# Advanced Classes

## RichText

* Key functions
  + Support different underlying string representations by making RichText a template class (e.g., cse::String, std::wstring)
    - This would also support non-string containers, which might enable interesting use cases?
  + Add additional operations to RichText which respect the formatting, like insert and erase
  + Add move semantics
  + Add a UpdateContainer method which allows the caller to manipulate the underlying string/container with a lambda function, and RichText will update the formatting as needed (within reason). For example, split up format across insertions, move format on deletions, or keep format on substitutions.
* Similar classes
* Expected challenges
  + Using RichText
  + Figuring out how the user manipulated the string might be difficult, and might be an unintuitive interface for the caller.
* Error conditions
  + We should add asserts to make sure that the IndexSet and the underlying container don’t fall out of sync
* Other class projects
  + StaticString might be a good use-case for the RichText template
  + It might be interesting to use RichText<std::vector<DynamicString>>. You couldn’t use DynamicString directly since it might change length, but you could apply format properties to individual DynamicStrings

## TextSerializer

This is a new class that we think will better prepare us for the project than adding additional functionality to other classes. TextSerializer takes a RichText instance and an OutputBackend (new abstract class), and serializes the text to the output format.

* Key functions
  + The serializer will go through RichText and make whatever calls are necessary to the output backend to serialize a particular format, and then assemble it all into the final serialized string
  + TextSerializer<T> will serialize RichText<T> into T
  + Output backends can choose not to implement all methods and just output unstyled text fo certain formats
  + Design should make implementing each OutputBackend as simple as possible
  + Automatically get the correct type of value out of the FormatProperty variant based on the implementation in OutputBackend
  + Support a header and footer for formats which might require extra info to create a working document
    - We might use this to include color stylesheets for HTML, for example
* Similar classes
  + std::format might be useful in TextSerializer itself and in OutputBackend implementations
  + Emacs org-mode’s org-export
* Expected challenges
  + Choosing the order to generate output for properties might be difficult. We might need to construct a graph from IndexSet and then generate the innermost formats first with DFS, like:
    - “**hello world**”  
      
  + Currently RichText uses a string to differentiate its format properties, so exporters would need to check for each supported property individually. We might change this to an enum which only supports the properties supported in our app, which would make RichText less flexible, but would better accommodate our needs for the application.
  + If we want to support more complex formats, we might need a Document class which could store a tree of RichText, which would make TextSerializer
  + It’s not completely clear what the input and output of the methods of OutputBackend would be, and how TextSerializer should merge together the outputs of overlapping properties. For something like ANSI for example, to disable one property we have to reset all the others, then re-apply the remaining properties.
    - This might require different modes on TextSerializer
  + We’ll have to reconcile exactly overlapping properties by some sort of priority
* Error conditions
  + RichText has an FormatProperty has an invalid type or value
* Other class projects
  + Graph might be useful for determining property generation ordering
  + While on the surface level it seems like Serializer might help, TextSerializer is really more of a “dispatcher” class than an actual serializer, whereas the OutputBackend implementations (which need to be created for each output format) will be doing the hard work.