

# Kara Effector 3.2



**Kara Effector 3.2**  
**Effector Book [Vol. 1]**  
**Tomos de 1 a 20**

**30 de Mayo de 2014**

<http://www.karaeffector.blogspot.com>

<http://www.facebook.com/karaeffector>

<http://www.youtube.com/user/victor8607>

<http://www.youtube.com/user/NatsuoDCE>



## Kara Effector - Effector Book:

# Kara Effector 3.2: Effector Book Vol. I [Índice]

## Effector Book [Tomo I]:

	Pág.
• Presentación e Introducción	1
• Descargar Kara Effector 3.2	2
• Instalar Kara Effector 3.2	2
- Primer método	2
- Segundo método	3
• Usar Kara Effector 3.2	3
<b>Ventana de Inicio</b>	3
- Apply to Style	3
- Selection Effect	4
- Using Tags Filter	4
- Line Comment	5
- Save Configuration	5
- Style Manager	5
- Cancel	6
- Apply Selection	6
• Instalar Nuevos Efectos	6

## Effector Book [Tomo II]:

	Pág.
<b>Ventana de Inicio</b>	
- Modify	1
• <b>Ventana de Modificación</b>	2
<b>Template Type</b>	2
- Line	3
- Syl	3
- Furi	3
- Char	3
- Translation Line	3

	Pág.
- Translation Word	3
- Translation Char	3
<b>Tiempos del Efecto</b>	
- Line Start Time	3
- Line End Time	3
<b>Centros en los Ejes</b>	
<b>[Posición]</b>	
- Center in "X"	4
- Center in "Y"	4
<b>Alineación de la Líneas fx</b>	
- Align [\an]	4
<b>Capa de las Líneas fx</b>	
- Layer	4
<b>Posición y Movimiento</b>	
- Pos in "X"	4
- Pos in "Y"	4
- Times Move	5
<b>Objetos Karaoke</b>	
- Return [fx]	5
<b>Repeticiones</b>	
- Loop	5
<b>Tamaño</b>	
- Size	5
<b>Colores y Transparencias</b>	
<b>[Shapes]</b>	
- Primary Color Shape	5
- Border Color Shape	5
- Shadow Color Shape	5
<b>Agregar Nuevos Tags</b>	
- Add Tags	5
<b>Lenguaje de los Tags Agregados</b>	
- LUA	6
- Automation Auto-4	6
<b>Generar y Guardar</b>	
<b>Efectos Nuevos</b>	
- Print Config [fx]	6
- Template Folder [fx]	7
- New [fx] Name	7

## Kara Effector - Effector Book:

	Pág.
<b>Funciones y Variables</b>	
- var.once	8
- var.line	8
- var.syl	8
- var.furi	8
- var.word	8
- var.char	8
- var.loop	8
<b>Configuraciones “Check”</b>	
- Nobalnk [fx]	9
- Vertical Kanji [fx]	9
- Save Configuration	9
<b>Botones de Ejecución</b>	
- Apply [fx]	9
- Cancel	9
- Style Manager Colors	9
- Change Template Type	9
- Back <	9

	Pág.
- l.align	2
- l.outline	2
- l.shadow	2
- l.scale_x	2
- l.scale_y	2
- l.angle	2
- l.spacing	2
<b>Tamaño y Posición</b>	
- l.width	3
- l.left	3
- l.center	3
- l.right	3
- l.height	3
- l.top	3
- l.middle	3
- l.bottom	3
- l.descent	3
<b>Tiempos y Escritura</b>	
- l.start_time	3
- l.end_time	3
- l.text	4
- l.text_stripped	4

## Effector Book [Tomo III]:

	Pág.
• <b>Librería “l” [KE]</b>	1
<b>Editor de Estilos [Aegisub]</b>	2
- l.fontname	2
- l fontsize	2
- l.color1	2
- l.color2	2
- l.color3	2
- l.color4	2
- l.alpha1	2
- l.alpha2	2
- l.alpha3	2
- l.alpha4	2
- l.margin_l	2
- l.margin_r	2
- l.margin_v, _t, _b	2

• <b>Librería “fx” [KE]</b>	
<b>Ventana de Inicio</b>	4
- text.color1	4
- text.color2	4
- text.color3	4
- text.color4	4
- text.alpha1	4
- text.alpha2	4
- text.alpha3	4
- text.alpha4	4
<b>Ventana de Modificación</b>	4
- fx.start_time	4
- fx.end_time	5
- fx.fun_x	5

## Kara Effector - Effector Book:

	Pág.
- fx.fun_y	5
- fx.domain_i	5
- fx.domain_f	5
- fx.layer	5
- fx.maxloop_fx	5
- fx.sizex, fx.sizey	5
- shape.color1	5
- shape.color3	5
- shape.color4	5
- shape.alpha1	5
- shape.alpha3	5
- shape.alpha4	5
- maxj	8
- fx.loop_v	8
- fx.loop_h	8
- fx.loop_3	8

## Effector Book [Tomo IV]:

	Pág.
• Librería “fx” [KE]	1
Ventana de Modificación	1
- fx.center_x	1
- fx.center_y	1
- fx.scale_x	1
- fx.scale_y	1
- fx.align	1
- fx.move_x1, _x2, _x3, _x4	1
- fx.move_y1, _y2, _y3, _y4	1
- fx.movet_i, fx.movet_f	1
- xres	6
- yres	6
- ratio	6
- frame_dur	6
- line.i	6
- line.n	6
- line.index ( ii )	6
- text.color	6
- text.alpha	6

	Pág.
- test.style	7
- text.alpha0	7
- shape.color	7
- shape.alpha	7
- shape.style	7
- shape.alpha0	7
- module	7
- module1	7
- module2	7

## Effector Book [Tomo V]:

	Pág.
• Librería “math” [LUA]	1
- math.abs(x)	1
- math.acos(x)	1
- math.asin(x)	1
- math.atan(x)	1
- math.atan2(y, x)	1
- math.ceil(x)	1
- math.cos(x)	1
- math.cosh(x)	1
- math.deg(x)	1
- math.exp(x)	2
- math.floor(x)	2
- math.mod(x, y)	2
- math.log(x)	2
- math.log10(x)	2
- math.max(x, ...)	2
- math.min(x, ...)	2
- math.pi	2
- math.pow(x, y)	2
- math.rad(x)	2
- math.random(m, n)	2
- math.sin(x)	2
- math.sinh(x)	2
- math.sqrt(x)	2
- math.tan(x)	2

## Kara Effector - Effector Book:

	Pág.
- math.tanh(x)	2
• <b>Librería “math” [KE]</b>	2
- math.R(m, n)	2
- math.Rfake(m, n, i)	2
- math.round(n, dec)	2
- math.distance(x1, y1, x2, y2)	2
- math.angle(x1, y1, x2, y2)	2
- math.polar(a, radius, Return)	3
- math.point(n, x, y, xi, yi, xf, yf)	3
- math.factk(n)	3
- math.bezier(Return, ...)	3
 <b>Tablas</b>	 3
• <b>Librería “table” [LUA]</b>	5
- table.concat(t, separador)	5
- table.insert(t, i, elemento)	5
- table.maxn(t)	5
- table.remove(t, i)	5
- table.sort(t)	5
 <b>Librería “table” [KE]</b>	 5
- table.inside(t, e)	5
- table.index(t, e)	5
- table.compare(t1, t2)	6
- table.disorder(t)	6
- table.concat1(t, ...)	6
- table.concat2(t, ...)	6
- table.replay(n, ...)	6
- table.count(t, e)	7
- table.pos(t, e)	7
- table.retire(t, ...)	7
- table.inserttable(t1, t2, i)	7
- table.reverse(t)	7
- table.cyclic(t)	7
- table.op(t, mode)	7

## Effector Book [Tomo VI]:

	Pág.
• <b>Objetos</b>	1
- <b>Objetos: String</b>	1
- <b>Objetos: Value</b>	2
• <b>Librería “string” [LUA]</b>	3
- string.byte(s)	3
- string.char(...)	3
- string.find(s, ptr)	3
- string.format(s, ...)	3
- string.gsub(s, ptr, replace, n)	3
- string.len(s)	4
- string.lower(s)	4
- string.rep(s, n)	4
- string.reverse(s)	4
- string.sub(s, i, j)	4
- string.upper(s)	4
• <b>Abreviaturas de Funciones</b>	5
- pi	5
- sin	5
- cos	5
- tan	5
- asin	5
- acos	5
- atan	5
- sinh	5
- cosh	5
- tanh	5
- log	5
- ln	5
- abs	5
- floor	5
- ceil	5
- deg	5
- rad	5
- r	5
- R	5
- Rf	5

## Kara Effect - Effector Book:

	Pág.
- Rand	5
- format	5
- F	5
• Teoría del Color RGB	5
• Teoría del Color HSV	7

## Effector Book [Tomo VII]:

	Pág.
• Librería "random" [KE]	2
- random.color	2
- random.colorvc	2
- random.alpha	2
- random.alphava	2
- random.e	2
- random.unique	2

## Effector Book [Tomo VIII]:

	Pág.
• Funciones Compartidas	
Automation Auto-4 y KE	1
- ass_color	2
- ass_alpha	2
- ass_style_color	2
- alpha_from_style	2
- color_from_style	2
- HSV_to_RGB	2
- HSL_to_RGB	2
- interpolate	2
- interpolate_color	2
- interpolate_alpha	2
- maxloop(new_loop)	2
- relayer(new_layer)	2
- retime	2

	Pág.
• Función retime – modos [KE]	1
- preline	3
- line	3
- postline	3
- preword	3
- start2word	3
- word	3
- word2end	3
- postword	3
- presyl	3
- start2syl	3
- syl	3
- syl2end	3
- postsyl	3
- prefuri	3
- start2furi	3
- furi	3
- furi2end	3
- postfuri	3
- prechar	3
- start2char	3
- char	3
- char2end	3
- postchar	3
- linepct	3
- wordpct	3
- sylpct	3
- furipct	3
- charpct	3
- set o abs	3

## Effector Book [Tomo IX]:

	Pág.
• Librías Funciones de Tiempo	1
- HMS_to_ms	1
- ms_to_HMS	2

## Kara Effector - Effector Book:

	Pág.
- time_to_frame	2
- frame_to_ms	2
- frame_to_HMS	3
- time_mid1	3
- time_mid2	4
- time_li	4
- time_lo	4
• <b>Librería “tag” [KE]</b>	4
- tag.clip	4

## Effector Book [Tomo X]:

	Pág.
• <b>Librería “tag” [KE]</b>	1
- loop_h	2
- tag.iclip	3
- tag.clip2	3
- tag.iclip2	3
- tag.movevc	3
- tag.movevci	6
- tag.only	6

	Pág.
- color.to_RGB	1
- color.to_HVS	1
- color.vc	2
- alpha.va	2
- color.r	2
- alpha.r	2
- color.rc	2
- color.gradientv	2
- alpha.gradientv	3
- color.gradienth	3
- alpha.gradienth	3
- color.vc_to_c	4
- alpha.va_to_a	4
- color.c_to_vc	4
- alpha.a_to_va	4
- color.interpolate	4
- alpha.interpolate	4
- color.delay	4
- alpha.delay	5
- color.movedelay	5
- color.set	6
- alpha.set	8

## Effector Book [Tomo XI]:

	Pág.
• <b>Librería “tag” [KE]</b>	1
- tag.set	1
- tag.glitter	3
- tag.oscill	4
- tag.ipol	7

## Effector Book [Tomo XIII]:

	Pág.
• <b>Librerías “Color” y “Alpha” [KE]</b>	1
- color.mask	1
- alpha.mask	2
- color.movemask	2
- alpha.movemask	3
- color.setmovemask	3
- color.movemaskv	4
- alpha.movemaskv	4
- color.masktable	4
- alpha.masktable	4
- color.module	4

## Effector Book [Tomo XII]:

	Pág.
• <b>Librerías “Color” y “Alpha” [KE]</b>	1

## Kara Effector - Effector Book:

	Pág.
- alpha.module	5
- color.module1	5
- alpha.module1	6
- color.module2	6
- alpha.module2	6

## Effector Book [Tomo XVII]:

	Pág.
• Librería “shape” [KE]	1
- shape.array	1
- shape.animated	3
- shape.animated2	6

## Effector Book [Tomo XIV]:

	Pág.
• Librería “shape” [KE]	1
- Shapes Prediseñadas del KE	2
- shape.rotate	5
- shape.reflect	6

## Effector Book [Tomo XVIII]:

	Pág.
• Librería “shape” [KE]	1
- shape.config	1
- shape.incenter	5
- shape.centerpos	5
- shape.trajectory	6

## Effector Book [Tomo XV]:

	Pág.
• Librería “shape” [KE]	1
- shape.displace	2
- shape.origin	2
- shape.oblique	3
- shape.reverse	4
- shape.ratio	4
- shape.size	5
- shape.info	6

## Effector Book [Tomo XIX]:

	Pág.
• Librería “shape” [KE]	1
- shape.Ltrajectory	1
- shape.Ctrajectory	2
- shape.Rtrajectory	3
- shape.Strajectory	3
- shape.movevc	3
- shape.moveci	6
- shape.multi1	6

## Effector Book [Tomo XVI]:

	Pág.
• Librería “shape” [KE]	1
- shape.array	2

## Effector Book [Tomo XX]:

	Pág.
• Librería “shape” [KE]	1
- shape.multi2	1
- shape.multi3	2
- shape.multi4	3

## Kara Effector - Effector Book [Tomo I]:

---

# Kara Effector 3.2: Effector Book Vol. I [Tomo I]

---

# Kara Effector 3.2:

El **Kara Effector** es un archivo **LUA** que tiene la finalidad de hacer más simple la labor de hacer **Efectos Karaoke** y de **Líneas de Traducción**. Contiene Efectos prediseñados que pueden ser aplicados directamente o modificados según uno quiera. Dichos Efectos prediseñados pueden tomarse como un punto de partida para crear nuevos y diferentes Efectos.

El **Kara Effector** nos da la posibilidad de aplicar uno o más Efectos a la vez las veces que queramos ya que posee una amplia librería con variables y funciones que pueden hacer prácticamente cualquier tipo de Efecto.

Se recomienda evitar cualquier tipo de modificación tanto en las Librerías de **Kara Effector** como en su archivo principal, a menos que sea una recomendación nuestra o estén completamente seguros de los que están haciendo, ya que lo más seguro es que deje de funcionar de manera correcta.

El **Kara Effector** ofrece dos tipos de lenguajes en los cuales desarrollar y modificar los Efectos, se pueden hacer en lenguaje **LUA** si ya lo dominan o en **Automation Auto-4** que es el lenguaje de los efectos que normalmente se hacen en el **Aegisub**.

El **Kara Effector** cuenta con múltiples herramientas que harán de la experiencia de hacer Efectos en **Aegisub** una tarea simple, al mismo tiempo que irán aprendiendo a hacer sus propios y originales Efectos y a la vez ingeniosas y elegantes combinaciones.

Todo aquello que deban saber del **Kara Effector** estará consignado en este libro, lo mismo que en un par de canales en **You Tube** que hemos dispuesto con dicho fin, lo mismo que en su **Blog Oficial**:

<http://www.KaraEffector.blogspot.com>

<http://www.facebook.com/KaraEffector>

<http://www.youtube.com/user/Victor8607>

<http://www.youtube.com/user/NatsuoDCE>

El Autor: **Vict8r Kara**

## Kara Effector - Effector Book [Tomo I]:

# Descarga Kara Effector 3.2

En la anterior versión del **Kara Effector** (3.1), en link de descarga solo era dado a aquellos que enviaban un correo privado en mi canal de **YT**. En esta ocasión la descarga será totalmente libre, ya sea desde el **Blog Oficial** o en el siguiente link:

[Kara Effector 3.2](#)

# Instala Kara Effector 3.2

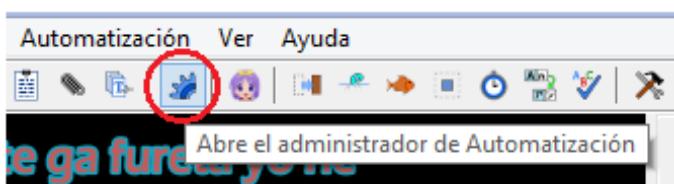
Hay dos formas distintas de hacer uso del **Kara Effector** en el **Aegisub**. El **Kara Effector 3.2** consta de 3 archivos **LUA** y un archivo **.ass** que es simplemente un karaoke test:

Effector-3.2.lua	330.475
Effector-utils-lib-3.2.lua	194.959
Effector-newfx-3.2.lua	8.923
Effector-3.2-test.ass	5.251

### PRIMER MÉTODO DE INSTALACIÓN:

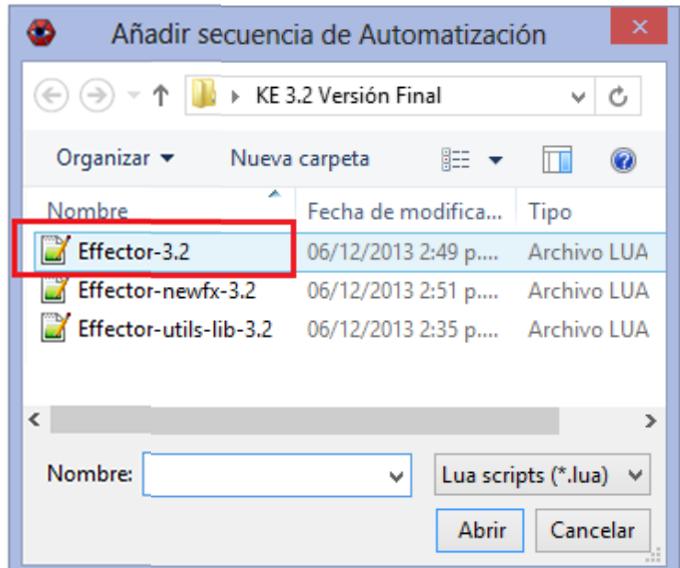
Al descargar el archivo, descomprimimos la carpeta y la ubicamos en la posición que más nos guste dentro de nuestro equipo, por lo general la ubicación recomendada es en el **ESCRITORIO**, pero cualquier lugar está bien, siempre que no olviden el lugar en donde la pegaron.

El primer paso es abrir el **Aegisub** y pulsamos el siguiente botón:

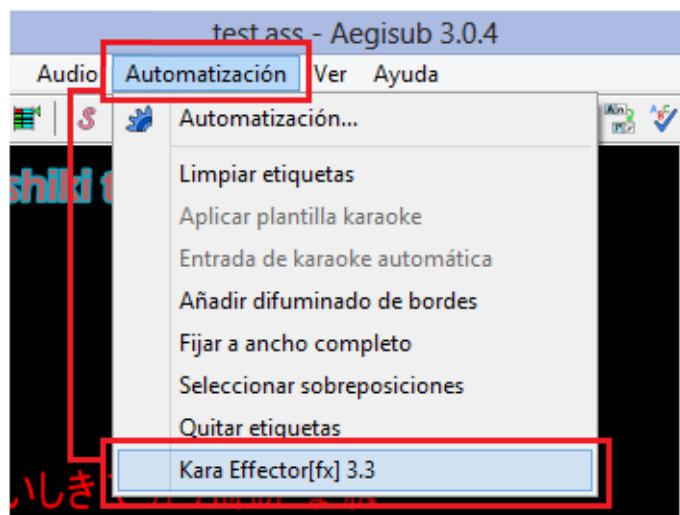


Y luego el botón donde pone: "Añadir"

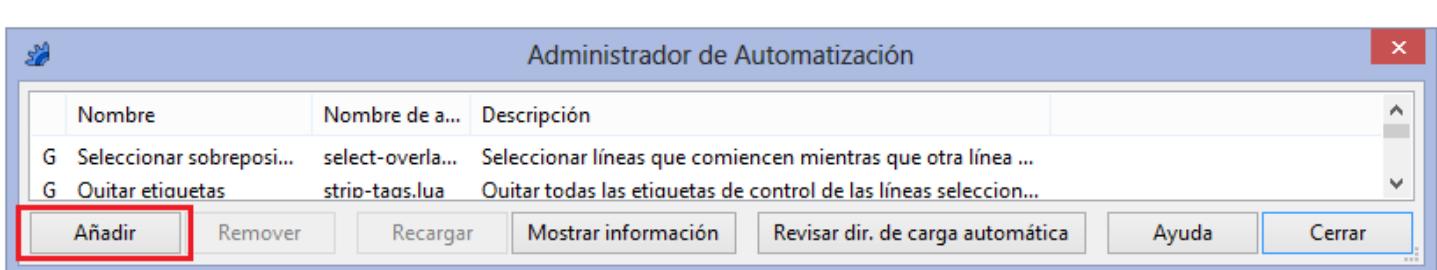
Una vez que pulsamos el botón "Añadir" se abre una ventana en donde podremos buscar la carpeta del **Kara Effector**, y de los tres archivos **LUA** solo cargaremos el archivo que pone: **Effector-3.2.lua**



Hecho esto, ya podemos visualizarlo en el **Aegisub** y empezar a usar el **Kara Effector**:



Este es la primera forma de instalar en **Kara Effector 3.2** en el **Aegisub**, a continuación mostraré la segunda forma de hacerlo ya que a algunos les parece molesto repetir este procedimiento cada vez que queramos usarlo.

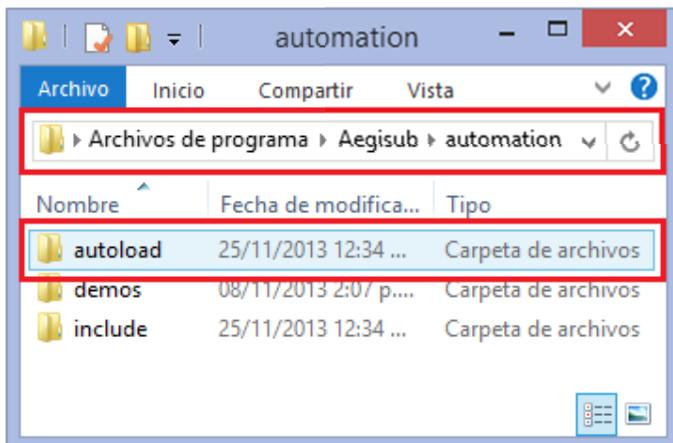


## Kara Effector - Effector Book [Tomo I]:

### SEGUNDO MÉTODO DE INSTALACIÓN:

Este segundo método es la forma de instalarlo de forma permanente en el **Aegisub** y consta de dos simples pasos que mostraré a continuación:

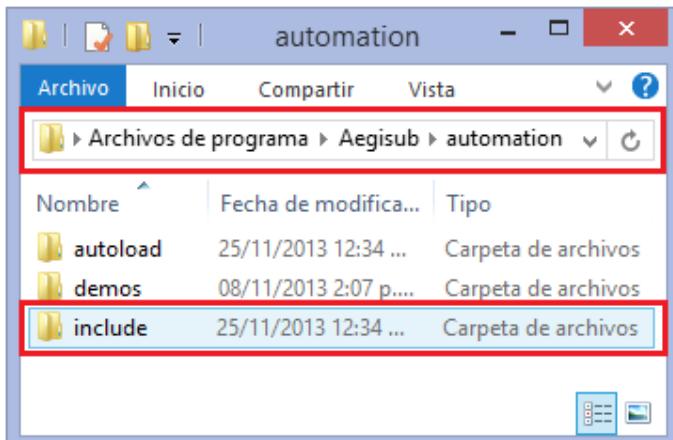
1. Se copia el archivo **Effector-3.2.lua** y se pega en la siguiente carpeta del Aegisub:



2. Se copian los otros dos archivos LUA del Kara Effector:

**Effector-utils-lib-3.2.lua**  
**Effector-newfx-3.2.lua**

Y se pegan en la siguiente carpeta del Aegisub:



Con estos dos simples pasos ya queda instalado el **Kara Effector** de forma permanente en el **Aegisub**, para poder hacer uso de él las veces que lo necesitemos para hacer nuestros Efectos Karaokes, o aplicar algún Efecto o Estilo especial en nuestras Líneas de Subtítulos.

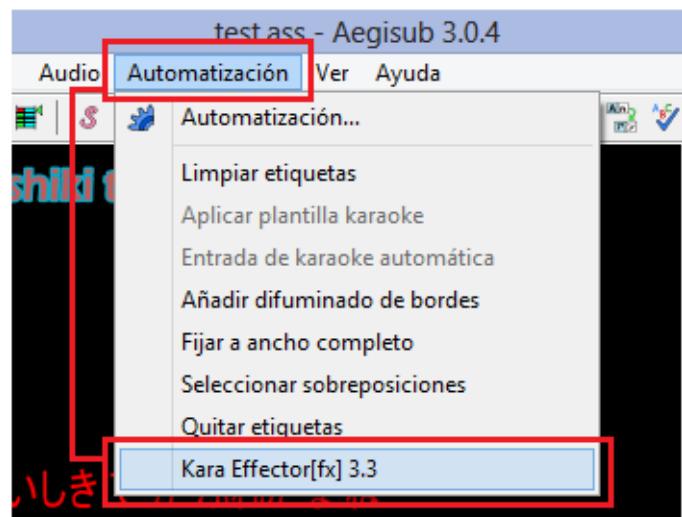
Ya es decisión de cada uno la forma en la que instalará el **Kara Effector** en el **Aegisub**, pero sea cual sea el método que elijan, éste funcionará de igual forma en ambos casos.

A partir de este punto ya podemos empezar a ver al **Kara Effector** como una de las opciones de Efectos del **Aegisub** automatizados y solo es cuestión de elegir los Efectos.

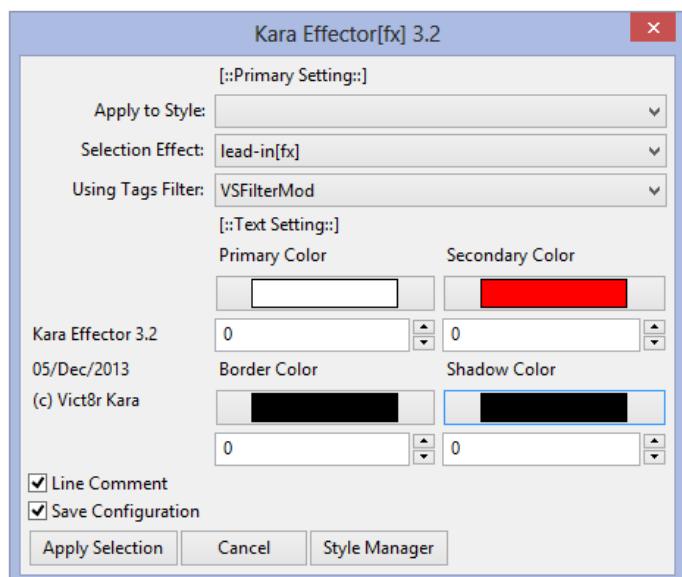
## Usa Kara Effector 3.2

Para empezar a usar el **Kara Effector 3.2** debemos abrir el **Aegisub**, y en comparación con el **Kara Effector 3.1** que solo podía ser usado en las versiones 2.1.8 y 2.1.9 del **Aegisub**, la versión 3.2 es compatible con todas las versiones del **Aegisub** existentes hasta la actualidad. (**Aegisub 3.1.3**)

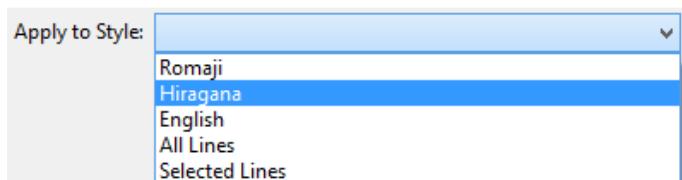
Una vez hayamos abierto el **Aegisub**, pulsamos el botón de **Automatización**:



Y se abrirá la Ventana de Inicio del **Kara Effector**:



### Apply to Style:



## Kara Effector - Effector Book [Tomo I]:

La opción “**Apply to Style**” es una ventana desplegable (**dropdown**), que al pulsarla saldrán los listados de los estilos que contiene el archivo .ass que abrimos en el **Aegisub**, excepto los dos últimos de esa lista. En la imagen anterior, solo los tres primeros de esa lista son los estilos de las líneas del archivo .ass, o sea:

Estilo: **Romaji**

Estilo: **Hiragana**

Estilo: **English**

Se puede ver más claramente en esta captura:

#	Inicio	Final	Estilo	Texto
1	0:00:34.30	0:00:39.31	Romaji	*Su*re*chi*ga*u *i*shi*
2	0:00:40.70	0:00:45.79	Romaji	*Ts*uka*ma*e*ru *yo *
3	0:00:45.92	0:00:54.56	Romaji	*Mo*to*me *a*u *ko*k*
4	0:00:02.43	0:00:08.16	Hiragana	*こ*ど*く*な*ほ*ほ*を*
5	0:00:08.33	0:00:13.19	Hiragana	*よ*あ*け*の*け*は*い*
6	0:00:13.54	0:00:18.39	Hiragana	*わ*た*し*を*そ*ら*え*
7	0:00:02.43	0:00:08.16	English	Mis mejillas están manchadas
8	0:00:08.33	0:00:13.19	English	Pero puedo sentir la llegada c
9	0:00:13.54	0:00:18.39	English	Que me atrae hacia los cielos

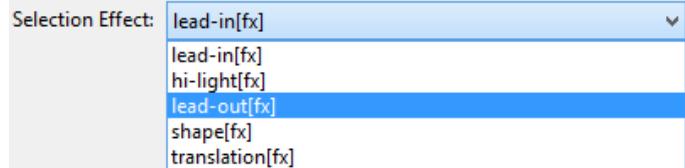
En pocas palabras, en “**Apply to Style**” escogemos el Estilo de las Líneas a las que le aplicaremos los Efectos. Al escoger un estilo, el Efecto se aplicará a todas las Líneas en nuestro archivo que tengan ese Estilo seleccionado.

La opción “**All Lines**”, como es evidente al traducirlo al español (**Todas las Líneas**), aplica el efecto a todas las Líneas de nuestro archivo .ass, excepto aquellas Líneas que estén comentadas y que no sean de tipo “**Dialogo**”, es decir que el Efecto no se aplicará a aquellas líneas que sean “**template**” como la “**template line**” o las “**code**” como las “**code once**” o “**code syll**”.

La opción “**Selected Lines**” como su nombre nos lo hace saber (Líneas Seleccionadas), aplica el Efecto a todas aquellas líneas que previamente hayamos seleccionado. Las líneas se seleccionan “iluminando” las líneas como tradicionalmente lo hacemos al seleccionar un texto o simplemente manteniendo presionado **Ctrl + Clip Derecho** de este modo podemos escoger a nuestra conveniencia aquellas líneas que queremos que se aplique nuestro efecto.

Una vez ya tengamos claro a qué Estilo aplicaremos un Efecto o a cuáles líneas en especial, el siguiente paso es seleccionar el tipo de Efecto que le aplicaremos a dichas líneas. A continuación veremos cómo podemos hacerlo.

### Selection Effect:



Como previamente había comentado, en esta pestaña desplegable es dónde podemos seleccionar el tipo de Efecto que aplicaremos a las líneas seleccionadas de nuestro archivo .ass. Son 5 tipos distintos de Efectos y a continuación los explicaré:

**lead-in[fx]**: son aquellos Efectos considerados como de “**Entrada**”, es decir que son los efectos que se aplican antes del Efecto de la Sílaba Activa, en el caso de los Efectos Karaoke, o los efectos de antelación justo antes de que una línea quede estable en el caso de las líneas de traducción del mismo karaoke.

**hi-light[fx]**: son los Efectos que le aplicamos a la Sílaba Activa, son los Efectos de Karaoke que llevan el ritmo de la canción y podría decirse que son efectos exclusivos de las Líneas Karaoke.

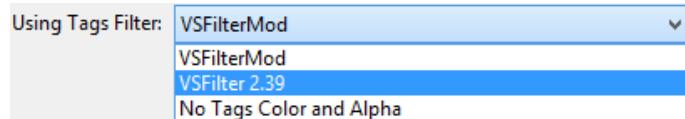
**lead-out[fx]**: son los Efectos de “**Salida**” son la contraparte de los Efectos tipo lead-in, y son los Efectos que se le hacen a las Sílabas luego de haber sido activadas en las Líneas de Karaoke o los Efectos que se hacen al final de una Línea de Traducción normal.

**shape[fx]**: son los Efectos que incluyen figuras hechas en el **AssDraw3**, conocidas como “**shapes**” y dependiendo de los tiempos del Efecto, pueden ser de **Entrada**, de **Salida** o de **Sílaba Activa**. También se pueden usar efectos tipo shape[fx] en las **líneas de traducción** y en los **subtítulos normales**.

**translation[fx]**: Son los Efectos que se pueden aplicar a todas las Líneas no comentadas de tipo Dialogo de nuestro archivo .ass. Generalmente son usados en las Líneas de Traducción, pero también pueden ser usados en las Líneas de Karaoke ya que este tipo de Efectos no exige que las Líneas estén separadas por Sílabas, por Palabras o por Caracteres.

En este momento ya tenemos elegidos tanto el **Estilo** como **tipo de Efecto** que usaremos. El siguiente paso es:

### Using Tags Filter:



## Kara Effector - Effector Book [Tomo I]:

En esta pestaña desplegable seleccionamos el formato en el que saldrán en nuestro Efecto los Tags de Colores y de Transparencias (alpha).

**VSFilterMod:** al elegir esta opción, hacemos que los tags de colores y transparencias saldrán en el formato del Filtro que lleva este nombre (**VSFilterMod**), ejemplo:

\1vc(&HFFFFFF&,&HFFFFFF&,&HFFFFFF&,&HFFFFFF&)

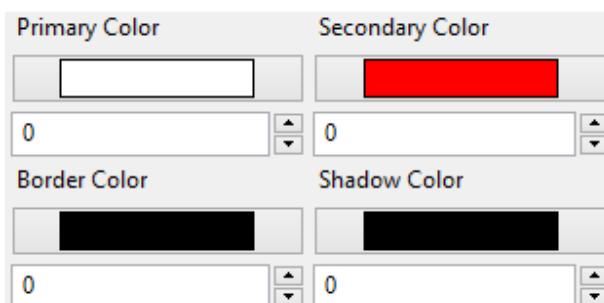
\3va(&H00&,&H00&,&H00&,&H00&)

**VSFilter 2.39:** esta opción hace que los tags de colores y transparencias en nuestro Efecto queden en el formato que por Default lo expresa el **Aegisub**, ejemplo:

\1c&HFFFFFF&

\3a&H00&

**No Tags Color and Alpha:** lo que hace esta opción es evitar que los colores y transparencias de la sección “**Text Setting**” salgan en nuestro Efecto. Esta opción solo afecta a estos ocho valores que se encuentran en la primera ventana del **Kara Effector**:



Estos ocho valores del “**Text Setting**” son cuatro colores y cuatro transparencias en donde están los colores primario y secundario, el color del borde y el color de la sombra, además de los respectivos valores de transparencia de estos colores. Los valores de las transparencias van desde 0 hasta 255, sabiendo que 0 es totalmente visible y 255 es completamente invisible.

### Line Comment – Save Configuration:

Line Comment  
 Save Configuration

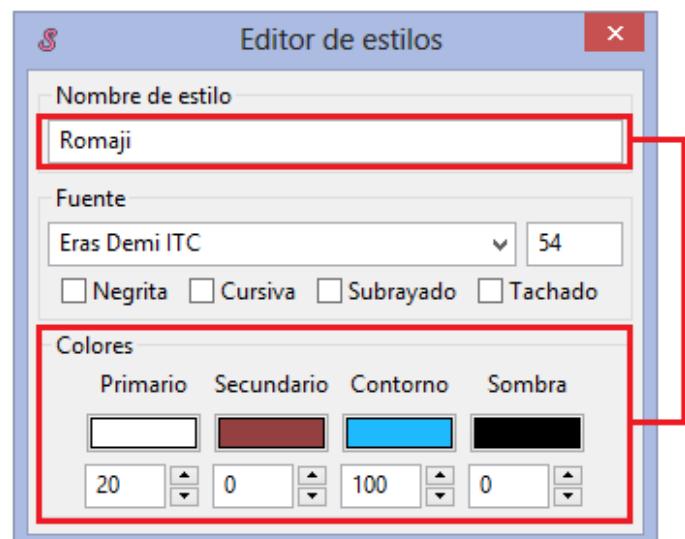
Son opciones tipo “checkbox”, o sea un botón de marcar y desmarcar. Al marcar la opción de “**line Comment**” lo que hace es Comentar las Líneas a las que le hayamos aplicado el Efecto, es decir, en caso de que queramos aplicar más de un Efecto a un mismo Estilo o Líneas seleccionadas, debemos mantener esta opción sin marcar, para que al aplicar los Efectos, las Líneas no se comenten y puedan seguir activas para aplicarle nuevos Efectos.

Por otro lado, la opción “**Save Configuration**” hace posible que todas aquellas modificaciones que hayamos hecho en cuanto a la selección del Estilo, tipo de Efecto, colores y demás, se conserven tal cual, al momento de pretender aplicar más Efectos.

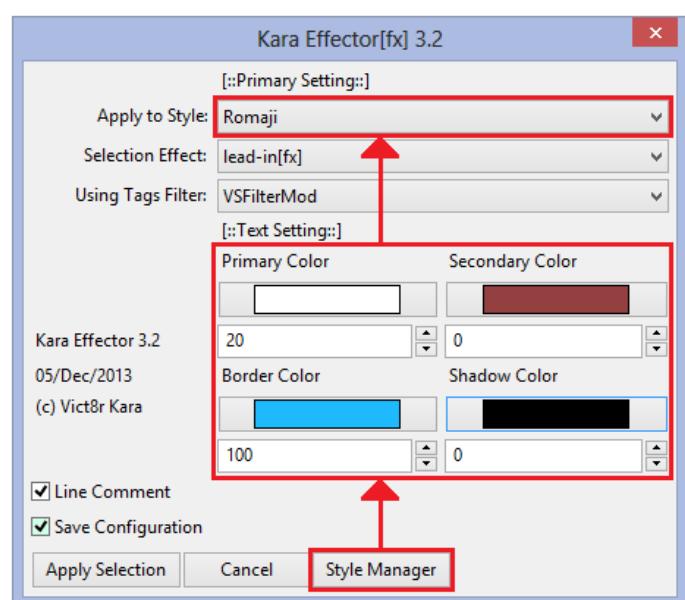
Y ya en la parte inferior de la primera ventana del **Kara Effector**, aparecen los botones de ejecución, los cuales explicaré de derecha a izquierda:

Apply Selection    Cancel    Style Manager

**Style Manager:** este botón hace que los valores de colores y transparencias del Estilo elegido, se copien en la ventana del **Kara Effector**. En el **Editor de Estilos del Aegisub** es en donde editamos generalmente estos valores:



Como se ve en la siguiente imagen, los valores de los colores y las transparencias del Estilo, son copiados:



## Kara Effector - Effector Book [Tomo I]:

Una duda razonable sería: ¿por qué la necesidad de copiar los valores de los colores y transparencias que ya había ajustado previamente?

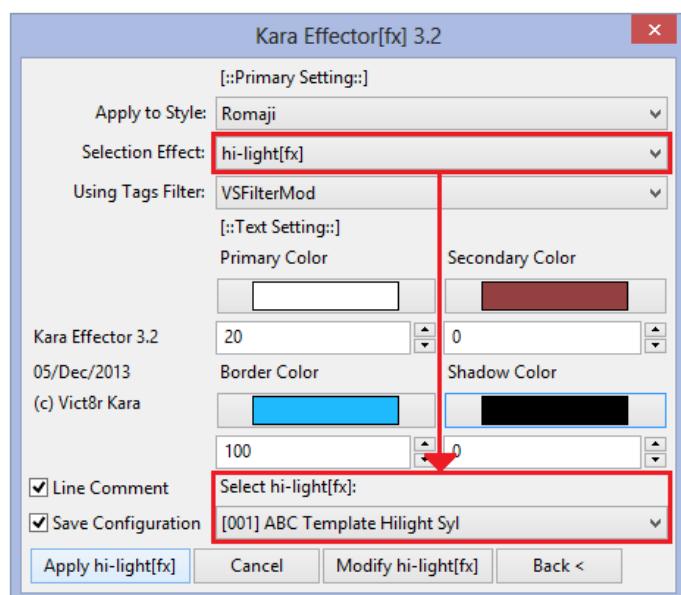
Y bueno, pasa que a veces creemos que los valores que elegimos en el Estilo en cuanto a colores y transparencias serán los definitivos, pero al aplicar un Efecto no quedamos conforme con los resultados y quizás haya la necesidad de modificar algunos de esos valores.

Otra razón sería la posibilidad de darle los valores de los colores y transparencias originales a un determinado Efecto y otros valores ligeramente modificados a un Efecto diferente. En fin, podría decir que a medida que se vayan familiarizando con el **Kara Effector**, se irán dando cuenta que no es simplemente un lujo, sino que puede llegar a ser de gran utilidad a la hora de modificar los Efectos o de crear otros nuevos.

**Cancel:** no hay mucho que decir de este botón, ya que es más que clara su función. Este botón cierra la ventana de inicio del **Kara Effector**.

