Kara Effector 3.2:
Effector Book
Vol. II [Tomo XXVI]

Kara Effector 3.2:

En este **Tomo XXVI** continuaremos viendo las funciones de la **librería text** que empezamos en el Tomo pasado. Esta librería contiene una serie de funciones interesantes que espero que con la ayuda de esta documentación, le puedan sacar el máximo provecho a la hora de llevar a cabo sus proyectos, no solo karaokes, sino también en la edición de los subtítulos.

Librería Text [KE]:

text.to_shape(Text, Scale, Tags)

Esta función, como su nombre lo indica, convierte un string de texto a **shape**.

El parámetro **Text** es el string de texto que queremos convertir a **shape** y su valor por default es el texto por default dependiendo el **Template Type**, como ya lo hemos visto en las anteriores funciones.

El parámetro **Scale** es un número entero positivo que hace referencia a las escala de la **shape** en la que se convertirá el texto. Las opciones recomendadas son:

- 1
- 2
- 4
- 8
- 16

De hecho, la escala puede ser cualquier otra potencia de 2, como 32, 64, 128 o superiores, pero eso hará que el texto se convierta en una **shape** demasiado grande, lo que generará un código muy extenso por línea, lo que hará que el script quedé muy pesado. Su valor por default es 1.

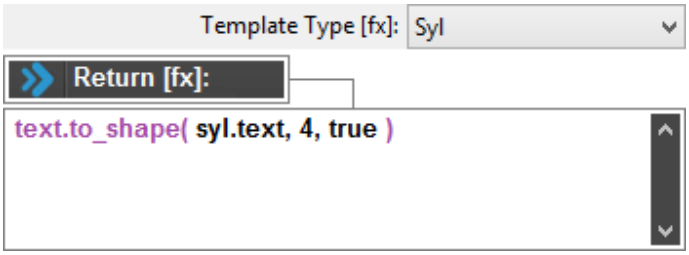
El parámetro **Tags** que añade 3 tags a la **shape** generada:

- **\an** ← Alineación respecto a la línea
- **\pos** ← Posición en la línea
- **\p** ← Proporción de la **shape**

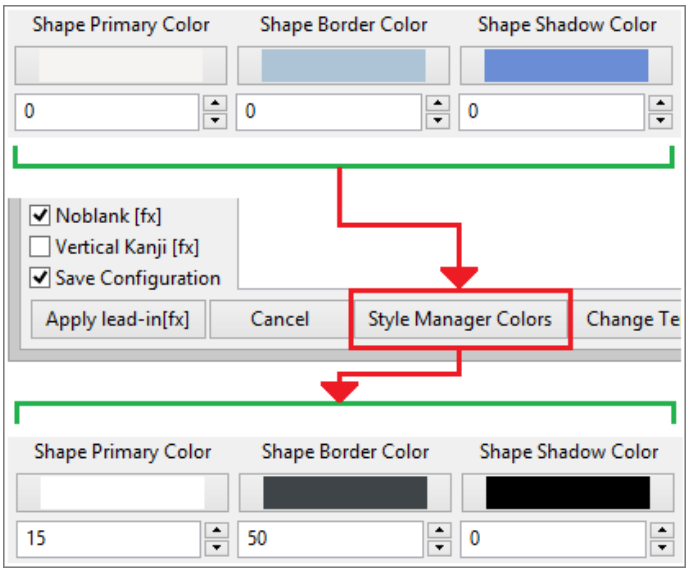
Estos tags se agregan a la **shape** con el fin ver en pantalla al texto convertido en **shape**, en su posición y alineación correcta sin necesidad de calcular otra cosa más en la **Ventana de Modificación** del **KE**. Este parámetro es un **boleano**, que de no estar en la función, se asumirá cono **nil** (nulo), o sea que no le añadirá ningún tag a la **shape** generada, es decir que **nil** es su valor por default. Y para añadir los tags mencionados, en el parámetro **Tags** debemos poner la palabra **true** (verdadero).

