buffer, this configuration runs the risk of underflow errors when parsing fields.

The following code corrects this by allocating a buffer of sufficient size to store everything available in the input stream in the `bb` buffer:

```
if (count == bb.capacity) {
  // Buffer not big enough, allocate one 4 times larger and fill at offset
  var tooSmall = scala.collection.mutable.ListBuffer.empty[ByteBuffer]
  var lastWrite = 0
  while (count == bb.capacity()) {
    // Remember where we started
    bb.flip()
    bb.position(lastWrite)

    // Save old buffer and allocate anew
    tooSmall += bb
    bb = ByteBuffer.allocate(count * 4)

    // Leave space to copy the old buffers back to this one
    bb.position(count)
    lastWrite = count

    // Read in as much as possible
    count += in.read(bb)
  }
  // bb now holds enough space for the entire buffer starting from a position at the end of the previous buffer's
size
  // so copy over the other buffers in tooSmall to fill in the gap
  bb.flip()
  tooSmall.foreach(b => { bb.put(b) } )
  bb.position(0)
}
else {
   // Buffer is sufficiently sized
   bb.flip()

}
```

The problem with this solution is that it has the potential to grossly over-allocate a buffer just to store the entire contents of the stream.

What's ideally desired is something more akin to a sliding window that only loads enough data at any given time to satisfy the current field request. In other words, it should only allocate more space in a `lazy`, just in time fashion.

Unfortunately, this change would negate the test case for `TestBufferAllocations` in `sub-projects/core/srcTest/daffodil/dsom/TestBufferAllocation.scala` and it's not yet clear how to inject a custom `sizeHint` into the `InStreamFromByteChannel` in order to properly test a `lazy` evaluation of a reasonably sized buffer.

The correct solution, with a test for if the buffer contains the end of the file and `lazy` allocations to get more data when the data possible will take some time as more analysis of the code is needed.

However, a similar problem exists in the `UNICODE` / `ICU IBM` internationalization libraries with their buffer allocations. Also, the code in `fillCharBufferUntilDelimiterOrEnd` and `fillCharBufferWithPatternMatch` (`sub-projects/core/src/daffodil/grammar/Parser.scala`) are 90% similar and so code reuse and consolidation would be a goal of the allocation cleanups.