Hasta este punto ya tenemos claro el Estilo o las Líneas a las que le aplicaremos un Efecto, el tipo de Efecto que haremos, el formato en el que quedarán los valores de los colores y las transparencias, incluso si queremos mantener dichos valores o los queremos modificar, pero lo que aún no hemos visto es dónde podemos elegir el Efecto en particular que aplicaremos, y para ello es que está el siguiente botón:

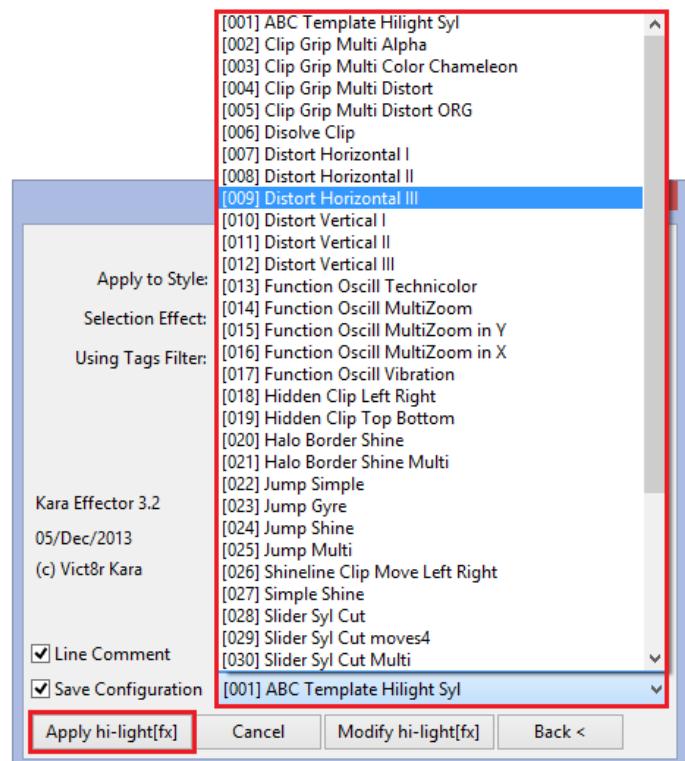
**Apply Selection:** este botón modifica la ventana de inicio del Kara Effector para poder visualizar el listado de Efectos según el tipo que hayamos elegido, también modifica los botones de ejecución para poder aplicar el Efecto elegido o para que podamos modificarlo:



En este ejemplo, el tipo de Efecto que elegí fue **hi-light** y por ello al pulsar el botón “**Apply Selection**”, el nombre de la nueva pestaña desplegable que apareció luego de modificarse la ventana de inicio del **Kara Effector** es:

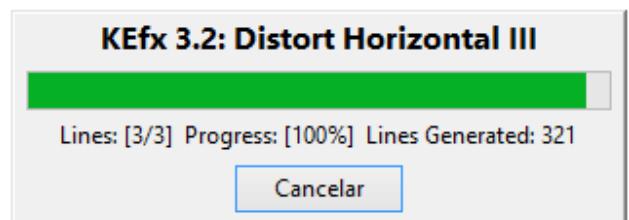
### Select hi-light[fx]

Y como su nombre lo indica, es donde ya podemos elegir el Efecto tipo **hi-light** que aplicaremos al Estilo o a las Líneas seleccionadas:



El botón “**Back <**” hace que podamos volver a seleccionar el tipo de Efecto que usaremos, y ya por último, una vez escogido el Efecto que aplicaremos, pulsamos el botón “**Apply hi-light[fx]**” (el “**hi-light**” es porque fue el tipo de Efectos que había elegido, si por el contrario hubiera elegido por ejemplo el tipo de Efecto tipo **shape**, el botón pondría: “**Apply shape[fx]**”).

Dependiendo del peso del Efecto, algunos de ellos se aplicarán más rápidos que otro. Al momento de aplicarse un Efecto veremos la siguiente ventana informativa:



Hay tres datos distintos que nos ofrece esta ventana, el primero nos dice a cuántas líneas ya se les ha aplicado el Efecto de todas en total. En este caso ya completó 3 de 3.

## Kara Effector - Effector Book [Tomo I]:

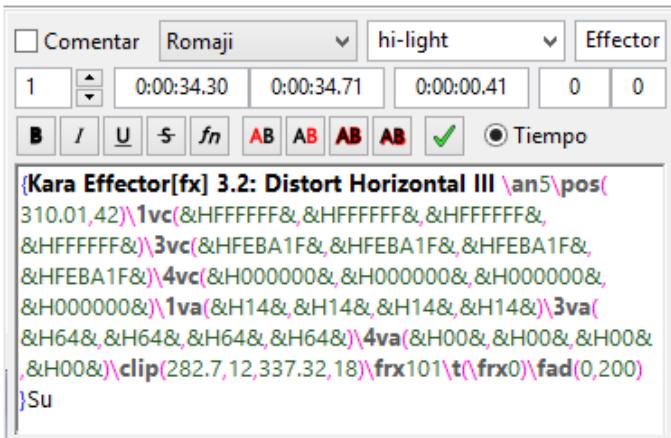
El siguiente dato es el progreso y está medido en el porcentaje total hasta que el Efecto se aplique en su totalidad.

Por último, el tercer dato nos muestra la cantidad de líneas que está generando el Efecto en cuestión.

Una vez que se haya aplicado el Efecto, ya podemos ver las líneas que generó, en donde el Actor pone el tipo de Efecto que seleccionamos (**hi-light** en este caso), y en la columna de Efecto siempre pondrá “**Effector [fx]**”. De ese modo es fácil identificar cuáles fueron las nuevas líneas que se generaron:

#	L	Inicio	Final	Estilo	Actor	Efecto	Texto
6	0	0:00:13.54	0:00:18.39	Hiragana			*わ*た*し*を*
7	0	0:00:02.43	0:00:08.16	English			Mis mejillas están
8	0	0:00:08.33	0:00:13.19	English			Pero puedo sentir
9	0	0:00:13.54	0:00:18.39	English			Que me atrae hac
10	1	0:00:34.30	0:00:34.71	Romaji	hi-light	Effector [fx]	*Su
11	1	0:00:34.30	0:00:34.71	Romaji	hi-light	Effector [fx]	*Su
12	1	0:00:34.30	0:00:34.71	Romaji	hi-light	Effector [fx]	*Su
13	1	0:00:34.30	0:00:34.71	Romaji	hi-light	Effector [fx]	*Su
14	1	0:00:34.30	0:00:34.71	Romaji	hi-light	Effector [fx]	*Su
15	1	0:00:34.30	0:00:34.71	Romaji	hi-light	Effector [fx]	*Su
16	1	0:00:34.30	0:00:34.71	Romaji	hi-light	Effector [fx]	*Su

Al mirar en unas de las líneas generadas, podemos ver claramente el nombre del programa, la versión y el nombre del Efecto que hayamos seleccionado:



En este caso en particular los valores de los colores y de transparencias están en formato del **VSFilterMod**, pero como ya había explicado anteriormente, todo depende de nuestra elección.

Siguiendo esta serie de instrucciones se pueden aplicar los Efectos que por default trae el **Kara Effector** con solo unas pocas modificaciones que son posibles hacer desde la Ventana de Inicio. Para la próxima entrega veremos la Ventana de Modificación y creación de Efectos y también cada una de las partes que la componen. Es todo por ahora y me despido con la ilusión de haber aclarado las

dudas que hasta este momento hayan tenido. No olviden visitar el Blog Oficial del **Kara Effector**, lo mismo que los canales de **You Tube** que están en la primera página de esta entrega.

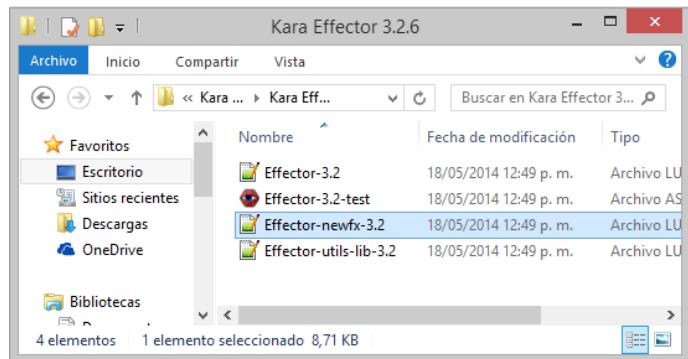
## Instalar Nuevos Efectos Kara Effector 3.2

Ya por último les dejamos los pasos a seguir para instalar en el **Kara Effector** los efectos nuevos que compartimos en el **Blog**, o aquellos que son compartidos por varios de los usuarios que ya hacen uso del **Kara Effector**.

1. Abrimos el archivo en donde está el código del efecto y copiamos todo el contenido:

```
Sakura_Blur_I = tableuplicate(Shape_Box); table.inbox(Sakura_Blur_I,"Syl",true,false, "#FFFFFF", "#5D406D", "#6A8DD6", "255", "40", "0", "l.start_time + syl.start_time - 200", "l.start_time + syl.end_time + 300", "", "", "", "syl.center", "syl.middle", "", "", "5", "0", "fx.pos_x", "fx.pos_y", "", "1", "10", "shape.flower5t", "format('\\\\\\fr%\\\\\\t(\\\\\\fr0)\\\\\\t(0,200,\\\\\\fscx75\\\\\\fscy75)\\\\\\fad(200,300)\\\\\\bord4\\\\\\blur6', 0.08*fx.dur)", ""); table.insert(shape_fx_library, Sakura_Blur_I); table.insert(shape_fx, "Sakura Blur I")
```

2. Ubicamos el archivo **Effector-newfx-3.2.lua** y lo abrimos:

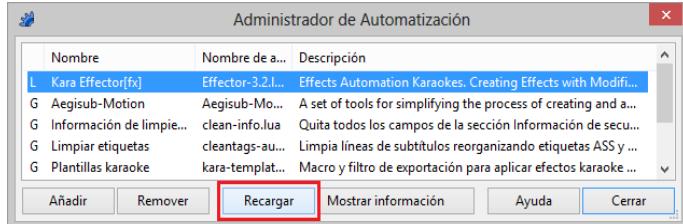


3. Pegamos el código del efecto al final del archivo **Effector-newfx-3.2.lua** (no necesariamente debe ser al final, pero así es más simple ubicarlo):

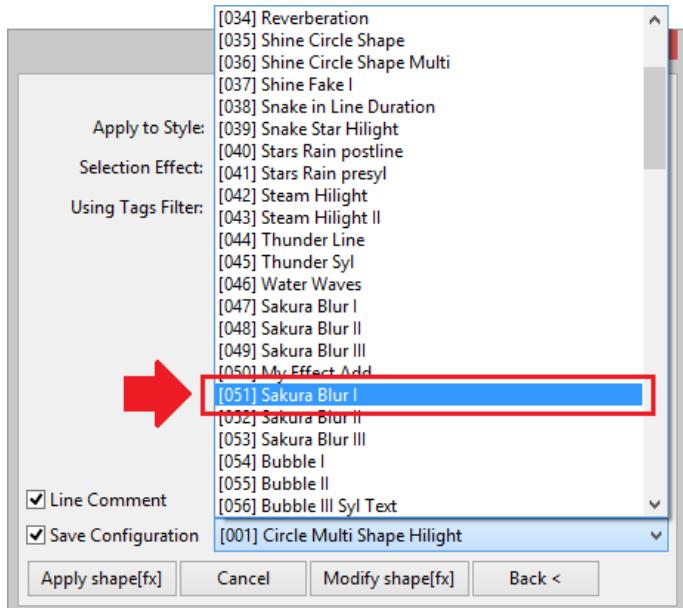
```
--Shape fx new list
Sakura_Blur_I = tableuplicate(Shape_Box); table.inbox(Sakura_Blur_I,
Sakura_Blur_II = tableuplicate(Shape_Box); table.inbox(Sakura_Blur_II,
Sakura_Blur_III = tableuplicate(Shape_Box); table.inbox(Sakura_Blur_III,
24
25
26
27
28
29
30
31
--Translation fx new list
ABC_Translation_Line = tableuplicate(Trans_Box); table.inbox(ABC_Translation_
Move_Line_Top = tableuplicate(Trans_Box); table.inbox(Move_Line_Top,
Sakura_Blur_I = tableuplicate(Shape_Box); table.inbox(Sakura_Blur_I,
```

## Kara Effector - Effector Book [Tomo I]:

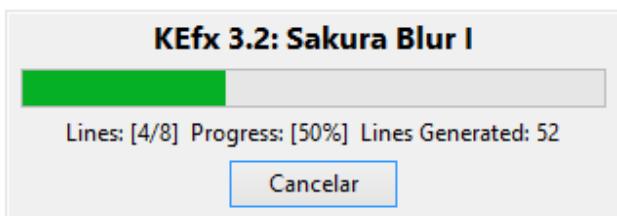
4. Si previamente ya teníamos abierto el **Aegisub**, tendremos que recargar el script del Kara Effector. En caso de no tenerlo abierto, el **script** se cargará de forma automática:



5. Abrimos el **Kara Effector** y buscamos el efecto dependiendo del tipo de efecto, en este caso es uno tipo **shape** y es en ese listado en donde quedara instalado y listo para ser usado:



Al aplicar el efecto veremos algo como esto:



Y en el vídeo ya podemos ver los resultados del efecto:



Con el final de este **Tomo I** se dan por concluidos los pasos preliminares de uso e instalación del **Kara Effector**. 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, exponer alguna duda o hacer alguna sugerencia. Pueden visitarnos y dejar su comentario en nuestra página de **Facebook**:

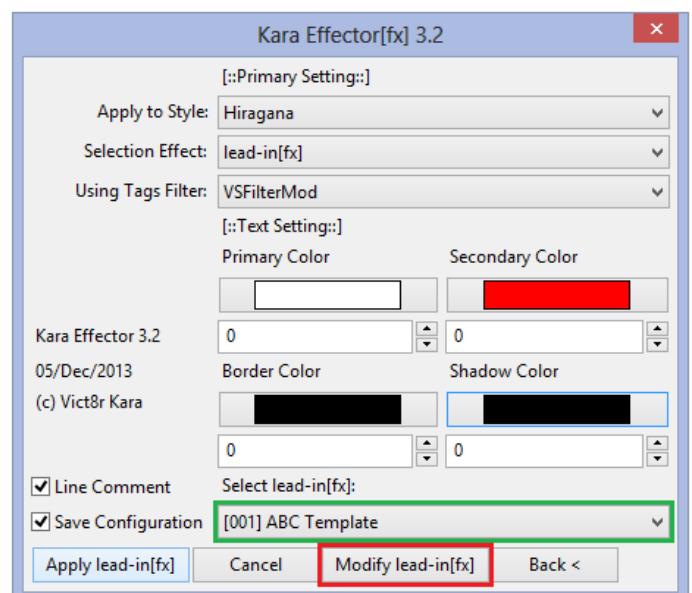
[www.facebook.com/karaeffector](http://www.facebook.com/karaeffector)

# Kara Effector 3.2: Effector Book Vol. I [Tomo III]

## Kara Effector 3.2:

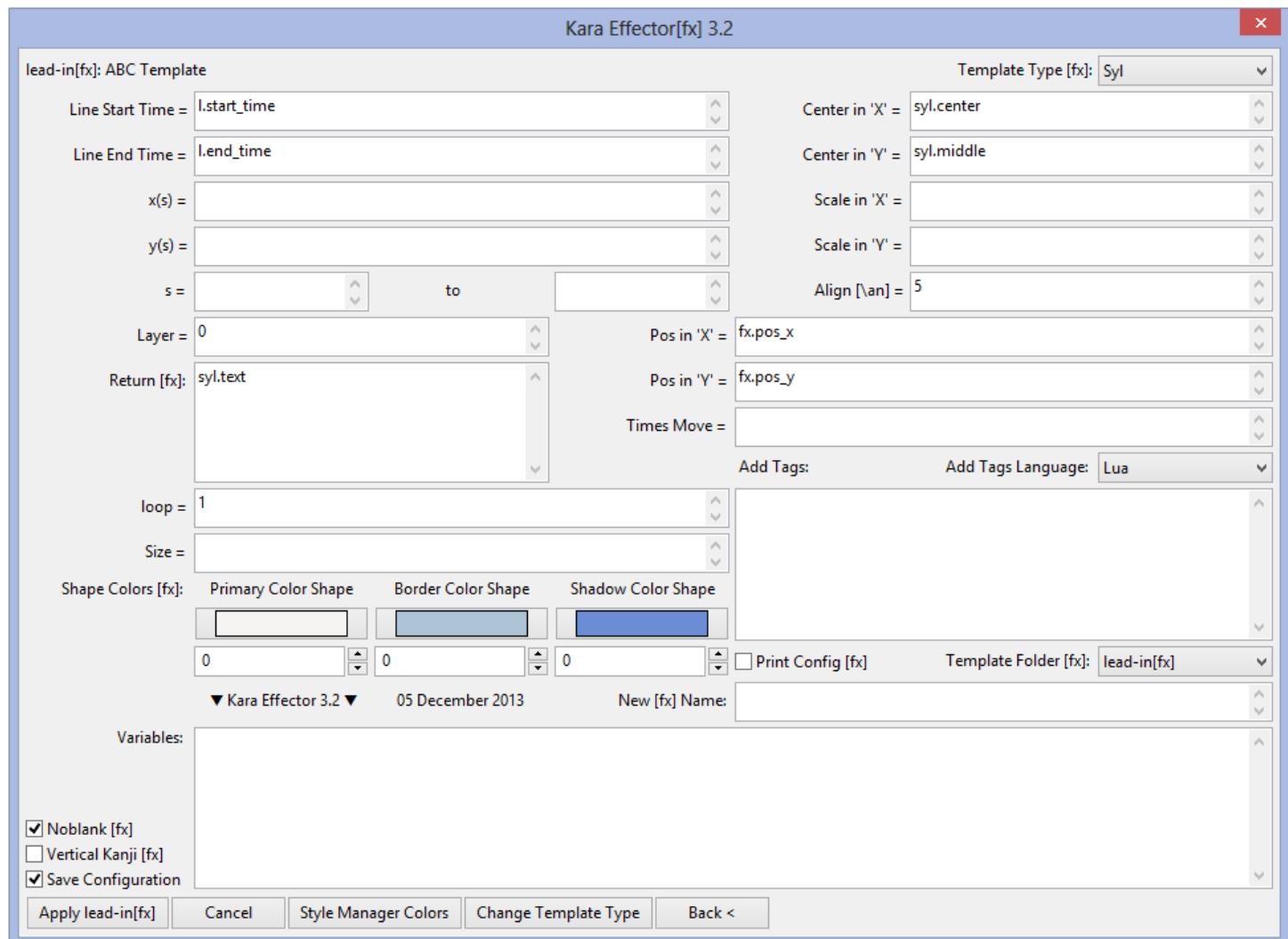
En la anterior entrega, vimos cómo descargar, instalar y aplicar los Efectos que vienen en el **Kara Effector** por Default. Las pocas modificaciones que se pueden hacer en la ventana de inicio no van más allá del cambio de los valores en los colores y la transparencia. En esta entrega veremos cómo abrir la **Ventana de Modificación** y de creación de nuevos Efectos, que consta de muchas más opciones que la ventana de inicio pero no por ello es más compleja de usar o entender.

Esta segunda ventana se puede visualizar al pulsar el botón de “**Modify**” y el resto del nombre de este botón lo hereda del tipo de Efecto que hayamos seleccionado:



El pulsar el botón “**Modify**” hace que la ventana de inicio del **Kara Effector** de transforme en la segunda ventana:

## Kara Effector - Effector Book [Tomo II]:



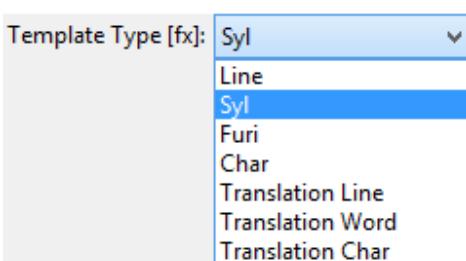
La **Ventana de Modificación** contiene toda la información concerniente al Efecto previamente seleccionado y a continuación haré una descripción de cada una de los elementos que lo conforman.

### Template Type [fx]:

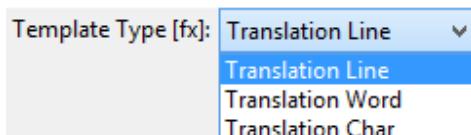


Es una pestaña desplegable en donde seleccionamos el modo en que se aplicará el Efecto y dependiendo el tipo de Efecto, las opciones de esta pestaña son las siguientes:

Para los Efectos tipo “**shape[fx]**” son:



Y para los Efectos tipo “**translation**” las opciones son:



Ahora veremos qué hace cada una de las opciones de esta pestaña desplegable y cómo sacarle el mejor provecho a la hora de decidirnos por una de ellas.

## Kara Effector - Effector Book [Tomo II]:

---

### Template Type [fx]: Line

La opción **Line** hace que se genere una única línea de Efecto por cada línea a la que se le aplique el mismo.

### Template Type [fx]: Syl

La opción **Syl** genera una Línea de Efecto por cada Sílaba de cada Línea a la que se le aplique un Efecto.

### Template Type [fx]: Furi

La opción **Furi** genera una Línea de Efecto por cada **Furigana** que tenga cada Línea que haya sido seleccionada para aplicarle un Efecto. Es necesario, más adelante, saber qué es y cómo se sincroniza un **Furigana**, para incluirlo como parte de nuestros Efectos.



De adelanto, los **furiganas** son los símbolos que están en rojo en la anterior imagen, y son **hiraganas** que indican cómo se pronuncia el **kanji** que está justo debajo de ellos. Esto genera más dudas, ya que para explicar un término, incluí otros dos nuevos, pero seguramente también harán parte de las próximas entregas. Se aclararán términos como: **Romaji**, **Kanji**, **Hiragana**, **Katakana**, **Kana**, **Furigana** y aquellos otros que de pronto puedan ser pertinentes a la hora de hacer Karaokes de gran calidad.

### Template Type [fx]: Char

La opción **Char** genera una Línea de Efecto por cada **Carácter** (letra, número, signo de puntuación, espacios y tabulaciones, símbolos matemáticos y demás) que haya en cada Línea que haya sido seleccionada para aplicarle un Efecto.

### Template Type [fx]: Translation Line

La opción **Translation Line** genera una Línea de Efecto por cada Línea de Traducción a la que se le aplique un Efecto. Esta opción al igual que las próximas dos, están pensadas para que sean usadas solo en las Líneas de Traducción.

---

### Template Type [fx]: Translation Word

La opción **Translation Word** genera una Línea de Efecto por cada Palabra de cada Línea a la que se le aplique un Efecto.

### Template Type [fx]: Translation Char

La opción **Translation Char** genera una Línea de Efecto por cada Carácter de cada Línea a la que se le aplica un Efecto.

Un Efecto en el **Kara Effector** no está necesariamente atado a un único Tipo de Plantilla (**Template Type**), ya que puede ser cambiado por otro a gusto de cada quien y dependiendo de los resultados que cada uno quiera obtener, o sea que es posible pasar un Efecto de **Template Syl** a **Template Char**, por ejemplo, si uno así lo quiere.

---

### Tiempos del Efecto:

Para modificar los tiempos de un Efecto hay dos opciones, una de ellas es desde las celdas de texto ("**textbox**") dispuestas para ese fin:

Line Start Time =	<input type="text" value="l.start_time"/>
Line End Time =	<input type="text" value="l.end_time"/>

**Line Start Time:** es el tiempo de inicio de las Líneas que genera un Efecto. En el ejemplo de la imagen el tiempo que pone es "**l.start\_time**" que es equivalente al tiempo de inicio original de cada Línea que haya sido seleccionada para aplicarle un Efecto.

**l.start\_time = line.start\_time**

Por ejemplo **l.start\_time + 1000** quiere decir que el tiempo de inicio de las Líneas de Efectos generadas será 1000 milisegundos (1000 ms = 1 segundo) después del tiempo de inicio original de las Líneas seleccionadas para aplicarle el Efecto.

**Line End Time:** es el tiempo final de las Líneas que genera un Efecto. En el ejemplo de la imagen el tiempo que pone es "**l.end\_time**" que es equivalente al tiempo final original de cada Línea que haya sido seleccionada para aplicarle un Efecto.

**l.end\_time = line.end\_time**

## Kara Effector - Effector Book [Tomo II]:

La otra forma de modificar los tiempos es con la función **retime**, que es una de las muchas funciones que pueden ser empleadas en el **Kara Effector** y que veremos en los próximos volúmenes.

### Centros en los Ejes (Posición):

Center in 'X' =	syl.center
Center in 'Y' =	syl.middle

Los Centros en los Ejes son celdas de texto en donde las coordenadas “x” y “y” de la posición son los centros para ubicar las Líneas de Efecto generadas.



La ventaja de estas y de todas las celdas de texto del **Kara Effector** es poder usar todas las operaciones matemáticas para modificar a nuestro gusto los valores que queremos usar. Las anteriores dos variables son solo dos de una lista de variables que también veremos en los próximos tomos.

### Alineación de las Líneas de Efecto:

Align [\an] =	5
---------------	---

**Align [\an]**: es la alineación del Efecto respecto a los centros de las Líneas seleccionadas para aplicar el mismo. Indica el punto de referencia que se alinea con los centros seleccionados en las dos celdas anteriores (**Center in 'X'** y **Center in 'Y'**).

En realidad la Alineación se ajusta a la posición final que tendrá la Línea de Efecto generada en el caso de las posiciones estáticas o a las trayectorias del movimiento.

### Capa de las Líneas de Efecto:

Layer =	0
---------	---

**Layer**: es el número de la capa que tendrá la Línea de Efecto generada. Para modificar en número de la capa se puede hacer en la celda de texto asignada para ello o con la función **relayer**, que también veremos más adelante.

The screenshot shows the Kara Effector interface. At the top, there is a toolbar with various icons and a dropdown menu labeled "Actor". Below the toolbar, a dropdown menu is open, showing the number "1" selected. A red box highlights this selection. To the right of the dropdown is a table titled "Layer" with columns for "#", "L", "Inicio", "Final", and "Estilo". The table contains several rows, with the last row (row 17) also highlighted with a red box. The table entries are as follows:

#	L	Inicio	Final	Estilo
13	0	0:00:28.52	0:00:34.30	Hiragana
14	0	0:00:34.30	0:00:39.31	Hiragana
15	0	0:00:40.70	0:00:45.79	Hiragana
16	1	0:00:45.92	0:00:54.56	Hiragana
17	0	0:00:02.43	0:00:08.16	English

La función de la capa (**layer**) es determinar qué se verá encima de qué, en nuestro vídeo, entre dos o más objetos karaoke (Caracteres, Sílabas, Palabras, Líneas de Texto o Shapes) que coinciden total o parcialmente en su posición.

### Posición y Movimiento:

Pos in 'X' =	fx.pos_x
Pos in 'Y' =	fx.pos_y
Times Move =	

**Pos in 'X'**: es la o las coordenadas con respecto a la posición final o del movimiento de la línea de Efecto con respecto al eje “x”. Para el caso de más de una coordenada, éstas se separan con comas (,) o con punto y coma (;).

**fx.pos\_x** = coordenada en “x” de la línea de Efecto

**Pos in 'Y'**: es la o las coordenadas con respecto a la posición final o del movimiento de la línea de Efecto con respecto al eje “y”. Para el caso de más de una coordenada, éstas se separan con comas (,) o con punto y coma (;).

**fx.pos\_y** = coordenada en “y” de la línea de Efecto

## Kara Effector - Effector Book [Tomo II]:

**Times Move:** para el caso del movimiento, se requiere un tiempo de inicio y un tiempo final para el mismo. Los tiempos son medidos en milisegundos (ms).

Si la celda de texto del **Times Move** está vacía, entonces el tiempo de inicio del movimiento es cero (0) y el tiempo final será el mismo que la duración total de la línea de Efecto (**fx.dur**). Los dos parámetros para el tiempo de inicio y final de un movimiento van separados por coma (,) o por punto y coma (;).

**fx.pos\_x** y **fx.pos\_y** son dos variables de la librería fx que hacen referencia a muchos de los valores del **Kara Effector** que podemos utilizar en nuestros efectos. En el próximo **Tomo** veremos éstas y más librerías que nos harán aún más simple la tarea de hacer un Efecto Karaoke.

### Objetos Karaoke:



**Return [fx]:** es la parte visible en el vídeo que se verá de un Efecto Karaoke. Puede ser un Caracter, una Sílaba, una Palabra, una Línea de Texto, un Número o un Símbolo matemático, un Hiaraga, Furigana, Katakana, Kanji y/o Shape.

### Repeticiones:

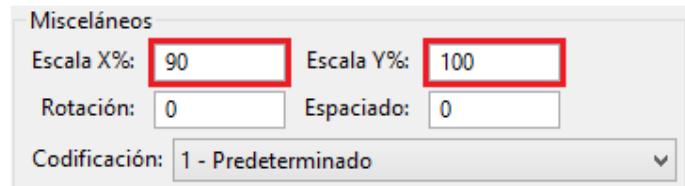


**Loop:** es la cantidad de repeticiones que se harán de un Efecto Karaoke. Si esta celda de texto está vacía, su valor por default es 1. Para algunas funciones del **Kara Effector** a veces se usan dos parámetros en el **loop**, para estos casos las repeticiones totales del Efecto será el producto de los dos parámetros. La cantidad de repeticiones de un Efecto se puede determinar de dos maneras, una es con la celda de texto **loop** y la otra es con la función **maxloop**, que también es otra de las funciones que veremos más adelante junto con las que ya se han mencionado antes.

### Tamaño:

**Size =**

**Size:** es el porcentaje del tamaño del Objeto Karaoke. En el caso de que esta celda de texto esté vacía, se toman los porcentajes del tamaño que están en el estilo de las Líneas seleccionadas:



Lo que en tags sería: → \fscx90\fscy100

En el caso de que en **Size** haya un solo valor, éste se toma para las dos escalas, ejemplo:

Size = 45 → \fscx45\fscy45

Y para cuando haya dos valores, entonces el primero es el porcentaje para la escala en "x" y el segundo para la escala en "y", ejemplo:

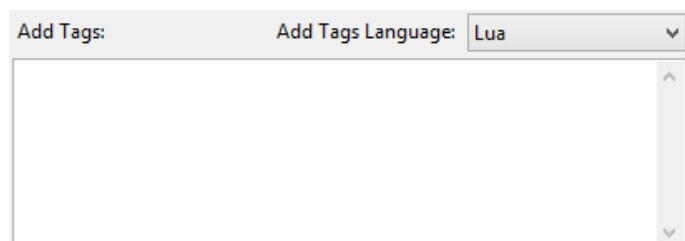
Size = 70, 92 → \fscx70\fscy92

### Colores y Transparencias de las Shapes:



Cuando en **Return [fx]** ponemos una **shape**, los colores y transparencias que se verán en nuestro Efecto serán los de la anterior imagen.

### Agregar Tags:



## Kara Effector - Effector Book [Tomo II]:

**Add Tags:** es una celda de texto que nos da la posibilidad de agregar nuevos tags a un Efecto, al igual que funciones. Ahora la versión 3.2 del **Kara Effector** ofrece la opción de añadir tags en el lenguaje que queramos, o sea, se pueden añadir en lenguaje **LUA** o en lenguaje **Automation Auto-4**.



Ejemplo en lenguaje **LUA**:

```
Add Tags: string.format("\\"pos(%s,%s)", syl.center + 100, syl.middle)
```

A screenshot of the Kara Effector software interface showing LUA code in the 'Add Tags' text area. The code is: `string.format("\\"pos(%s,%s)", syl.center + 100, syl.middle)`.

Este mismo tag en lenguaje **Automation Auto-4** sería:

```
Add Tags: \pos(!$center + 100,$middle)
```

A screenshot of the Kara Effector software interface showing Automation Auto-4 code in the 'Add Tags' text area. The code is: `\pos(!\$center + 100,\$middle)`.

La adaptación del **Kara Effector** para reconocer el lenguaje **Automation Auto-4** es ideal para todos aquellos que ya se familiarizan con ese método de hacer Efectos, ya que se usa exactamente igual que en el **Aegisub** cuando se hacen Efectos Karaoke. En el **Aegisub**, una plantilla de Efecto podría ser:



En el **Kara Effector** no cambiaría absolutamente nada:

```
Add Tags: Add Tags Language: Automation Auto-4  
!retime("syl",0,0)!\\an5\\pos($center,$middle)\\fad(300,300)
```

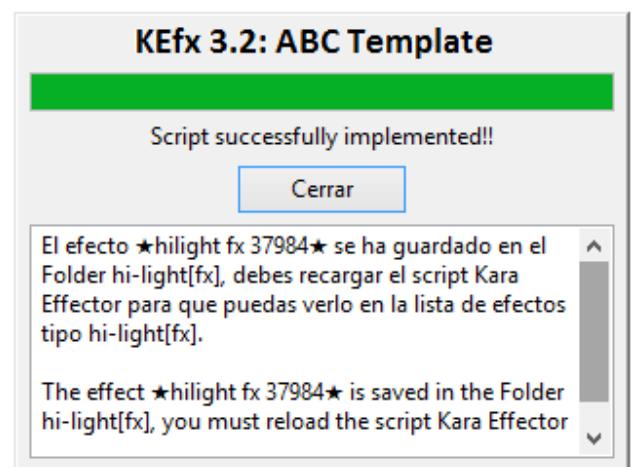
A screenshot of the Kara Effector software interface showing Automation Auto-4 code in the 'Add Tags' text area. The code is: `!retime("syl",0,0)!\\an5\\pos(\$center,\$middle)\\fad(300,300)`.

### Generar y Guardar Efectos Nuevos:



Al hacer Efectos nuevos a partir de la modificación de alguno ya hecho en el **Kara Effector** o al crear uno nuevo partiendo desde ceros, genera la necesidad de guardarlos para luego ser usados posteriormente. Pensando en esto, están estas tres opciones que son muy simples de usar:

**Print Config [fx]:** es el **checkbox** que decide si un Efecto se genera normalmente o si en vez de ello, se guarda de forma permanente en el **Kara Effector**. Si esta opción no está marcada, el Efecto se genera en el **Aegisub**, de lo contrario, las configuraciones del Efecto se **"imprimen"** en el **Effector-newfx.lua** directamente o en el **Aegisub** para posteriormente ser copiado de forma manual en el **Effector-newfx.lua**. Al guardarlo directamente de salir un aviso como este:



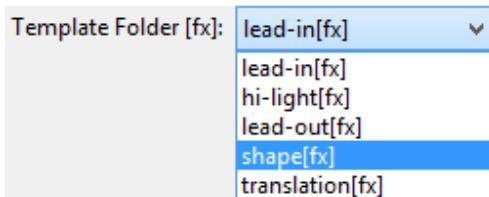
Este recuadro es la señal inequívoca de que el Efecto se ha guardado de forma satisfactoria en el archivo LUA del Effector, **Effector-newfx.lua**, que es el archivo que está destinado para los Efectos nuevos. De lo contrario, el Efecto se imprime en el **Aegisub** y se pega manualmente en el **Effector-newfx.lua**, se verá algo como esto:

23	0:00:40.70	0:00:45.79	English		Me afarraré a ti y n
24	0:00:45.92	0:00:54.56	English		Dos corazones que
25	0:00:00.00	0:00:00.00	Romaji	Effector [Fx] Config	hilight_fx_37922 =

El método para hacer que los Efectos nuevos se guarden directamente es muy simple, pero prefiero que quede explicado más claramente en los siguientes tomos.

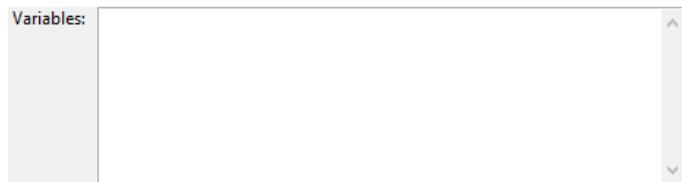
## Kara Effector - Effector Book [Tomo II]:

**Template Folder [fx]:** es el lugar de destino del nuevo Efecto, o sea que al guardarlo aparece en el listado de tipos de Efectos que elegimos en esta opción:



**New [fx] Name:** es la celda de texto donde escribimos el nombre del nuevo Efecto. Se pueden usar letras y números, pero no signos de puntuación ni caracteres especiales del **ASCII**. Si no escribimos nada en esta celda e imprimimos el nuevo Efecto, el **Kara Effector** le asigna un nombre único por Default a cada Efecto que se imprima sin que escribamos el nombre.

### Funciones y Variables:



Es una de las celdas de texto más importantes ya que aquí podemos declarar tanto variables como funciones y arreglos, en general se pueden definir el resto de cosas que no podamos hacer directamente desde el resto de las celdas.

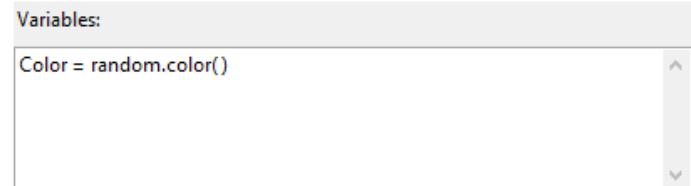
Declarar un variable es tan simple como asignarle el nombre que queramos, seguido del signo “igual” (=) y por último el valor que tendrá dicha variable. Para los siguientes ejemplos inventé una variable llamada “**Color**” y el valor asignado es uno random.

**random.color** es una función del **Kara Effector** y como su nombre ya lo hace prever, retorna un color al azar de entre todos los del espectro, excepto el blanco y el negro. Esta y las demás funciones del **Kara Effector** se verán con ejemplos en los próximos tomos.

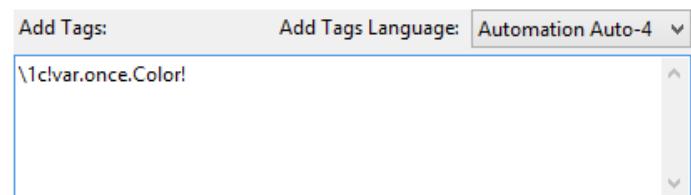
Una misma variable declarada se puede “llamar” de diversas maneras, en caso de que una variable tenga un valor constante, no importaría de qué forma se llame, su valor seguirá siendo el mismo. Ejemplo:

Angle = 2\*math.pi

La variable “Angle” siempre tendrá el mismo valor sin importar de qué ni de cuántas veces se llame a ser usada. Caso distinto es cuando se usa un valor random como el del siguiente ejemplo:



Luego de declarar la variable ya puede ser llamada en cualquiera de las celdas de texto del **Kara Effector**, en este ejemplo lo hice en “**Add Tags**”:



**var.once:** es la forma de llamar a una variable para que se ejecute una sola vez en todo el karaoke

- Línea 1: **Kodoku na hoho wo nurasu nurasu kedo**  
Línea 2: **Yoake no kehai ga shizuka ni michite**  
Línea 3: **Watashi wo sora e maneku yo**

El random del color se hizo una sola vez y por eso el color primario de todas las sílabas de todas las líneas es el mismo, como se puede ver en la imagen anterior.

Este ejemplo lo he hecho en lenguaje **Automation Auto-4** y el color random que generó la variable lo usé para el color primario:

\1c!var.once.Color!

Este mismo ejemplo hecho, pero en lenguaje **LUA** tendría dos formas principales de hacerse:

1. string.format("\1c%s", var.once.Color)
2. "\1v..var.once.Color

Ambas hacen exactamente lo mismo y ya es decisión de cada uno elegir la que más se le facilite o guste.

## Kara Effector - Effector Book [Tomo II]:

**var.line:** el random se ejecuta, pero una vez por cada línea a la que le apliquemos el Efecto. A diferencia del **var.once** que solo hizo el random una sola vez:

Add Tags: Add Tags Language: Automation Auto-4

```
\1cvar.line.Color!
```

El random generó para cada línea un color diferente:

Línea 1: **Kodoku na hoho wo nurasu nurasu keto**  
Línea 2: **Yoake no kehai ga shizuka ni michite**  
Línea 3: **Watashi wo sora e maneku yo**

**var.syl:** el random se ejecuta una vez por cada sílaba de cada línea:

Add Tags: Add Tags Language: Automation Auto-4

```
\1cvar.syl.Color!
```

El random generó un color diferente para cada sílaba:

**Kodoku na hoho wo nurasu nurasu keto**

**var.furi:** cumple la misma función que **var.syl**, con la diferencia de que genera el random una vez por cada Furigana que tengan las líneas seleccionadas:

Add Tags: Add Tags Language: Automation Auto-4

```
\1cvar.furi.Color!
```

**var.word:** es similar al **var.syl**, pero genera el random una vez por cada palabra de cada línea seleccionada:

Add Tags: Add Tags Language: Automation Auto-4

```
\1cvar.word.Color!
```

El random generó un color diferente para cada palabra:

**Kodoku na hoho wo nurasu nurasu keto**

**var.char:** el random se ejecuta una vez por cada carácter de cada línea:

Add Tags: Add Tags Language: Automation Auto-4

```
\1cvar.char.Color!
```

**Kodoku na hoho wo nurasu nurasu keto**

**var.loop:** el random se ejecuta una vez por cada loop que hayamos puesto como número de repeticiones para un Efecto:

Add Tags: Add Tags Language: Automation Auto-4

```
\1cvar.loop.Color!
```

El **var.once**, **var.line**, **var.word**, **var.syl**, **var.furi**, **var.char** y **var.loop**; adquiere relevancia, como lo mencionaba antes, cuando declaramos una variable o alguna función que incluya uno o más valores random. De otra manera los valores de las variables serían constantes y el valor al llamar a ese tipo de variables sería el mismo sin importar el “**var**” que se use. Para variables constantes está la opción de llamar a la variable directamente por su nombre como se ve en el siguiente ejemplo:

Add Tags: Add Tags Language: Automation Auto-4

```
\1cColor!
```

La forma de llamar una variable declarada en “Variables” depende mucho del resultado que se quiera obtener. En esta celda de texto, las variables y funciones, lo mismo que los arreglos (array) pueden ir separados por comas (,) o por punto y coma (;), pero para que sea posible que sean llamadas por su nombre original (como en el ejemplo anterior), deben ir separadas por punto y coma (;).

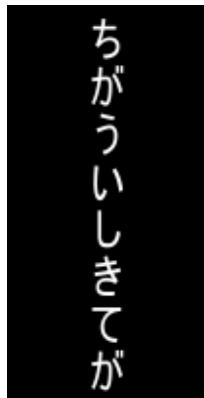
## Kara Effector - Effector Book [Tomo II]:

### Configuraciones “Check”:

- Noblank [fx]
- Vertical Kanji [fx]
- Save Configuration

**Noblank [fx]:** al estar marcada, esta opción NO toma en cuenta a las Sílabas que están en Blanco o Vacías, al igual que Líneas en Blanco. Para poder tener en cuenta a estos valores en blanco, lo que debemos hacer es “desmarcar” esta opción.

