

Kara Effector 3.2:

Effector Book


Vol. II [Tomo XXXVII]



Kara Effector 3.2:

En este **Tomo XXXVII** continuaremos viendo más de los Recursos disponibles en el **Kara Effector**, 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 para la edición de las líneas de subtítulos.

Recursos [KE]:

 Recursos [KE]:

— III —

Actualización

Nuevos Modos de la Función R [KE]:

Los nuevos modos acá documentados de la función **R** están disponibles a partir de la versión **3.2.9.5** del **KE**, y tienen la finalidad de *ampliar las posibilidades a la hora de generar valores numéricos aleatorios*.

La primera actualización consiste en un tercer parámetro de la siguiente forma:

R(Val1, Val2, Step)

El parámetro **Step** es un número entero no mayor a la diferencia entre **Val2** y **Val1**, y lo que hace es marcar la distancia entre los valores que retornará la función.

Ejemplo:

R(40, 90, 10)

En este caso, lo que hace el **Step = 10** es que la función retorne un valor aleatorio entre 40 y 90, pero de 10 en 10, o sea que los posibles valores que podrían ser retornados por la función **R** son:

- 40
- 50
- 60
- 70
- 80
- 90

Ejemplo:

R(-21, 35, 7)

Ahora el **Step** tiene un valor de 7 unidades, entonces el valor retornado es un valor aleatorio entre -21 y 35, pero de 7 en 7.

El valor por default del parámetro **Step** es 1, y en el caso de usar este parámetro en la función, debe ser mayor que cero.

Usemos o no el parámetro **Step** en la función, ésta siempre retornará números enteros, y basado en esta particularidad, vienen las siguientes actualizaciones de la función:

- **Rd**
- **Rc**
- **Rm**

Rd hace que la función **R** retorne valores redondeados en décimas.

Ejemplo:

Rd(2, 10)

La función retorna un número aleatorio entre 2 y 10 con una precisión de hasta una décima:

- 2.6
- 5.4
- 9
- 3.1
- 10

Rc hace que la función **R** retorne valores redondeados en centésimas.

Ejemplo:

Rc(-3, 5)

La función retorna un número aleatorio entre -3 y 5 con una precisión de hasta una centésima:

- -1.63
- 1.48
- 0
- 0.31
- 4.92

Rm hace que la función **R** retorne valores redondeados en milésimas.

Ejemplo:

Rm(1, 2.43)

La función retorna un número aleatorio entre 1 y 2.43 con una precisión de hasta una milésima:

- 2.326
- 1
- 1.934
- 2

También podemos usar el parámetro **Step** en la función, en los tres anteriores modos documentados, lo que hará que las posibilidades aumenten aún más.

Todas estas actualizaciones de la función **R** nos servirán de apoyo para muchas de las funciones ya documentadas y para algunas más que aún no hemos visto, y que de a poco aprenderemos a sacarle el máximo provecho.

Recursos [KE]:

Actualización

Abreviaciones de algunos tags [KE]:

Las siguientes cinco abreviaciones hacen posible que al llamar un solo tag, podamos aplicar dos o más de ellos al mismo tiempo. Las abreviaciones son las siguientes:

- **\fscxy**
- **\frxy**
- **\frxz**
- **\fryz**
- **\frxyz**

\fscxy es equivalente a los tags **\fscx** y **\fscy** al tiempo, y hay dos formas diferentes de hacerlo. Ejemplo:

LUA:

- “\fscxy120” = \fscx120\fscy120
- “\fscxy80” = \fscx80\fscy80

Automation Auto-4:

- \fscxy200 = \fscx200\fscy200
- \fscxy25 = \fscx25\fscy25

O sea que siempre que usemos valores constantes en esta abreviatura, dichos valores serán los mismos para los tags a los que equivale. En el caso de que queramos poner un valor aleatorio (random), debemos poner la función junto con la abreviatura. Ejemplo:

LUA:

- “\fscxyR(100,200)” = \fscx132\fscy157
- “\fscxyRd(10,50)” = \fscx34.9\fscy18.7

Automation Auto-4:

- \fscxyRm(0,1) = \fscx0.854\fscy0.051
- \fscxyRc(23,37) = \fscx25.67\fscy33.78

Entonces, al adjuntarle a la abreviación la función random, garantizamos que a los tags equivalentes les corresponda un valor diferente para cada uno de forma aleatoria.

