Getting started is very simple:

- attach TextWithEvents component to UI's GO
- paste text into text field
- type <a href=nameOfAction>your link</a>. Look at the end of text field in the Lighting scene for example&better explanation
- **Optional step:** adjust variables in TextWithEvents component (like curve, fonts etc.) or in generated child's TextButton component (like Transition mode – NOTE – sprite swap is unsupported, fade duration etc.)
- hit play mode
- hover cursor over your link
- ???
- **PROFIT**

Documentation

TextWithEvents - fields

- `public AnimationCurve m_CurveData = AnimationCurve.Linear(0, 0, 1, 0);`
  ```
  This field store curve data, text follow this curve;
  ```

- `public float phase = 0;`
  ```
  This field store offset in x dimension for curve; since curve data is
  normalized to [-1,1] in y dimension and [0,1] in x dimension, you could
  use trigonometry logic to make programmatic/Mecanim animation on curve;
  ```

- `public float curveMultiplier = 100;`
  ```
  This field store multiplier for y dimension in curve data, so if
  constrain [-1,1] is waaay too less, you could type e.g. 200, what will
  push limit to [-2,2] and so on. Useful for animation too;
  ```

- `public RectTransform rectT;`
  ```
  This field store reference (cache) to rectTransform on the same GO;
  ```

- `public Dictionary<string, List<Link>> linksList;`
  ```
  This field store all links, Key represent value from href attribute.
  Not intended to use by user but public is still required since this
  field is used by TextButton component.
  ```

- `public bool onlyColorChanged = false;`
  ```
  Field used for optimisation - true value prevent rebuilding and parsing
  whole text mesh when Link apply new text with changed color only;
  ```

- `public static int lastClickedIndex;`
  ```
  This field store last clicked index in string. NOTE: this is static
  field which mean that index will be shared between all instances of
  TextWithEvents. Simpliest way to know which text component was clicked
  is using OnClick event in TextButton component with parent
  TextWithEvents as argument;
  ```

TextWithEvents – properties

- `public new string text{get; set;};`
  ```
  Workaround for lack of OnValueChanged event in Text component; Use this
  if you need change visible text;
  ```

TextButton – fields

- `public TextWithEvents targetText;`
  `This field store reference to parent's TextWithEvents component;`

TextButton – methods

- `public bool IsRaycastLocationValid(Vector2 sp, Camera eventCamera);`
  `Implemented from IcanvasRaycastFilter interface.` *`Sp`* `store screen point,` *`eventCamera`* `store camera which draw UI stuff;`

- `public void WrapperForValidation();`
  `Editor only helper. Dont use this, on build player this is removed;`

Link – fields

- `public int[] linkStartAt;`
  `This field store info about start (0 index) and end (1 index) of link`

- `public TweenRunner<ColorTween> ColorTweener;`
  `This field store tweener for tweening color between state if fade duration is higher than 0; However dont change instance of this since this is initialized internally;`

Link – methods

- `public void SetColor(Color color);`
  `Set link's color immediately with given` *`color;`*

- `public void CrossFadeColor(Color targetColor, float duration, bool ignoreTimeScale, bool useAlpha, bool useRGB);`
  `Set link's color with respect to interactable state (if interactable is true, target color will be given from normal state, otherwise color from disable state will be used) and fade` *`duration`* `with given` *`targetColor`*`. Set true to` *`ignoreTimeScale`* `if you need fade` *`duration`* `independent to time scale,` *`useAlpha`* `too if you need alpha in color,` *`useRGB`* `too if you need rest channels in color;`

- `public void Reset();`
  `Use this if you need immediately reset link's color with respect to interactable state (if interactable is true, target color will be given from normal state, otherwise color from disable state will be used);`