**Vertical Kanji [fx]:** al marcar esta opción, las líneas de karaoke de un Efecto saldrán en pantalla de forma Vertical en vez de horizontalmente, que es la forma en que salen tradicionalmente. Es recomendado para los Kanjis o el resto de los pictogramas japoneses:



**Save Configuration:** cumple la misma función que su opción gemela de la ventana de inicio del **Kara Effector**, o sea que conserva las modificaciones hechas en la segunda ventana a un Efecto, siempre y cuando no recarguemos el **Kara Effector**.

### Botones de Ejecución:

- Apply lead-in[fx]
- Cancel
- Style Manager Colors
- Change Template Type
- Back <

**Apply [fx]:** aplica el Efecto seleccionado

**Cancel:** cierra el **Kara Effector**

**Style Manager Colors:** asigna los valores de los colores y transparencias del Estilo de las Líneas seleccionadas a los colores y transparencias **Shape**, en el caso en que queramos usar dichos valores. Es decir que cumple la misma función que la opción **Style Manager** de la ventana de inicio del **Kara Effector**.

**Change Template Type:** cambia el tipo de Plantilla del Efecto. En **Template Type** elegimos el tipo de Plantilla a la que queremos pasar el Efecto, acto seguido, pulsamos el botón **Change Template Type** e inmediatamente todas las variables se convertirán, como por ejemplo, de `syl.center` a `char.center`, de `line.duration` a `word.duration`.

Esta opción facilita la tarea de cambiar un Efecto de un tipo de Plantilla a otra, ya que también podemos hacerlo de forma manual.

Esta opción debe tomarse con cierta calma ya que en algunas ocasiones no queremos cambiar alguna variable en especial y debemos estar atento cuando no queramos que esto suceda.

**Back <:** nos lleva de vuelta a la ventana de inicio del **Kara Effector** en caso de que hayamos pasado por alto alguna modificación en dicha ventana o que queramos cambiar de Efecto sin la necesidad de cancelar y volver a abrir.

Este es un breve repaso de los elementos que hacen parte de la segunda ventana del **Kara Effector**. No teman si hasta este punto aún tienen cosas que no han quedado del todo claras, de hecho cuento con ello, y por eso la necesidad de los próximos tomos, en donde veremos a profundidad los ítems anteriormente mencionados en este tomo y aquellos que aún ignoran.

Es todo por ahora, recordándoles que pueden escribirnos sus dudas y comentarios en el Blog Oficial del **Kara Effector** y/o en nuestros canales de **You Tube**.

**Kara Effect - Effect Book [Tomo III]:**

---

---

## **Kara Effect 3.2: Effect Book Vol. I [Tomo III]**

---

---

## **Kara Effect 3.2:**

Y ha llegado la hora de empezar a ver las Librerías con las que cuenta en **Kara Effect 3.2** y para ello he dispuesto una serie de ejemplos y listados para tratar que cada una de ellas quede lo más clara posible. El conocimiento y dominio de las Librerías es indispensable a la hora de hacer y modificar Efectos de alta calidad.

### **Librería "I" [KE]:**

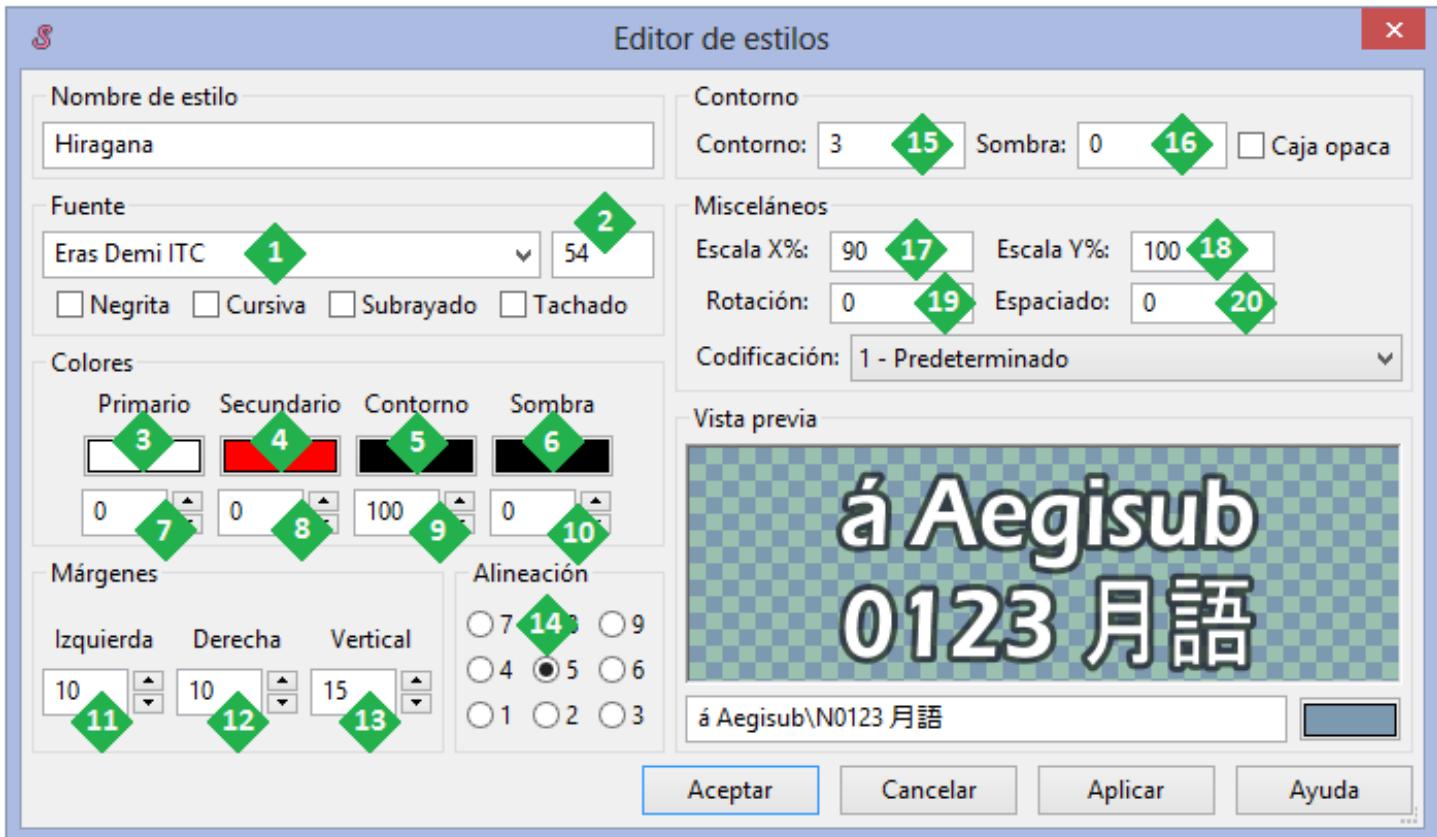
Es un listado de variables y constantes que hacen referencia a valores de la "Línea", es decir, a cada una de las líneas del archivo .ass.

La mayoría de los valores de esta Librería son los que asignamos en el Estilo de Línea en el editor de estilos del **Aegisub**:

---

---

## Kara Effector - Effector Book [Tomo III]:



1. **I.fontname**
2. **I fontsize**
3. **I.color1**
4. **I.color2**
5. **I.color3**
6. **I.color4**
7. **I.alpha1**
8. **I.alpha2**
9. **I.alpha3**
10. **I.alpha4**
11. **I.margin\_l**
12. **I.margin\_r**
13. **I.margin\_v**, **I.margin\_t**, **I.margin\_b**
14. **I.align**
15. **I.outline**
16. **I.shadow**
17. **I.scale\_x**
18. **I.scale\_y**
19. **I.angle**
20. **I.spacing**

Los anteriores 20 valores de la Librería "I" no necesitan mucha explicación, ya que son los que siempre usamos cuando creamos y modificamos un nuevo Estilo de Línea.

Un pequeño ejemplo sería:

```
Add Tags: Add Tags Language: Lua
string.format("\t(0,200,\fscx200)\t(200,1000,\fscx%s)", I.scale_x)
```

En el ejemplo, se aumenta la escala en el eje "x" en 200% desde 0 hasta 200 ms y luego desde 200 ms a 1000 ms la escala en el eje "x" vuelve a la proporción que tiene en el Estilo, es decir, 90% (como se puede apreciar en la imagen anterior del Estilo, ítem 17).

Veamos este mismo ejemplo, pero ahora en lenguaje **Automation Auto-4**. Aunque la forma de escribirlo cambie dado los dos tipos de lenguajes, el resultado es el mismo:

```
Add Tags: Add Tags Language: Automation Auto-4
\t(0,200,\fscx200)\t(200,1000,\fscx!I.scale_x!)
```

## Kara Effector - Effector Book [Tomo III]:

Y continuando con los valores de la Librería “I”, ahora veremos los que son referentes a las Líneas de Texto y a su posición en el vídeo.

**I.width:** es el Ancho medido en pixeles de cada una de las Líneas.



**I.left:** es la Distancia medida en pixeles desde la parte izquierda del vídeo hasta la parte izquierda de la Línea.



**I.center:** es la Distancia medida en pixeles desde la parte izquierda del vídeo hasta el centro de la Línea.



**I.right:** es la Distancia medida en pixeles desde la parte izquierda del vídeo hasta la parte derecha de la Línea.



**I.height:** es la Altura medida en pixeles de cada una de las Líneas.



**I.top:** es la Distancia medida en pixeles desde la parte superior del vídeo hasta la parte superior de la Línea.



Notarán que la parte superior de la Línea no coincide con la parte superior de sus letras más grandes, esto se debe a que hay un espacio extra que sirve para separar las Líneas verticalmente cuando estas están una encima de la otra. Ya que de otro modo las Líneas se verían pegadas.

**I.middle:** es la Distancia medida en pixeles desde la parte superior del vídeo hasta la mitad de la altura de la Línea.



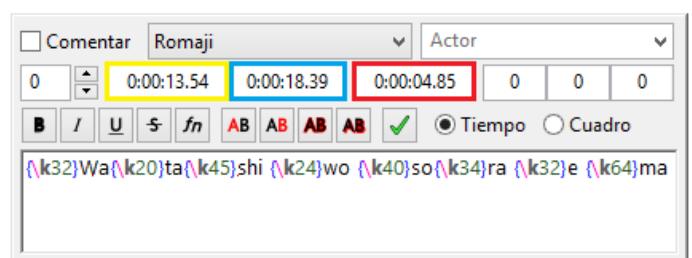
**I.bottom:** es la Distancia medida en pixeles desde la parte superior del vídeo hasta la parte inferior de la Línea.



**I.descent:** es la Distancia medida en pixeles desde la parte superior de la Línea hasta la parte superior Real de la misma. Es la Distancia que separa verticalmente una Línea de otra.



Para terminar, los valores de la Librería “I” que hacen falta son los que tienen referencia con el tiempo y su forma de escritura:



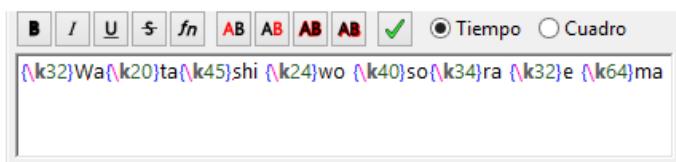
**I.start\_time:** (recuadro amarillo de la anterior imagen) es el tiempo de inicio medido en ms (milisegundos) de cada Línea.

**I.end\_time:** (recuadro azul) es el tiempo final medido en (ms) de cada Línea.

## Kara Effector - Effector Book [Tomo III]:

**I.duration:** (recuadro rojo) es la duración total medida en (ms) de cada Línea de Dialogo del archivo .ass.

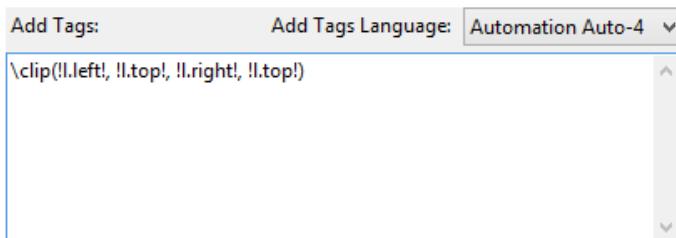
**I.text:** es todo lo que está escrito en la Línea, incluidos los tags:



**I.text\_stripped:** es similar a **I.text**, pero con la diferencia que no tiene en cuenta a los tags que tenga dicha Línea:

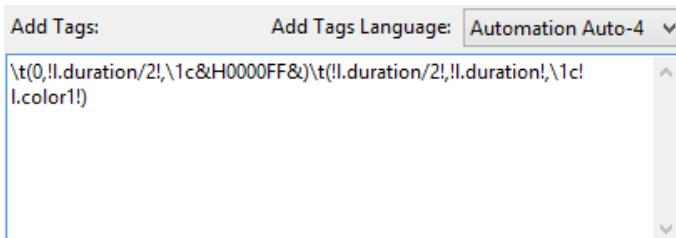


El uso de cada uno de los valores de la Librería “I” o de las próximas librerías que veremos, facilitará la forma de hacer Efectos que en principio serían mucho más complejos. Acá otro corto ejemplo de cómo poder usar la Librería “I”:



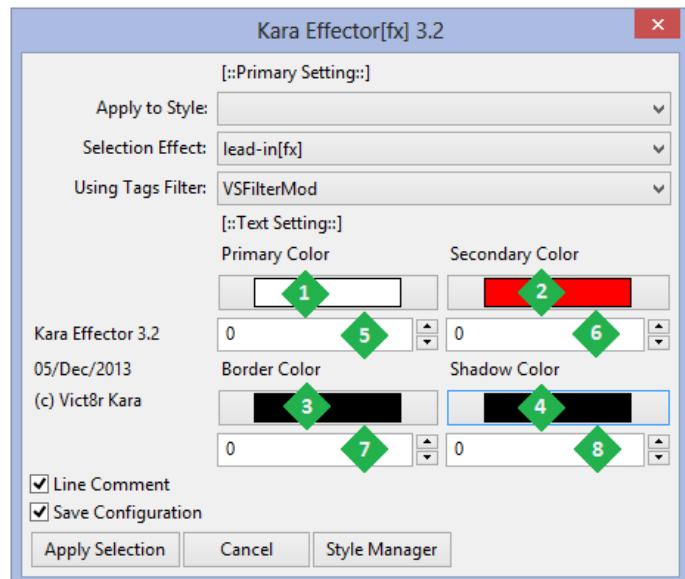
Es un clip en donde es visible toda la Línea.

En este otro ejemplo usamos **I.duration** para generar una transformación del color primario. Desde 0 hasta la mitad de la duración, el color primario se transforma en Rojo (&H0000FF&: Rojo en formato .ass) y desde la mitad de la duración hasta la duración total, regresa a su color original asignado en el Estilo (**I.color1**):

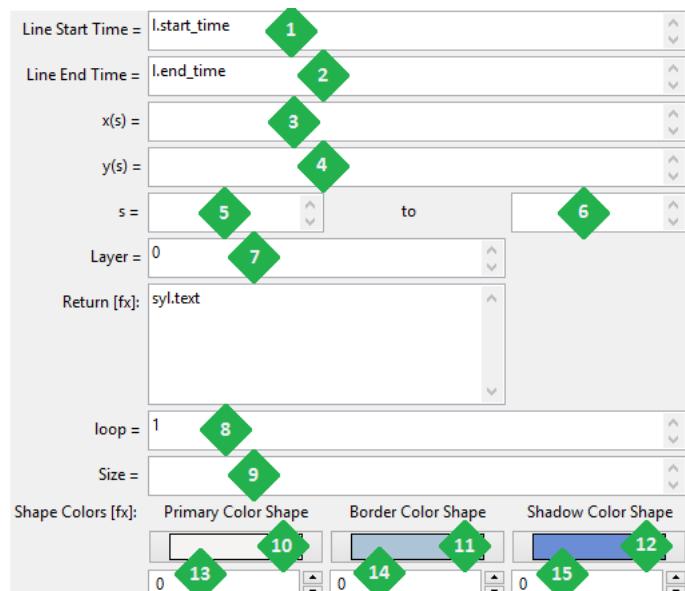


## Librería “fx”:

Es la Librería que contiene los valores del Kara Effector.



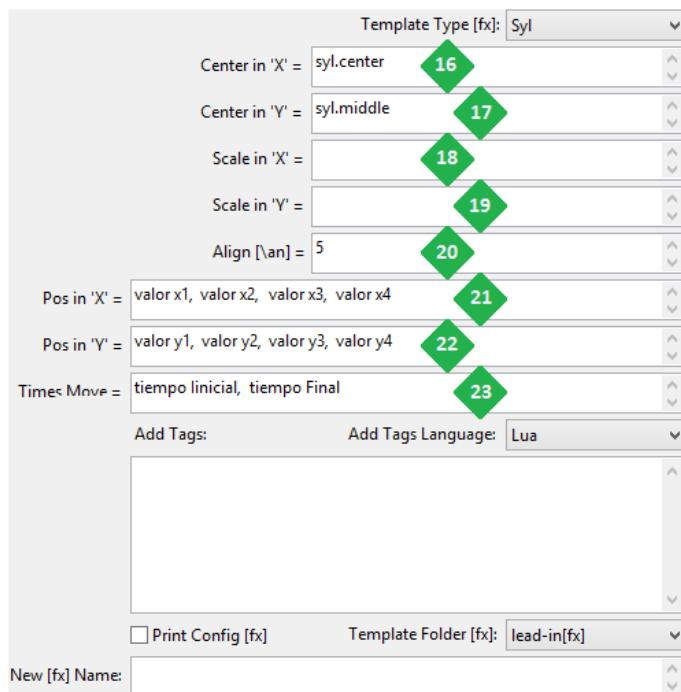
1. **text.color1**
2. **text.color2**
3. **text.color3**
4. **text.color4**
5. **text.alpha1**
6. **text.alpha2**
7. **text.alpha3**
8. **text.alpha4**



1. **fx.start\_time**

## Kara Effector - Effector Book [Tomo III]:

2. **fx.end\_time**
3. **fx.fun\_x**
4. **fx.fun\_y**
5. **fx.domain\_i**
6. **fx.domain\_f**
7. **fx.layer**
8. **fx.maxloop\_fx**
9. **fx.sizeX, fx.sizeY**
10. **shape.color1**
11. **shape.color3**
12. **shape.color4**
13. **shape.alpha1**
14. **shape.alpha3**
15. **shape.alpha4**



16. **fx.center\_x**
17. **fx.center\_y**
18. **fx.scale\_x**
19. **fx.scale\_y**
20. **fx.align**
21. **fx.move\_x1, fx.move\_x2, fx.move\_x3, fx.move\_x4**
22. **fx.move\_y1, fx.move\_y2, fx.move\_y3, fx.move\_y4**
23. **fx.movet\_i, fx.movet\_f**

No parecen ser muchas, pero son suficientes. Ahora veremos una pequeña explicación de cada una de ellas.

Bueno, considero que los valores de la ventana de inicio del **Kara Effector** no necesitan de mucha explicación ya que son los colores y transparencias que tendrán por default cada una de las Líneas de Efecto generadas.

**fx.start\_time:** es el tiempo de inicio de cada una de las Líneas generadas por un Efecto. Puede ser modificado directamente desde su respectiva celda de texto o con la función **retime** desde “**Add Tags**” o alguna función hecha en “**Variables**”.

**fx.end\_time:** es el tiempo final de cada una de las Líneas generadas por un Efecto. Este valor puede ser modificado directamente desde su respectiva celda de texto o con la función **retime** desde “**Add Tags**” o alguna función hecha en “**Variables**”. La variable **fx.start\_time** puede ser usada en esta celda de texto, ejemplo:

Line Start Time = `I.start_time + math.random(-2000, 2000)`  
Line End Time = `fx.start_time + 500`

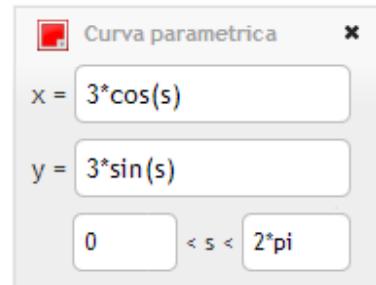
O sea que **fx.start\_time** es un valor aleatorio entre -2000 ms y 2000 ms respecto al tiempo de inicio original de cada Línea seleccionada para aplicarle un Efecto (**I.start\_time**).

Por otro lado, **fx.end\_time** pone que será igual al tiempo en donde inició la Línea de fx (**fx.start\_time**), sumado a 500 ms. Es fácil deducir que el **fx.dur** de todas las Líneas que se generen con estos dos tiempos, siempre será de 500 ms. (véase la siguiente variable).

**fx.dur:** es la duración total de cada Línea fx que sea generada por un Efecto. Es equivalente a la siguiente diferencia:

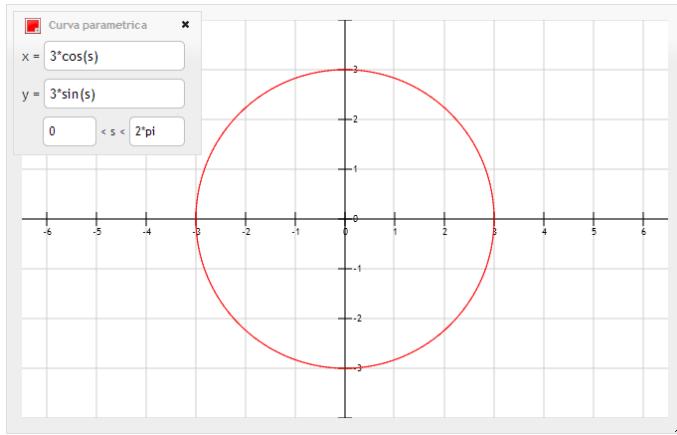
$$\text{fx.dur} = \text{fx.start_time} - \text{fx.end_time}$$

**fx.fun\_x:** es la ecuación paramétrica de “x” en términos de “s” (donde “s” es el dominio de la función). Ejemplo:



## Kara Effector - Effector Book [Tomo III]:

En la anterior imagen están las ecuaciones paramétricas de "x" y "y" de un Círculo de Radio 3. También podemos ver el dominio "s" que va desde 0 a  $2\pi$ :



La anterior gráfica fue generada en <http://fooplot.com>

En el Kara Effector pondríamos así:

```
x(s) = 3*cos(s)  
y(s) = 3*sin(s)  
s = 0 to 2*pi
```

Las escalas en el **Kara Effector**, tanto en el eje "x" como en el "y" son por default 1, es decir que este círculo se verá en el vídeo muy pequeño, ya que el Radio del Círculo es solo de 3 pixeles. Hay dos formas de aumentar su tamaño: o se aumenta el Radio del Círculo directamente en las ecuaciones paramétricas o se aumentan las escalas en las celdas de texto que están al lado derecho de estas, todo depende del gusto y del nivel de habilidad adquirido de cada uno.

En estas celdas de texto podemos modificar las Escalas en ambos ejes de una determinada gráfica:

```
Scale in 'X' =  
Scale in 'Y' =
```

Olivaba mencionar que este tipo de Efecto se hacen con Shapes y con un **loop** lo suficientemente grande para que se generen la gráficas.

Para un ejemplo guiado en el **Kara Effector** tomando como base las ecuaciones paramétricas del Círculo, sería:

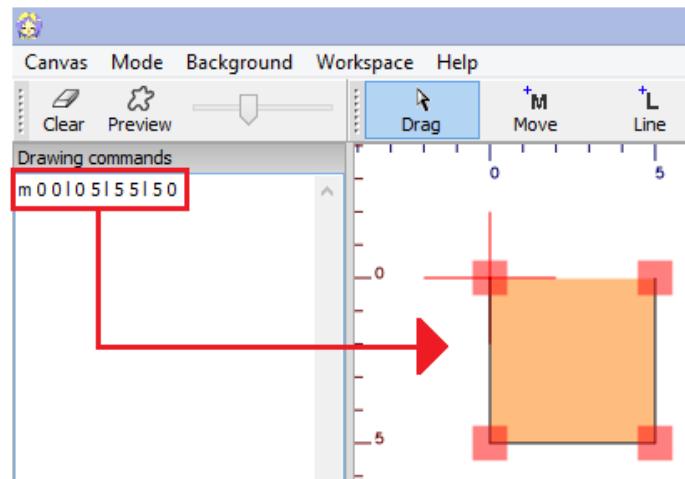
- A. Definimos las ecuaciones paramétricas y el dominio de las funciones ("s"):

```
x(s) = 3*cos(s)  
y(s) = 3*sin(s)  
s = 0 to 2*pi
```

- B. Aumentamos las escalas en 10 para que el Radio del Círculo pase de 3 pixeles a  $3 \times 10 = 30$  pixeles. (Se pude experimentar con distintos valores en las escalas y ver cómo varían las gráficas):

```
Scale in 'X' = 10  
Scale in 'Y' = 10
```

- C. Elegimos una **Shape** y la ponemos en **Return [fx]**:



En mi caso, dibujé un Cuadrado en el **AssDraw3**, de 5 pixeles de lado y copié el código de la Shape en **Return [fx]**:

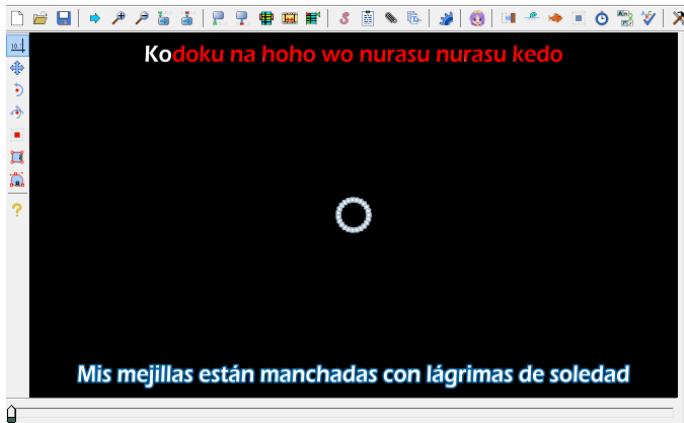
```
Return [fx]: m 00105155150
```

- D. Aumentamos el **loop** para que generará la gráfica:

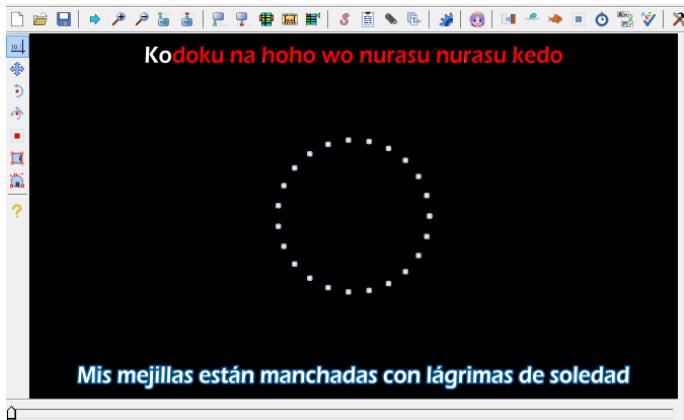
```
loop = 24
```

Para este ejemplo puse en **Template Type** la opción "Line" para que se genere un solo Círculo por cada Línea:

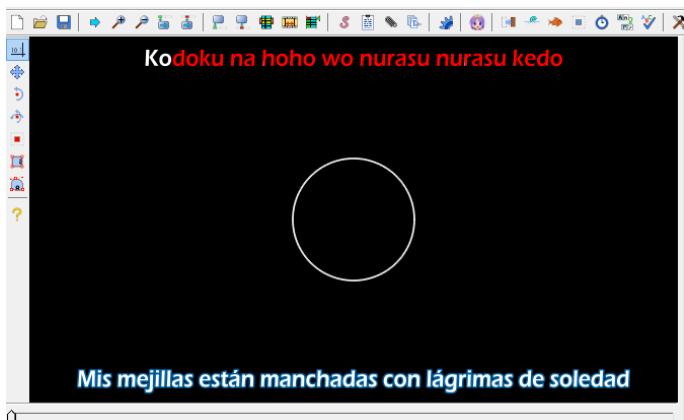
## Kara Effector - Effector Book [Tomo III]:



Ya se puede ver en el vídeo el Círculo que se generó, pero parece que 30 pixeles para el Radio es poco para poder ver los 24 cuadraditos (loop) de 5 X 5 pixeles que lo conforman, por ello aumentaré las escalas en ambos ejes a 50 y volveré a aplicar el Efecto:



Ahora sí se pueden apreciar los cuadraditos que generan el Círculo. La cantidad de cuadrados, lo mismo que el valor de las escalas, tamaños (**size**) y colores, se pueden modificar para que la gráfica sea más continua, ejemplo:



Es un ejemplo de cómo “graficar” en el **Kara Effector**.

Las configuraciones para generar el anterior Círculo son:

lead-in[fx]: ABC Template

Line Start Time = l.start\_time

Line End Time = l.end\_time

x(s) =  $3\cos(s)$

y(s) =  $3\sin(s)$

s = 0 to  $2\pi$

Layer = 0

Return [fx]: m 0 0 1 0 5 1 5 1 5 0

Pos in 'X' = Times Move =

loop = 240

Size = 70

Shape Colors [fx]: Primary Color Shape Border Color Shape Shadow Color Shape

0 255 0

Template Type [fx]: Line

Center in 'X' = line.center

Center in 'Y' = line.middle

Scale in 'X' = 40

Scale in 'Y' = 40

Align [\an] = 5

Pos in 'X' = fx.pos\_x

Pos in 'Y' = fx.pos\_y

En próximos ejemplos veremos cómo hacer un Efecto a partir de la gráfica de las ecuaciones paramétricas.

**fx.fun\_y:** es la ecuación paramétrica de “y” en términos de “s” (donde “s” es el dominio de la función). Ejemplo:

Curva parametrica

x =  $3\cos(s)$

y =  $3\sin(s)$

0 < s <  $2\pi$

Con el ejemplo anterior del Círculo, esta variable de la Librería “fx” no necesita muchas más explicación y por ello iremos directamente a las siguientes variables.

## Kara Effector - Effector Book [Tomo III]:

**fx.domain\_i:** es el inicio del dominio “s” de las ecuaciones paramétricas. Si en esta celda de texto no pone nada, el inicio del dominio por default es 0.

**fx.domain\_f:** es el final del dominio “s” de las ecuaciones paramétricas. Si en esta celda de texto no pone nada, el final del dominio por default es 1.

No necesariamente el inicio del dominio “s” debe ser menor que el final, lo convencional es que sí lo sea para que la gráfica se dibuje normalmente de menor a mayor, de lo contrario de dibujará al revés. Este orden cobra importancia cuando hacemos animaciones con las gráficas y las vemos en el vídeo.

**fx.layer:** es la capa de cada una de las Líneas de fx y decide la prioridad en el vídeo de dos o más Objetos Karaoke que coinciden de forma total o parcial en su posición o trayectoria de movimiento.

**fx.maxloop\_fx:** este valor equivale a la cantidad total de repeticiones de cada una de las Líneas fx. Se puede usar con este nombre o con uno más conocido por aquellos que ya conocen algo de **Automation Auto-4: maxj**

$$\text{maxj} = \text{fx.maxloop_fx}$$

El valor de **maxj** (Máximo valor de “j”) se puede modificar directamente en la celda de texto destinada para ello o con la función **maxloop**.

En el caso de haber un solo valor en esta celda, el valor de **maxj** es ese mismo valor:

loop = 2

En este caso, **maxj** = 2

Para el caso de dos valores:

loop = 2, 5

Para este caso, **maxj** =  $2 \times 5 = 10$

Y para el caso de que haya tres valores:

loop = 2, 5, 4

Y para este último caso, **maxj** =  $2 \times 5 \times 4 = 40$

3 es el número máximo de valores que se pueden poner en esta celda de texto. Cuando alguno de ellos o todos, no están, su valor por default es 1.

Más adelante veremos por qué la necesidad de más de un valor en esta celda de texto, pero por ahora les mostraré la variable asignada a cada uno de ellos, ejemplo:

loop = 2, 5, 4

**fx.loop\_v:** (el 2 en la imagen) es el loop vertical.

**fx.loop\_h:** (el 5 en la imagen) es el loop horizontal.

**fx.loop\_3:** (el 4 en la imagen) es un loop de respaldo.

Como lo mencionaba antes, si alguno de estos tres valores no está, por default es 1. Lo que quedaría:

$$\text{fx.maxloop_fx} = \text{fx.loop_v} * \text{fx.loop_h} * \text{fx.loop_3}$$

$$\text{maxj} = \text{fx.loop_v} * \text{fx.loop_h} * \text{fx.loop_3}$$

$$\text{maxj} = \text{fx.maxloop_fx}$$

**fx.sizex:** es el porcentaje del tamaño con respecto al eje “x” asociado al tag \fscx, ejemplo:

Size = 80

En este ejemplo, **fx.sizex** valdría 80 y en el Efecto se verán los siguientes tags: \fscx80\fscy80

Es decir que si solo hay un valor en esta celda de texto, el porcentaje en el eje “y” será el mismo que para el eje “x”, o sea 80% para los dos ejes.

**fx.sizey:** es el porcentaje del tamaño con respecto al eje “y” asociado al tag \fscy, ejemplo:

Size = 80, 125

O sea: **fx.sizex** = 80 y **fx.sizey** = 125

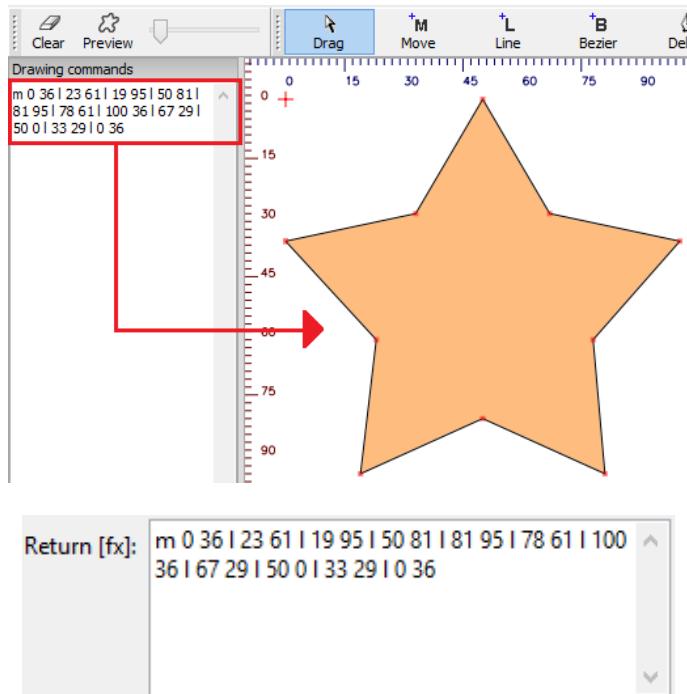
Y en tags: \fscx80\fscy125

Si esta celda de texto está vacía los valores por default de estas dos variables son:

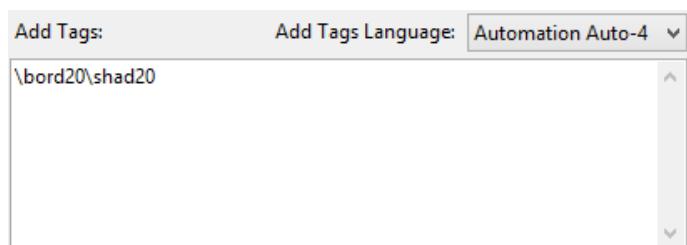
$$\text{fx.sizex} = \text{l.scale\_x} \quad \text{y} \quad \text{fx.sizey} = \text{l.scale\_y}$$

## Kara Effector - Effector Book [Tomo III]:

Para las variables de la Librería “fx” concernientes a las figuras hechas en el **AssDraw3** usaré la siguiente Shape:

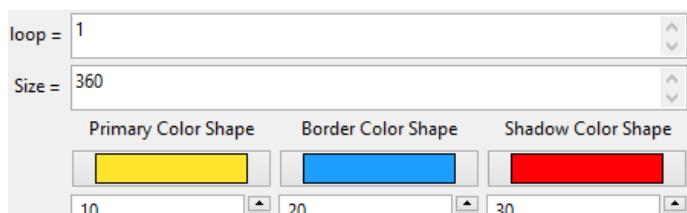


Y le agregaré los siguientes tags:

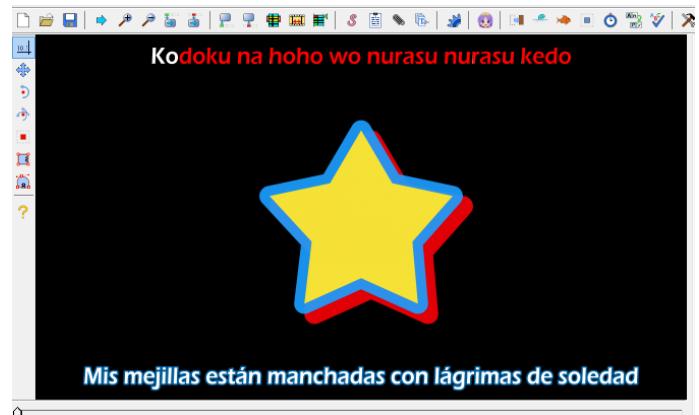


O sea, 20 pixeles para el tamaño del borde y otros 20 para el tamaño de la sombra.

Además de esto, las siguientes configuraciones:



Y al aplicar el Efecto, se verá en pantalla la Shape con las configuraciones que le di:



**shape.color1:** es el color Primario de la Shape (amarillo).

**shape.color3:** es el color del Borde de la Shape (azul).

**shape.color4:** es el color de la Sombra de la Shape (rojo).

**shape.alpha1:** es la transparencia del color Primario de la Shape (10 para este ejemplo).

**shape.alpha3:** es la transparencia del color del Borde de la Shape (20).

**shape.alpha4:** es la transparencia del color de la Sombra de la Shape (30).

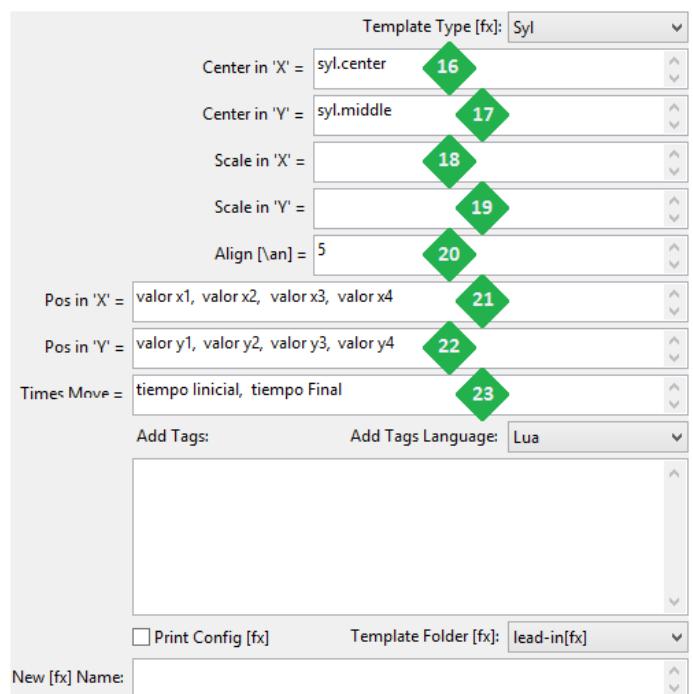
Para el **Tomo IV** terminaremos con la explicación de las variables restantes de la Librería “fx” y veremos más de las librerías del **Kara Effector**. No olviden visitarnos en el **Blog Oficial** lo mismo que en los canales de **YouTube** para descargar los nuevos Efectos o si quieren comentar, exponer alguna duda o hacer alguna sugerencia.

# Kara Effector 3.2: Effector Book Vol. I [Tomo IV]

# Kara Effector 3.2:

El inicio de este **Tomo IV** veremos la segunda parte de la Librería “fx” de **Kara Effector**. Estas son las variables que quedaron sin explicar del **Tomo** anterior y a continuación veremos un poco más de cada una de ellas:

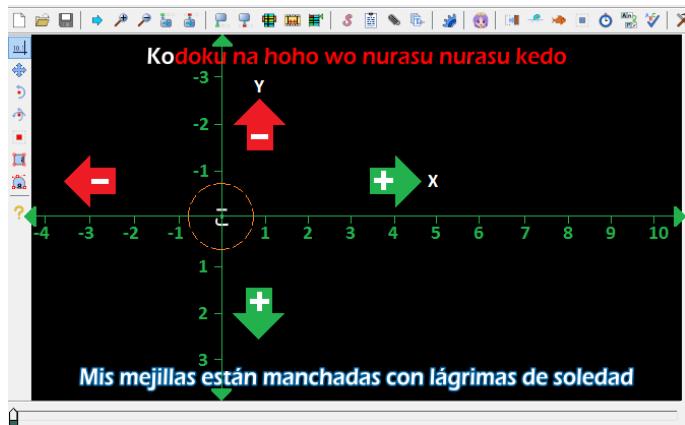
## Librería “fx” [KE]:



16. `fx.center_x`
17. `fx.center_y`
18. `fx.scale_x`
19. `fx.scale_y`
20. `fx.align`
21. `fx.move_x1, fx.move_x2, fx.move_x3, fx.move_x4`
22. `fx.move_y1, fx.move_y2, fx.move_y3, fx.move_y4`
23. `fx.movet_i, fx.movet_f`

## Kara Effector - Effector Book [Tomo IV]:

**fx.center\_x:** es el Centro medido en pixeles con respecto al eje “x” y hace las veces de la Abscisa al Origen de un sistema de coordenadas cartesianas para cada Línea de fx.



En la imagen anterior dibujé las coordenadas cartesianas con el origen en el centro del Hiragana “ko”:



La coordenada en “x” de ese origen sería **syl.center** que como su nombre lo indica, es el centro de la Sílaba. Nótese que en los archivos .ass, con respecto al eje “y”, hacia arriba es negativo y hacia abajo es positivo, es decir que este eje está invertido.