Ejemplo:

Llamamos a la función en **Return [fx]:**



Si aplicamos hasta este punto el ejemplo, el texto saldrá convertido en **shape** y por ende saldrá con los colores de las Shapes y no con los colores de los textos. Recordemos el procedimiento para convertir los colores a los asignados en el estilo:

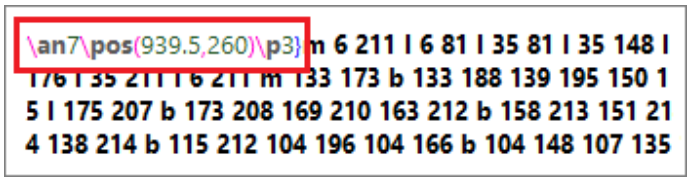


Ahora al aplicar, podremos ver el texto convertido a **shape**, pero con los colores del texto. En apariencia no se notan las diferencias entre el texto original y la shape a la que se transformó:

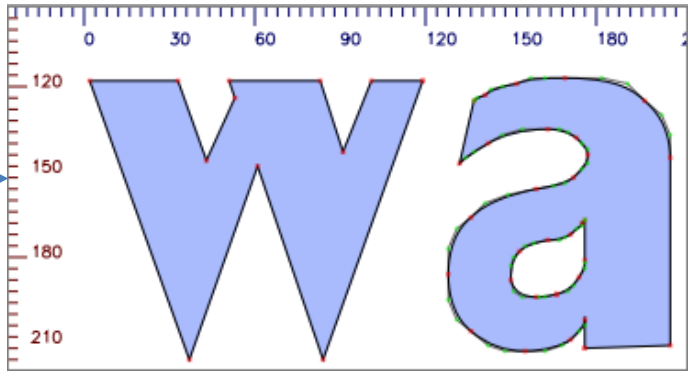
Al ver las líneas generadas, notamos que en lugar de cada Sílaba, está la **shape** de la misma:

	Dos corazones que se buscan conforman este sueño
Effector [Fx]	m 6 211 l 6 81 l 35 81 l 35 148 l 65 118 l 93 118 l 57 154
Effector [Fx]	m 85 81 l 85 211 l 56 211 l 56 206 b 54 207 52 209 49 2
Effector [Fx]	m 6 211 l 6 81 l 35 81 l 35 148 l 65 118 l 93 118 l 57 154
Effector [Fx]	m 84 144 l 84 211 l 62 211 l 62 150 b 62 146 61 143 58
Effector [Fx]	m 84 144 l 84 212 l 62 211 l 62 150 b 62 146 61 143 58
Effector [Fx]	m 84 144 l 84 212 l 62 211 l 62 150 b 62 146 61 143 58
Effector [Fx]	m 119 118 l 84 216 l 61 148 l 37 216 l 2 118 l 33 118 l 4

Y como pusimos "**true**" en el parámetro Tags, entonces la **shape** generada viene con los 3 tags que la posicionan:



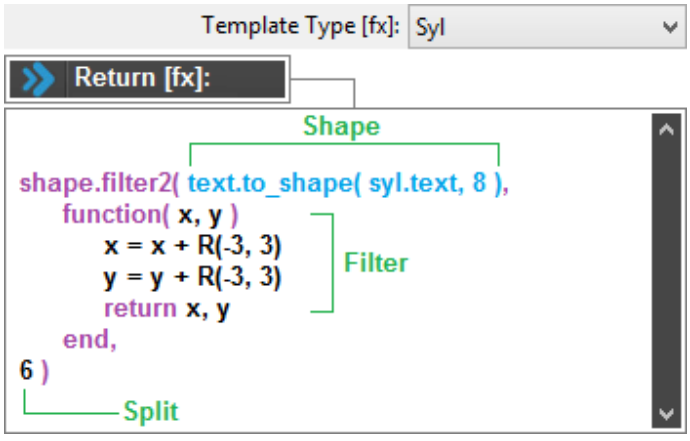
Al copiar el código de una de las Shapes generadas, y pegarlo en el **ASSDraw3**, veremos algo como esto:



La ventaja de poder convertir el texto en una **shape**, es que hecho una vez esto, le podemos aplicar cualquier función de la **librería shape**:

Ejemplo:

Para usar el texto como una shape convencional, debemos omitir al parámetro **Tags**, para que la función no retorne a los 3 tags junto con la **shape**:



shape.filter2:

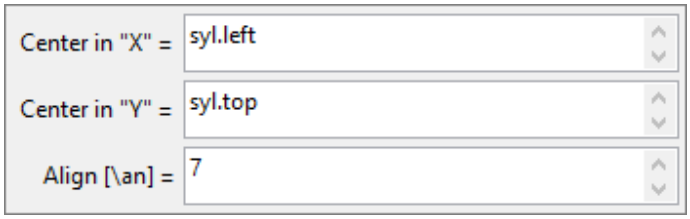
- **Shape**: el texto aumentado en un factor de 8.
- **Filter**: una función que hace que cada coordenada, tanto "x" como "y", queden desplazados de su posición original, por un valor aleatorio entre -3 y 3.
- **Split**: es el valor de la longitud de cada segmento recto en la que se dividirá la shape, en este caso al texto convertido en shape. Tiene en este ejemplo un valor de 6 px.