Las otras cuatro abreviaturas corresponden a los siguientes tags:

- **\frxy** = \frx\fry
- **\frxz** = \frx\frz
- **\fryz** = \fry\frz
- **\frxyz** = \frx\frz\frz

Y de forma similar a los ejemplos de vistos en la abreviatura **\fscxy**, son aplicables las mismas condiciones, tanto para los valores constantes, como para los valores aleatorios al usar la función **R** o alguna de sus nuevas modificaciones.

Estas cinco abreviaciones no consisten un efecto en sí mismas, pero nos ayudan a ahorrar trabajo en los mismos. Disponibles en el **KE** versión **3.2.9.5** o superior.

Recursos [KE]:

Actualización

tags progresivos [KE]:

Esta es otra de las actualizaciones que se pueden usar a partir de la versión **3.2.9.5** del **Kara Effector**, y lo que hace es interpolar todos los valores dentro de una tabla adjunta a un tag, respecto al **module1** y al **module**, según así lo dispongamos.

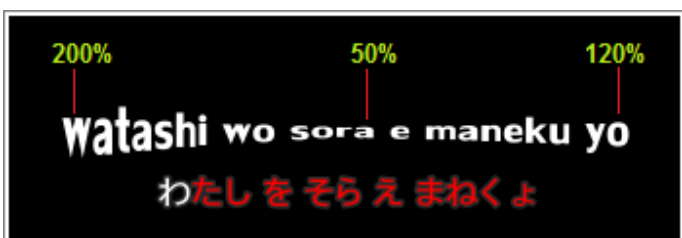
Ejemplo:

Template Type [fx]: Char

Add Tags Language: Lua

"\fscy{200, 50, 120}"

Dado que es un **Template Type: Char**, cada una de los caracteres de la línea tomarán los valores en el tag `\fscy` empezando desde el 200%, pasando por el 50% hasta llegar al 120%:



La interpolación de los diferentes porcentajes en la escala de la Font respecto al eje "y" se hizo en relación al **module1**.

Ejemplo:

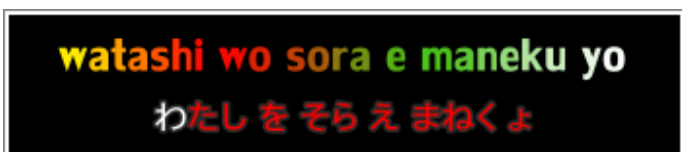
Template Type [fx]: Char

Add Tags Language: Automation Auto-4

\1c{"&H00FFFF&", "&H0000FF&", "&H16C047&", "&HFFFFFF&"}

Los colores de la tabla adjunta son:

- **&H00FFFF&** = Amarillo
- **&H0000FF&** = Rojo
- **&H16C047&** = Verde
- **&HFFFFFF&** = Blanco



De los dos ejemplos vistos anteriormente podemos deducir que al adjuntar normalmente la **tabla** al tag, los valores de la misma se interpolarán respecto al **module1**. Ahora para que los valores de la tabla se interpolen respecto al **module** (respecto al **loop**) debemos añadir el signo menos (-) justo en medio del tag y la **tabla**:

Ejemplo:

Template Type [fx]: Syl

Pos in "X" = fx.pos_x + math.polar(360*j/maxj, 70, "x")

Pos in "Y" = fx.pos_y + math.polar(360*j/maxj, 70, "y")

loop = 12

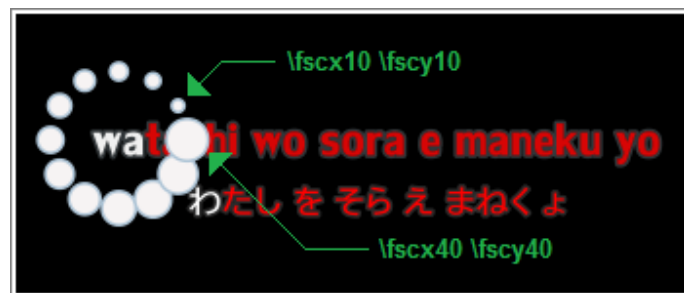
Return [fx]: shape.circle

Y en **Add Tags** ponemos:

Add Tags Language: Automation Auto-4

\fscxy-{10, 40}

Aplicando, con los tiempos de un **hi-light**, veremos esto:



Al usar una de las abreviaciones vistas hace poco se ahorra aún más trabajo, dado que:

\fscxy-{10, 40} = \fscx-{10, 40}\fscy-{10, 40}

Entonces los valores de la **tabla** se aplicarán a dichos tags de forma progresiva, interpolándose desde el 10% hasta llegar al 40%, como se puede ver en la imagen anterior.