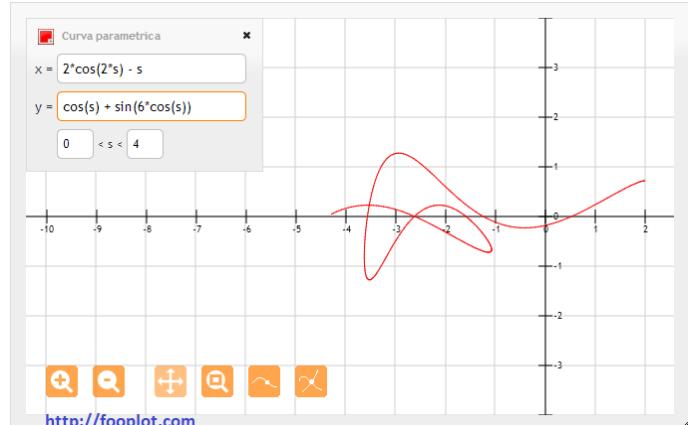
**fx.center\_y:** es el Centro medido en pixeles con respecto al eje “y” y hace las veces de la Ordenada al Origen de un sistema de coordenadas cartesianas para cada Línea de fx.

Como podemos ver en el ejemplo del Hiragana “ko”, sería **syl.middle**, que es el centro de la Sílaba, pero medido verticalmente.

**syl.center** y **syl.middle** son variables de la Librería “syl” que originalmente ya viene en el Aegisub por default, pero que también explicaré en los próximos tomos, ya que he agregado algunas variables más a esta Librería y es necesario que sepamos en qué consisten dichas variables.

**fx.scale\_x:** junto a **fx.scale\_y**, es la Escala con respecto al eje “x” de las gráficas que se generen con ecuaciones paramétricas.

**fx.scale\_y:** es la Escala con respecto al eje “y” de las gráficas que se generen con ecuaciones paramétricas.



Acá hay otro ejemplo de una Gráfica generada por ecuaciones paramétricas, y son este tipo de gráficas las que serán afectadas por las anteriores Escalas. El valor por default de ambas Escalas es 1.

Una vez conocidas las variables **fx.center\_x**, **fx.center\_y**, **fx.fun\_x**, **fx.fun\_y**, **fx.scale\_x** y **fx.scale\_y**; es importante que veamos las siguientes variables:

**fx.pos\_x:** es la coordenada en “x” de la posición final de la Línea fx luego de haber asignado valores a las variables **fx.center\_x**, **fx.center\_y**, **fx.fun\_x**, **fx.fun\_y**, **fx.scale\_x** y **fx.scale\_y**.

$$fx.pos_x = fx.center_x + fx.fun_x * fx.scale_x$$

**fx.pos\_y:** es la coordenada en “y” de la posición final de la Línea fx luego de haber asignado valores a las variables **fx.center\_x**, **fx.center\_y**, **fx.fun\_x**, **fx.fun\_y**, **fx.scale\_x** y **fx.scale\_y**.

$$fx.pos_y = fx.center_y + fx.fun_y * fx.scale_y$$

Veamos un ejemplo para tener claro lo de la Posición final:

Center in 'X' =	<input type="text" value="syl.center"/>		
Center in 'Y' =	<input type="text" value="syl.middle"/>		
x(s) =	<input type="text" value="cos(s)"/>	Scale in 'X' =	<input type="text" value="2"/>
y(s) =	<input type="text" value="sin(s)"/>	Scale in 'Y' =	<input type="text" value="4"/>
Pos in 'X' =	<input type="text" value="fx.pos_x"/>		
Pos in 'Y' =	<input type="text" value="fx.pos_y"/>		

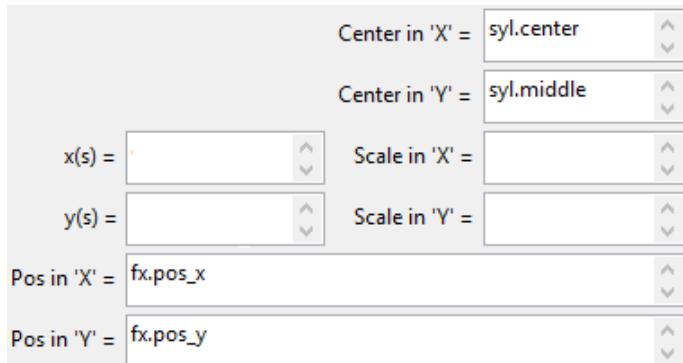
## Kara Effector - Effector Book [Tomo IV]:

Para este ejemplo obtendríamos los siguientes valores:

**fx.pos\_x = syl.center + cos(s) \* 2**

**fx.pos\_y = syl.middle + sin(s) \* 4**

El valor por default de **fx.fun\_x** y **fx.fun\_y** es 0, teniendo en cuenta esto, para el siguiente ejemplo tenemos:

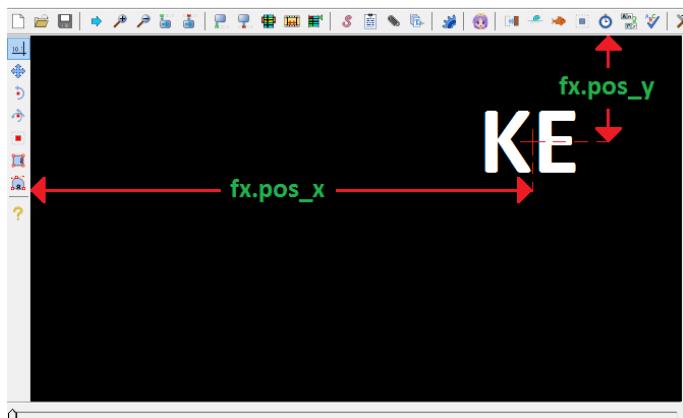


**fx.pos\_x = syl.center + 0 \* 1 = syl.center**

**fx.pos\_y = syl.middle + 0 \* 1 = syl.middle**

Los ceros de las anteriores ecuaciones corresponden a los valores por default de **fx.fun\_x** y **fx.fun\_y**, ya que en la imagen ambas celdas están vacías. Los unos son los valores por default de **fx.scale\_x** y **fx.scale\_y**, dado que también están vacías sus respectivas celdas de texto.

Entonces concluimos que **fx.pos\_x** y **fx.pos\_y** son los centros de la posición final de cada una de las Líneas fx:

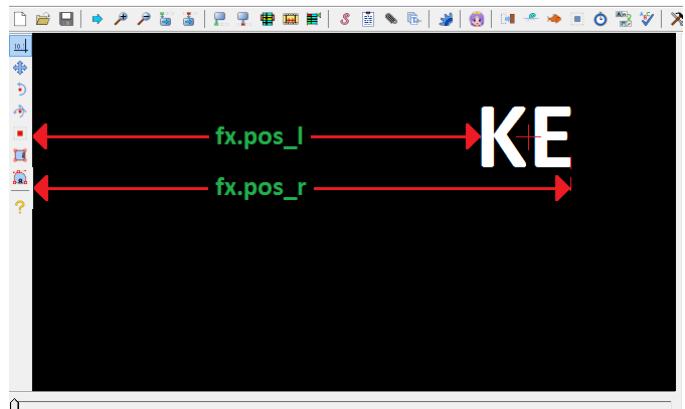


Una vez aclarados los colores de las anteriores dos variables, será más simple entender las cuatro siguientes, ya que dependen directamente de ellas.

**fx.pos\_l** y **fx.pos\_r**: (left y right) son la parte izquierda y derecha de **fx.pos\_x** respectivamente.

La izquierda y la derecha de **fx.pos\_x** dependen del “Template Type”, es decir que depende del tipo de plantilla del Efecto. Ejemplo:

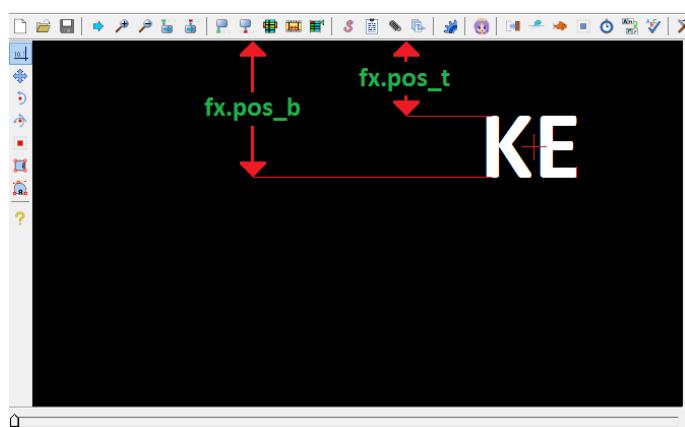
Si la plantilla es tipo “**Syl**” entonces **fx.pos\_l** y **fx.pos\_r** serán la izquierda y la derecha de la Sílaba, sin importar en dónde esté esa Sílaba:



Para Template Type “**Line**”, entonces **fx.pos\_l** y **fx.pos\_r** serán la izquierda y la derecha de la Línea de Texto. Lo mismo pasaría para los demás tipos de Plantillas: “**Word**”, “**Furi**”, “**Char**” y las demás.

**fx.pos\_t** y **fx.pos\_b**: (top y bottom) son la parte superior e inferior de **fx.pos\_y** respectivamente.

Como en el caso de las dos variables anteriores, la parte superior e inferior depende únicamente de la posición final:



**fx.align**: es la alineación de la Línea fx. Debe ser un valor entero entre 1 y 9, inclusive. El valor por default de esta variable es 5. 5 es la alineación recomendada para aquellos que aún no tienen mucha experiencia a la hora de hacer Efectos Karaoke, pero todas son importantes.

## Kara Effector - Effector Book [Tomo IV]:

**fx.move\_x1, fx.move\_x2, fx.move\_x3, fx.move\_x4:** son la cuatro posibles coordenadas con respecto al eje "x" de las posición o de la trayectoria del movimiento de la Línea fx. Son las coordenadas en "x" de los tags: \pos, \move, \moves3, \moves4 y \mover.

**fx.move\_y1, fx.move\_y2, fx.move\_y3, fx.move\_y4:** son la cuatro posibles coordenadas con respecto al eje "y" de las posición o de la trayectoria del movimiento de la Línea fx. Son las coordenadas en "y" de los tags: \pos, \move, \moves3, \moves4 y \mover.

Veamos algunos ejemplos de la anteriores ocho variables y de a poco las iremos aclarando:

Pos in 'X' =	fx.pos_x - 45
Pos in 'Y' =	fx.pos_y

**fx.move\_x1 = fx.pos\_x - 45**

**fx.move\_y1 = fx.pos\_y**

Los valores por default de **fx.move\_x1** y **fx.move\_y1** son, **fx.pos\_x** y **fx.pos\_y**, respectivamente. Del ejemplo de la anterior imagen, su resultado en la Línea fx sería el tag \pos, ya que solo hay una coordenada para ambos ejes.

Este ejemplo también retornaría un tag \pos:

Pos in 'X' =	
Pos in 'Y' =	fx.pos_y

**fx.move\_x1 = fx.pos\_x** ← por Default

**fx.move\_y1 = fx.pos\_y**

Para el caso de que ambas celdas de texto estén vacías, no retornaría ningún tag de posición ni de movimiento:

Pos in 'X' =	
Pos in 'Y' =	

El valor por default de **fx.move\_x2**, **fx.move\_y2** y todas las superiores, es siempre la variable inmediatamente a cada una de ellas:

Pos in 'X' =	
Pos in 'Y' =	fx.pos_y, fx.pos_y + 20

**fx.move\_x1 = fx.pos\_x** ← por Default

**fx.move\_x2 = fx.move\_x1** ← por Default

**fx.move\_y1 = fx.pos\_y**

**fx.move\_y2 = fx.pos\_y + 20**

El ejemplo anterior daría como resultado un tag \move, ya que habría dos coordenadas para cada eje.

Para el próximo ejemplo, veremos el caso cuando ambos tienen tres coordenadas o al menos una celda de texto tiene tres coordenadas:

Pos in 'X' =	fx.pos_x, fx.pos_x - math.random(-30, 70)
Pos in 'Y' =	fx.pos_y, fx.pos_y + 25, fx.pos_y - 25

**fx.move\_x1 = fx.pos\_x**

**fx.move\_x2 = fx.pos\_x - math.random(-30,70)**

**fx.move\_x3 = fx.move\_x2** ← por Default

**fx.move\_y1 = fx.pos\_y**

**fx.move\_y2 = fx.pos\_y + 25**

**fx.move\_y3 = fx.pos\_y - 25**

El tag que resultaría de este ejemplo sería un \moves3. Para el caso en el que al menos una de las dos celdas de texto tenga cuatro coordenadas, veamos este ejemplo:

Pos in 'X' =	50, 120, -80, syl.center
Pos in 'Y' =	200, 400, 680, syl.middle

**fx.move\_x1 = 50**

**fx.move\_x2 = 120**

**fx.move\_x3 = -80**

**fx.move\_x4 = syl.center**

**fx.move\_y1 = 200**

**fx.move\_y2 = 400**

**fx.move\_y3 = 680**

**fx.move\_y4 = syl.middle**

## Kara Effector - Effector Book [Tomo IV]:

Para el caso del ejemplo anterior obtendríamos como tag un \moves4, lo que quiere decir que el tag que resulta de la combinación de **Pos in 'X'** y **Pos in 'Y'** depende de cuál de estas dos celdas de texto tenga más coordenadas.

Y para el caso del tag \mover hacemos lo siguiente:

Pos in 'X' =	x1, x2, Angle1, Angle2, Radius1, Radius2
Pos in 'Y' =	y1, y2

**fx.movet\_i**, **fx.movet\_f**: son los tiempo de inicio y final para los tags de movimiento: \move, \moves3, \moves4 y \mover:

Times Move =	t1, t2
--------------	--------

“t1” representa el tiempo de inicio del movimiento y “t2” el tiempo final. Sus valores por default son 0 y **fx.dur**, respectivamente.

Veamos algunos ejemplos:

Pos in 'X' =	x1, x2, x3
Pos in 'Y' =	y1, y2, y3
Times Move =	t1, t2

→ \moves3(x1, y1, x2, y2, x3, y4, t1, t2)

El anterior es un ejemplo sencillo de cómo obtener un \moves3 junto con los parámetros de tiempo incluidos.

Pos in 'X' =	x1
Pos in 'Y' =	y1, y2, y3
Times Move =	

→ \moves3(x1, y1, x1, y2, x1, y4)

Este ejemplo ilustra cómo **fx.move\_x2** y **fx.move\_x3** son **fx.move\_x1** por default, y al no haber parámetros de tiempo, entonces no salen en el tag.

Pos in 'X' =	x1, x2
Pos in 'Y' =	y1, y2
Times Move =	t1

→ \move(x1, y1, x2, y2, t1, fx.dur)

Y en este otro ejemplo vemos que **fx.movet\_f** equivale a **fx.dur** (duración de la Línea fx) por default.

Todo tag de posición o de movimiento que hagamos utilizando estas tres celdas de texto:

Pos in 'X' =	fx.pos_x
Pos in 'Y' =	fx.pos_y
Times Move =	

Se verá reflejado en las Líneas fx:

B	I	U	S	fn	AB	AB	AB	AB	✓	<input checked="" type="radio"/> Tiempo	<input type="radio"/> Cuadro
(Kara Effector[fx] 3.2: ABC Template \an5\pos(308,34,42))do											

Pero no es la única forma de usar los tags de posición y movimientos, ya que desde **Add Tags** también lo podemos hacer, de hecho, se si hace desde **Add Tags**, entonces cualquier otro tag de posición o de movimiento es anulado:

Pos in 'X' =	fx.pos_x
Pos in 'Y' =	fx.pos_y
Times Move =	
Add Tags:	Add Tags Language: Lua
\move(10,20,30,40)	

El **Kara Effector** detecta de forma automática que hay un tag de movimiento (\move) en **Add Tags**, entonces anula el tag \pos que se iba a generar por las configuraciones de **Pos in 'X'** y **Pos in 'Y'**:

0	0:00:02.43	0:00:08.16	0:00:05.73	0	0	0					
B	I	U	S	fn	AB	AB	AB	AB	✓	<input checked="" type="radio"/> Tiempo	<input type="radio"/> Cuadro
(Kara Effector[fx] 3.2: ABC Template \move(10,20,30,40))Ko											

Si por algún motivo se coloca más de un tag de posición o de movimiento en **Add Tags**, el **Kara Effector** solo tomará en cuenta el último de ellos:

Add Tags:	Add Tags Language: Automation Auto-4
\move(100,200,120,300,0,1000)\pos(\$center,\$middle)	

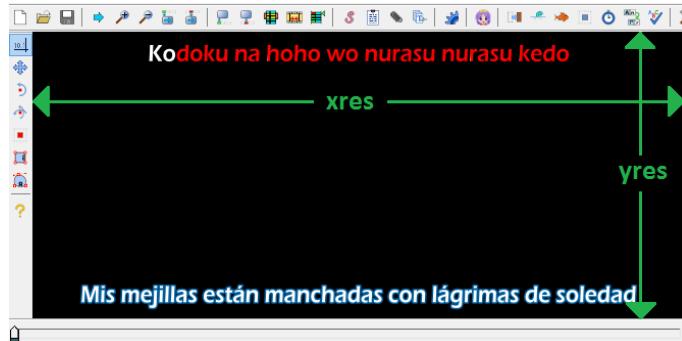
En este caso solo se toma en cuenta el tag \pos ya que es el último y el tag \move no saldrá en la Línea de fx.

## Kara Effector - Effector Book [Tomo IV]:

Con los anteriores ejemplos concluye la explicación de la Librería "fx" pero hay otra serie de variables y valores propios del **Kara Effector** que veremos a continuación:

**xres** y **yres**: son las dimensiones medidas en pixeles del vídeo que se esté usando al momento de aplicar un Efecto y sus valores por default son 1280 y 720 p.

**xres** es el ancho del vídeo y **yres**, el alto:



**ratio**: es la Proporción que usa el **Kara Effector** para que un Efecto funcione exactamente igual sin depender de las dimensiones del vídeo. Al aplicar un Efecto con un vídeo abierto, el valor del **ratio** es: **xres/1280**, para el caso contrario su valor por default es 1.

**frame\_dur**: es la duración medida en milisegundos de cada uno de los cuadros (**frames**) del vídeo que se esté usando al momento de aplicar un Efecto. Su valor por default es de **41.708** ms.

**line.i**: es el Contador numérico de cada una de las Líneas seleccionadas a las que le aplicaremos un Efecto. Es similar a la variable **syl.i**, salvo que esta última es el contador numérico de las Sílabas que contiene una Línea.

**line.n**: es la cantidad total de Líneas seleccionadas para aplicar un Efecto. Es similar a la variable **syl.n**, salvo que esta última es la cantidad total de Sílabas que contiene una Línea.

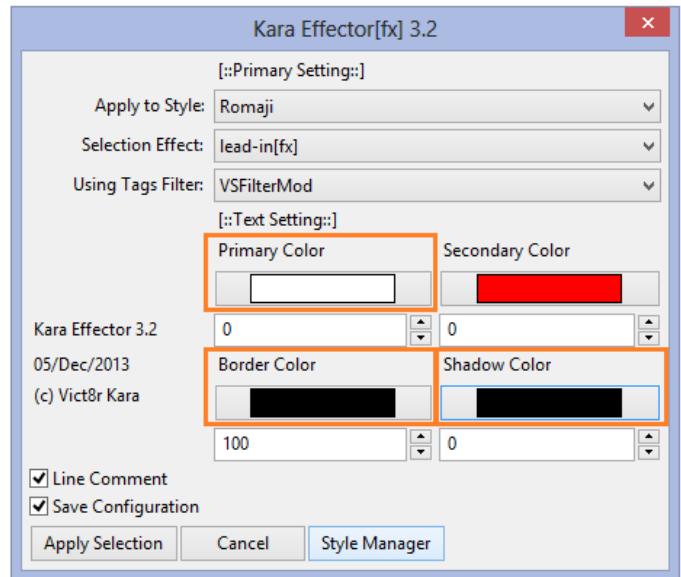
**line.index**: es el contador de todas las Líneas de Dialogo de un archivo .ass y también puede ser llamado como: **ii**

**text.color**: son los tres tag de color de la ventana de inicio del Kara Effector. En Lenguaje **Automation Auto-4** sería:

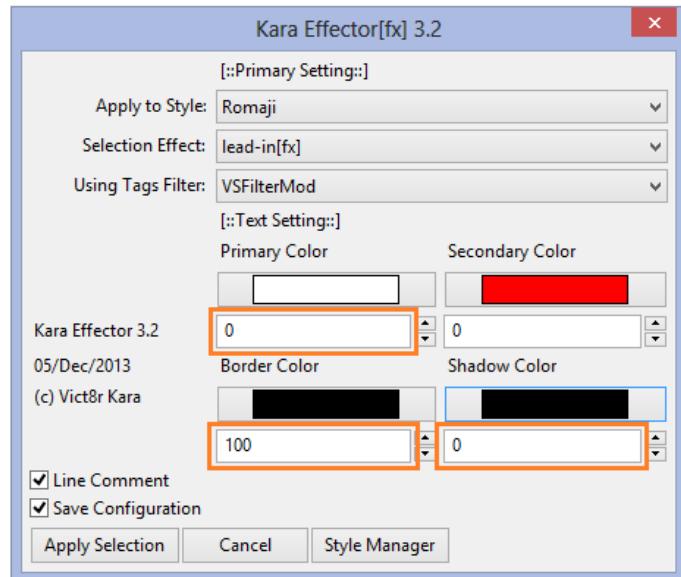
```
\1c!text.color1!\3c!text.color3!\4c!text.color4!
```

El formato en que se ven depende del filtro seleccionado.

Es decir que **text.color** equivale a estos tres colores:



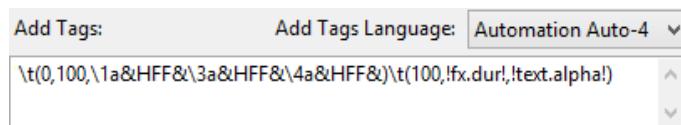
**text.alpha**: es similar a **text.color**, pero hace referencia a las transparencias de los tres colores anteriormente mencionados:



En Lenguaje **Automation Auto-4** sería:

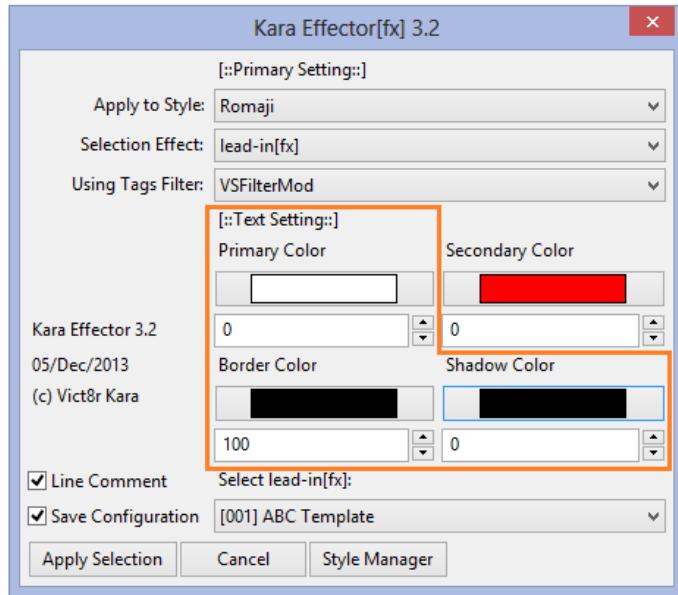
```
\1a!text.alpha1!\3a!text.alpha3!\4a!text.alpha4!
```

Tanto **text.color** y **text.alpha** pueden servir para volver a las configuraciones de la ventana de inicio luego de alguna transformación de colores y/o de transparencias, ejemplo:



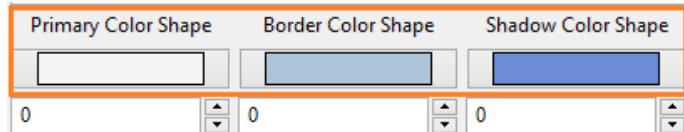
## Kara Effector - Effector Book [Tomo IV]:

**text.style:** es la unión de **text.color** y **text.alpha**, o sea que hace referencia a estos seis valores de la Ventana de Inicio del **Kara Effector**:



**text.alpha0:** equivale a \alpha&HFF&.

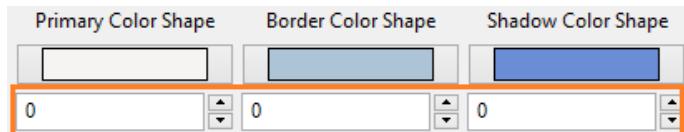
**shape.color:** son los tres tag de color de las Shapes:



En Lenguaje Automation Auto-4 sería:

```
\1c!shape.color1!\3c!shape.color3!\4c!shape.color4!
```

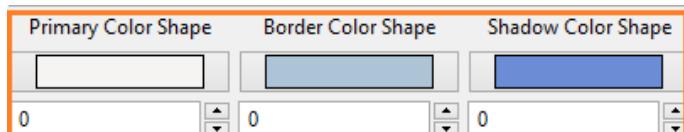
**shape.alpha:** son los tres tag de transparencia de las Shapes:



En Lenguaje Automation Auto-4 sería:

```
\1a!shape.alpha1!\3a!shape.alpha3!\4a!shape.alpha4!
```

**shape.style:** es la unión de **shape.text** y **shape.alpha**:



**shape.alpha0:** equivale a \alpha&HFF&.

**module:** es la interpolación equidistante de los valores numéricos entre 0 y 1 con respecto al loop de un Efecto.

$$\text{module} = (j - 1) / (\text{maxj} - 1)$$

**module1:** es la interpolación equidistante de los valores numéricos entre 0 y 1 con respecto a la cantidad de sílabas de las Líneas seleccionadas para un Efecto.

$$\text{module1} = (\text{syl.i} + \text{module} - 1) / \text{syl.n}$$

**module2:** es la interpolación equidistante de los valores numéricos entre 0 y 1 con respecto a la cantidad de Líneas seleccionadas para un Efecto.

$$\text{module2} = (\text{line.i} + \text{module1} - 1) / \text{line.n}$$

Y hemos llegado al final de las variables de la Librería "fx" sin antes mencionarles que aún hay algunas funciones de la misma que he decidido explicar en futuros **Tomos** para una mejor comprensión. De a poco vamos revelando los secretos del **Kara Effector** y espero que no se pierdan las próximas entregas.

A medida que avanzamos en los **Tomos**, profundizamos cada vez más en el mundo del **Kara Effector** y contamos con más herramientas para hacer nuestros propios Efectos. No olviden visitarnos en el **Blog Oficial** lo mismo que en los canales de **YouTube** para descargar los nuevos Efectos o si quieren comentar, exponer alguna duda o hacer alguna sugerencia.

## Kara Effector - Effector Book [Tomo V]:

# Kara Effector 3.2: Effector Book Vol. I [Tomo V]

# Kara Effector 3.2:

En este **Tomo V** nos adentramos más en el mundo de las Librerías, tanto de **LUA** como las del **Kara Effector**. Algunas funciones solo serán mencionadas y otras más tendrán ejemplos para una mayor comprensión, de todas maneras aquellas que no queden claras, más adelante las veremos con mayor detenimiento.

No es importante memorizarlas todas, pero ayuda el hecho de que tengamos una idea de qué hace cada función y para qué nos podría servir una variable en especial.

## Librería "math" [LUA]:

Es la librería de las funciones matemáticas de **LUA** y es de mucha utilidad. A continuación veremos las funciones y valores más usados, al menos para hacer Efectos. Omití algunas funciones y valores que consideré que no serían relevantes para usar en el **Kara Effector**, pero de todas maneras son fáciles de conseguir en la web.

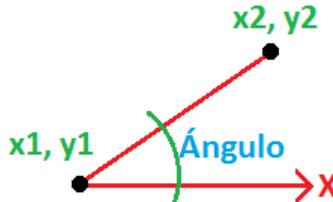
<b>math.abs(x)</b>	Retorna el Valor Absoluto de x
<b>math.acos(x)</b>	Retorna el Arco Coseno de x en un ángulo medido en radianes
<b>math.asin(x)</b>	Retorna el Arco Seno de x en un ángulo medido en radianes
<b>math.atan(x)</b>	Retorna el Arco Tangente de x en un ángulo medido en radianes
<b>math.atan2(y, x)</b>	Retorna el Arco Tangente de y/x en un ángulo medido en radianes
<b>math.ceil(x)</b>	Retorna el número entero mayor más cercano a x
<b>math.cos(x)</b>	Retorna el Coseno del ángulo x medido en radianes
<b>math.cosh(x)</b>	Retorna el Coseno Hiperbólico de x
<b>math.deg(x)</b>	Retorna el valor de x, convertido de radianes a sexagesimal. Ej: <b>math.deg(math.pi) = 180</b>

## Kara Effector - Effector Book [Tomo V]:

<b>math.exp(x)</b>	Retorna el valor de $e^x$ . $e = 2.718281\dots$
<b>math.floor(x)</b>	Retorna el número Entero Menor más cercano a x
<b>math.mod(x, y)</b>	Retorna el Residuo de $x/y$ , o sea que retorna el Modulo entre x e y
<b>math.log(x)</b>	Retorna el Logaritmo Natural (base e) de x
<b>math.log10(x)</b>	Retorna el Logaritmo Decimal (base 10) de x
<b>math.max(x, ...)</b>	Retorna el número mayor de entre todos los parámetros
<b>math.min(x, ...)</b>	Retorna el número menor de entre todos los parámetros
<b>math.pi</b>	Retorna el valor de pi. $\pi = 3.14159\dots$
<b>math.pow(x, y)</b>	Retorna el valor de $x^y$ . o sea x elevado a la y
<b>math.rad(x)</b>	Retorna el valor de x, convertido de sexagesimal a radianes
<b>math.random(m, n)</b>	Retorna un Número Aleatorio. En el caso de haber dos valores (m, n), retorna un número aleatorio entre esos dos valores. En el caso de un valor (m), retorna un número aleatorio entre 1 y ese valor. Y para el caso de no tener ningún valor, retorna un número decimal aleatorio entre 0 y 1
<b>math.sin(x)</b>	Retorna el Seno del ángulo x medido en radianes
<b>math.sinh(x)</b>	Retorna el Seno Hiperbólico de x
<b>math.sqrt(x)</b>	Retorna la Raíz Cuadrada de x. se puede remplazar con $x^{0.5}$
<b>math.tan(x)</b>	Retorna la Tangente del ángulo x medido en radianes
<b>math.tanh(x)</b>	Retorna la Tangente Hiperbólica de x

necesidad de “ampliar” la Librería “math”. Las siguientes funciones y variables hacen parte de la Librería “math”, pero son solo de uso exclusivo del **Kara Effector**.

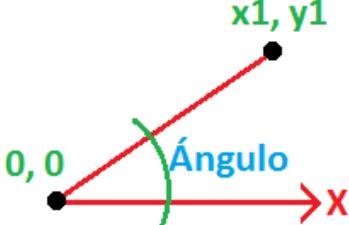
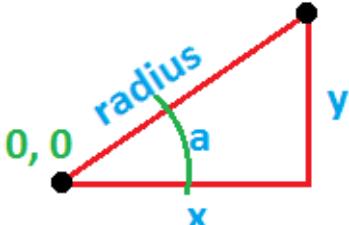
También he omitido algunas funciones de la ampliación de la Librería “math” del **Kara Effector**, pero es porque no son para usar en los Efectos, sino que son para ayudar a otras funciones a hacer su trabajo.

<b>math.R(m, n)</b>	Cumple prácticamente la misma función que <b>math.random</b> , pero con la diferencia que genera un número aleatorio por cada vez que se aplica el Efecto. La forma abreviada es: <b>R(m, n)</b>
<b>math.Rfake(m, n, i)</b>	Parecida a <b>math.random</b> , pero el random se genera una única vez para todos los Efectos que se use, además consta de un tercer parámetro opcional, que hace que el random haga más operaciones antes de retornar un resultado. La forma abreviada es: <b>Rf(m,n,i)</b>
<b>math.round(n, dec)</b>	Redondea un número n según las cantidad de cifras decimales que indiquemos (dec). Si no está el parámetro “dec” entonces retorna el entero más cercano
<b>math.distance(x1, y1, x2, y2)</b>	Retorna la Distancia entre dos puntos P1=(x1, y1) y P2=(x2, y2). También se puede usar así: <b>math.distance(x1, y1)</b> En este caso retorna la Distancia entre P1=(0, 0) y P2(x1, y1)
<b>math.angle(x1, y1, x2, y2)</b>	Retorna el Ángulo que forma el segmento formado por los dos puntos, y el tramo positivo del eje “x”:   También se puede usar así: <b>math.angle(x1, y1)</b> En este caso retorna el Ángulo que forma el segmento formado

## Librería “math” [KE]:

La anterior Librería es la que ya viene por default el lenguaje **LUA**, pero para hacer alguno de los Efectos parece que ésta se queda corta, es por ello que hubo la

## Kara Effector - Effector Book [Tomo V]:

	por los puntos $P1 = (0, 0)$ y $P=(x1, y1)$ , y el tramo positivo del eje "x":
	
<b>math.polar (a, radius, Return)</b>	Retorna las coordenadas Polares que forma el ángulo ( <b>a</b> ) y el radio ( <b>radius</b> ). El parámetro <b>Return</b> puede ser "x" o "y" según la coordenada que queremos que retorne. Si <b>Return</b> no está, retorna ambas coordenadas.
	
<b>math.point (n, x, y, xi, yi, xf, yf)</b>	Retorna un conjunto de Puntos. <b>n</b> : es el tamaño del conjunto <b>x</b> : es el rango horizontal <b>y</b> : es el rango vertical <b>xi, yi</b> : es el primer punto <b>xf, yf</b> : es el último punto
<b>math.factk(n)</b>	Retorna $n!$ , es decir que retorna el factorial de <b>n</b>
<b>math.bezier (Return, ...)</b>	Retorna una <b>Curva Bezier</b> a partir de una serie de puntos o el trazado de una Shape. Es una función exclusiva para Efectos con Shapes.

Para ver la siguiente Librería es importante que antes tengamos claro el concepto de "**tabla**". Una **tabla** es un conjunto de elementos, también se le conoce como "arreglo".

El nombre de una **tabla** es asignado por uno mismo y sus elementos están dentro de llaves ("{}"). Veamos un corto ejemplo:

**Letras** = {"A", "B", "C", "D"}

En este ejemplo, la **tabla** se llama "Letras" y tiene cuatro elementos. Los elementos de una **tabla** están separados por coma (,) o por punto y coma (;), no importa cuál de las dos se use.

Para usar los elementos de la **tabla** "Letras", se hace con el nombre de la **tabla** seguido del número de la posición del elemento entre corchetes:

**Letras** = {"A", "B", "C", "D"}

**Letras[1]** = "A"

**Letras[2]** = "B"

**Letras[3]** = "C"

**Letras[4]** = "D"

Otra forma de definir la tabla de este ejemplo y aun así tener el mismo resultado sería:

**Letras** = {[1] = "A", [2] = "B", [3] = "C", [4] = "D"}

Y por si fuera poco, todavía hay una tercera forma de definir la **tabla** "Letras". Se define vacía o con algunos de sus elementos y luego se le "insertan" el resto:

**Letras** = {}

**Letras[1]** = "A"

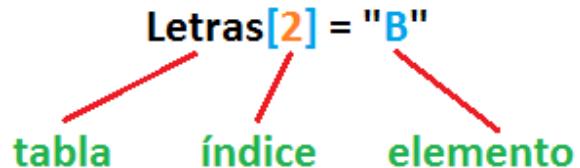
**Letras[2]** = "B"

**Letras[3]** = "C"

**Letras[4]** = "D"

**Letras** = {"A", "B", "C", "D"}

El número de la posición que ocupa un elemento dentro de una **tabla**, se le llama **índice**:



**Letras[2] = "B"**

tabla      índice      elemento

El tamaño de una **tabla** es la cantidad de elementos que tenga la misma. Para obtener el tamaño de una **tabla** se escribe el signo número seguido del nombre de la **tabla**:

## Kara Effector - Effector Book [Tomo V]:

#Letras = 4

Veamos un ejemplo en el **Kara Effector** con este tipo de **tabla**:

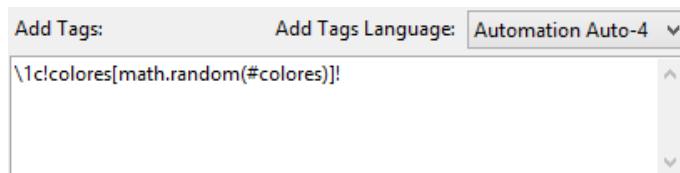
**Paso 1.** Definimos una **tabla** en la celda de texto Variables:

Variables: colores = {"&H00FF20&", "&HFD440E&", "&H9C03FE&"}

Le di por nombre “colores” y contiene tres colores: verde, azul y lila. De esta **tabla** se infiere que su tamaño es:

#colores = 3

**Paso 2.** Usar los elementos de la **tabla** en Add Tags:



Para este ejemplo, usé los elementos de la **tabla** con un random. Una pequeña explicación de la expresión usada sería:

- \1c!colores[math.random(#colores)]!
- #colores = 3
- \1c!colores[math.random(3)]!
- \1c!colores[math.random(1,3)]!

Como es un random, los posibles resultados son:

- \1c!colores[1]! ← verde
- \1c!colores[2]! ← azul
- \1c!colores[3]! ← lila

Y en el vídeo veríamos algo más o menos como esto:



Este es solo un pequeño ejemplo de cómo usar los tres elementos de esta **tabla**, pero hay muchas formas más y en futuros ejemplos veremos esas formas de hacerlo. Si al momento de hacer este, no se ve en el vídeo como se ve en la imagen anterior, les recomiendo volver a descargar el **Kara Effector** en el **Blog Oficial**.

Los elementos de la **tabla** “colores” del ejemplo anterior también pueden ser declarados por medio de un nombre, lo que me da pie para mostrarles otra forma de declarar una **tabla**:

```
colores = {  
    verde = "&H00FF20&",  
    azul = "&HFD440D&",  
    lila = "&H9C03FE&"  
}
```

Y la forma de “llamar” a los elementos declarados de esta forma es:

- colores.verde
- colores.azul
- colores.lila

O sea, el nombre de la **tabla** seguido de un punto (.) y luego el nombre que le hayamos dado a cada elemento.

El nombre que se le da a cada elemento de una **tabla**, en este caso la **tabla** “colores”, también es asignado a gusto propio. Lo pude haber hecho de esta otra forma y sería lo mismo:

```
colores = {  
    v = "&H00FF20&",  
    a = "&HFD440D&",  
    l = "&H9C03FE&"  
}
```

Veamos un ejemplo de cómo usar los valores de esta **tabla**: (en lenguaje **Automation Auto-4**)

\3c!colores.v!

O sea que el color del **Borde** (\3c) sería verde. Este mismo ejemplo pero en lenguaje **LUA** sería: (recordemos que hay dos formas de hacerlo)

- string.format("\3c%s", colores.v)
- "\3c"..colores.v

## Kara Effector - Effector Book [Tomo V]:

Lo anterior es solo una pequeña introducción al mundo de las **tablas**, pero el concepto es mucho más amplio de lo que hasta ahora hemos visto y de apoco les mostraré lo que aun reste por ver.

Ya con el concepto de **tabla** un poco más claro, veremos la siguiente Librería:

### Librería “table” [LUA]:

Es la librería de las funciones de LUA que hacen referencia a las **tablas**. Esta Librería también tiene una ampliación en el **Kara Effector** y por ahora solo explicaré las funciones que son útiles para hacer Efectos:

<b>table.concat</b> (t, separador)	Retorna los elementos de una tabla, pero concatenados. Si el parámetro “separador” no está, simplemente concatena a los elementos. Ejemplo: T = {"a", 7, "FF"} <b>table.concat(T)</b> = "a7FF" Ejemplo con “separador”: T = {"a", 7, "FF"} <b>table.concat(T, " ++ ")</b> = "a ++ 7 ++ FF"
<b>table.insert</b> (t, i, elemento)	Inserta un elemento asignado a una tabla. Si el parámetro “i” no está, inserta al elemento al final de la tabla. Ejemplo: A = {2, 4, 6, 8, 10} <b>table.insert(A, "&amp;HFF&amp;")</b> Entonces: A = {2, 4, 6, 8, 10, "&HFF&"} Si para este mismo ejemplo ponemos el parámetro “i”, sería: <b>table.insert(A, 3, "&amp;HFF&amp;")</b> Entonces: A = {2, 4, "&HFF&", 6, 8, 10} Nótese cómo el elemento se insertó en la posición 3.
<b>table.maxn(t)</b>	Cumple la misma función que <b>#t</b> , o sea que retorna el tamaño de la tabla t.
<b>table.remove(t, i)</b>	Remueve el elemento de la posición “i” de la tabla “t”. Ej: B = {1, 3, 5, 7} <b>table.remove(B, 2)</b> Entonces: B = {1, 5, 7}

<b>table.sort(t)</b>	Organiza de menor a mayor los elementos de una tabla, sí y solo sí todos los elementos son números. Ejemplo: D = {5, 4, 7, 9, 2, 1} <b>table.sort(D)</b> Entonces: D = {1, 2, 4, 5, 7, 8} Los elementos de la tabla D ahora están organizados de menor a mayor.
----------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Es momento de mencionar que las funciones de la Librería “table” se usan para crear nuevas funciones. Crear una nueva función es una herramienta para hacer Efectos que el **Aegisub**, el lenguaje **LUA** y **Kara Effector** nos ofrecen. Hacer funciones será la parte final de este curso, ya que cuando cada uno pueda hacer sus propias funciones, le resta muy poco por aprender acerca de Efectos Karaoke, por eso es importante ir viendo todos estos conceptos, para que cuando llegue el momento de usarlos a pleno, no quedaremos tan perdidos.

### Librería “table” [KE]:

Es la ampliación de la Librería “table” con que cuenta el **Kara Effector** y algunas de sus funciones nos sirven para hacer Efectos y las restantes para crear nuevas funciones.