Y al aplicar sucederá lo siguiente:

El texto está convertido en una **shape**, que a su vez fue deformado por el filtro de la función **shape.filter2**, pero está 8 veces más grande de lo que debería estar y no tiene ni la alineación ni la posición correcta. Y para solucionar esto debemos hacer lo siguiente:

Ejemplo:

Modificamos los puntos de referencia a las posiciones **Left**, **Top** (Izquierda, Superior). Esta modificación depende del **Template Type**. Y cambiamos la alineación del texto a 7:



Ahora nos resta añadir el tag **\p** que es el que nos dará la proporción correcta de la shape para que tenga el tamaño real del texto, y lo debemos hacer así:

Return [fx]:

```
"{\\p4}" .. shape.filter2( text.to_shape( syl.text, 8 ),
        function( x, y )
            x = x + R(-3, 3)
            y = y + R(-3, 3)
            return x, y
        end,
    6 )
```

Y al aplicar veremos el texto deformado y en su posición y alineación correcta:

Y con un poco más de pixeles en la distorsión en el filtro:

En el anterior ejemplo vimos cómo el tag de proporción \p4 es el que le corresponde a una escala de 8 para el texto. La siguiente tabla nos mostrará la proporción correspondiente a cada una de las diferentes escalas:

Escala del Texto	Proporción \\p
1	1
2	2
4	3
8	4
16	5

Como las Shapes se modifican por medio de una función, de un filtro, las posibilidades son infinitas y todo dependerá de la creatividad de cada uno.

Para el siguiente ejemplo, ya no convertiremos a cada Sílaba una por una sino a la línea de texto completa, y para ello ya saber qué hacer, modificar el **Template Type**:

Ejemplo:

Recordemos que **minx** es el mínimo valor de “x” y que **w_shape** es el ancho medido en pixeles de la **shape**:

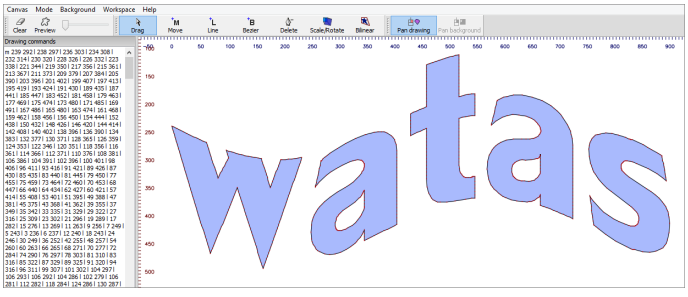
Template Type [fx]: Line

Return [fx]:

```
"{\\p4}" .. shape.filter2( text.to_shape( nil, 8 ),
        function( x, y )
            modx = (x - minx)/w_shape
            y = y + 60*sin(12*pi*modx)
            return x, y
        end,
    6 )
```

Al poner **nil** en el parámetro **Text** de **text.to_shape**, ésta lo asumirá por su valor por default, que en modo **Line** es: **line.text_stripped**

Y vemos cómo la línea de texto completa (en **shape**) se distorsiona gracias a la función del filtro:



text.do_shape(Text, Shape, Scale, Mode, Tags)

Esta función es algo similar a la anterior, y de hecho el principio es el mismo, ya que convierte el texto ingresado en **shape**. Esta función contiene dentro de sí a un filtro que hace que el texto, luego de haber sido convertido a **shape**, adopte la forma de otra **shape** ingresada en el parámetro **Shape**.

El parámetro **Text** es el texto para convertir a **shape** y su valor por default es el texto por default dependiendo el **Template Type**, como ya lo hemos visto en las anteriores funciones.

El parámetro **Shape** es la nueva figura que adoptará el texto una vez convertido en **shape**. Su valor por default es una **shape** recta equivalente a no aplicarle ningún filtro al texto, lo que en principio no tiene mucho sentido, ya que si usamos esta función es precisamente para convertir el texto en una **shape** nueva.