A este mismo ejemplo le podemos aplicar la cantidad de tags progresivos, abreviaciones o normales que queramos, todo depende de los resultados que estemos necesitando, ejemplo:

\fscxy-{10, 40}\1c{"&H00FFFF&", "&H0000FF&"}

Las combinaciones posibles son infinitas, solo resta probar.

Recursos [KE]: Actualización

Nuevas opciones en la función tag.oscill

Esta es una de las funciones más amplia y completa de la librería **tag** y aun así se puede seguir expandiendo. Las actualizaciones que veremos a continuación nos dan más posibilidades en el tercer parámetro de la función **tag.oscill**, que es donde añadimos los tags.

Ejemplo:

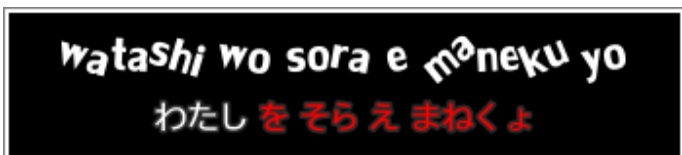
Add Tags Language: Automation Auto-4

```
tag.oscill( fx.dur, 200, "\\frz( R(-45, 45) )" )
```

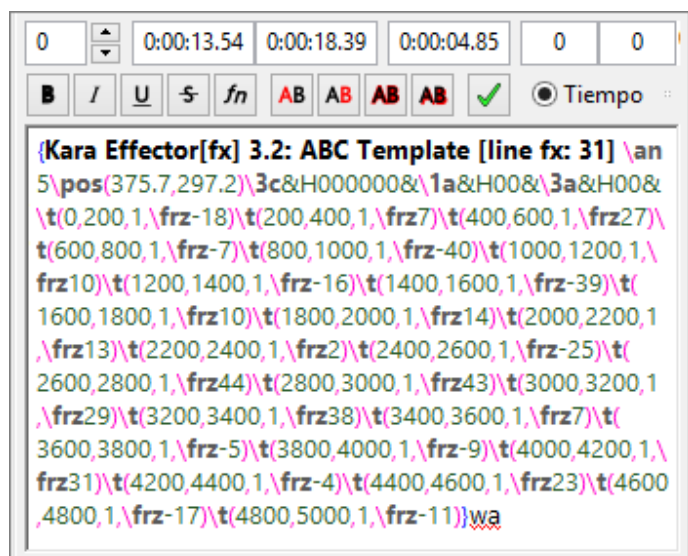
Lo que debemos hacer es adjuntar unos paréntesis luego del tag para poder poner dentro de él la función **R** con los parámetros que queramos.

Para este ejemplo en particular la función generará una serie de transformaciones de 200 ms de duración y en cada una de ellas aparecerá el tag **\frz** con un valor aleatorio entre -45° y 45°. Es decir que esta actualización hace que la función **R** se lleve a cabo una y otra vez dentro de cada una de las transformaciones que genera la función **tag.oscill**:

Una vez aplicado el efecto veremos una sucesión de giros respecto al eje "z", en transformaciones de 200 ms de duración:



Así se vería una de las líneas generadas:



Ejemplo:

Add Tags Language: Automation Auto-4

```
tag.oscill( fx.dur, 200, "\\fax( Rc(-0.2, 0.2) )\\fay( Rc(-0.2, 0.2) )" )
```

Entonces, entre comillas simples o dobles ponemos el o los tags que deseamos, y adjunto a cada uno de ellos abrimos paréntesis para que dentro de ellos pongamos la función **R** con los valores que más se adecuen al efecto necesitado.

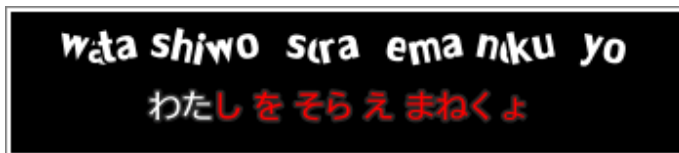
Y la segunda actualización de la función **tag.oscill** consiste en poder utilizar dentro de los tags al contador "i" de las transformaciones generadas al aplicar. Este contador parte desde 0 y va aumentando progresivamente de uno en uno según las transformaciones que genere la función.

Ejemplo:

Add Tags Language: Automation Auto-4

```
tag.oscill( fx.dur, 320, "\\fr( 20*(-1)^(i + syl.i) )" )
```

En este caso el contador "i" hará que los valores se vayan alternando:



Es todo por ahora para el Tomo XXXVII. 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