<b>table.inside(t, e)</b>	Retorna <b>true</b> (verdadero) en el caso que el elemento “e” esté en la tabla “t”. En caso contrario retorna <b>false</b> (falso). Ejemplo: P = {"a", "b", "c"} <b>table.inside(P, "c")</b> = <b>true</b> <b>table.inside(P, "e")</b> = <b>false</b>
<b>table.index(t, e)</b>	Retorna el <b>índice</b> del elemento “e” en el caso de que dicho elemento pertenezca a la tabla “t”. En caso contrario retorna de nuevo al elemento. Ejemplo: R = {13, 42, 63, 34, 25} <b>table.index(R, 42)</b> = 2 Retorna 2, porque R[2] = 42 <b>table.index(R, "AA")</b> = "AA" Retorna "AA", porque: <b>table.inside(R, "AA")</b> = <b>false</b> O sea que "AA" no está en R

## Kara Effector - Effector Book [Tomo V]:

<b>table.compare(t1, t2)</b> <p>Retorna <b>true</b> (verdadero) en el caso de que la tabla <b>t1</b> sea igual que la tabla <b>t2</b>, de otro modo retorna <b>false</b> (falso). Ejemplo:  <math>A = \{1, 2, 3\}</math>  <math>B = \{1, 2, 3, 4\}</math>  <math>\text{table.compare}(A, B) = \text{false}</math>  <math>C = \{7, "a"\}</math>  <math>D = \{7, "a"\}</math>  <math>\text{table.compare}(C, D) = \text{true}</math></p>	<b>table.concat1(t, ...)</b> <p>Retorna la tabla <b>t</b> con todos sus Elementos concatenados a (...).  Ejemplo 1: (tabla y un elemento)  <math>A = \{"a", "b", "c"\}</math>  <math>B = \text{table.concat1}(A, 9)</math>  <math>B = \{"9a", "9b", "9c"\}</math>  Ejemplo 2: (tabla y más de un elemento)  <math>F = \{1, 2, 3\}</math>  <math>G = \text{table.concat1}(F, a, b)</math>  <math>G = \{a1, b1, a2, b2, c1, c2\}</math>  Ejemplo 3: (tabla con tabla)  <math>M = \{4, 6, 8\}</math>  <math>N = \{f, g\}</math>  <math>O = \text{table.concat1}(M, N)</math>  <math>O = \{f4, g4, f6, g6, f8, g8\}</math></p> <p>En estos ejemplos vemos cómo los tres puntos pueden ser un solo elemento o varios, también pueden ser una tabla</p>
<b>table.disorder(t)</b> <p><b>t</b> puede ser una tabla o un número entero positivo mayor o igual que 2.  Para el caso en que <b>t</b> sea una tabla, retorna a la misma tabla con sus elementos en desorden. Los elementos se desordenan de forma aleatoria (random). Ej:  <math>H = \{"a", "b", "c", 3, 9\}</math>  <math>G = \text{table.disorder}(H)</math>  Un posible resultado sería:  <math>G = \{"b", 9, "a", 3, "c"\}</math>  Si <b>t</b> es un entero positivo, entonces la función crea una tabla de números consecutivos desde 1 hasta <b>t</b>, para luego desordenar esos números. Ej:  <math>G = \text{table.disorder}(6)</math>  Un posible resultado sería:  <math>G = \{3, 5, 6, 2, 1, 4\}</math>  El número total de resultados es <b>#t!</b>, o sea el factorial del tamaño de la tabla. Para este último ejemplo sería:  <math>6! = 720</math> posibilidades</p>	<b>table.concat2(t, ...)</b> <p>Es similar a <b>table.concat1</b> y la diferencia se notará en los ejemplos.  Ejemplo 1: (tabla y un elemento)  <math>A = \{"a", "b", "c"\}</math>  <math>B = \text{table.concat1}(A, 9)</math>  <math>B = \{"9a", "9b", "9c"\}</math>  Ejemplo 2: (tabla y más de un elemento)  <math>F = \{1, 2, 3\}</math>  <math>G = \text{table.concat1}(F, a, b)</math>  <math>G = \{a1b1, a2b2, c1c2\}</math>  Ejemplo 3: (tabla con tabla)  <math>M = \{4, 6, 8\}</math>  <math>N = \{f, g, h\}</math>  <math>O = \text{table.concat1}(M, N)</math>  <math>O = \{f4g4h4, f6g6h6, f8g8h8\}</math></p>
<p>He decidido hacer un pequeño paréntesis acá para aclarar algunos conceptos que nos ayudarán a entender mucho de lo que se viene. El lenguaje <b>Automation Auto-4</b> es una modificación del lenguaje <b>LUA</b> para que reconozca las variables <b>Dólar (\$)</b> y las operaciones hechas dentro de los signos de admiración (!!). En pocas palabras, el lenguaje <b>Automation Auto-4</b> está basado en el lenguaje <b>LUA</b>.</p> <p>Otro de los conceptos que quería mencionar es uno que en ocasiones nos encontramos en los parámetros que requiere una función, y son tres puntos seguidos (...). Los tres puntos seguidos pueden ser una variable, una tabla, un elemento o un listado de elementos y/o una combinación de éstos. Estos tres puntos se colocan para indicar que podemos poner cuantas cosas queramos.</p>	<b>table.replay(n, ...)</b> <p>Retorna una tabla con <b>n</b> veces repetidas a (...).  Ejemplo 1:  <math>A = \text{table.replay}(4, "a")</math>  <math>A = \{"a", "a", "a", "a"\}</math>  Ejemplo 2:  <math>B = \text{table.replay}(3, f, g, h)</math>  <math>B = \{f, g, h, f, g, h, f, g, h\}</math>  Ejemplo 3:  <math>C = \{7, 8, 9\}</math>  <math>D = \text{table.replay}(2, C)</math>  <math>D = \{7, 8, 9, 7, 8, 9\}</math>  Se nota cómo se repiten <b>n</b> veces el o los elementos ingresados</p>

## Kara Effector - Effector Book [Tomo V]:

<b>table.count(t, e)</b>	Retorna el número de veces en el que el elemento <b>e</b> aparece en la tabla <b>t</b> . En el caso en que el elemento <b>e</b> no esté en <b>t</b> , retorna 0. Ejemplo: A = {a, b, a, 7, a, 8, 9, a} <b>table.count(A, a) = 4</b> <b>table.count(A, c) = 0</b> <b>table.count(A, b) = 1</b>	<b>table.reverse(t)</b> Retorna a la tabla <b>t</b> , pero con el <b>índice</b> invertido. Ejemplo 1: A = {3, 4, 5, 6, 7} B = <b>table.reverse(A)</b> B = {7, 6, 5, 4, 3} Ejemplo 2: C = {f, 5, 3, a, m, 2, 9} D = <b>table.reverse(C)</b> D = {9, 2, m, a, 3, 5, f}
<b>table.pos(t, e)</b>	Retorna una tabla con el o los <b>índices</b> del elemento <b>e</b> en la tabla <b>t</b> . En el caso de que el elemento <b>e</b> no esté en <b>t</b> , retorna una tabla vacía. Ejemplo: A = {a, b, a, 7, a, 8, 9, a} B = <b>table.count(A, a)</b> B = {1, 3, 5, 8} C = <b>table.count(A, c)</b> C = {} D = <b>table.count(A, b)</b> D = {2}	<b>table.cyclic(t)</b> Genera un “ <b>ciclo</b> ” con todos los elementos de la tabla <b>t</b> . Ejemplo 1: A = {2, 4, 6, 8} B = <b>table.cyclic(A)</b> B = {2, 4, 6, 8, 6, 4, 2} Ejemplo 2: C = {a, 4, 2, d, 5, b} D = <b>table.cyclic(C)</b> D = {a, 4, 2, d, 5, b, 5, d, 2, 4, a}
<b>table.retire(t, ...)</b>	Retorna la tabla <b>t</b> , pero retira a (...) de la tabla en el caso en que están en ella. Ejemplo 1: A = {a, b, a, 7, a, 8, 9, a} B = <b>table.retire(A, a)</b> B = {b, 7, 8, 9} Ejemplo 2: C = <b>table.retire(A, a, 7, 8)</b> C = {b, 9} Ejemplo 3: D = {7, 8, 9} E = <b>table.retire(A, D)</b> E = {a, b, a, a, a}	<b>table.op(t, mode)</b> Genera operaciones con los elementos de la tabla <b>t</b> . Dichas operaciones dependen del modo “ <b>mode</b> ”. Esta función es exclusiva para las tablas con elementos numéricos. Ejemplo 1: <b>mode</b> = “sum” Suma los elementos de la tabla A = {9, 1, 16, 4} <b>table.op(A, “sum”)</b> = 9 + 1 + 16 + 4 = 30 Ejemplo 2: <b>mode</b> = “concat” Une a los elementos de la tabla <b>table.op(A, “concat”)</b> = 14916 Ejemplo 3: <b>mode</b> = “average” Obtiene un promedio de la tabla <b>table.op(A, “average”)</b> = (9 + 1 + 16 + 4) / #A = 30 / 4 = 7.5 Ejemplo 4: <b>mode</b> = “min” Da al menor de los elementos <b>table.op(A, “min”)</b> = 1 Ejemplo 5: <b>mode</b> = “max” Da al mayor de los elementos <b>table.op(A, “max”)</b> = 16 Ejemplo 6: <b>mode</b> = “add” Adiciona un tercer parámetro a cada uno de los elementos B = <b>table.op(A, “add” -2)</b> B = {9 - 2, 1 - 2, 16 - 2, 4 - 2} B = {7, -1, 14, 2}
<b>table.inserttable(t1, t2, i)</b>	Inserta los elementos de la tabla <b>t2</b> en la tabla <b>t1</b> a partir del <b>índice</b> <b>i</b> , o en el caso de no estar el parámetro <b>i</b> , se insertan al final de la tabla <b>t1</b> . Ejemplo 1: A = {6, 7, 8} B = {a, b, c, d} C = <b>table.inserttable(A, B, 3)</b> C = {6, 7, a, b, c, d, 9} Los elementos de la tabla <b>B</b> se insertaron en la tabla <b>A</b> , a partir del <b>índice</b> 3 (la tercera posición). Ejemplo 2: D = {5, 4, 1}      E = {f, g} F = <b>table.inserttable(D, E)</b> F = {5, 4, 1, f, g}	

## Kara Effector - Effector Book [Tomo V]:

---

---

Omití algunas funciones de la ampliación de la Librería “**table**” por un par de motivos: para ver algunos conceptos previos y para poder darle más espacio y mayor número de ejemplos para total comprensión.

Este ha sido un **Tomo** cargado con mucha teoría, teoría que no necesariamente deben memorizar ni dominar toda al 100%, pero es mejor tener a la mano el medio para consultar alguna duda, que necesitar un concepto y no saber en dónde buscar.

Lo que con certeza les puedo asegurar es que el tener claro cuáles son y para qué sirve cada una de las variables y funciones de las Librerías vistas en esta series de tomos, les dará las herramientas necesarias para crear sus propias funciones, en donde las posibilidades son infinitas y los Efectos que se pueden lograr con cada una de ellas no tienen comparación.

En el **Kara Effector** la teoría es tan importante como la práctica, ambas deben ir de la mano, ya que sin una de ellas la otra no sería suficiente. De a poco iremos haciendo un equilibrio entre ellas y por eso en los próximos tomos irá aumentando el número de ejemplos para ponerlos en práctica y aumentar nuestras destrezas.

---

Espero que en este tramo del camino ya hayan podido ver algunos de los Efectos que por default vienen en el **Kara Effector** y hayan podido entender un poco mejor en qué consiste cada uno de ellos, y eso gracias a los conceptos, variables y funciones vistas. No olviden visitarnos en el **Blog Oficial** lo mismo que en los canales de **YouTube** para descargar los nuevos Efectos o dejar algún comentario, exponer alguna duda o hacer alguna sugerencia.

---

---

# Kara Effector 3.2: Effector Book Vol. I [Tomo VI]

---

## Kara Effector 3.2:

En el **Tomo VI** veremos un poco más de los lenguajes **LUA** y **Automation Auto-4**, ya que el saber más de ellos nos ayudará a comprender la estructura de cualquier Efecto hecho en el **Kara Effector** o incluso, cualquier otro hecho en **Aegisub**.

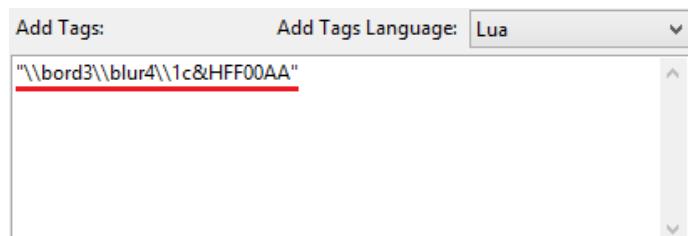
Entender un Efecto es el primer paso para desarrollar uno propio, tanto en el **Aegisub** o en **Kara Effector**. Espero que lo visto en este Tomo sea de fácil comprensión para todos.

## OBJETOS

Para hacer un Efecto Karaoke o una simple Plantilla con tags para un archivo de Subtítulos en lenguaje **LUA**, en el **Kara Effector** o en **Automation Auto-4**, solo existen dos **HERRAMIENTAS**, dichas Herramientas reciben el nombre de “strings” y “values”.

## OBJETOS: STRING

Un **string** (cadena) es un carácter o sucesión de ellos que carecen de algún tipo de valor. En lenguaje **LUA** un **string** es todo aquello que esté entre comillas, ya sean dobles o simples. Veamos algunos ejemplos de strings en **LUA**:



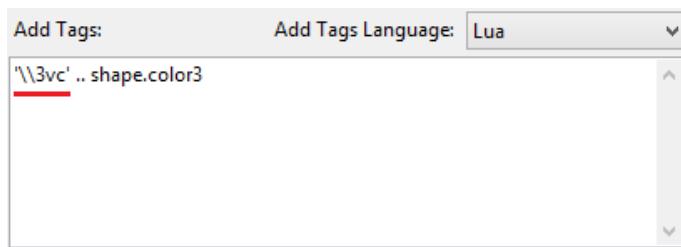
Add Tags: Add Tags Language: Lua

```
"\"\\bord3\\blur4\\1c&HFF00AA\""
```

En este caso, todo aquello que está entre las comillas dobles es un **string**, ya que para el lenguaje **LUA** esto sería lo mismo que una palabra cualquiera, sin valor alguno.

## Kara Effector - Effector Book [Tomo VI]:

En el siguiente ejemplo veremos los dos tipos de **Objetos**:



```
'\\3vc' .. shape.color3
```

Lo que está entre las comillas simples es un **string**, el resto es **value**.

En **LUA**, si algo está entre comillas es un **string**, incluso si es un número o si hay operaciones matemáticas, ejemplo:

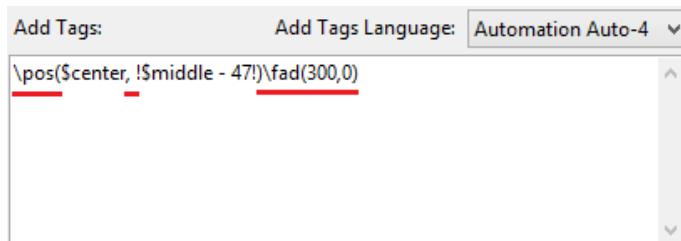
"12 + 3"

Al estar entre comillas no se realiza la operación (suma).

12 + 3

Ya sin las comillas, LUA ejecuta la operación y devuelve el resultado (15).

Por otro lado, en **Automation Auto-4**, un **string** también es fácil de reconocer. Un **string** es todo aquello que no está precedido del signo dólar (variables dólar) ni tampoco está dentro de los signos de admiración. Ejemplos:



```
\pos($center, !$middle - 47!)\\fad(300,0)
```

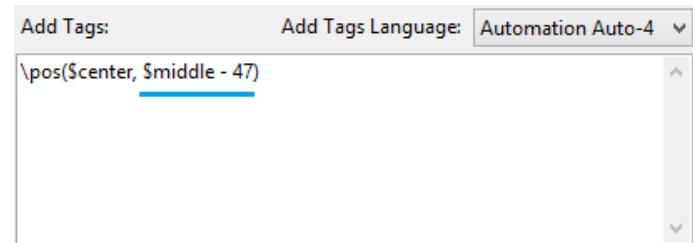
De esta expresión, los únicos **Objetos** que no son un **string** en lenguaje **Automation Auto-4** son:

\$center

!\$middle – 47!

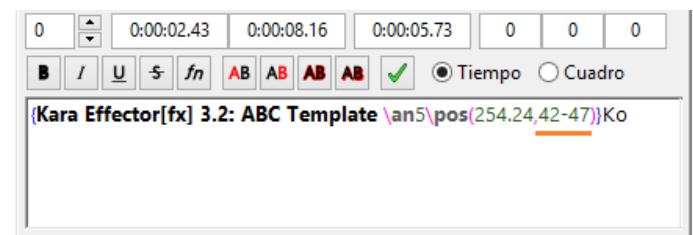
El resto, incluidos los paréntesis y las comas, son un **string** y por ende carecen de valor operacional. Podemos afirmar que los tags con que hacemos los Efectos también son un **string**, aunque para el **Aegisub** sí tengan valor operacional o funcional, ya que ni en **LUA** ni en **Automation Auto-4** se pueden hacer operaciones con ellos.

Un **error** común que se comete a veces en el lenguaje **Automation Auto-4** es intentar hacer una operación sin poner los signos de admiración:



```
\pos($center, $middle - 47)
```

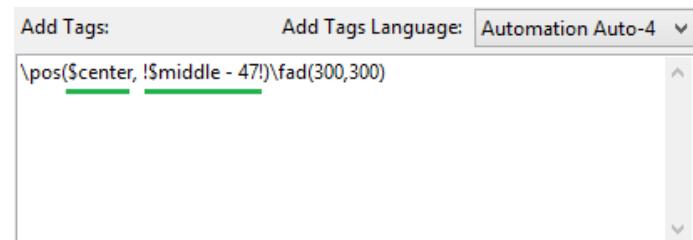
Al no poner los signos de admiración, el **Kara Effector** da por sentado que estamos escribiendo un **string** y por ello no ejecuta la operación que habíamos pensado:



Se ve claramente cómo queda expresada la sustracción (resta), pero no devolvió el resultado de la misma. **Aegisub** toma en cuenta solo al 42 y omite el resto.

## OBJETOS: VALUE

El **value** es todo lo que tiene algún tipo de valor, ya sea numérico, operacional o funcional. Un **value** es todo lo que no es un **string**. En **LUA** un **value** es todo aquello que no está dentro de las comillas simple o dobles. De forma similar, en **Automation Auto-4**, un **value** es todo aquello que no está precedido del signo dólar (variable dólar) ni tampoco está dentro de los signos de admiración.



```
\pos($center, !$middle - 47!)\\fad(300,300)
```

En verde podemos ver dos ejemplos de **values** (valores) en lenguaje **Automation Auto-4**.

## Kara Effector - Effector Book [Tomo VI]:

Add Tags:      Add Tags Language: **Lua**

```
"\\bord" .. l.outline + 2
```

En **LUA**, como ya lo sabemos, sino está dentro de las comillas, en este caso dobles, es un **value**. Los dos puntos seguidos (..) tienen valor operacional (concatenación) y la variable **l.outline** tiene un valor numérico (espesor del borde medido en pixeles).

Ahora veamos un cuadro con los diferentes tipos de **value** y algunos ejemplos:

TIPO DE VALUE	EJEMPLOS
<b>Numérico</b>	Pi, e, 1276, syl.center, line.middle
<b>Operación</b>	+, -, /, *, ^, .., %, :
<b>Función</b>	math.random, string.format
<b>Tabla</b>	A = {1, 2, 'F'} B = {"&HFF&", "&H00&" } C = {[1] = 43; [2] = 86} D = {}
<b>Operador Lógico</b>	for, if, while, repeat, or, and
<b>Variable</b>	Color1 = "&HFF0000&" Radius = 6*pi Linevc = ""

Una vez claro los conceptos de **string** y **value** ya podemos seguir viendo las Librerías que aun nos faltan. La que veremos a continuación es referente a los **strings** y con las funciones que hay en ellas notarán que los **string** son más que simples objetos que carecen de valor.

## Librería “string” [LUA]

<b>string.byte(s)</b>	Retorna el <b>Valor Numérico</b> del carácter <b>s</b> según su equivalente decimal en la <a href="#">Tabla ASCII</a> Veamos un ejemplo de la Tabla: <table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>64</td><td>@</td><td>96</td><td>'</td></tr> <tr> <td>65</td><td>A</td><td>97</td><td>a</td></tr> <tr> <td>66</td><td>B</td><td>98</td><td>b</td></tr> <tr> <td>67</td><td>C</td><td>99</td><td>c</td></tr> <tr> <td>68</td><td>D</td><td>100</td><td>d</td></tr> </table> Vemos que el valor decimal de “@” es 64, así como el de “b” es 98.	64	@	96	'	65	A	97	a	66	B	98	b	67	C	99	c	68	D	100	d
64	@	96	'																		
65	A	97	a																		
66	B	98	b																		
67	C	99	c																		
68	D	100	d																		

	<b>Ejemplo 1:</b> <b>string.byte(“A”)</b> = 65 <b>Ejemplo 2:</b> Str = “C” <b>string.byte(Str)</b> = 67 Los valores decimales de la <b>Tabla ASCII</b> van desde el 0 hasta el 255. El modo abreviado de usar esta función es así: <b>Ejemplo 3:</b> s = “B” <b>s:byte()</b> = 66 <b>Ejemplo 4:</b> s = “d” <b>s:byte()</b> = 100 Esta función tiene más modo de uso, pero al menos por el momento no serán de gran utilidad y de ahí que las haya omitido.																				
<b>string.char(...)</b>	Retorna el carácter asignado por la <b>Tabla ASCII</b> de un número entre 0 y 255. Ejemplo: <b>string.char(65)</b> = “A” <b>string.char(66)</b> = “B” <b>string.char(65, 66, 100)</b> = “ABd”																				
<b>string.find(s, ptr)</b>	Retorna las posiciones del inicio y final del string <b>s</b> dentro de un string mayor <b>ptr</b> . <b>Ejemplo 1:</b> L = “Demo lua string” s = “lua” <b>string.find(s, L)</b> = 6, 8 6 es el inicio y 8 es el final: <table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>D</td><td>e</td><td>m</td><td>o</td><td></td><td>I</td><td>u</td><td>a</td><td></td><td>s</td> </tr> <tr> <td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td></td><td></td> </tr> </table> La manera abreviada es: <b>s:find(L)</b> = 6, 8 Si el string <b>s</b> no pertenece al string <b>ptr</b> , entonces retorna <b>nil</b>	D	e	m	o		I	u	a		s	1	2	3	4	5	6	7	8		
D	e	m	o		I	u	a		s												
1	2	3	4	5	6	7	8														
<b>string.format(s, ...)</b>	Inserta valores al string <b>s</b> . El modo general de asignar los valores es por medio del string “%s”. <b>Ejemplo 1:</b> <b>string.format(“\\be%s”, l.shadow)</b> Es decir que inserta el valor de <b>l.shadow</b> al string “\\be%s”. Hay dos formas abreviadas de usar esta función en el Kara Effector: <ul style="list-style-type: none"> <li>• <b>format(s, ...)</b></li> <li>• <b>F(s, ...)</b></li> </ul> <b>Ejemplo 2:</b> <b>F(“\\pos(%s,%s)”, syl.center, 20*pi)</b>																				

## Kara Effector - Effector Book [Tomo VI]:

	<p>Por cada vez que aparezca “%s” en el string, se debe asignar el valor que se insertará en esa posición.</p> <p>Este modo es el más usado en el Kara Effector y es la función string que más se usa en él. Los demás modos no son tan relevantes para hacer Efectos Karaokes.</p>		<p>Ejemplo 2: s = “EL SOL” s:<b>lower()</b> = “el sol”</p>
<b>string.gsub(s, ptr, replace, n)</b>	<p>Reemplaza al string <b>s</b> que pertenece al string <b>ptr</b>, por un valor u otro string llamado <b>replace</b>, <b>n</b> cantidad de veces.</p> <p>El modo abreviado de uso es:</p> <p><b>ptr:gsub(s, replace, n)</b></p> <p><b>n</b> es un entero positivo, es opcional en la función.</p> <p>Ejemplo 1: ptr = “el sol en la mañana” ptr:<b>gsub(“a”, “X”) = “el sol en IX mXñXnX”</b></p> <p>La función remplazó al carácter “a” por “X” todas las veces.</p> <p>Ejemplo 2: ptr = “el sol en la mañana” ptr:<b>gsub(“a”, “X”, 2)</b> = “el sol en IX mXñana”</p> <p>El remplazo se hizo solo dos veces, dado que <b>n = 2</b></p> <p>Esta función también es mucho más extensa de lo que aparenta, pero de eso nos ocuparemos más adelante.</p>	<p>Repite al string <b>s</b> una <b>n</b> cantidad de veces.</p> <p>El modo abreviado es:</p> <p><b>s:rep(n)</b></p> <p>Ejemplo 1: s = “Demo” s:<b>rep(3) = “DemoDemoDemo”</b></p> <p>Ejemplo 2: s = “lua ” s:<b>rep(5) = “lua lua lua lua lua”</b></p>	
<b>string.reverse(s)</b>		<p>Invierte al string <b>s</b>.</p> <p>El modo abreviado es:</p> <p><b>s:reverse()</b></p> <p>Ejemplo 1: s = “Demo” s:<b>reverse() = “omeD”</b></p> <p>Ejemplo 2: s = “La Luna y el Sol” s:<b>reverse() = “oS le y anuL aL”</b></p>	
<b>string.sub(s, i, j)</b>		<p>Recorta al string <b>s</b> desde la posición <b>i</b> hasta la posición <b>j</b>.</p> <p>Tanto <b>i</b> como <b>j</b> son números enteros y pueden ser positivos o negativos. Al ser positivos la posición se empieza a contar de izquierda a derecha, si son negativos, se cuenta al revés.</p> <p>El modo abreviado es:</p> <p><b>s:sub(i, j)</b></p> <p>Ejemplos: s = “El Sol” s:<b>sub(1, -1) = “El sol”</b> s:<b>sub(1, 1) = “E”</b> s:<b>sub(-3, -1) = “Sol”</b> s:<b>sub(2, -1) = “l Sol”</b></p> <p>El parámetro <b>j</b> es opcional s:<b>sub(-2) = “o”</b> s:<b>sub(4) = “S”</b></p>	
<b>string.lower(s)</b>	<p>Retorna la cantidad de caracteres que tiene el string <b>s</b>.</p> <p>El modo abreviado es:</p> <p><b>s:len()</b></p> <p>Ejemplo 1: s = “&amp;HFFFFF&amp;” s:<b>len() = 9</b></p> <p>Ejemplo 2: s = “El sol” s:<b>len() = 6</b></p> <p>No olvidemos que el espacio entre palabra y palabra también es un carácter.</p>	<p>Retorna al string <b>s</b> con todas las letras mayúsculas en él convertidas en minúsculas.</p> <p>El modo abreviado es:</p> <p><b>s:lower()</b></p> <p>Ejemplo 1: s = “La Noche” s:<b>lower() = “la noche”</b></p>	<p>Retorna al string <b>s</b> con todas las letras minúsculas en él convertidas en mayúsculas.</p> <p>El modo abreviado es:</p> <p><b>s:upper()</b></p> <p>Ejemplo 1: s = “La Noche” s:<b>upper() = “LA NOCHE”</b></p> <p>Ejemplo 2: s = “¡¿Qué?, no puedes!” s:<b>upper() = “¡¿QUÉ?, NO PUEDES!”</b></p>

## Kara Effector - Effector Book [Tomo VI]:

De esta Librería también omití un par de funciones que quizás, al necesitarlas más adelante, las explique a cada una de ellas, aunque al ser una biblioteca de **LUA** se hace un poco más simple hallar información en la web sobre ellas.

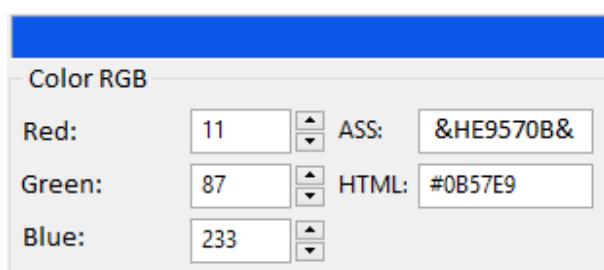
Terminada esta librería, es momento para ver las formas abreviadas de las funciones comúnmente usadas en el **Kara Effector**:

ABREVIATURA	FUNCIÓN
pi	math.pi
sin	math.sin
cos	math.cos
tan	math.tan
asin	math.asin
acos	math.acos
atan	math.atan
sinh	math.sinh
cosh	math.cosh
tanh	math.tanh
log	math.log10
ln	math.log
abs	math.abs
floor	math.floor
ceil	math.ceil
deg	math.deg
rad	math.rad
r	math.random
R	math.R
Rf	math.Rfake
rand	math.random
format	string.format
F	string.format

Todas las anteriores abreviaturas de las funciones aplican tanto en lenguaje **LUA** como en **Automation Auto-4**.

A continuación veremos un poco de dos de las **teorías de color** que maneja el formato .ass que nos servirán para entender las siguientes Librerías.

### TEORÍA DEL COLOR RGB (Red, Green, Blue):



De la anterior imagen vemos cómo el tono de azul que está en la parte superior de la misma, está formado por tres valores de rojo, verde y azul:

Tono	Valor Decimal	Valor Hexadecimal
Rojo	11	0B
Verde	87	57
Azul	233	E9

En formato .ass el tono de azul del ejemplo anterior sería:

&HE9570B&

De forma general, todo color en formato .ass tiene la siguiente estructura:

&H [Azul] [Verde] [Rojo] &

La estructura de un color en formato .ass es similar a la del formato **HTML**, pero los tonos invertidos, como se ve en la imagen anterior:

#0B57E9

El **Kara Effector** tiene una función que convierte un color de formato **HTML** a .ass:

color.ass("#0B57E9") = &HE9570B&

Veamos un listado de colores conocidos en formato **HTML** que con la anterior función ya sabemos cómo pasarlo a formato .ass. En la Tabla aparece el color, el nombre en inglés y el código del mismo en formato **HTML**:

AliceBlue	#F0F8FF
AntiqueWhite	#FAEBD7
Aqua	#00FFFF
Aquamarine	#7FFFDD
Azure	#F0FFFF
Beige	#F5F5DC
Bisque	#FFE4C4
Black	#000000
BlanchedAlmond	#FFEBBC
Blue	#0000FF
BlueViolet	#8A2BE2
Brown	#A52A2A
BurlyWood	#DEB887
CadetBlue	#5F9EA0
Chartreuse	#7FFF00
Chocolate	#D2691E
Coral	#FF7F50
CornflowerBlue	#6495ED

## Kara Effector - Effector Book [Tomo VI]:

---

Cornsilk	# FFF8DC
Crimson	# DC143C
Cyan	# 00FFFF
DarkBlue	# 00008B
DarkCyan	# 008B8B
DarkGoldenrod	# B8860B
DarkGray	# A9A9A9
DarkGreen	# 006400
DarkKhaki	# BDB76B
DarkMagenta	# 8B008B
DarkOliveGreen	# 556B2F
DarkOrange	# FF8C00
DarkOrchid	# 9932CC
DarkRed	# 8B0000
DarkSalmon	# E9967A
DarkSeaGreen	# 8FBBC8F
DarkSlateBlue	# 483D8B
DarkSlateGray	# 2F4F4F
DarkTurquoise	# 00CED1
DarkViolet	# 9400D3
DeepPink	# FF1493
DeepSkyBlue	# 00BFFF
DimGray	# 696969
DodgerBlue	# 1E90FF
Firebrick	# B22222
FloralWhite	# FFFAF0
ForestGreen	# 228B22
Fuchsia	# FF00FF
Gainsboro	# DCDCDC
GhostWhite	# F8F8FF
Gold	# FFD700
Goldenrod	# DAA520
Gray	# 808080
Green	# 008000
GreenYellow	# ADFF2F
Honeydew	# F0FFF0
HotPink	# FF69B4
IndianRed	# CD5C5C
Indigo	# 4B0082
Ivory	# FFFFF0
Khaki	# F0E68C
Lavender	# E6E6FA
LavenderBlush	# FFF0F5
LawnGreen	# 7CFC00
LemonChiffon	# FFFACD
LightBlue	# ADD8E6
LightCoral	# F08080
LightCyan	# E0FFFF
LightGoldenrodYellow	# FAFAD2

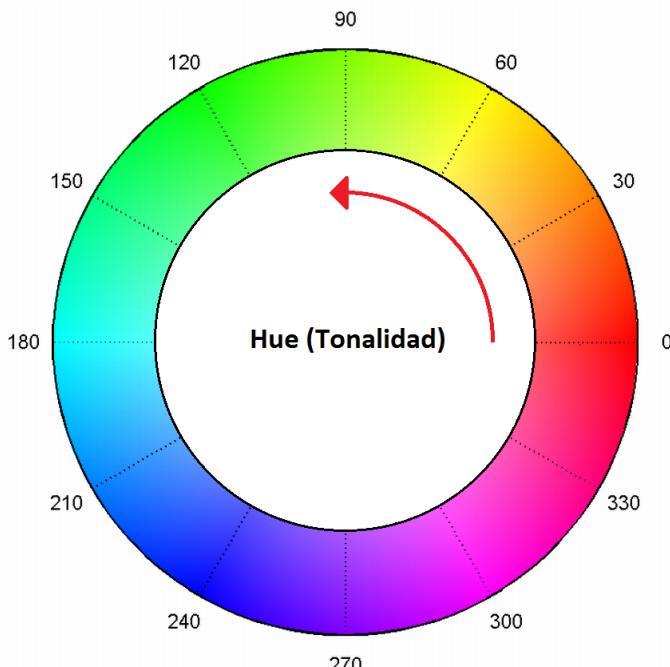
LightGray	# D3D3D3
LightGreen	# 90EE90
LightPink	# FFB6C1
LightSalmon	# FFA07A
LightSeaGreen	# 20B2AA
LightSkyBlue	# 87CEFA
LightSlateGray	# 778899
LightSteelBlue	# B0C4DE
LightYellow	# FFFFFE0
Lime	# 00FF00
LimeGreen	# 32CD32
Linen	# FAF0E6
Magenta	# FF00FF
Maroon	# 800000
MediumAquamarine	# 66CDAA
MediumBlue	# 0000CD
MediumOrchid	# BA55D3
MediumPurple	# 9370DB
MediumSeaGreen	# 3CB371
MediumSlateBlue	# 7B68EE
MediumSpringGreen	# 00FA9A
MediumTurquoise	# 48D1CC
MediumVioletRed	# C71585
MidnightBlue	# 191970
MintCream	# F5FFFFA
MistyRose	# FFE4E1
Moccasin	# FFE4B5
NavajoWhite	# FFDEAD
Navy	# 000080
OldLace	# FDF5E6
Olive	# 808000
OliveDrab	# 6B8E23
Orange	# FFA500
OrangeRed	# FF4500
Orchid	# DA70D6
PaleGoldenrod	# EEE8AA
PaleGreen	# 98FB98
PaleTurquoise	# AFEEEE
PaleVioletRed	# DB7093
PapayaWhip	# FFEFD5
PeachPuff	# FFDAB9
Peru	# CD853F
Pink	# FFC0CB
Plum	# DDA0DD
PowderBlue	# B0E0E6
Purple	# 800080
Red	# FF0000
RosyBrown	# BC8F8F
RoyalBlue	# 4169E1

## Kara Effector - Effector Book [Tomo VI]:

SaddleBrown	#8B4513
Salmon	#FA8072
SandyBrown	#F4A460
SeaGreen	#2E8B57
SeaShell	#FFF5EE
Sienna	#A0522D
Silver	#C0C0C0
SkyBlue	#87CEEB
SlateBlue	#6A5ACD
SlateGray	#708090
Snow	#FFFAFA
SpringGreen	#00FF7F
SteelBlue	#4682B4
Tan	#D2B48C
Teal	#008080
Thistle	#D8BFD8
Tomato	#FF6347
Transparent	#FFFFFF
Turquoise	#40E0D0
Violet	#EE82EE
Wheat	#F5DEB3
White	#FFFFFF
WhiteSmoke	#F5F5F5
Yellow	#FFFF00
YellowGreen	#9ACD32

### TEORÍA DEL COLOR HSV (Hue, Saturation, Value):

**Hue:** es la tonalidad de un color, cuyo valor es un real desde 0 hasta 360, como vemos en la imagen:



En la anterior imagen vemos cómo **Hue** = 0 equivale al rojo, **Hue** = 60 equivale al amarillo, **Hue** = 240 equivale al azul.

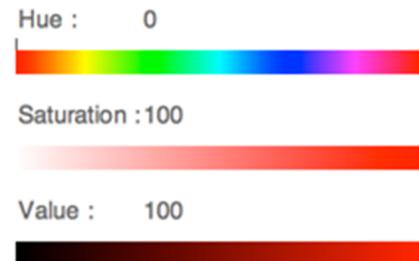
Veamos un ejemplo con el color Rojo. Con la teoría del color **RGB** sería (los valores de R, G y B son reales entre 0 como mínimo y 255 como máximo):



En formato HTML: **#FF0000**

En formato .ass: **&H0000FF&**

Pero en la teoría HSV sería:



- Hue = 0
- Saturation = 100
- Value = 100

Lo que da pie a las siguientes dos definiciones que hacen parte a la teoría HSV.

**Saturation:** es la cantidad de blanco que tendrá la tonalidad **Hue**. Su valor es un número real entre 0 y 100, donde 0 completamente blanco y 100 equivale a que la tonalidad no tendría nada de blanco.

**Value:** es la cantidad de negro que tendrá la tonalidad **Hue**. Su valor es un número real entre 0 y 100, donde 0 completamente negro y 100 equivale a que la tonalidad no tendría nada de negro.

Si **Saturation** es 0, no importa el valor de **Hue**, el color siempre será **Blanco**. De manera similar pasaría con **Value**, ya que si es 0, no importaría el valor de **Hue**, el color que resultaría siempre sería **Negro**.

## Kara Effector - Effector Book [Tomo VI]:

Para poner en práctica la teoría del color **HSV**, **Aegisub** ya trae por default un par de funciones que hacen posible convertir los tres valores (H, S y V) en un color en formato .ass, y la primera de ellas es:

**HSV\_to\_RGB(H, S, V):** convierte los valores de **H**, **S** y **V** en valores de 0 a 255 de Rojo, Verde y Azul. Es decir que esta función retorna tres resultados al mismo tiempo.

Para esta función de Aegisub, **Hue** sigue siendo un real entre 0 y 360, pero **Saturation** y **Value** son un número real entre 0 y 1.

Ejemplo 1:

R, G, B = **HSV\_to\_RGB(0, 1, 1)**

- R = 255
- G = 0
- B = 0

Ejemplo 2:

R, G, B = **HSV\_to\_RGB(264, 0.75, 0.2)**

- R = 28.05
- G = 12.75
- B = 51

Y la siguiente función del **Aegisub** es la que convierte estos tres valores en un color en formato .ass para que pueda ser usado en los tags de colores:

**ass\_color(R, G, B):** transforma los valores de R, G y B en un color en formato .ass: **&H[Azul][Verde][Rojo]&**

Ejemplos:

**ass\_color(255, 0, 0) = &H0000FF&**

**ass\_color(47, 82, 242) = &HF2522F&**

**ass\_color(158, 80, 153) = &H99509E&**

lo que quiere decir que debemos combinar estas dos funciones para convertir los valores de H, S y V en un color en formato .ass:

**ass\_color(HSV\_to\_RGB(H, S, V))**

veamos algunos ejemplos de esta combinación:

Estilo	Efecto	Texto
Default	template	<code>!_G.ass_color(_G.HSV_to_RGB(45, 1, 0.5))!</code>
Default	karaoke	
Default	fx	<code>&amp;H005F7F&amp;</code>
Default	fx	<code>&amp;H005F7F&amp;</code>

Recordemos que desde el **Aegisub**, para usar cualquiera de las funciones que por default vienen en él, éstas deben iniciar por “**\_G.**” como se ve en la imagen anterior. En el **Kara Effector** da igual si se coloca este prefijo o no, las funciones son reconocidas de las dos formas.

Un ejemplo práctico en el **Kara Effector** sería:

The screenshot shows two examples of the Kara Effector interface. Both examples use the 'ass\_color' function with 'Automation Auto-4' language. The first example uses a constant value (R=360) and a random value (0.5). The second example uses a random value (0.5) and a constant value (R=360).

```
"\\1c" .. ass_color(HSV_to_RGB(R(360), 1, 0.5))
```

```
\1class_color(HSV_to_RGB(R(360), 1, 0.5))
```

Uno más usando la función **string.format**:

The screenshot shows an example of the Kara Effector interface using the 'string.format' function with 'Lua' language. It formats a string to include a color using the 'ass\_color' function with random values.

```
format("\\1c%\\bord%", ass_color(HSV_to_RGB(45, 1, 1)), r(2, 5))
```

Es decir que con esta combinación de funciones podemos obtener un color con valores constantes, como en el ejemplo de la imagen anterior, o con valores aleatorios (random), como en las otras dos anteriores imágenes.

En este punto, el nivel de complejidad ha ido en aumento y cada vez tenemos un mayor conocimiento y contamos con más herramientas para desarrollar nuestros propios Efectos. La mayoría de las próximas funciones de las siguientes librerías tienen aplicación directamente para hacer nuevos Efectos y vendrán acompañadas con varios ejemplos y ejercicios prácticos. No olviden visitarnos en el **Blog Oficial** lo mismo que en los canales de **YouTube** para descargar los nuevos Efectos o dejar algún comentario, exponer alguna duda o hacer alguna sugerencia.