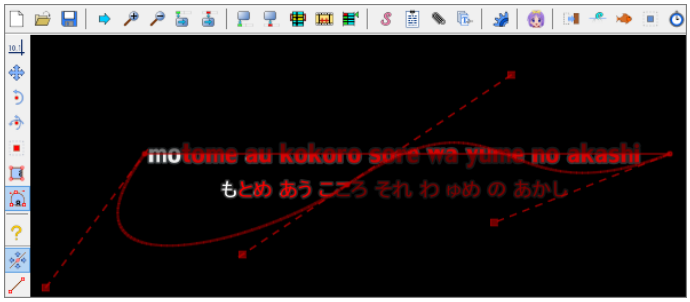
Los parámetros **Scale** y **Tags** cumplen exactamente la misma tarea que en la función anterior, que es la de darle proporción texto convertido en **shape** y añadirle los tags de posicionamiento al mismo.

El parámetro **Mode** es un número entero entre 1 y 4, para que la función haga la transformación en uno de sus cuatro modos habilitados. Su valor por default es 1.

Para los siguientes ejemplos, usaremos la función con un **Template Type: Line**, aunque con cualquiera es posible usar la función, ya que no importa el tipo de texto ingresado, ésta lo convertirá en shape.

Una forma sencilla de dibujar Shapes para esta función es usando la herramienta para recortar subtítulos en un área vectorial (**\clip**):

Y con solo dos bezier dibujé la siguiente curva usando dicha herramienta:



Este es el código de la anterior curva clip, el cual usaremos como **shape** para los siguientes ejemplos:

```
{\clip(m 204 298 b 31 532 375 474 558 352 845 160 815 418 1122 298)}
```

Y para mayor comodidad, definimos esta **shape** como una variable, que luego usaremos en la función:

Variables:

```
mi_shape = "m 204 298 b 31 532 375 474 558 352
           845 160 815 418 1122 298"
```

Ejemplo:

Usamos la función con **Mode** = 1

Template Type [fx]: Line

Return [fx]:

```
text.do_shape(
        line.text_stripped,     <-- Texto
        mi_shape,               <-- Shape
        8,                      <-- Escala
        1,                      <-- Modo
        true                    <-- Tags
)
```

Entonces la función convierte el texto a **shape** y luego lo obliga a adoptar la forma de la **shape** ingresada en el parámetro **Shape**:

Con **Mode** = 1, la función justifica al texto en el centro de la **shape**, de modo que queda repartido equitativamente, desde el centro hacia los extremos. El clip que dibujamos no se verá en la imagen, solo lo dejé para referencia.

Sin ese clip de referencia se debe ver así:

Con **Mode = 2**, el texto queda justificado a la derecha de la **shape** ingresada, según la longitud del mismo, que para este ejemplo sería: **line.width**

Con **Mode = 3**, el texto queda justificado a la izquierda de la **shape** ingresada:

Y con **Mode = 4**, el texto se deforma hasta alcanzar la longitud total de la **shape** ingresada:

Una quinta opción que tiene el parámetro **Mode** es la de ser una función filtro:

Ejemplo:

Template Type [fx]: Line

Return [fx]:

```
text.do_shape( nil, mi_shape, 8,
    function( x, y )
        y = y + R(-25, 25)
        return x, y
    end,
true )
```

Entonces la función asumirá la justificación de **Mode = 4**, y adicional a ello, aplicará el filtro a la **shape**:

Esta es otra función ideal para hacer algunos carteles pocos convencionales, como aquellos que tienen forma de arco:

O también para hacer logos que contengan texto que no esté en línea recta. Ejemplo:

Es todo por ahora para el **Tomo XXVI**. Intenten poner en práctica todos los ejemplos vistos y no olviden descargar la última actualización disponible del **Kara Effector 3.2** y visitarnos en el **Blog Oficial**, lo mismo que en los canales de **YouTube** para descargar los nuevos Efectos o dejar algún comentario. Pueden visitarnos y dejar su comentario en nuestra página de **Facebook**:

- www.karaeffector.blogspot.com
- www.facebook.com/karaeffector
- www.youtube.com/user/victor8607
- www.youtube.com/user/NatsuoKE
- www.youtube.com/user/karalaura2012