Subject: Re: Choosing the best way to go full UNICODE
Posted by cbpporter on Wed, 31 May 2017 10:26:22 GMT
View Forum Message <> Reply to Message

mirek wrote on Wed, 31 May 2017 13:07
Then the result is definitely indexable. Or am I missing something?

Yes, performance!

Unicode Strings are not indexable and indeed you can make them indexable with Vector<String> (or better).

But that is a bit of Sisyphean act. The conflict between them being non indexable and you forcing them to be indexable will result in performance and memory overhead. Like I said before, you can make a list indexable by traverse and store but you rarely would do this in practice, instead replace your random access algorithm with a linear traversal one if possible.

Now, there are some mighty complex algorithms which probably will call for this, where we will traverse and store.

But for the rest, I still think that traverse and store into a indexable structure is the worst case scenario.

You still traverse the string once, but do not store only the current code point and maybe have a few "last" positions to keep track of some other characters from previous positions.

And I would still advise the use of a StringWalker class, one that can seek to a random position, but ideally the algorithm will never use this capability! After a seek (or just initialization) it will store the current code point and a few more fields, like begin of the code point, size of the sequence. Then it as ++ and -- to go one code point up or down. This class or a separate one can do the same for glyphs.

Using such a class (or embedding this functionality directly into String) to traverse the string once from beginning to end to process each codepoint/glyph will have almost zero performance overhead.

The important part is not to make the confusion that such a class makes string index-able, i.e. only random seek if can't avoid it. And random seek with small jumps. Writing the algorithm in such a way that StringWalker is List<int>, not Vector<int>.