**Kara Effector - Effector Book [Tomo VII]:**

# **Kara Effector 3.2: Effector Book Vol. I [Tomo VII]**

# **Kara Effector 3.2:**

Para la fecha de publicación de este **Tomo VII** ya pueden descargar la más reciente actualización del **Kara Effector** que pone como fecha el 1 de Enero de 2014. La nueva actualización consiste en poder usar en nuestros Efectos los colores en formato **HTML** sin tener la necesidad de convertirlos a formato .ass y tener mayor diversidad de opciones al poder usar ambos formatos de manera libre.

La actualización fue inspirada al desarrollar el tema de las teorías de color que se usan en el **Aegisub** y les mostraré unos cuantos ejemplos de usarla:



Con esta actualización podemos copiar el código de un color en formato **HTML** y usarlo en el **Kara Effector**. En la web hay muchas páginas en donde podemos encontrar estos colores en dicho formato. Recomiendo la siguiente:

<http://www.computerhope.com/htmcolor.htm>

En esta web encontraremos un extenso listado de colores en formato **HTML** y así poder elegir el que más se ajuste al Efecto que aplicaremos:

#357EC7	<b>Slate Blue</b>
#488AC7	<b>Silk Blue</b>
#3090C7	<b>Blue Ivy</b>
#659EC7	<b>Blue Koi</b>
#87AFC7	<b>Columbia Blue</b>
#95B9C7	<b>Baby Blue</b>

## Kara Effector - Effector Book [Tomo VII]:

### Librería “random” [KE]:

Es una librería exclusiva del **Kara Effector** que consiste en una serie de funciones con resultados aleatorios que podemos aplicar en el desarrollo de los Efectos. La Librería “random” contiene las siguientes funciones:

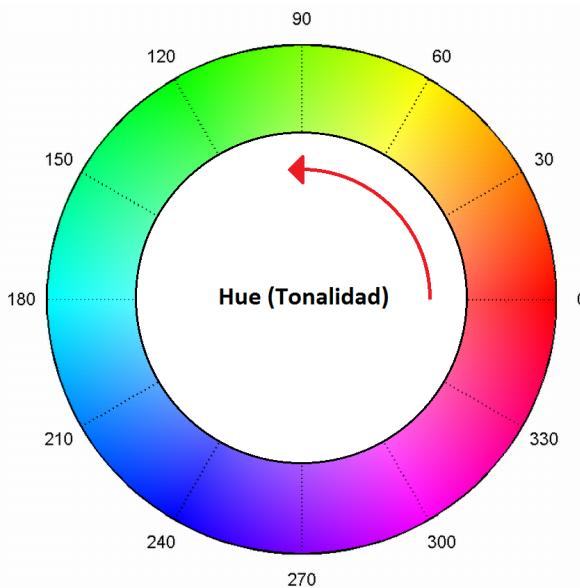
- **random.color**
- **random.colorvc**
- **random.alpha**
- **random.alphava**
- **random.e**
- **random.unique**

Y a continuación veremos en detalle en qué consiste cada una de ellas.

**random.color( H, S, V )**: retorna un color en formato .ass según la teoría HSV. **H**, **S** y **V** son parámetros opcionales y pueden ser valores numéricos o tablas.

Modo 1: sin ningún parámetro

**random.color()**: retorna un color en formato .ass correspondiente a un tono (**Hue**) al azar entre 0 y 360 de la teoría HSV.



En este modo (Modo 1), la función retorna un tono al azar, pero tanto **S** (Saturation) como **V** (Value) son constantes y el valor de cada uno de ellos es 100, o sea que el color será solo alguno de los de la imagen anterior.

En el **Kara Effector**, en un Efecto tipo “Syl”, podemos poner a prueba el Modo 1 de esta función:

```
Add Tags:          Add Tags Language: Automation Auto-4
\1clrandom.color()!
```

Entonces en el vídeo veremos algo como esto:

**Kodoku na hoho wo nurasu nurasu keto**

Para cada Sílaba la función generó un color aleatorio con un valor entre 0 y 360 del anterior círculo cromático.

Este es un ejemplo del color que puede retornar la función en este modo, siempre y cuando hayamos seleccionado la opción “**VSFilter 2.39**” o la opción “**No Tags Color and Alpha**” en **Using Tag Filter**:

```
0 0:00:02.43 0:00:08.16 0:00:05.73 0 0
B I U S fn AB AB AB AB ✓ ⚪ Tiempo
[Kara Effector[fx] 3.2: ABC Template \an5\pos(254.24,42
\1c&HFFFFFF&\3c&H000000&\4c&H000000&\1a&H00&
3a&H00&\4a&H00&\1c&H00FF61&)Ko
```

Si por el contrario, elegimos la opción “**VSFilterMod**”, el color se multiplicará cuatro veces y saldrán separados por comas dentro de un paréntesis (en formato **VSFilterMod**):

```
Using Tags Filter: VSFilterMod
0 0:00:02.43 0:00:08.16 0:00:05.73 0 0
B I U S fn AB AB AB AB ✓ ⚪ Tiempo ⚪ C
[Kara Effector[fx] 3.2: ABC Template \an5\pos(254.24,42
\1vc(&H9800FF&,&H9800FF&,&H9800FF&,&H9800FF&)Ko
```

Modo 2: con un parámetro constante

**random.color(H)**: retorna un color en formato .ass correspondiente al tono (**Hue**) entre 0 y 360 que le hayamos asignado a la función. Ejemplo:

**random.color(112) = “&H00FF21&”**

## Kara Effector - Effector Book [Tomo VII]:

### Kodoku na hoho wo nurasu nurasu keto

En el círculo cromático vemos que el valor 112 le corresponde a un todo de verde claro. En este modo, la función retorna un color constante correspondiente al **Hue** del valor que le ingresemos.

Usar la función **random.color** para que retorne un color constante es ideal para hacer degradaciones con los colores **HSV**. Ejemplo: (**Template Type**: Syl)

```
Add Tags: Add Tags Language: Automation Auto-4
\nrandom.color( 15 + 55*syl.i/syl.n )!
```

Es un degradado **HSV** con los valores desde 15 para la primera Sílaba, hasta 70 ( $15 + 55 = 70$ ) para la última:

### Kodoku na hoho wo nurasu nurasu keto

Modo 3: con una Tabla como parámetro

**random.color({H\_i, H\_f})**: retorna un color en formato .ass correspondiente a un valor aleatorio entre **H\_i** y **H\_f**. Ejemplo:

**random.color({270, 320})** = "&HFF00AA&"

### Kodoku na hoho wo nurasu nurasu keto

Vemos que el tono (**Hue**) que resulta corresponde a un valor aleatorio entre 270 y 320 del círculo cromático **HSV**. O sea que cuando algún parámetro es una Tabla, entonces la función hace un Random entre los dos valores de la Tabla, cada vez que se ejecute la función.

Si combinamos los tres Modos de la función **random.color** (ausencia de uno o más de sus parámetros, uno o más parámetros constantes y por último, que uno o más de los parámetros sean una Tabla), obtenemos todo un mundo nuevo de posibilidades (15 en total):

- **random.color()**
- **random.color(H)**

- **random.color({H\_i, H\_f})**
- **random.color(H, S)**
- **random.color(H, {S\_i, S\_f})**
- **random.color({H\_i, H\_f}, S)**
- **random.color({H\_i, H\_f}, {S\_i, S\_f})**
- **random.color(H, S, V)**
- **random.color(H, S, {V\_i, V\_f})**
- **random.color(H, {S\_i, S\_f}, V)**
- **random.color({H\_i, H\_f}, S, V)**
- **random.color(H, {S\_i, S\_f}, {V\_i, V\_f})**
- **random.color({H\_i, H\_f}, S, {V\_i, V\_f})**
- **random.color({H\_i, H\_f}, {S\_i, S\_f}, V)**
- **random.color({H\_i, H\_f}, {S\_i, S\_f}, {V\_i, V\_f})**

En resumen:

- el parámetro **H** es un valor entre 0 y 360 o una tabla con dos elementos numéricos entre 0 y 360. Su valor por default es aleatorio entre 0 y 360.
- **S** y **V** son, o un valor entre 0 y 100 o una tabla con dos elementos numéricos entre 0 y 100. Sus valores por default son siempre de 100 para cada uno.

Ejemplos:

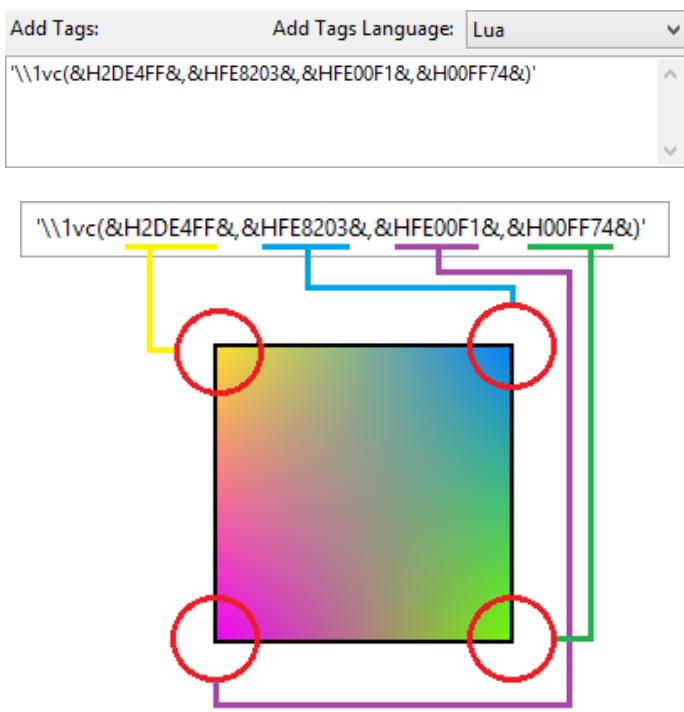
- **random.color(115)**
- **random.color({30, 220})**
- **random.color(304, 70)**
- **random.color(216, {60, 80})**
- **random.color({320, 340}, 90)**
- **random.color({125, 150}, {50, 100})**
- **random.color(0, 55, 92)**
- **random.color(86, 10, {0, 72})**
- **random.color(328, {60, 80}, 64)**
- **random.color({80, 120}, 77, 81)**
- **random.color(143, {0, 100}, {50, 80})**
- **random.color({45, 77}, 44, {66, 100})**
- **random.color({0, 105}, {70, 90}, 16)**
- **random.color({170, 204}, {55, 76}, {47, 88})**

Recomiendo poner a prueba en el **Kara Effector**, a cada uno de los anteriores ejemplos con el fin de practicar con esta función hasta que se familiaricen con cada uno de sus modos y opciones de uso. Esta es la primera función de la Librería "**random**" y restan cinco más por ver.

## Kara Effector - Effector Book [Tomo VII]:

**random.colorvc( H, S, V ):** es similar a **random.color** con la única diferencia que retorna un color en formato del filtro **VSFiltermod**.

Un color en formato **VSFilterMod** está compuesto por cuatro colores para cada uno de sus cuadrantes. Ejemplo:



O sea que todo color en formato **VSFilterMod** tiene la siguiente estructura:

**(color SI, color SD, color II, color ID)**

- **color SI:** Superior Izquierdo
- **color SD:** Superior Derecho
- **color II:** Inferior Izquierdo
- **color ID:** Inferior Derecho

Y lo que hace la función **random.colorvc** es usar la función **random.color** para generar a cada uno de esos cuatro anteriores colores. Los modos y todas las combinaciones posibles de la función **random.colorvc** son exactamente los mismos que en **random.color**, vista anteriormente:

- **random.colorvc( )**
- **random.colorvc(H)**
- **random.colorvc({H\_i, H\_f})**
- **random.colorvc(H, S)**
- **random.colorvc(H, {S\_i, S\_f})**
- **random.colorvc({H\_i, H\_f}, S)**

- **random.colorvc({H\_i, H\_f}, {S\_i, S\_f})**
- **random.colorvc(H, S, V)**
- **random.colorvc(H, S, {V\_i, V\_f})**
- **random.colorvc(H, {S\_i, S\_f}, V)**
- **random.colorvc({H\_i, H\_f}, S, V)**
- **random.colorvc(H, {S\_i, S\_f}, {V\_i, V\_f})**
- **random.colorvc({H\_i, H\_f}, S, {V\_i, V\_f})**
- **random.colorvc({H\_i, H\_f}, {S\_i, S\_f}, V)**
- **random.colorvc({H\_i, H\_f}, {S\_i, S\_f}, {V\_i, V\_f})**

No sobra recordarles que para usar colores en formato **VSFilterMod** debemos seleccionar esa opción en **Using Tags Filter**, de otro modo el **Kara Effector** convertirá el color a formato **VSFilter 2.39**.

**random.alpha(A1, A2):** retorna una transparencia (**alpha**) aleatoria en formato .ass (**VSFilter 2.39**) desde el valor de **A1** hasta **A2**.

Los parámetros **A1** y **A2** pueden ser números reales entre 0 y 255, o valores **alpha** entre "&H00&" y "&HFF&".

Modo 1: sin parámetros

**random.alpha():** retorna una transparencia al azar entre totalmente visible (0 en decimal o "&H00" en formato .ass), hasta totalmente invisible (255 en decimal o "&HFF&" en formato .ass).

Modo 2: parámetros numéricos

**random.alpha(A1, A2):** retorna una transparencia aleatoria entre los valores numéricos de **A1** y **A2**.  
Ejemplos:

- **random.alpha(55, 142)**
- **random.alpha(0, 200)**
- **random.alpha(180, 255)**

Modo 3: parámetros **alpha** en formato .ass

**random.alpha(A1, A2):** retorna una transparencia aleatoria entre los valores **alpha** (.ass) de **A1** y **A2**.  
Ejemplos:

- **random.alpha("&HA3&", "&HED&")**
- **random.alpha("&H0C&", "&H7F&")**

## Kara Effector - Effector Book [Tomo VII]:

Una especie de Modo 4 sería combinar los modos 2 y 3. Ejemplos:

- `random.alpha(120, "&HED&")`
- `random.alpha("&H0C&", 215)`

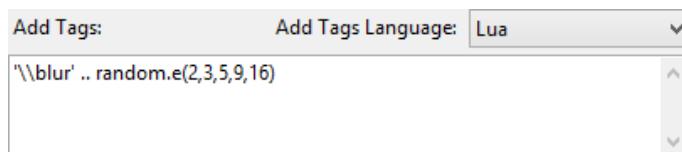
**random.alphava(A1, A2):** es muy similar a la anterior función y retorna una transparencia (**alpha**) aleatoria en formato **VSFilterMod** desde el valor de **A1** hasta **A2**. Los Modos y formas de uso son los mismos que usamos en **random.alpha**

Para cada una de las cuatro transparencias (alpha), que conforman a una transparencia en formato **VSFilterMod**, se ejecuta el random que hace que sea prácticamente imposible que las cuatro sean iguales. Ejemplos:

- `random.alphava(90, 180)`
- `random.alphava(0, 100)`
- `random.alphava("&HC4&", "&HFF&")`
- `random.alphava("&H00&", "&H86&")`
- `random.alphava(12, "&HAA&")`
- `random.alphava("&HDF&", 125)`

**random.e(...):** retorna un parámetro al azar de entre todos los que se le hayan ingresado o un elemento al azar, entre los elementos de una tabla que se le haya ingresado como parámetro.

Ejemplo 1:



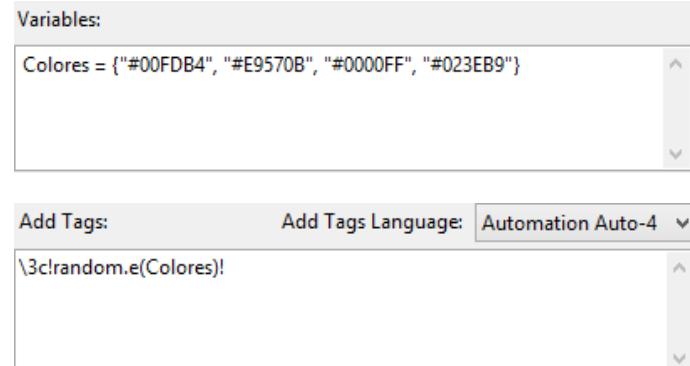
Add Tags: Add Tags Language: Lua

```
'\blur' .. random.e(2,3,5,9,16)
```

La función retornaría alguno de esos cinco valores que se le ingresaron, seleccionado de forma aleatoria, o sea que los posibles resultados serán:

- `\blur2`
- `\blur3`
- `\blur5`
- `\blur9`
- `\blur16`

Ejemplo 2:



Variables:

```
Colores = {"#00FDB4", "#E9570B", "#0000FF", "#023EB9"}
```

Add Tags: Add Tags Language: Automation Auto-4

```
\3clrandom.e(Colores)!
```

Declaré una tabla llamada “Colores”, entonces la función **random.e** selecciona un elemento al azar de entre los cuatro que tiene la tabla.

**random.unique(T, i):** primero desordena la posición de los elementos de la tabla **T** para posteriormente retornar al elemento **T[i]**. **T** puede ser una tabla propiamente dicha o un número entero positivo. En el caso de que **T** sea un entero, entonces se crea una tabla con los números desde 1 hasta **T**.

Ejemplo 1:

**random.unique(syl.n, syl.i):** primero crea una tabla con los números consecutivos desde 1 hasta **syl.n**:

$$T = \{1, 2, 3, 4, 5, 6, 7, 8, \dots, syl.n\}$$

Luego desordena la posición de los elementos de la tabla de manera aleatoria:

$$T = \{5, 8, 4, syl.n, 2, 6, 3, \dots, 1, 7\}$$

Por último, retorna **T[syl.i]**:

- $T[1] = 5$
- $T[2] = 8$
- $T[3] = 4$
- $T[4] = syl.n$
- $T[syl.n - 1] = 1$
- $T[syl.n] = 7$

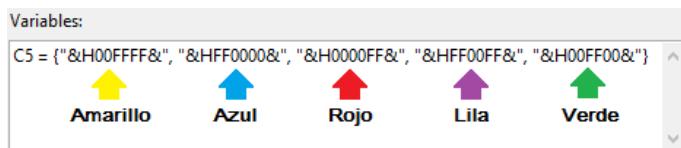
O sea que la función **random.unique** nunca repite un resultado, a menos que sea usada una cantidad mayor de veces que el tamaño de la tabla.

## Kara Effector - Effector Book [Tomo VII]:

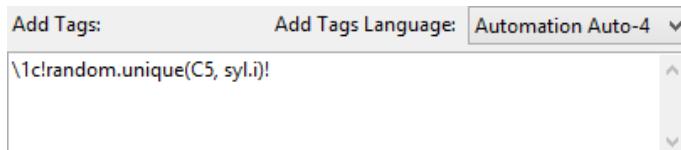
---

Ejemplo 2:

Declaramos una tabla con cinco colores dispuestos de la siguiente forma:



Si intentamos usar un color para cada Sílaba, y si una Línea de Texto tiene más de cinco Sílabas, entonces esto hace imposible que la función **random.unique** le asigne un único color de esta tabla a cada una de las Sílabas. Lo que hace la función es agotar las opciones y luego repite el mismo patrón una y otra vez:



Como la tabla “**C5**” de nuestro ejemplo solo tiene cinco elementos, el resultado **C5[6]** o cualquier otro mayor no existiría, entonces la función repite los resultados:

**Kodoku na hoho wo nurasu nurasu keto**

Para esta línea de texto la función reorganizó los colores de la tabla en forma aleatoria así:

1. Rojo
2. Amarillo
3. Lila
4. Verde
5. Azul

Y vemos que el patrón, al menos en esta línea, se repitió tres veces:

**Kodoku na hoho wo nurasu nurasu keto**

---

**1            2            3**

Esta se podría considerar como la función más compleja de la Librería “**random**”, pero con un poco de práctica se irán acostumbrando y encontrarán la forma de darle una aplicación en algunos de sus Efectos y proyectos.

---

Con la culminación de la Librería “**random**” se da por terminado el **Tomo VII**, con la recomendación de poner en práctica los ejemplos vistos en él. No olviden descargar la más reciente 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, exponer alguna duda o hacer alguna sugerencia.

---

**Kara Effector - Effector Book [Tomo VIII]:**

# **Kara Effector 3.2: Effector Book Vol. I [Tomo VIII]**

## **Kara Effector 3.2:**

Esta es la entrega del **Tomo VIII** y cada vez estamos más cerca de comprender a cada una de las herramientas con las que podemos contar en el **Kara Effector** a la hora de hacer un Efecto para nuestros Subtítulos. Siguiendo con este orden de ideas, continuaremos viendo las funciones y variables que aún nos restan por ver.

### **Las Funciones Compartidas de Automation Auto-4 y del Kara Effector:**

En el **Aegisub** se pueden usar una serie de funciones que hacen parte de la Librería del **Automation Auto-4**, que lleva por nombre: **utils auto-4**

La Librería **utils auto-4** es un archivo .lua que esta en la carpeta “**include**” del **Aegisub**, y en ella están todas las funciones que se pueden usar para hacer Efectos en los **Templates**:

C:\Program Files (x86)\Aegisub\automation\include		
Nombre	Fecha de modifica...	Tipo
cleantags	23/06/2013 3:59 p....	Archivo LUA
clipboard	23/06/2013 3:59 p....	Archivo LUA
karaskel	05/12/2011 12:06 ...	Archivo LUA
karaskel-auto4	05/12/2011 12:06 ...	Archivo LUA
re	23/06/2013 3:59 p....	Archivo LUA
unicode	23/06/2013 3:59 p....	Archivo LUA
utils	23/06/2013 3:59 p....	Archivo LUA
utils-auto4	23/06/2013 3:59 p....	Archivo LUA

## Kara Effector - Effector Book [Tomo VIII]:

Todas la funciones de esta Librería se pueden usar en el **Kara Effector**. Esas funciones son:

- **ass\_color(r, g, b)**
- **ass\_alpha(a)**
- **ass\_style\_color(r, g, b, a)**
- **alpha\_from\_style(scolor)**
- **color\_from\_style(scolor)**
- **HSV\_to\_RGB(H, S, V)**
- **HSL\_to\_RGB(H, S, L)**
- **interpolate(pct, min, max)**
- **interpolate\_color(pct, first, last)**
- **interpolate\_alpha(pct, first, last)**

La explicación de estas y otras funciones del **Aegisub** están en su **Web Oficial**:

[http://docs.aegisub.org/manual/Automation\\_4\\_utils.lua](http://docs.aegisub.org/manual/Automation_4_utils.lua)

Algunas de las anteriores funciones, me imagino que las han usado o las han visto en algún Efecto, es por eso que no profundizaré en detalles con ellas, o no al menos por ahora.

Hay otras funciones que también pertenecen al **Aegisub** y se pueden usar en el **Kara Effector**:

- **maxloop(new\_loop)**
- **relayer(new\_layer)**
- **retime(modo, add\_start, add\_end)**

**maxloop( n )**: esta función determina el cantidad de veces que se repetirá un Línea fx. Ejemplo:

Add Tags: Add Tags Language: Lua  
maxloop(3, '\\fad(200,0)'

Entonces cada Línea fx se triplica:

23	0:00:40.70	0:00:45.79	English					Me aferraré a
24	0:00:45.92	0:00:54.56	English					Dos corazones
25	0:00:02.43	0:00:08.16	Romaji	lead-in	Effector [Fx]	*Ko		
26	0:00:02.43	0:00:08.16	Romaji	lead-in	Effector [Fx]	*do		
27	0:00:02.43	0:00:08.16	Romaji	lead-in	Effector [Fx]	*ku		
28	0:00:02.43	0:00:08.16	Romaji	lead-in	Effector [Fx]	*na		
29	0:00:02.43	0:00:08.16	Romaji	lead-in	Effector [Fx]	*ho		

Otro Ejemplo, en Lenguaje **Automation Auto-4**:

Add Tags: Add Tags Language: Automation Auto-4  
!maxloop(\$sdur/10)!!retime("syl",0,0){\an5\b1}

Es decir que maxloop cumple una función muy similar a la celda de texto “loop”, pero con la diferencia de que la función **maxloop** se puede usar dentro de las nuevas funciones que se pueden hacer en la celda de texto “Variables”.

**relayer( n )**: esta función determina el número de la capa de cada una de las líneas generadas por un efecto. El entero “n” determina el número de la capa.

Un ejemplo sencillo en lenguaje **LUA** podría ser:

Add Tags: Add Tags Language: Lua  
relayer(syl.i, '\\blur2'

Y este sería el resultado:

número de la capa



23	0	0:00:40.70	0:00:45.79	English				Me aferraré a
24	0	0:00:45.92	0:00:54.56	English				Dos corazones
25	1	0:00:02.43	0:00:08.16	Romaji	lead-in	Effector [Fx]	*Ko	
26	2	0:00:02.43	0:00:08.16	Romaji	lead-in	Effector [Fx]	*do	
27	3	0:00:02.43	0:00:08.16	Romaji	lead-in	Effector [Fx]	*ku	
28	4	0:00:02.43	0:00:08.16	Romaji	lead-in	Effector [Fx]	*na	
29	5	0:00:02.43	0:00:08.16	Romaji	lead-in	Effector [Fx]	*ho	

O sea que el número de la capa de cada línea de efecto generada equivale a la posición de la sílaba dentro de la línea karaoke, esto es **syl.i**

Recordemos que el número de la capa (**layer**) determina la visibilidad de dos o más objetos que coinciden parcial o totalmente en su posición en el vídeo. Un objeto en el vídeo (ya sea una letra, una sílaba, una línea, una shape o lo que sea que pongamos en nuestras líneas en el **Aegisub**), se verá en frente del otro, si o solo si su número de capa es mayor a la del otro objeto con el que coincide en su posición o, si tienen el mismo número de capa y su número de línea en el script del **Aegisub** es mayor.

## Kara Effector - Effector Book [Tomo VIII]:

Y para usar esta función en lenguaje **Automation Auto-4** lo único que debemos hacer es escribirla dentro de los ya conocidos signos de admiración:



Es decir que relayer cumple una función muy similar a la celda de texto “**layer**”, pero con la diferencia de que la función relayer se puede usar dentro de las nuevas funciones que se pueden hacer en la celda de texto “**variables**”.

**retime(modo, add\_start, add\_end)**: es una de las funciones más importantes y más usadas del lenguaje **Automation Auto-4**, ya que “**re-timea**” los tiempos de las líneas de efecto generadas y es la función que hace posible que categorizemos nuestros efectos en entradas, efectos se sílaba activa, efectos de salida y demás tipos de efectos dependiendo del **modo** que usemos en la función.

Esta función se puede usar también en el **Kara Effector** exactamente de la misma manera que se usa en el **Aegisub**, con la significativa diferencia de que en el **Kara Effector**, esta función tiene muchos más **modos** que la versión original en **Automation Auto-4**.

Los **modos** de la función **retime** que se pueden usar en **Automation Auto-4** son 10:

1. preline
2. line
3. postline
4. presyl
5. start2syl
6. syl
7. syl2end
8. postsyl
9. sylpct
10. set or abs

Estos 10 **modos** se pueden usar en el **Kara Effector** de la misma manera que se hace en el **Aegisub**, ya sea en lenguaje **LUA** o en **Automation Auto-4**.

Los modos de la función **retime** que se pueden aplicar en el **Kara Effector** son 29:

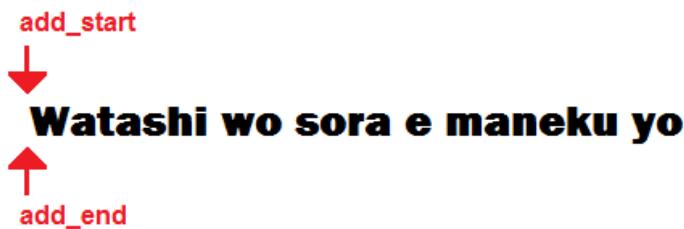
1. preline
2. line
3. postline
4. preword
5. start2word
6. word
7. word2end
8. postword
9. presyl
10. start2syl
11. syl
12. syl2end
13. postsyl
14. prefuri
15. start2furi
16. furi
17. furi2end
18. postfuri
19. prechar
20. start2char
21. char
22. char2end
23. postchar
24. linepct
25. wordpct
26. sylpct
27. furipct
28. charpct
29. set or abs

**retime(“preline”, add\_start, add\_end)**: el prefijo “pre” es la abreviatura de “previous” (previo, anterior, antes de) y “line” es “línea” en español. O sea que “preline” significa: antes de la línea.

El inicio y final del modo “**preline**” equivalen al tiempo de inicio de la línea karaoke (line.start\_time). En este mismo orden de ideas, **add\_start** es el tiempo en milisegundos que se agregará o sustraerá al inicio, y **add\_end** es el tiempo en milisegundos que se agregará o sustraerá al tiempo final, que como ya lo había mencionado, es el mismo que el tiempo de inicio para este modo. Para sustraer tiempo, ya sea al inicio, al final o a ambos, ponemos valores negativos dentro de la función **retime**.

## Kara Effector - Effector Book [Tomo VIII]:

La siguiente gráfica muestra en dónde se encuentran por default el **add\_start** y el **add\_end** en el modo “**preline**”:



Lo que equivale a: **retime("preline", 0, 0)**

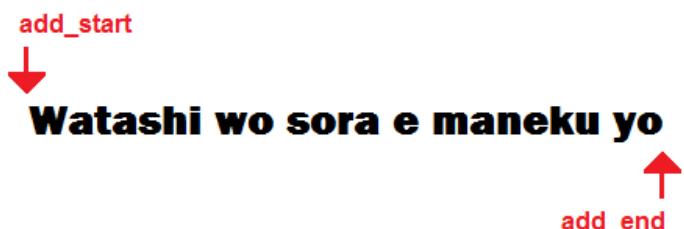
El modo “**preline**” puede ser usado en todo tipo de efecto o **Template Type**. Los recordamos:

- Line
- Syl
- Furi
- Char
- Translation Line
- Translation Word
- Translation Char

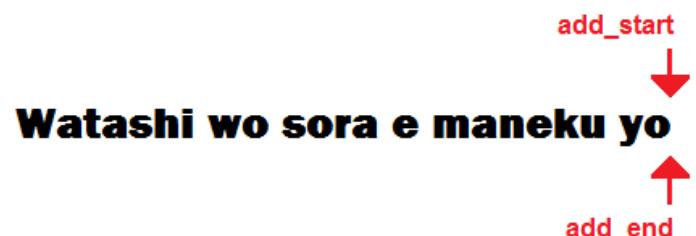
Algunos ejemplos de la función **retime** en modo “**preline**”:

- **retime("preline", -200, 0)**
- **retime("preline", -300, 200)**
- **retime("preline", 400, 1000)**
- **retime("preline", 0, 2000)**
- **retime("preline", -800, -100)**

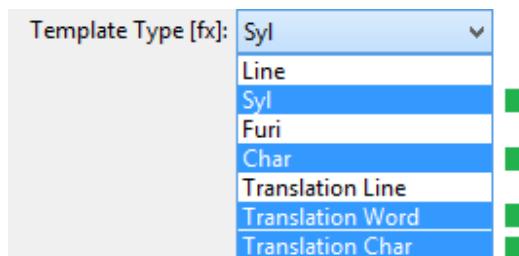
**retime("line", add\_start, add\_end)**: el tiempo de inicio equivale al tiempo de inicio de la línea karaoke (line.start\_time) y el tiempo final, al tiempo final de cada una de las líneas de karaoke (line.end\_time). Este **modo** está habilitado para todos los **Template Type**.



**retime("postline", add\_start, add\_end)**: “post” de posterior, o sea, el tiempo después de la línea. El tiempo de inicio equivale al tiempo final de la línea karaoke (line.end\_time) y el tiempo final, también al tiempo final de cada una de las líneas de karaoke (line.end\_time). Este **modo** está habilitado para todos los **Template Type**.



**retime("preword", add\_start, add\_end)**: literalmente “antes de la palabra”. El tiempo de inicio equivale al tiempo de inicio de cada una de las palabras en cada línea karaoke (line.start\_time + word.start\_time) y el tiempo final, también al tiempo de inicio de cada una de las palabras en cada línea. Este **modo** está habilitado para los siguientes **Template Type**: Syl, Char, Translation Word y Translation Char.



**retime("start2word", add\_start, add\_end)**: literalmente “desde el inicio hasta la palabra”. El tiempo de inicio equivale al tiempo de inicio de cada una de las líneas (line.start\_time) y el tiempo final, al tiempo de inicio de cada **palabra** (line.start\_time + word.start\_time).

## Kara Effector - Effector Book [Tomo VIII]:

Este modo está habilitado para los siguientes Template Type: Syl, Char, Translation Word y Translation Char.

add\_start



**Watashi wo sora e maneku yo**



add\_end

retime("word", add\_start, add\_end): literalmente “palabra”. El tiempo de inicio equivale al tiempo de inicio de cada una de las palabras en cada línea karaoke (line.start\_time + word.start\_time) y el tiempo final, al tiempo final de cada una de las palabras (line.start\_time + word.end\_time). Este modo está habilitado para los siguientes Template Type: Syl, Char, Translation Word y Translation Char.

add\_start



**Watashi wo sora e maneku yo**



add\_end

retime("word2end", add\_start, add\_end):

literalmente “desde la palabra hasta el final”. El tiempo de inicio equivale al tiempo final de cada una de las palabras de cada línea karaoke (line.start\_time + word.end\_time) y el tiempo final, al tiempo final de cada una de las líneas karaoke (line.end\_time). Este modo está habilitado para los siguientes Template Type: Syl, Char, Translation Word y Translation Char.

add\_start



**Watashi wo sora e maneku yo**



add\_end

retime("postword", add\_start, add\_end):

literalmente “después de la palabra”. El tiempo de inicio equivale al tiempo final de cada una de las palabras en cada línea karaoke (line.start\_time + word.end\_time) y el tiempo final, también al tiempo final de cada una de las palabras (line.start\_time + word.end\_time). Este modo está habilitado para los siguientes Template Type: Syl, Char, Translation Word y Translation Char.

add\_start



**Watashi wo sora e maneku yo**



add\_end

retime("presyl", add\_start, add\_end): literalmente “antes de la sílaba”. El tiempo de inicio equivale al tiempo de inicio de cada una de las sílabas en cada línea karaoke (line.start\_time + syl.start\_time) y el tiempo final, también al tiempo de inicio de cada una de las sílabas en cada línea. Este modo está habilitado para los siguientes Template Type: Syl, Furi, Char y Translation Char.

add\_start



**Watashi wo sora e maneku yo**



add\_end

retime("start2syl", add\_start, add\_end):

literalmente “desde el inicio hasta la sílaba”. El tiempo de inicio equivale al tiempo de inicio de cada una de las líneas (line.start\_time) y el tiempo final, al tiempo de inicio de cada sílaba (line.start\_time + syl.start\_time). Este modo está habilitado para los siguientes Template Type: Syl, Furi, Char y Translation Char.

add\_start



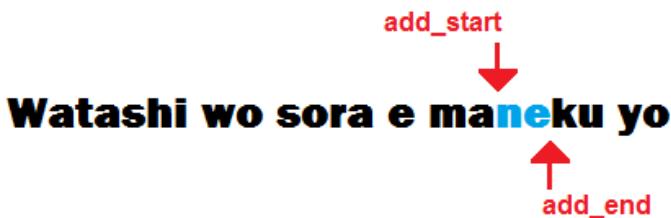
**Watashi wo sora e maneku yo**



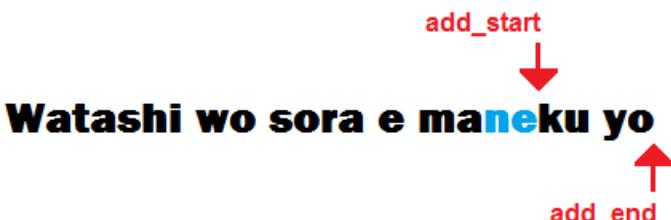
add\_end

## Kara Effector - Effector Book [Tomo VIII]:

`retime("syl", add_start, add_end)`: literalmente “sílaba”. El tiempo de inicio equivale al tiempo de inicio de cada una de las sílabas en cada línea karaoke (`line.start_time + syl.start_time`) y el tiempo final, al tiempo final de cada una de las sílabas (`line.start_time + syl.end_time`). Este modo está habilitado para los siguientes Template Type: Syl, Furi, Char y Translation Char.



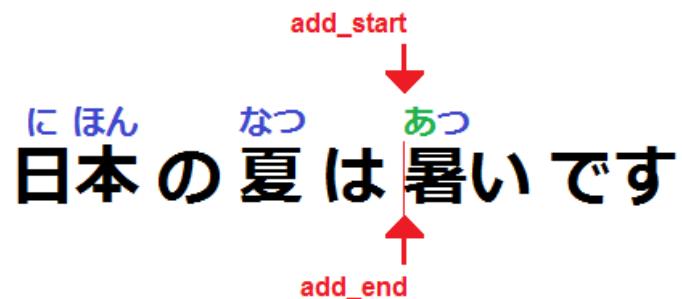
`retime("syl2end", add_start, add_end)`: literalmente “desde la sílaba hasta el final”. El tiempo de inicio equivale al tiempo final de cada una de las sílabas de cada línea karaoke (`line.start_time + syl.end_time`) y el tiempo final, al tiempo final de cada una de las líneas karaoke (`line.end_time`). Este modo está habilitado para los siguientes Template Type: Syl, Furi, Char y Translation Char.



`retime("postsyl", add_start, add_end)`: literalmente “después de la sílaba”. El tiempo de inicio equivale al tiempo final de cada una de las sílabas en cada línea karaoke (`line.start_time + syl.end_time`) y el tiempo final, también al tiempo final de cada una de las sílabas (`line.start_time + syl.end_time`). Este modo está habilitado para los siguientes Template Type: Syl, Furi, Char y Translation Char.



`retime("prefuri", add_start, add_end)`: literalmente “antes del furigana”. El tiempo de inicio equivale al tiempo de inicio de cada uno de los furiganas en cada línea karaoke (`line.start_time + furi.start_time`) y el tiempo final, también al tiempo de inicio de cada una de los furiganas en cada línea.



`retime("start2furi", add_start, add_end)`: literalmente “desde el inicio hasta el furigana”. El tiempo de inicio equivale al tiempo de inicio de cada una de las líneas (`line.start_time`) y el tiempo final, al tiempo de inicio de cada furigana (`line.start_time + furi.start_time`).



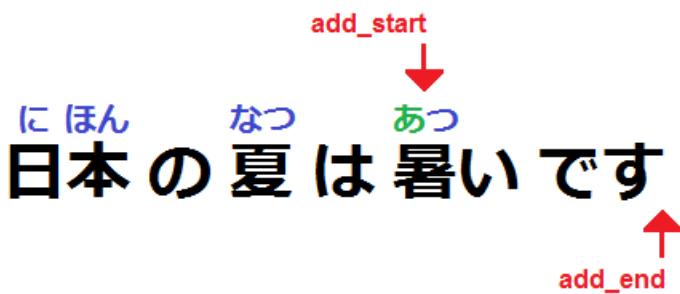
`retime("furi", add_start, add_end)`: literalmente “furigana”. El tiempo de inicio equivale al tiempo de inicio de cada uno de los furiganas en cada línea karaoke (`line.start_time + furi.start_time`) y el tiempo final, al tiempo final de cada uno de los furiganas (`line.start_time + furi.end_time`).



## Kara Effector - Effector Book [Tomo VIII]:

`retime("furi2end", add_start, add_end):`

literalmente “desde el **furigana** hasta el final”. El tiempo de inicio equivale al tiempo final de cada uno de los furiganas de cada línea karaoke (line.start\_time + furi.end\_time) y el tiempo final, al tiempo final de cada una de las líneas karaoke (line.end\_time).



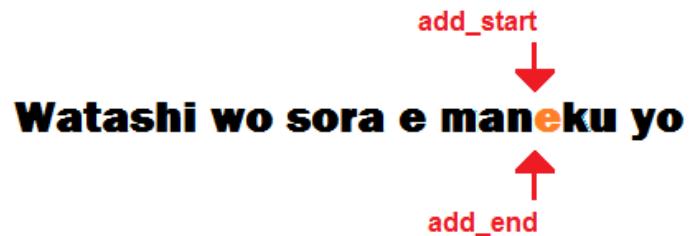
`retime("postfuri", add_start, add_end):`

literalmente “después del **furigana**”. El tiempo de inicio equivale al tiempo final de cada uno de los furiganas en cada línea karaoke (line.start\_time + furi.end\_time) y el tiempo final, también al tiempo final de cada uno de los furiganas (line.start\_time + furi.end\_time).



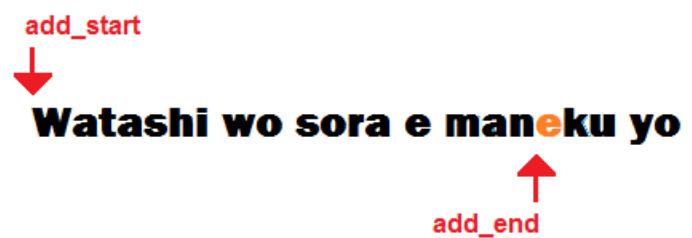
Olivaba mencionar que los 5 modos furiganas solo se pueden aplicar al **Template Type**: Furi, ya que no tiene mucho sentido aplicarle este tipo de modos a los demás **Template Type**.

`retime("prechar", add_start, add_end):` literalmente “antes del **caracter**”. El tiempo de inicio equivale al tiempo de inicio de cada uno de los caracteres en cada línea karaoke (line.start\_time + char.start\_time) y el tiempo final, también al tiempo de inicio de cada uno de los caracteres en cada línea. Este **modo** está habilitado para los siguientes **Template Type**: Char y Translation Char.

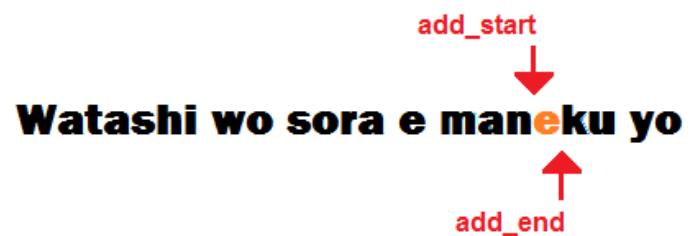


`retime("start2char", add_start, add_end):`

literalmente “desde el inicio hasta el **caracter**”. El tiempo de inicio equivale al tiempo de inicio de cada una de las líneas (line.start\_time) y el tiempo final, al tiempo de inicio de cada **caracter** (line.start\_time + char.start\_time). Este **modo** está habilitado para los siguientes **Template Type**: Char y Translation Char.



`retime("char", add_start, add_end):` literalmente “**caracter**”. El tiempo de inicio equivale al tiempo de inicio de cada uno de los caracteres en cada línea karaoke (line.start\_time + char.start\_time) y el tiempo final, al tiempo final de cada uno de los caracteres (line.start\_time + furi.end\_time). Este **modo** está habilitado para los siguientes **Template Type**: Char y Translation Char.

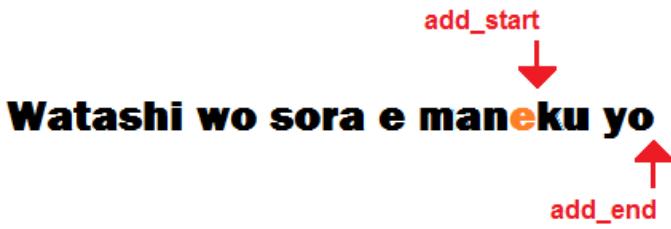


`retime("furi2end", add_start, add_end):`

literalmente “desde el **furigana** hasta el final”. El tiempo de inicio equivale al tiempo final de cada uno de los furiganas de cada línea karaoke (line.start\_time + furi.end\_time) y el tiempo final, al tiempo final de cada una de las líneas karaoke (line.end\_time). Este **modo** está

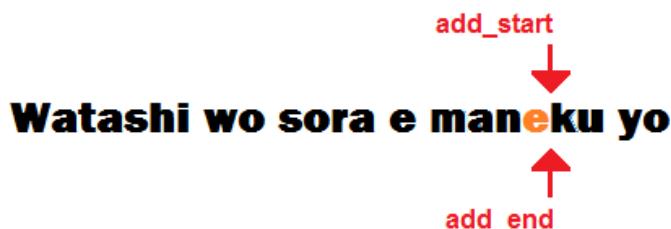
## Kara Effector - Effector Book [Tomo VIII]:

habilitado para los siguientes **Template Type**: Char y Translation Char.



### retime("postchar", add\_start, add\_end):

literalmente “después del **furiganaTemplate Type**: Char y Translation Char.



A continuación veremos los 5 modos “**pct**” de la función **retime**, que son: “**linepct**”, “**wordpct**”, “**sylpct**”, “**fuript**” y “**charpt**”.

“**pct**” es la sigla de “**percent**” (porcentaje en español). El 0% equivale al inicio del modo y el 100% al tiempo final, o sea, al total de la duración.

### retime("linepct", 0, 100):

usada con estos dos valores (0 y 100) es lo mismo que el modo “line”.



La diferencia real es cuando se usan otros valores para que se adicionen o resten del tiempo inicial de la línea.

Ejemplo:

- line.start\_time = 1200
- line.duration = 3600 ms
- line.end\_time = 4800
- retime("linepct", 0, 50)

0% de 3600 = 0

50% de 3000 = 1800, entonces:

1200 + 0 = 1200 ← sería el tiempo de inicio del re-timeo

1200 + 1800 = 3000 ← sería el tiempo final del re-timeo

De manera muy similar funcionan los otros 4 modos “**pct**”.

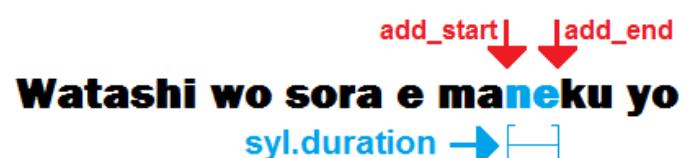
### retime("wordpct", 0, 100):

usada con estos dos valores (0 y 100) es lo mismo que el modo “word”.



### retime("sylpct", 0, 100):

usada con estos dos valores (0 y 100) es lo mismo que el modo “syl”.



### retime("fuript", 0, 100):

usada con estos dos valores (0 y 100) es lo mismo que el modo “furi”.



## Kara Effector - Effector Book [Tomo VIII]:

**retime("charpt", 0, 100):**

usada con estos dos valores (0 y 100) es lo mismo que el modo "char".

add\_start | add\_end

**Watashi wo sora e maneku yo**  
char.duration → H

Y por último, está el modo "set" o "abs", que es lo mismo. Y hace referencia al tiempo absoluto con respecto al cero del vídeo, ejemplo:

**retime("set", 1200, 23100):** entonces el tiempo de inicio del re-timeo será 1200 ms y el tiempo final será 23100 ms.

No pareciera ser sencillo memorizar los **29 modos** de la función **retime**, pero es solo cuestión de práctica y de un poco de sentido común dado el nombre de cada uno de los modos. Y como ya lo había mencionado, esta función es de gran utilidad y más para aquellos que ya la han usado en **Automation Auto-4** y están familiarizados con los 10 modos por default del **Aegisub**.

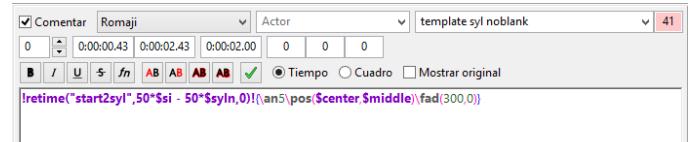
Estos son dos cortos ejemplos de cómo usar la función **retime** dependiendo del lenguaje que más nos guste, **LUA** o **Automation Auto-4**:

Add Tags:      Add Tags Language: **Lua**  
`retime("start2word", 0, 300), "\fad(0,300)"`

Add Tags:      Add Tags Language: **Automation Auto-4**  
`!retime("syl2end", -100, 600)! \fad(0,300)`

Y a manera de práctica, pueden usar el resto de los modos para que sepan qué uso darle acada uno de ellos y crear efectos de entrada (**lead-in**), de sílaba activa (**hi-light**), de salida (**lead-out**) o la combinación que deseen. Las posibilidades son muchas y los efectos también.

El **Kara Effector** también nos da la posibilidad de pegar en él algún **Template** copiado desde el **Aegisub**. Ejemplo:



Inicio	Final	Estilo	Efecto	Texto
0:00:00.43	0:00:02.43	Romaji	template syl noblank	!retime("start2syl",50*\$si - 50*\$syln,0)!*
0:00:02.43	0:00:08.16	Romaji		*Ko*do*ku *na *ho*ho *wo *nu*ra*su *nu*ra*su *ke*ko
0:00:08.33	0:00:13.19	Romaji		*yo*a*ke *no *ke*ha*ki *ga *shiz*zu*ka *ni *mi*chi*te

En el caso de que nos guste este Template en particular, que de casualidad es de un Efecto que nos gusta, entonces lo copiamos tal cual y lo pegamos en el **Kara Effector** usando el lenguaje **Automation Auto-4**, así:

Add Tags:      Add Tags Language: **Automation Auto-4**  
`!retime("start2syl",50*$si - 50*$syln,0)!{\\an5\\pos($center,$middle)}\\fad(300,0)`

Y al aplicar el Efecto, el resultado será el mismo que en el **Aegisub**, lo que implica una gran ventaja desde donde se lo mire.

El **Tomo VIII** se despide con la recomendación de poner en práctica los ejemplos vistos en él. No olviden descargar la más reciente 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, exponer alguna duda o hacer alguna sugerencia.

# Kara Effector 3.2: Effector Book Vol. I [Tomo IX]

# Kara Effector 3.2:

En el **Tomo IX** seguiremos profundizando en las librerías del **Kara Effector**, ya que eso nos dará las herramientas necesarias para aumentar nuestro nivel al momento de hacer un Efecto.

El **Kara Effector** tiene muchas funciones, pero no todas ellas son para hacer Efectos. Hay algunas funciones que nos sirven de apoyo para crear otras nuevas, también hay funciones que hacen que el **Kara Effector** pueda llevar a cabo su tarea y hay otras que nos ayudan en la generación de los Efectos.

En las librerías que veremos a continuación hay de todo tipo de funciones y para todos los gustos. Como siempre digo, a la final todo depende de qué queremos hacer para tener claro por cuál función nos decidiremos o cuál es la que mejor se ajusta a nuestro proyecto.

## Librería: Funciones de Tiempo [KE]

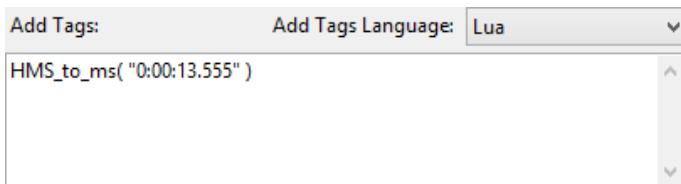
**HMS\_to\_ms( Time\_HMS )**: esta función convierte un tiempo dado, de formato **HMS** (horas, minutos y segundos) a formato ms (milisegundos). El tiempo **HMS** debe ser ingresado entre comillas, ya sean comillas sencillas o dobles. Ejemplo:

The screenshot shows the Kara Effector interface. At the top, there's a toolbar with various icons. Below it is a timeline with a playhead at 0:00:13.555. A red box highlights this timestamp. To the right of the timeline is a numeric value '325'. Below the timeline is a table with columns: #, L, Inicio, Final, Estilo, and Texto. The table contains three rows of data:

#	L	Inicio	Final	Estilo	Texto
1	1	0:00:02.43	0:00:08.16	Romaji	*Ko*do*ku *na *ho*ho
2	1	0:00:08.33	0:00:13.19	Romaji	*Yo*za*ke *no *ke*ha*
3	1	0:00:13.54	0:00:18.39	Romaji	*Wa*ta*shi *wo *so*ra

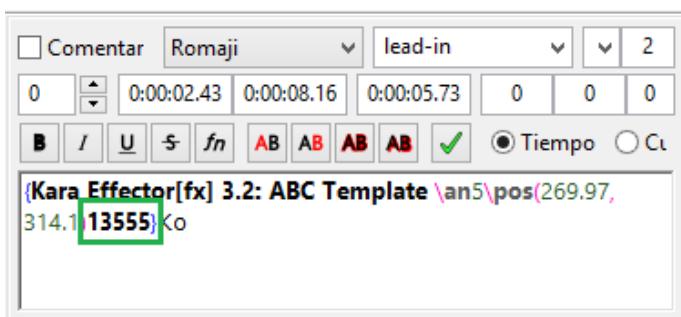
## Kara Effector - Effector Book [Tomo IX]:

Copiamos el tiempo que está dentro del recuadro rojo en la anterior imagen, y lo pegamos dentro de la función, sin olvidar colocarlo entre comillas:



```
Add Tags: Add Tags Language: Lua  
HMS_to_ms( "0:00:13.555" )
```

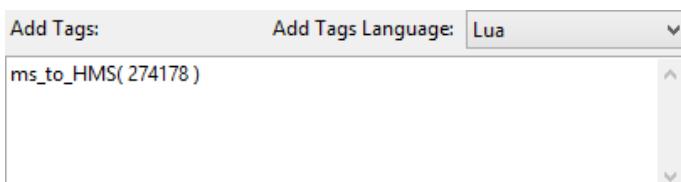
Y la función convertirá ese tiempo que estaba en formato **HMS** a formato ms, cuyo resultado cuenta como un valor numérico:



O sea que: **HMS\_to\_ms( "0:00:13.55" ) = 13555 ms**

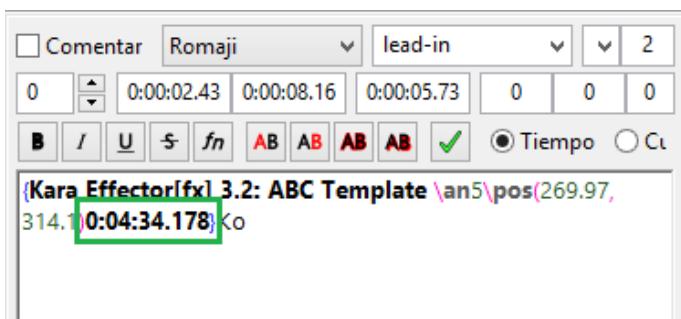
---

**ms\_to\_HMS( Time\_ms )**: esta función convierte un tiempo dado, de formato ms (milisegundos) a formato **HMS** (horas, minutos y segundos). El tiempo ms debe ser ingresado como un valor numérico, sin comillas. Ejemplo:



```
Add Tags: Add Tags Language: Lua  
ms_to_HMS( 274178 )
```

Obtenemos:

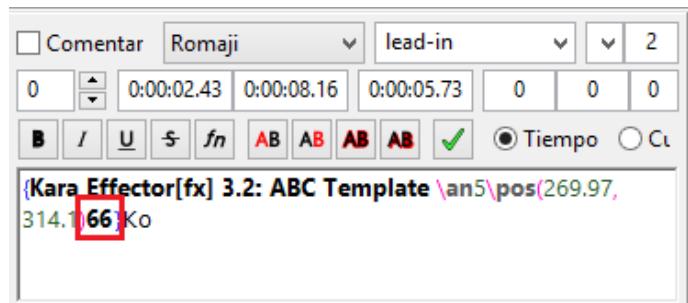


**time\_to\_frame( time )**: convierte un tiempo dado, ya sea en formato **HMS** o ms, en la cantidad de “frames” (cuadros) que ocuparía ese tiempo en el video que se está usando para aplicar un Efecto.

Ejemplo con tiempo en formato ms (milisegundos):



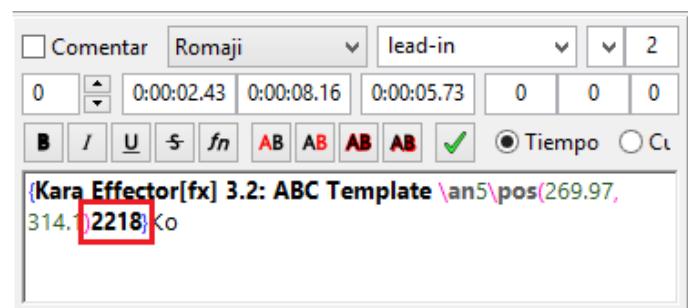
```
Add Tags: Add Tags Language: Lua  
time_to_frame(2728)
```



Ejemplo con tiempo en formato **HMS**:



```
Add Tags: Add Tags Language: Lua  
time_to_frame('0:01:32.486')
```



De lo anterior se concluye que 2728 ms equivalen a 66 frames (cuadros), y que en “0:01:32:486” (1 minuto, 32 segundos y 486 ms) hay 2218 frames.

---

**frame\_to\_ms( frames )**: esta función convierte el número de la cantidad de frames a ms (milisegundos). La cantidad de frames debe ser ingresada como un valor numérico, sin las comillas, ejemplo:

## Kara Effector - Effector Book [Tomo IX]:

Add Tags: Add Tags Language: Lua  
frame\_to\_ms( 150 )

Comentar Romaji lead-in 2  
0 0:00:02.43 0:00:08.16 0:00:05.73 0 0 0  
**B I U S fn AB AB AB AB ✓**  Tiempo  C.  
Kara Effector[fx] 3.2: ABC Template \an5\pos(269.97,  
314.1 6257) Ko

**frame\_to\_HMS( frames )**: esta función convierte el número de la cantidad de frames a un tiempo en formato **HMS** (horas, minutos y segundos). La cantidad de frames debe ser ingresada como un valor numérico, sin las comillas, ejemplo:

Add Tags: Add Tags Language: Lua  
frame\_to\_HMS( 150 )

Comentar Romaji lead-in 2  
0 0:00:02.43 0:00:08.16 0:00:05.73 0 0 0  
**B I U S fn AB AB AB AB ✓**  Tiempo  C.  
Kara Effector[fx] 3.2: ABC Template \an5\pos(269.97,  
314.1 0:00:06.257) Ko

**time\_mid1( delay )**: esta función, dependiendo el tipo de Efecto (este y los siguientes ejemplos están hechos con **Template Type: Syl**, pero se puede usar para todos los tipos menos el Line y Template Line), si se usa en el tiempo de inicio de la Línea, hace que las sílabas aparezcan de forma progresiva, desde los extremos hacia el centro de la Línea, como se puede apreciar en el siguiente ejemplo:

Line Start Time = `I.start_time + time_mid1( 50 )`



Esta función adicionada al tiempo de inicio de la Línea, nos sirve para hacer efectos **lead-in** (efectos de entrada). El tiempo que separa la aparición de una sílaba con respecto de la otra, es el **delay** (retraso), que para este ejemplo fue de 50 ms:

25	0	0:00:02.89	0:00:08.16	Romaji	lead-in	Effector [Fx]	*Ko
26	0	0:00:02.94	0:00:08.16	Romaji	lead-in	Effector [Fx]	*do
27	0	0:00:02.99	0:00:08.16	Romaji	lead-in	Effector [Fx]	*ku
28	0	0:00:03.04	0:00:08.16	Romaji	lead-in	Effector [Fx]	*na
29	0	0:00:03.09	0:00:08.16	Romaji	lead-in	Effector [Fx]	*ho
30	0	0:00:03.14	0:00:08.16	Romaji	lead-in	Effector [Fx]	*ho

Lo que muestra la imagen anterior es que la diferencia entre 0:00:02.890 y 0:00:02.940 es de 50 ms, que fue el **delay** usado para este ejemplo. Y como ya se había dicho antes, esta función también puede ser usada con los **Template Type**: Furi, Char, Translation Char y Translation Word; con resultados similares.

A mayor **delay**, mayor será el tiempo en que aparezca una sílaba (caracter o palabra, dependiendo del Template Type) con respecto de la inmediatamente anterior. O sea que uno decide el retraso de la función dependiendo del resultado que queremos obtener.

La función **time\_mid1( delay )** usada en el tiempo final de la Línea, nos ayuda a hacer efectos **lead-out** (de salida) y hace que las sílabas desaparezcan desde los extremos hacia el centro de la línea, ejemplo:

Line End Time = `I.end_time + time_mid1( 50 )`



O sea que la función **time\_mid1( delay )** nos sirve para hacer efectos, tanto **lead-in** como **lead-out**, dependiendo en qué tiempo de la Línea la usemos: **Line Start Time** o **Line End Time**, tiempo de inicio y final, respectivamente.

## Kara Effector - Effector Book [Tomo IX]:

**time\_mid2( delay )**: esta función, dependiendo el tipo de Efecto, si se usa en el tiempo de inicio de la Línea, hace que las sílabas aparezcan de forma progresiva, desde el centro hacia los extremos de la Línea. Es la función opuesta a **time\_mid1**, como se puede apreciar en el siguiente ejemplo:

Line Start Time =



También se puede usar esta función en los Template Type: Furi, Char, Tranlation Char y Translation Word

La función **time\_mid2( delay )** usada en el tiempo final de la Línea, nos ayuda a hacer efectos **lead-out** (de salida) y hace que las sílabas desaparezcan desde el centro hacia los extremos de la Línea. Ejemplo:

Line End Time =



**time\_li( delay )**: esta función es similar a las dos funciones anteriormente vistas. Esta función se puede sumar o restar al tiempo de inicio de la Línea y por eso su nombre: **time\_li (time lead-in)**. Y al igual que las dos anteriores funciones, se puede usar en los **Template Type**: Syl, Furi, Char, Template Char y Translation Word. Para este ejemplo usaré **Template Type**: Syl

Modo suma: hace que las sílabas aparezcan de forma progresiva de izquierda a derecha en la Línea.

Line Start Time =



Modo resta: hace que las sílabas aparezcan de forma progresiva de derecha a izquierda en la Línea.

Line Start Time =



**time\_lo( delay )**: es similar a **time\_li**. Esta función se puede sumar o restar al tiempo final de la Línea y por eso su nombre: **time\_lo (time lead-out)**. Y al igual que las anteriores funciones, se puede usar en los **Template Type**: Syl, Furi, Char, Template Char y Translation Word.

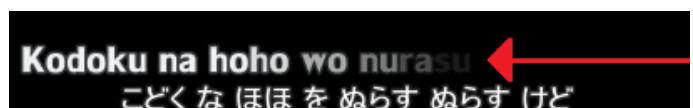
Modo suma: hace que las sílabas desaparezcan de forma progresiva de izquierda a derecha en la Línea.

Line End Time =



Modo resta: hace que las sílabas desaparezcan de forma progresiva de derecha a izquierda en la Línea.

Line End Time =



Con esta función culmina la librería de las funciones de tiempo del Kara Effector y da paso a la próxima librería.

## Librería: tag [KE]

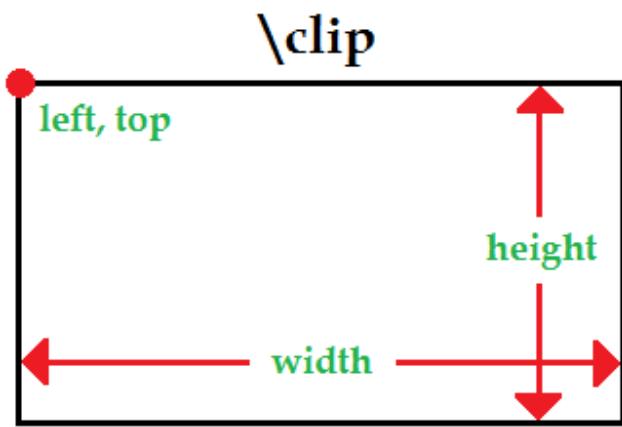
Esta librería contiene una serie de funciones enfocadas en los Tags que usamos a la hora de hacer Efectos.

**tag.clip( left, top, width, height, mode )**: crea uno o más clip's rectangulares dependiendo del **loop** asignado, con posición y medidas específicas. Los cinco parámetros a ingresar en la función son opcionales, ya que cada uno de ellos tiene valor por default en caso de ser necesario.

## Kara Effector - Effector Book [Tomo IX]:

**left y top** son las coordenadas 'x' y 'y' respectivamente del punto de origen del clip, que es el punto superior izquierdo del rectángulo que lo generará.

**width y height** son el ancho y el alto del rectángulo que generará al clip, respectivamente:



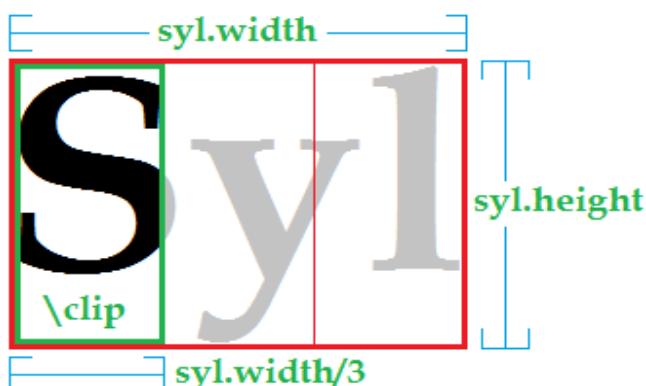
Veamos algunos ejemplos para empezar a aclarar los conceptos hasta acá vistos:

```
Add Tags:          Add Tags Language: Lua
tag.clip( syl.left, syl.top, syl.width/3, syl.height )
```

- **left** = syl.left
- **top** = syl.top
- **width** = syl.width/3
- **height** = syl.height



Solo queda visible un tercio del ancho de la sílaba, ya que para el ejemplo se usó **syl.width/3** como el ancho del clip:



Si nuestro efecto es un **Template Type: Syl**, y queremos que toda la sílaba sea totalmente visible dentro del clip, debemos usar los siguientes valores:

```
Add Tags:          Add Tags Language: Lua
tag.clip( syl.left, syl.top, syl.width, syl.height )
```

Los anteriores valores también son los valores por default en el **Template Type: Syl**, o sea que lo podemos usar con el mismo resultado de la siguiente forma:

```
Add Tags:          Add Tags Language: Lua
tag.clip()
```

Es decir, que los valores por default que usará la función dependen del Template Type.

Para Template Type: **Line** y **Translation Line**, los valores por default de **tag.clip( )** son: line.left, line.top, line.width y line.height

Para el Template Type: **Translation Word**, los valores por default de **tag.clip( )** son: word.left, word.top, word.width y word.height

Para el Template Type: **Syl**, los valores por default de **tag.clip( )** son: syl.left, syl.top, syl.width y syl.height

Para el Template Type: **furi**, los valores por default de **tag.clip( )** son: furi.left, furi.top, furi.width y furi.height

Y para Template Type: **Char** y **Translation Char**, los valores por default de **tag.clip( )** son: char.left, char.top, char.width y char.height

Hasta este punto pareciera que la función **tag.clip** no tiene nada de especial, o al menos que no tiene algo distinto a lo que el tag "\clip" podría hacer por sí solo, pero es aquí en donde la celda de texto "**loop**" entra en escena y nos muestra las ventajas de la función, dependiendo de los resultados que deseemos en nuestros efectos.

## Kara Effector - Effector Book [Tomo IX]:

A continuación veremos las distintas combinaciones que podemos hacer con la función **tag.clip** a partir de los valores que usemos en la celda de texto “**loop**”. Ejemplo:

- **loop:** 4
- Template Type: **Translation Word**

loop = 4

Add Tags: Add Tags Language: Lua

tag.clip( )

La función **tag.clip** creará 4 clip's horizontales dentro de rectángulo, según los parámetros que hayamos ingresado a la función, en este ejemplo está por default y como es un Template Type: **Translation Word**, entonces tiene las dimensiones exactas de cada una de las palabras en cada una de las líneas:



Con solo colocar cualquier valor en la celda de texto “**loop**” obtendremos tantos **clip's horizontales** como los necesitemos. Para el siguiente ejemplo veremos cómo obtener clip's verticales con la función **tag.clip**

- **loop:** 1, 3
- Template Type: **Char**

loop = 1, 3

Add Tags: Add Tags Language: Lua

tag.clip( )



Lo que hace el loop usado de ese modo es que la función **tag.clip** cree **clip's verticales** dentro del rectángulo hecho por las medidas ingresadas y como es un **Template Type: Char** y usamos los valores por default de la función, hará que cada carácter quede dentro de los tres clip's creados.

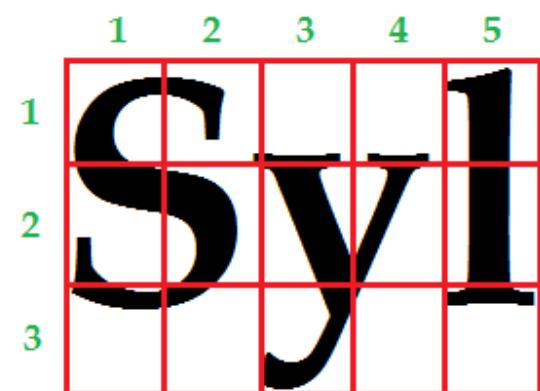
En el próximo ejemplo veremos la forma de hacer **clip's reticulares** (como en forma de grilla o cuadrícula):

- **loop:** 3, 5
- Template Type: **Syl**

loop = 3, 5

Add Tags: Add Tags Language: Lua

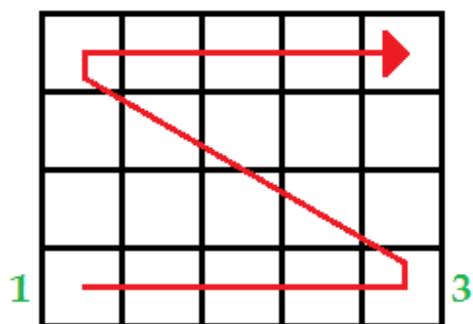
tag.clip( )



La retícula creada por la función **tag.clip** será de **3** clip's horizontales por **5** verticales y la cantidad total del **loop** será:  $3 \times 5 = 15$

El parámetro **mode** en la función **tag.clip** es un número que asiganmos con el fin de determinar el orden de los clip's. Son **8 modes** y los veremos a cada uno de ellos:

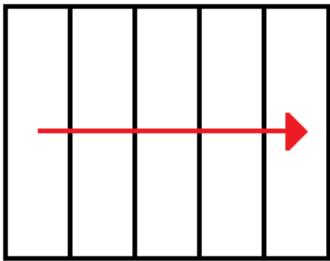
**Modo: 13**



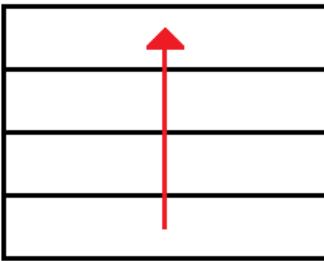
## Kara Effector - Effector Book [Tomo IX]:

- **modo 13:** la imagen anterior muestra el orden de los clip's en el **modo 13** de un **tag.clip** reticulado. En el caso de los clip's verticales, el **modo 13** hace que el primer clip sea el del extremo izquierdo y el último el del extremo derecho. Para los clip's horizontales el primero sería el del extremo inferior y el último el del extremo superior:

Modo: 13

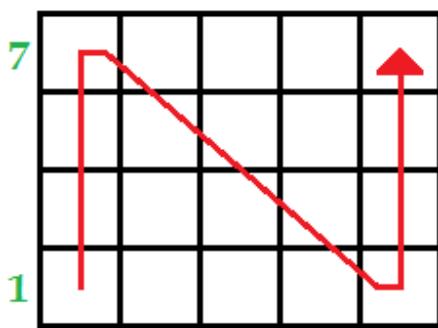


Modo: 13



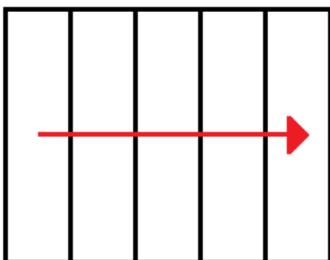
- **modo 17:** para un **tag.clip** reticulado, el orden de los clip's es como el de la siguiente imagen, es decir, el primre clip es el de la esquina inferior izquierda y el último es el de la esquina supeior derecha, siguiendo la trayectoria de la gráfica:

Modo: 17

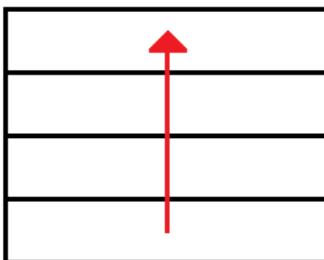


En el caso de los clip's verticales, el **modo 17** hace que el primer clip sea el del extremo izquierdo y el último el del extremo derecho. Para los clip's horizontales el primero sería el del extremo inferior y el último el del extremo superior:

Modo: 17

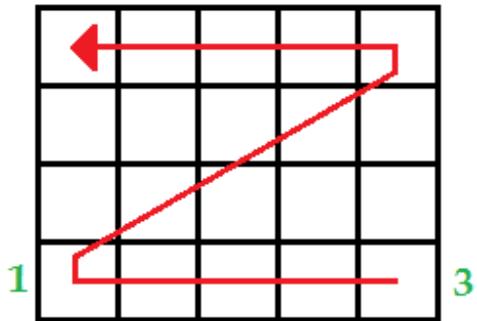


Modo: 17



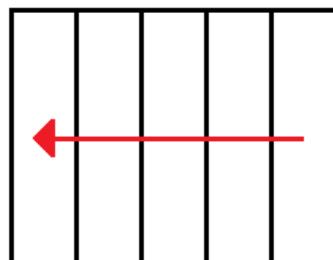
- **modo 31:** para un **tag.clip** reticulado, el orden de los clip's sería; el primer clip es el de la esquina inferior derecha y el último es el de la esquina supeior izquierda, siguiendo la trayectoria de la gráfica:

Modo: 31

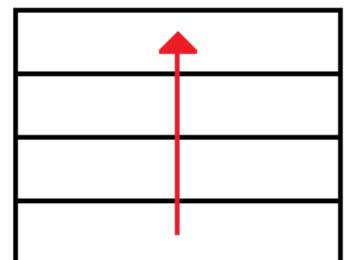


En el caso de los clip's verticales, el **modo 31** hace que el primer clip sea el del extremo derecho y el último el del extremo izquierdo. Para los clip's horizontales el primero sería el del extremo inferior y el último el del extremo superior:

Modo: 31

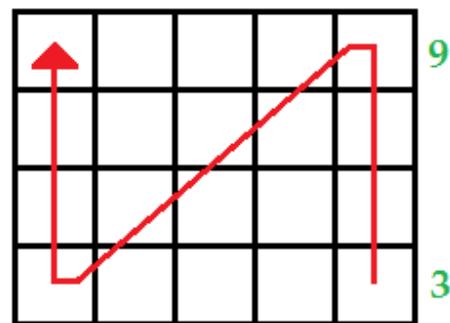


Modo: 31



- **modo 39:** para un **tag.clip** reticulado, el orden de los clip's sería; el primer clip es el de la esquina inferior derecha y el último es el de la esquina supeior izquierda, siguiendo la trayectoria de la gráfica:

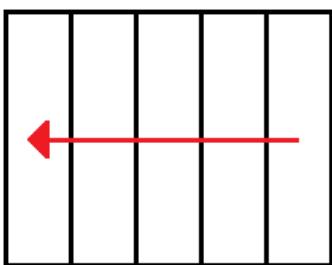
Modo: 39



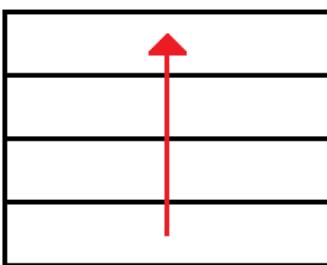
## Kara Effector - Effector Book [Tomo IX]:

En el caso de los clip's verticales, el **modo 39** hace que el primer clip sea el del extremo derecho y el último el del extremo izquierdo. Para los clip's horizontales el primero sería el del extremo inferior y el último el del extremo superior:

Modo: 39

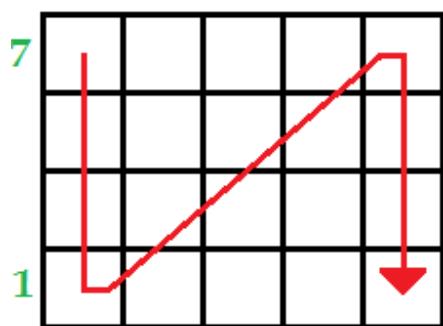


Modo: 39



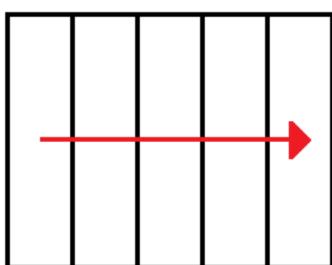
- **modo 71:** para un **tag.clip** reticulado, el orden de los clip's sería; el primer clip es el de la esquina superior izquierda y el último es el de la esquina inferior derecha, siguiendo la trayectoria de la gráfica:

Modo: 71

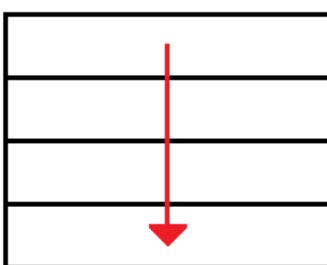


En el caso de los clip's verticales, el **modo 71** hace que el primer clip sea el del extremo izquierdo y el último el del extremo derecho. Para los clip's horizontales el primero sería el del extremo superior y el último el del extremo inferior:

Modo: 71



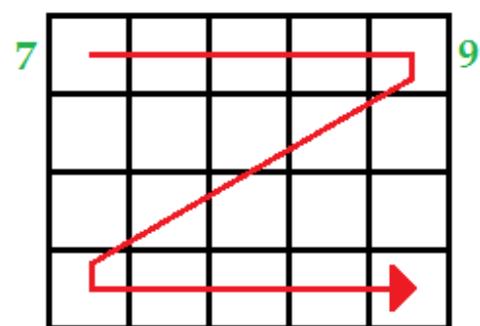
Modo: 71



- **modo 79:** es el modo por default de la función. para un **tag.clip** reticulado, el orden de los clip's

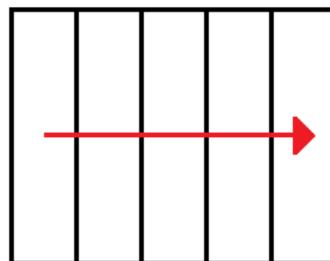
sería el siguiente; el primer clip es el de la esquina superior izquierda y el último es el de la esquina inferior derecha, siguiendo la trayectoria de la gráfica:

Modo: 79

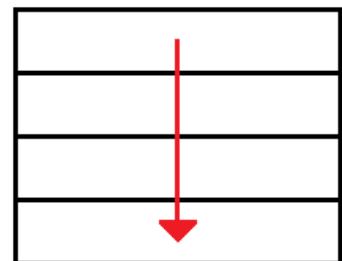


En el caso de los clip's verticales, el **modo 79** hace que el primer clip sea el del extremo izquierdo y el último el del extremo derecho. Para los clip's horizontales el primero sería el del extremo superior y el último el del extremo inferior:

Modo: 79

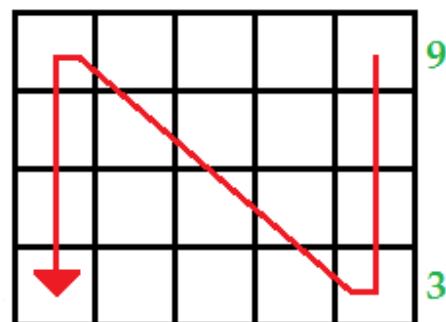


Modo: 79



- **modo 93:** para un **tag.clip** reticulado, el orden de los clip's sería; el primer clip es el de la esquina superior derecha y el último es el de la esquina inferior izquierda, siguiendo la trayectoria de la gráfica:

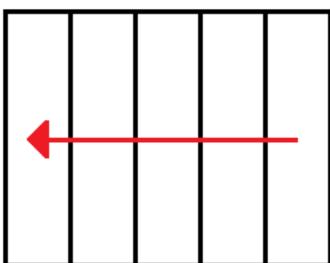
Modo: 93



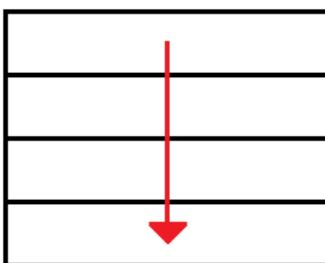
## Kara Effector - Effector Book [Tomo IX]:

En el caso de los clip's verticales, el **modo 93** hace que el primer clip sea el del extremo derecho y el último el del extremo izquierdo. Para los clip's horizontales el primero sería el del extremo superior y el último el del extremo inferior:

Modo: 93

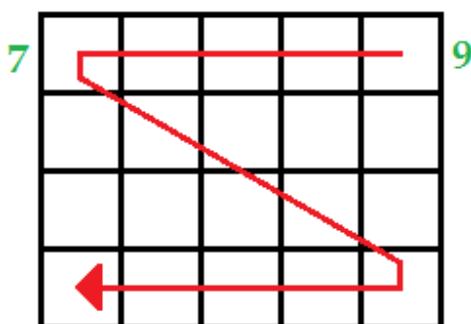


Modo: 93



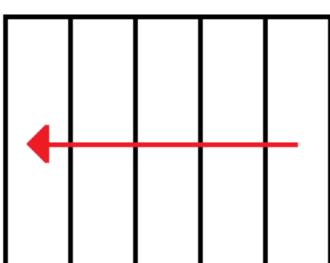
- **modo 97:** para un **tag.clip** reticulado, el orden de los clip's sería; el primer clip es el de la esquina superior derecha y el último es el de la esquina inferior izquierda, siguiendo la trayectoria de la gráfica:

Modo: 97

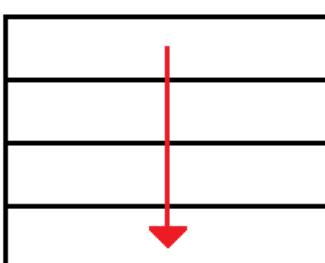


En el caso de los clip's verticales, el **modo 97** hace que el primer clip sea el del extremo derecho y el último el del extremo izquierdo. Para los clip's horizontales el primero sería el del extremo superior y el último el del extremo inferior:

Modo: 97



Modo: 97



Y el **modo 97** sería el último de ellos. Así que hay para elegir según sea nuestra necesidad en un Efecto.

Recordemos que el número **modo** lo escribirímos dentro de la función **tag.clip** como un valor numérico, ejemplos:

`tag.clip( fx.pos_l, fx.pos_t, syl.width, syl.height, 31 )`

`tag.clip( fx.pos_l, fx.pos_t, char.width, char.height, 97 )`

`tag.clip( fx.pos_l - 50, fx.pos_t, l.width + 100, l.height, 17 )`

Es todo por el momento y damos por terminado el **Tomo IX**. En el **Tomo X** del **Kara Effector** continuaremos no solo con la función **tag.clip** sino también con el resto de la Librería “**tag**”, ya que vale la pena dedicarle más de tiempo y espacio al estudio de estas funciones y todos los efectos que podemos hacer con ellas.

Intenten poner en práctica todos los ejemplos vistos en este **Tomo** y no olviden descargar 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, exponer alguna duda o hacer alguna sugerencia.

# Kara Effector 3.2: Effector Book Vol. I [Tomo X]

---

## Kara Effector 3.2:

Como lo había mencionado al final de la anterior entrega, este **Tomo X** es la continuación de la librería “**tag**”, que como las demás librerías del **Kara Effector**, es importante que sepamos en qué consiste cada una de sus funciones para poder sacarle el máximo provecho posible.

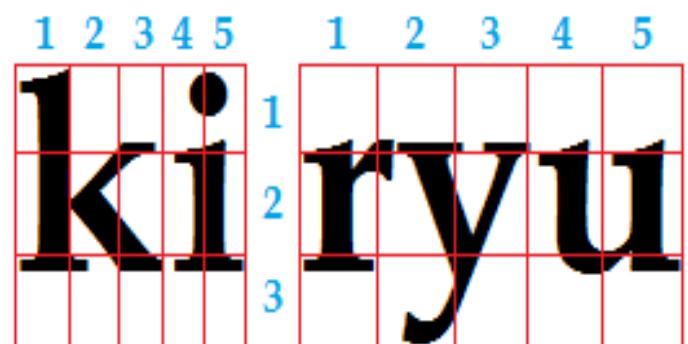
Sin más, retomaremos la librería “**tag**” en la función en donde nos habíamos quedado; la función **tag.clip**

### Librería tag [KE]:

Veremos la forma de hacer un **clip multiple cuadrado**, es decir que las dimensiones del clip sean las mismas. Ya sabemos cómo hacer un clip reticulado o de cuadricula con la función **tag.clip** y es de la siguiente forma, ejemplo:



Pero este método no garantiza que los clip's tengan las mismas dimensiones, es decir que sean cuadrados:



En la imagen anterior vemos cómo para algunas sílabas las proporciones de los clip's no son para nada cuadradas y eso es porque la cantidad de clip's verticales es constante, lo que no es ideal dados los distintos anchos de la Líneas, Palabras, Sílabas, Caracteres y demás.

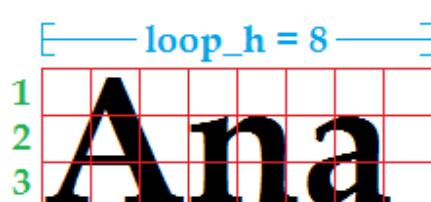
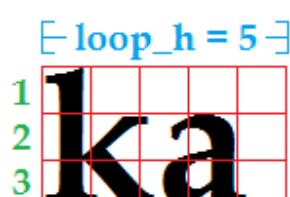
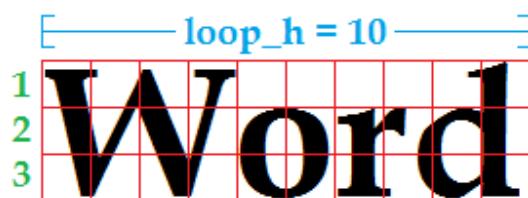
---

## Kara Effector - Effector Book [Tomo X]:

Entonces, para que los clip's sean cuadrados, les mostraré dos formas distintas de hacerlo. Y la primera forma de hacerlo es usando la variable **loop\_h**, así:

loop =

**loop\_h:** (variable) es un número entero calculado por el **Kara Effector**, teniendo en cuenta la cantidad de clip's verticales (del ejemplo anterior: 3), que hace que el ancho y el alto de los clip's tengan las mismas dimensiones y sean cuadrados, ejemplo:



Entonces el **loop\_h** varía en todos los casos con el fin de que los clip's den la función **tag.clip** sean cuadrados. La segunda forma de hacerlo es:

- Declaramos una variable con la medida en pixeles de las dimensiones de los clp's, ejemplo:

Variables:

- Dependiendo del **Template Type**, por ejemplo un **Template Type: Char**, debemos escribir en la celda de texto **loop**, así:

loop =

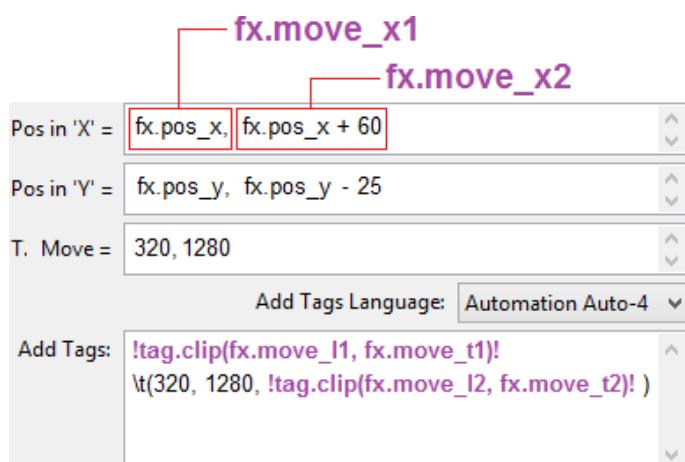
Si por ejemplo, un carácter mide 70 pixeles de ancho por 40 de alto, tendríamos:

- $40/\text{pix} = 40/10 = 4$
- $70/\text{pix} = 70/10 = 7$
- $4 \times 7 = 28$

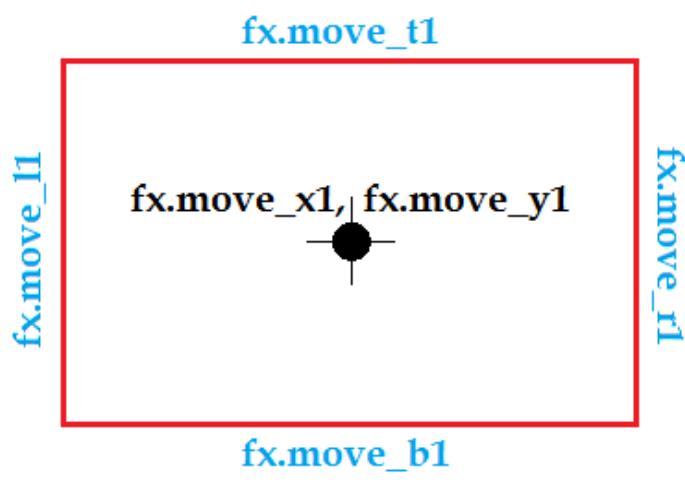
O sea que obtendríamos 28 clip's de  $10 \times 10$  pixeles de la función **tag.clip**, que a su vez sería la cantidad del **loop**, es decir que **maxj = 28**

A menor tamaño de los clip's, mayor será la cantidad del loop, y a mayor cantidad de clip's, mayor será la cantidad de recursos consumidos por la memoria RAM y la PC se hará mucho más lenta.

A continuación veremos cómo mover los clip's generados por la función **tag.clip** en el **Kara Effector**, usando algunas cosas que ya hemos aprendido hasta el momento.



**fx.move\_l1** es la coordenada de la posición izquierda de **fx.move\_x1**, y **fx.move\_t1** es la superior de **fx.move\_y1**:



## Kara Effector - Effector Book [Tomo X]:

Las dimensiones del rectángulo anterior dependerán del **Template Type**, por ejemplo, si es tipo Translation Word, entonces las dimensiones serán las de cada una de cada Palabra de cada Línea.

Desde y hacia dónde se mueve un clip es decisión de cada uno, según un efecto así lo requiera, lo más importante es saber cómo hacerlo y el poder contar con una función como **tag.clip** que nos facilita un poco esa labor.

---

**tag.iclip( left, top, width, height, mode ):** similar a **tag.clip**, pero con la leve diferencia que no hace clip's sino iclip's.

Para los que aún no están familiarizados con los iclip's y en qué consisten, recordemos qué son y para qué se usan.

- **clip:** es un rectángulo con posición y dimensiones específicas que hace visible únicamente a todo lo que esté dentro de dicho rectángulo. Ejemplo:



O sea que todo lo que está dentro del clip es lo que veremos y todo lo que esté por fuera de él quedará totalmente invisible.

- **iclip:** es un rectángulo con posición y dimensiones específicas que hace visible únicamente a todo lo que esté por fuera de dicho rectángulo. Ejemplo:



O sea que todo lo que está por fuera del iclip es lo que veremos y todo lo que esté por dentro de él quedará totalmente invisible.

La elección entre el clip y el iclip dependerá del efecto.

---

**tag.clip2( left, top, width, height ):** esta función es también similar a las función **tag.clip**, pero con la gran diferencia que siempre genera el mismo clip sin importar el

---

loop, es decir que no genera clip's verticales, horizontales ni reticulares, sino que siempre genera el mismo clip con las mismas dimensiones, es por eso que no necesita el parámetro **mode**.

---

**tag.iclip2( left, top, width, height ):** es similar a **tag.clip2**, pero con la diferencia que no genera clip's sino iclip's y es la última función de la librería “**tag**” que emplea clip's rectangulares por medio de coordenadas.

---

A continuación veremos dos funciones más basadas en **clip's**, pero esta vez no serán rectangulares sino basadas en **shapes**, es decir que están enfocadas en la figuras que dibujamos en el **AssDraw3**.

**tag.movevc( shape, x, y, Dx, Dy, t\_i, t\_f ):** similar a la función **tag.clip** con la diferencia que el clip que genera es una **shape**. Todos los parámetros de esta función, excepto el primero (**shape**), pueden tener valores por default. Veamos qué son y en qué consisten:

- **x:** coordenada con respecto al eje “x” que hace referencia a la posición en donde estará el centro de la **shape**, su valor por default es **fx.move\_x1**
- **y:** coordenada con respecto al eje “y” que hace referencia a la posición en donde estará el centro de la **shape**, su valor por default es **fx.move\_y1**
- **Dx:** cantidad de desplazamiento en pixeles con respecto al centro de la **shape**, con referencia al eje “x”. Su valor por default es **fx.move\_x2 - fx.move\_x1**

De no existir un **fx.move\_x2**, recordemos que su valor por default es **fx.move\_x1**, por lo que el valor por default del parámetro **Dx**, en este caso, sería: **fx.move\_x1 - fx.move\_x1 = 0**

- **Dy:** cantidad de desplazamiento en pixeles con respecto al centro de la **shape**, con referencia al eje “y”. Su valor por default es **fx.move\_y2 - fx.move\_y1**

De no existir un **fx.move\_y2**, recordemos que su valor por default es **fx.move\_y1**, por lo que el valor por default del parámetro **Dy**, en este caso, sería: **fx.move\_y1 - fx.move\_y1 = 0**

## Kara Effector - Effector Book [Tomo X]:

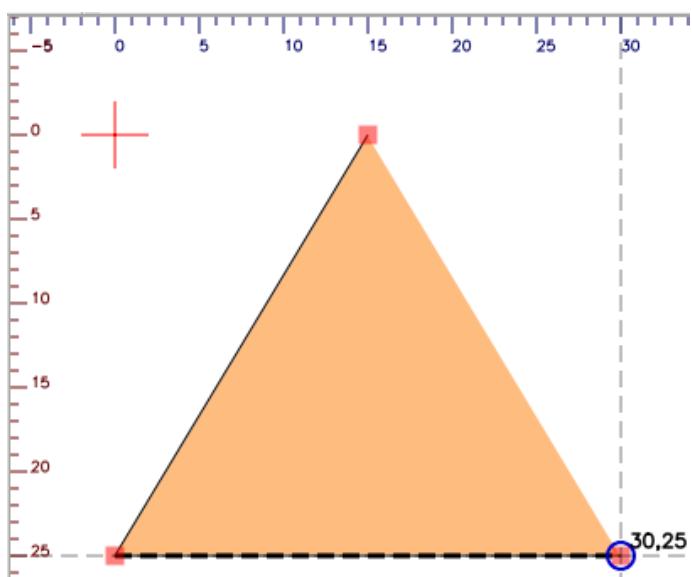
- **t\_i:** es el tiempo en que iniciará el movimiento del clip, en caso que decidamos que se mueva. Su valor por default es **fx.movet\_i**

De no existir un **fx.movet\_i**, recordemos que su valor por default es 0, por lo que en este caso el valor por default de **t\_i** sería 0.

- **t\_f:** es el tiempo en que finalizará el movimiento del clip, en caso que decidamos que se mueva. Su valor por default es **fx.movet\_f**

De no existir un **fx.movet\_f**, recordemos que su valor por default es **fx.dur**, por lo que en este caso el valor por default de **t\_f** sería **fx.dur**

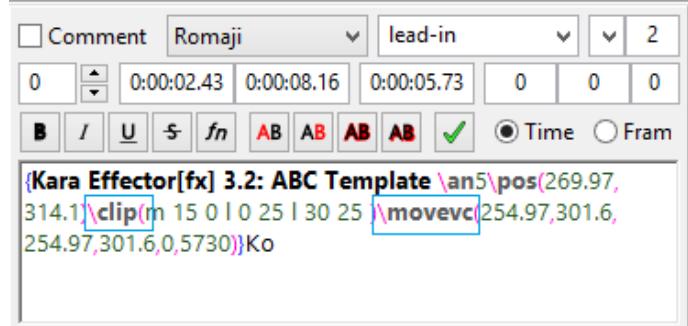
Para el siguiente ejemplo usará un simple triángulo hecho en el **AssDraw3**, como lo podemos ver en esta imagen:



El triángulo tiene 30 pixeles de ancho por 25 de alto. El siguiente paso es copiar el código de esa **shape** y pegarlo dentro de la función **tag.movevc** y dejaremos el resto de los parámetros por default. He usado un **Template Type: Syl**, pero pueden usar cualquiera de los de la lista:

Pos in 'X' =	fx.pos_x
Pos in 'Y' =	fx.pos_y
T. Move =	
Add Tags:	Add Tags Language: Lua
tag.movevc('m 15 0   0 25   30 25')	

La función **tag.movevc** siempre retorna dos tags, un **\clip** que contiene dentro de sí a la **shape** y un **\movevc** que le da la posición al **clip** y lo mueve si ese fuere el caso:



Como el tag de posición es un **\pos** entonces el **\movevc** solo posiciona al clip. Ahora veremos cómo queda el clip del triángulo en cada una de las Sílabas de la Línea:



Como el triángulo es mucho más pequeño que el tamaño de cada una de las sílabas, entonces éstas no alcanzan a ser totalmente visibles:



En cuanto el clip y el objeto karaoke al que afecta (ya sea una sílaba, una línea, carácter, palabra, shape o demás) se muevan al mismo tiempo, desde el mismo punto de inicio y hacia el mismo punto final; lo recomendable es dejar el resto de los parámetros de la función **tag.movevc**, por default. Ejemplo:

Hacemos un **\move** con posiciones iniciales random:

Pos in 'X' =	fx.pos_x + R(-30,30), fx.pos_x
Pos in 'Y' =	fx.pos_y + R(-40,40), fx.pos_y
T. Move =	0, 360

## Kara Effector - Effector Book [Tomo X]:

Usaremos la función **tag.movevc** con la misma shape y el resto de los parámetros por default:

```
Add Tags: Add Tags Language: Lua
tag.movevc('m 15 0 10 25 130 25')
```

De este ejemplo tendríamos que:

- $x = fx.\text{pos\_x} + R(-30,30)$
- $y = fx.\text{pos\_y} + R(-40,40)$
- $Dx = fx.\text{pos\_x} - fx.\text{pos\_x} + R(-30,30) = R(-30,30)$
- $Dy = fx.\text{pos\_y} - fx.\text{pos\_y} + R(-40,40) = R(-40,40)$
- $t\_i = 0$
- $t\_f = 360$

Y el resultado sería:



Que luego de 360 ms las sílabas y los clip's quedarán en su posición natural:



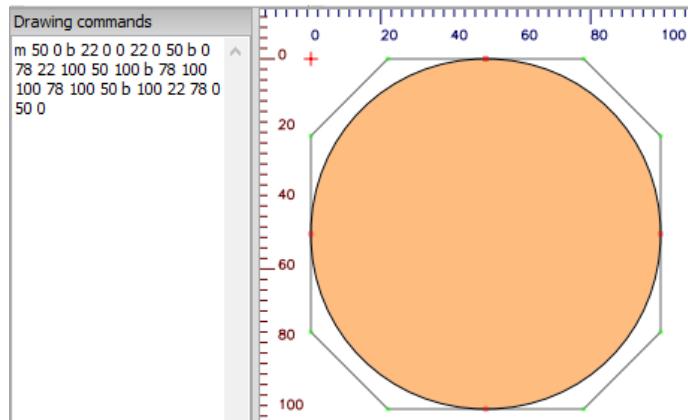
El tag **\movevc** solo es aplicable con el filtro **VSFilterMod**, de otro modo no se verá el efecto. El **\movevc** solo funciona de la mano de los tags **\pos** y **\move**, pero no lo hace con los tags **\moves3**, **\moves4** y **\mover**. En pocas palabras, el tag **\movevc** solo puede mover a un clip en línea recta, sin importar la dirección.

Ahora veremos un ejemplo de objeto estático y clip móvil, es decir, en donde tengamos que usar el resto de los parámetros de la función **tag.movevc**:

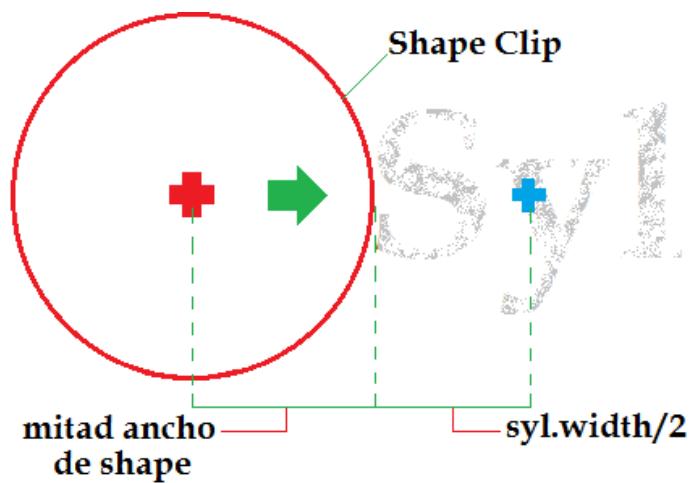
En un **Template Type: Syl**, le dejamos las posiciones por default que equivalen a **syl.center** y **syl.middle**:

Pos in 'X' =	fx.\text{pos\_x}
Pos in 'Y' =	fx.\text{pos\_y}

La **shape** que usaremos para el clip es un círculo de 100 px de ancho, dicha **shape** hace parte de las librerías del **Kara Effector** y se llama: **shape.circle**



Y la posición inicial del clip será justo al lado izquierdo de cada sílaba, como muestra la siguiente imagen:



Dicha posición inicial de clip sería:

- $x = fx.\text{pos\_x} - syl.\text{width}/2 - 50, fx.\text{pos\_y}$

Y la posición final del clip será:

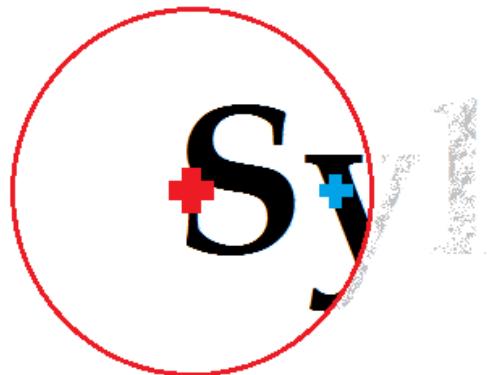
- $x = fx.\text{pos\_x}, fx.\text{pos\_y}$

```
Add Tags: Add Tags Language: Lua
tag.movevc(shape.circle, fx.\text{pos\_x} - syl.\text{width}/2 - 50, fx.\text{pos\_y}, syl.\text{width}/2 + 50, 0)
```

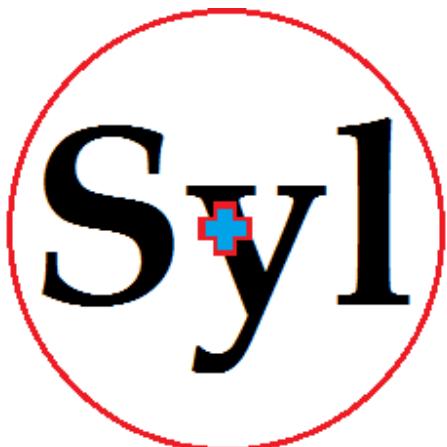
- **shape = shape.circle**
- **x = fx.\text{pos\\_x} - syl.\text{width}/2 - 50**
- **y = fx.\text{pos\\_y}**
- **Dx = syl.\text{width}/2 + 50**
- **Dy = 0**

## Kara Effector - Effector Book [Tomo X]:

Entonces, el círculo que hace de clip irá avanzando desde donde estaba inicialmente hasta que su centro coincida con el de la sílaba:



Y cuando se cumpla el tiempo total de la línea fx, la shape y la sílaba tendrán el mismo centro:



La función **tag.movevc** nos será de gran ayuda para hacer efectos con clip's de alto nivel, ya que de otra forma sería muy complicado lograr el control en cuanto a posición y movimiento cuando de clip's se trata.

---

**tag.movevc( shape, x, y, Dx, Dy, t\_i, t\_f )**: similar a **tag.movevc**, pero con la diferencia que no retorna un clip, sino un iclip.

---

**tag.only( condition, exit\_t, exit\_f )**: esta función retorna alguno de los dos parámetros (**exit\_t** o **exit\_f**) según el valor de verdad del parámetro **condition**. Si el valor de verdad de **condition** es verdadero (**true**), retorna al parámetro **exit\_t**, si es falso (**false**) retorna **exit\_f**.

Veamos los tipos de condiciones que podemos usar en la función **tag.only**:

- `==` igual a
- `~=` diferente a
- `<` menor que
- `>` mayor que
- `<=` menor o igual que
- `>=` mayor o igual que

Ejemplo:

```
Add Tags:          Add Tags Language: Lua
tag.only( word.i == 3, '\\1c&H0000FF&', '\\1c&FFFFFF&' )
```

Esto quiere decir que, si la Palabra en la línea es la número tres, su color primario será **Rojo** ('\\1c&H0000FF&'), pero si dicha condición es falsa, su color primario será **Blanco** ('\\1c&FFFFFF&'):



En la función **tag.only** solo el tercer parámetro (**exit\_f**) puede tener un valor por default, y este valor depende del tipo de objeto que sea el segundo parámetro (**exit\_t**). Si **exit\_t** es un **string**, entonces el valor por default de **exit\_f** es vacío (""), y si **exit\_t** es un **número**, el valor por default de **exit\_f** es cero (0). Ejemplos:

- **tag.only( syl.i <= 5, "\\\blur3" )**  
Como **exit\_t** es un **string** ("\\blur3"), entonces el valor por default de **exit\_f** será vacío. Esto quiere decir que, si la sílaba es una de las cinco primeras de la línea, la función retornará "\\blur3", de lo contrario no retornará nada.
- **i.end\_time + tag.only( char.i == char.n, 1200 )**  
Como **exit\_t** es un **número** (1200) entonces el valor por default de **exit\_f** será cero (0). Esto quiere decir que, si el carácter es el último de la línea (char.n), se sumarán 1200 ms al tiempo final de la línea, de lo contrario se le sumará cero.

## Kara Effector - Effector Book [Tomo X]:

Los seis tipos de condiciones vistos anteriormente tienen un único valor de verdad, ya sea verdadero (**true**) o falso (**false**), pero no puede tener ambos valores al mismo tiempo. Estas seis condiciones son conocidas como **condiciones simples** y al combinar dos o más de ellas se crean las **condiciones compuestas**.

### Condiciones simples:

- == igual a
- ~= diferente a
- < menor que
- > mayor que
- <= menor o igual que
- >= mayor o igual que

Y para obtener las **condiciones compuestas** es necesario usar **conectores**, la **conjunción (and)** o la **disyunción (or)**. Cada una de las dos mencionadas también tiene un único valor de verdad que dependerá del valor de verdad de cada una de las condiciones que la conforman.

### Tabla de verdad de la **conjunción**:

Caso	p	q	p and q
1	V	V	V
2	V	F	F
3	F	V	F
4	F	F	F

### Tabla de verdad de la **disyunción**:

Caso	p	q	p or q
1	V	V	V
2	V	F	V
3	F	V	V
4	F	F	F

### Ejemplos:

- ( syl.i >= 7 ) **and** ( syl.dur < 300 )
- ( R(2) == 1 ) **or** ( char.width > 80 )
- ( syl.text ~= "ke" ) **and** ( syl.i < 5 )
- ( word.n < 10 ) **or** (line.width > 600 )

Las posibles combinaciones son muchas, así que el tener claro qué es lo que queremos en nuestro efectos nos ayudará a decidirnos por alguna de ellas.

A continuación están las posibles **combinaciones** entre solo dos **condiciones** por medio de alguno de los dos **conectores (and u or)**:

Caso	Cond. 1	Conector	Cond.2
1	==	<b>and // or</b>	==
2	==	<b>and // or</b>	~=
3	==	<b>and // or</b>	<
4	==	<b>and // or</b>	>
5	==	<b>and // or</b>	<=
6	==	<b>and // or</b>	>=
7	~=	<b>and // or</b>	~=
8	~=	<b>and // or</b>	<
9	~=	<b>and // or</b>	>
10	~=	<b>and // or</b>	<=
11	~=	<b>and // or</b>	>=
12	<	<b>and // or</b>	<
13	<	<b>and // or</b>	>
14	<	<b>and // or</b>	<=
15	<	<b>and // or</b>	>=
16	>	<b>and // or</b>	>
17	>	<b>and // or</b>	<=
18	>	<b>and // or</b>	>=
19	<=	<b>and // or</b>	<=
20	<=	<b>and // or</b>	>=
21	>=	<b>and // or</b>	>=

21 posibles combinaciones, pero no se preocupen, no es necesario memorizarlas todas, ya que el solo hecho de saber el valor de verdad de los **conectores (and u or)** es más que suficiente y el resto es solo cuestión de aplicar un poco de lógica al asunto.

Con el final de la función **tag.only** se da por terminado el **Tomo X**, pero no la librería “tag”, ya que aún nos resta por ver unas cuantas funciones más de ella.

En el **Tomo XI** del **Kara Effector** continuaremos con el resto de la Librería “tag”. Intenten poner en práctica todos los ejemplos vistos en este **Tomo** 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, exponer alguna duda o hacer alguna sugerencia. También pueden visitarnos en nuestra página de **Facebook**: [www.facebook.com/karaeffector](http://www.facebook.com/karaeffector)

# Kara Effector 3.2: Effector Book Vol. I [Tomo XI]

---

## Kara Effector 3.2:

Este **Tomo XI** es la continuación de las funciones de la librería “**tag**”. No hay mucho más que se pueda decir acerca de esta librería, sino que como todas las vistas en los Tomos del **Kara Effector**, es muy importante conocerla y saber qué tipo de ayuda nos puede ofrecer cada una de las funciones que contiene.

### Librería **tag** [KE]:

**tag.set( times, events )**: esta función asume que todas las líneas habilitadas para aplicarle un Efecto son una sola y luego aplica una transformación de un 1 ms de duración de cada uno de los elementos de la **tabla “events”**, según los tiempos de la **tabla “times”**.

Las tablas “**times**” y “**events**” pueden ser ingresadas directamente en la función o declararlas a modo de variables en la celda de texto “**Variables**”. Veamos un ejemplo de los dos casos:

```
Add Tags: Add Tags Language: Lua
tag.set({\"0:00:10.00\", \"0:00:20.000\"}, {\"\\1c#00FF00\", \"\\1c#FF00FF\"})
```

No está de más decir que ambas tablas deben tener la misma cantidad de elementos. En este ejemplo las tablas fueron ingresadas directamente.

Los tiempos de la tabla “**times**” son copiados de la parte inferior del vídeo, en el Aegisub y posteriormente pegados entre comillas, no importando si son sencillas o dobles.

## Kara Effect - Effector Book [Tomo XI]:

Los tags en la **tabla “events”** también se deben escribir entre comillas, así como se puede apreciar en la anterior imagen. Como la función `tag.set` retorna transformaciones de los tags que están en la **tabla “events”**, entonces éstos deben ser tags que puedan ser “**animados**” por el tag “`\t`”, por ejemplo: `\bord`, `\blur`, `\3c`, `\alpha`, `\1a`, `\jitter`, `\fsvp`, `\clip`, `\clip`, `\fscx`, `\fsc`, `\fscy`, `\fsp`, `\shad`, entre otros.

Ahora veamos algunos ejemplos de tags que no pueden ser “**animados**” por el tag “`\t`”: `\pos`, `\move`, `\org`, `\moves3`, `\moves4`, `\movevc`, `\mover`, `\fad`, `\t`, entre otros.

Lo que hará la función en este ejemplo será que a los 10 s (“**0:00:10.000**”) contados desde el inicio del video (cero absoluto), convertirá al color primario (“`\1c`”) de su color por default a Verde (“#00FF00”) está en formato **HTML**, pero no es obligación que esté en este formato, también se podría hacer en formato **.ass**, o sea “**&H00FF00&**”):

#	L	Start	End	Style	Text
1	1	0:00:02.43	0:00:08.16	Romaji	*Ko*do*ku *na *ho*ho *wo *
2	1	0:00:08.33	0:00:13.19	Romaji	*Yo*a*ke *no *ke*ha*i *ga *
3	1	0:00:13.54	0:00:18.39	Romaji	*Wa*a*shi *wo *so*ra *e *
4	1	0:00:18.67	0:00:27.22	Romaji	*Ki*bo*u *ga *ka*na*ta *de *
5	1	0:00:28.52	0:00:34.30	Romaji	*Ma*yo*i *na*ga*ra *mo *ki *

En la imagen, como la línea 1 va desde los 2.43 s hasta los 8.16 s, entonces no se verá afectada por la función y el color primario de ésta será el que ya tiene por default, que en este ejemplo es Blanco:

### Kodoku na hoho wo nurasu nurasu kedo

Pero la línea 2, como va desde los 8.33 s hasta los 13.19 s, sí se verá afectada, a partir de los 10 s. O sea que el color primario de la línea 2 será Blanco desde que inicia (8.33 s) hasta los 10 s:

### Yoake no kehai ga shizuka ni michite

Y justo cuando el video llegue a los 10 s, el color primario cambiará de forma automática a Verde:

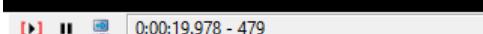
### Yoake no kehai ga shizuka ni michite

Como la línea 3 inicia en 13.54 s y finaliza en 18.39 s, su color primario será siempre el Verde:

### Watashi wo sora e maneku yo

En el caso de la línea 4 pasará algo similar a la línea 2, ya que su tiempo de inicio está en 18.67 s, entonces su color primario iniciará siendo Verde, pero al llegar a los 20 s (“**0:00:20.000**”) cambiará a Lila (“#FF00FF” en **HTML**), ya que su tiempo final está en 27.22 s:

### Kibou ga kanata de matteru sou da yo iku yo



### Kibou ga kanata de matteru sou da yo iku yo



Y desde la línea 4 en adelante, el color primario será siempre Lila, ya que eso es lo que dicta la función.

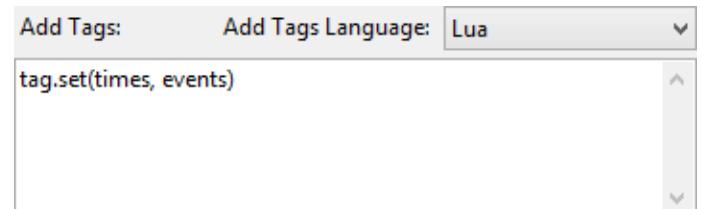
Si queremos declarar las variables de las dos tablas de la función (“**times**” y “**events**”), haríamos algo como:

Variables:

```
times = {"0:00:10.00", "0:00:20.000"}; events = {"\"1c#00FF00", "\"1c#FF00FF"}
```

Importante  
usar ";" en vez de ","

Y dentro de la función, en **Add Tags**:



Y sin importar cuál de los dos métodos usemos, los resultados serán los mismos. Es decir, que los tags que coloquemos en la **tabla “events”** sucederán de forma inmediata, según los tiempos que hayamos registrado en la **tabla “times”**. Esta función es ideal para hacer que nuestros efectos hagan cosas puntuales a medida que en el video al que le hacemos un karaoke, sucedan cambios que nos llamen la atención, como un cambio de color o de escena, cambios de estilos por un personaje y demás.

## Kara Effector - Effector Book [Tomo XI]:

El **lead-in** de la línea que se ve en la imagen, incluye una **shape** de Plumas para imitar el estilo de las plumas que salen en el vídeo:



La función **tag.set** nos serviría para hacer que las plumas de nuestro efecto desaparezcan exactamente en el mismo momento que lo hacen las del vídeo, haciendo algo como:

```
tag.set( {acá el tiempo exacto}, {"\alpha&HFF&"} )
```

Entonces en ese preciso momento las plumas quedarán invisibles gracias al tag \alpha. Como se podrán imaginar, las posibilidades son prácticamente infinitas y a gusto de cada uno de nosotros.

Un tag en la **tabla "events"** no necesariamente debe ser uno solo, pueden ser varios:

```
events = {"\fscxy125\3c&H000000&\blur4", "\shad2"}
```

Si queremos que una o más de las transformaciones no suceda de forma inmediata (ya que por default cada una de la transformaciones en esta función tarda solo 1 ms), el **Kara Effector** nos da dos opciones para ello:

- **Opción 1:** duración de la transformación en ms. Ej:

```
times = { "0:00:10.000", {"0:00:20.000", 460} }
```

Esto hará que la segunda transformación ya no dure 1 ms, sino que ahora tardará 460 ms.

- **Opción 2:** tiempo final de la transformación. Ej:

```
times = { "0:00:10.000", {"0:00:20.000", "0:00:21.810"} }
```

Esto hará que la segunda transformación ya no dure 1 ms, sino que ahora tardará 1810 ms, ya que:

$$0:00:21.810 - 0:00:20.000 = 1810 \text{ ms}$$

De lo anterior es fácil deducir que como la duración por default de cada una de las transformaciones de la función **tag.set** es 1 ms, se vería de la siguiente forma, ejemplo:

```
times = { {"0:00:10.000", 1}, {"0:00:20.000", 1} }
```

Con este método haremos que una transformación tarde exactamente el tiempo que necesitemos y no siempre 1 ms como lo hace por default. Recuerden que la cantidad de tiempos en la **tabla "times"** es ilimitada, y que por cada uno de esos tiempos debe haber un tag o serie de tags que le correspondan, para que las transformaciones sean posibles y la función no nos arroje un error.

**tag.glitter( time, add\_i, add\_f):** esta función hace transformaciones al azar en un tiempo determinado “time” que consisten en cambios bruscos de las dimensiones del objeto karaoke. Los parámetros “add\_i” y “add\_f” son opcionales.

El parámetro “time” puede ser un tiempo en ms, lo que hará que las transformaciones al azar se hagan en el lapso de tiempo entre 0 y dicho valor. Ejemplo:

```
tag.glitter(1200)
```

La otra forma que puede tener el parámetro “time” es en forma de **tabla**, en donde el primer elemento de la misma sea el tiempo en ms del inicio y el segundo elemento sea el tiempo final. Ejemplo:

```
tag.glitter({400, 2300})
```

O sea que las transformaciones al azar sucederán entre los 400 ms y los 2300 ms. Pongamos a prueba este mismo ejemplo:

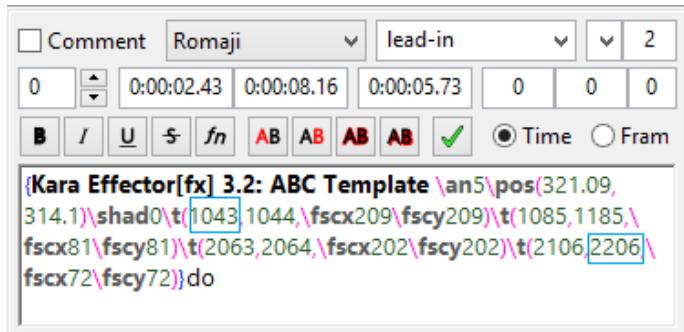
Add Tags: Add Tags Language: Lua

```
tag.glitter({400, 2300})
```

**Ko**do ku na **hho** wo nurasu nurasU ke do  
こどくな ほほ を ぬらす ぬらす けど

## Kara Effector - Effector Book [Tomo XI]:

Y verificamos si las transformaciones se realizaron en el rango de 400 a 2300 ms:

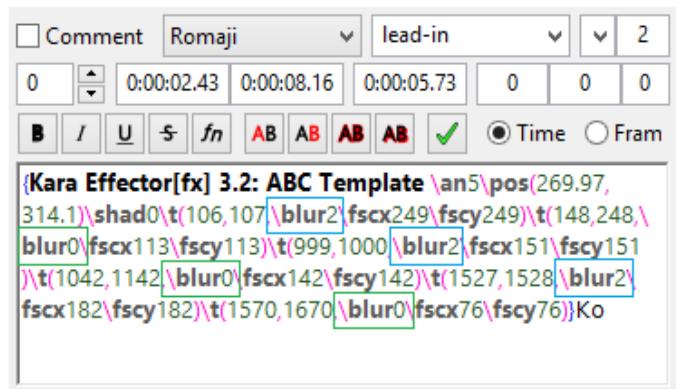


El valor por default de “time” es **fx.dur**, o sea, la duración total de cada una de las líneas fx:

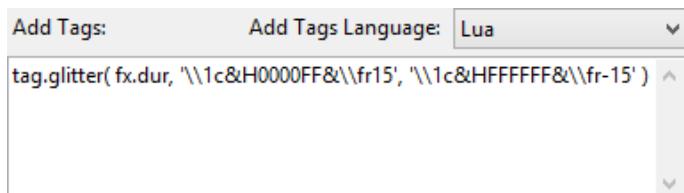
**tag.glitter( ) = tag.glitter( fx.dur )**

Las transformaciones que retorna la función **tag.glitter** vienen en cantidades pares. El parámetro “**add\_i**” es un tag o una serie de ellos que a la postre se agregarán a las transformaciones impares retornadas. De forma similar, el parámetro “**add\_f**” se agregará a las transformaciones pares retornadas. Ejemplo:

**tag.glitter( {0,1700}, “\\blur2”, “\\blur0” )**



El truco radica en que los tags de “**add\_i**” sean opuestos a los de “**add\_f**” para que una transformación “contradiga” a la otra. Si por ejemplo en “**add\_i**” colocamos un tag \1c, entonces debemos usar en “**add\_f**” otro tag \1c, pero con un color distinto, ejemplo:



Del anterior ejemplo, notamos cómo los dos colores son distintos (Rojo y Blanco), y cómo los ángulos son opuestos entre sí (\fr15 y \fr-15).

La función **tag.glitter** es muy útil para asemejar efectos de brillo en **shapes** pequeñas, ya que asemeja un efecto de “titileo” como el de las estrellas en el firmamento. Así que la pueden poner a prueba con la **shape** de una estrella con un tamaño en pixeles entre 5 y 10.

---

**tag.oscill( time, delay, ... )**: esta función genera transformaciones de duración “**delay**” en un lapso de tiempo “**time**”.

El parámetro “**time**” puede ser un valor en ms, lo que hará que las transformaciones se ejecuten desde cero hasta dicho valor, ejemplo:

- **time** = 1000
- **delay** = 200

Estos dos valores hacen que la función **tag.oscill** retorne cinco transformaciones:

1. de **0** a 200 ← Duración = 200
2. de 200 a 400 ← Duración = 200
3. de 400 a 600 ← Duración = 200
4. de 600 a 800 ← Duración = 200
5. de 800 a **1000** ← Duración = 200

El parámetro “**time**” también puede ser una **tabla**, en donde el primer elemento es el tiempo inicial y el segundo elemento es el tiempo final, ejemplo:

- **time** = {400, 1120}
- **delay** = 180

Estos dos valores hacen que la función retorne cuatro transformaciones:

1. de **400** a 580 ← Duración = 180
2. de 580 a 760 ← Duración = 180
3. de 760 a 940 ← Duración = 180
4. de 940 a **1120** ← Duración = 180

Entonces, el parámetro “**time**” puede ser tanto un **valor** numérico, como una **tabla** con dos valores que limiten el rango de tiempo en el cual se ejecutará la función.

## Kara Effector - Effector Book [Tomo XI]:

---

El parámetro “**delay**” tiene las dos mismas cualidades del parámetro “**time**”, de poder ser tanto un **valor** numérico como una **tabla** de valores.

Para un “**delay**” con valor numérico, nos sirve un ejemplo similar a los dos anteriores:

- **time** = 300
- **delay** = 100

Lo que generará tres transformaciones:

1. de 0 a 100      ← Duración = 100
2. de 100 a 200    ← Duración = 100
3. de 200 a 300    ← Duración = 100

cuando “**delay**” es una **tabla**, ampliamos las posibilidades de manipular más a la función **tag.oscill** y a los resultados que nos ofrece. Recordemos que una transformación se basa en el tag **\t**, y uno de los modos de uso del tag **\t** es:

**\t( t1, t2, accel, \...**

En donde el parámetro “**accel**” es la aceleración de la transformación, cuyo valor por default es 1. Si queremos modificar la aceleración en las transformaciones de la función **tag.oscill**, lo que debemos hacer es especificarla en el segundo elemento de la tabla “**delay**”, ejemplo:

- **time** = 450
- **delay** = {150, 0.8}

Lo que generará tres transformaciones, pero con la aceleración de 0.8:

1. **\t( 0, 150, 0.8, \...**      ← Duración = 150
2. **\t( 150, 300, 0.8, \...**    ← Duración = 150
3. **\t( 300, 450, 0.8, \...**    ← Duración = 150

Entonces decimos que el primer elemento de la **tabla “delay”** será la duración de cada transformación, y el segundo elemento equivale a la aceleración “**accel**” de las transformaciones.

El “**delay**” como **tabla** nos da otra tercera posibilidad, que es el “dilatar” la duración de cada transformación que se genere. El valor de dicha dilatación es el tercer elemento de la **tabla “delay”**, y este valor puede ser positivo, lo que hará que cada transformación dure más tiempo que la inmediatamente anterior; o si el valor es negativo, hará

---

que cada transformación dure cada vez menos tiempo que la transformación inmediatamente anterior. Ejemplo:

- **time** = {500, 1110}
- **delay** = {100, 1.2, 10}

o sea que:

- **Duración**      = 100
- **Aceleración**    = 1.2
- **Dilatación**    = 10

Este ejemplo generará cinco transformaciones con las siguientes características:

1. **\t( 500, 600, 1.2, \...**      ← Duración = 100
2. **\t( 600, 710, 1.2, \...**    ← Duración = 110
3. **\t( 710, 830, 1.2, \...**    ← Duración = 120
4. **\t( 830, 960, 1.2, \...**    ← Duración = 130
5. **\t( 960, 1100, 1.2, \...**    ← Duración = 140

Es notorio cómo cada transformación dura 10 ms más que la transformación inmediatamente anterior, ya que ese fue el valor asignado como “dilatación”.

Hasta este punto, en la función **tag.oscill**, el “**delay**” no solo decide la duración de cada transformación, sino también su frecuencia. Si por ejemplo tenemos un “**delay**” de 200 ms, lo que significaría que la duración de las transformaciones generadas será de 200 ms, y que cada transformación empezará 200 ms después de haber iniciado la transformación inmediatamente anterior.

Para modificar la frecuencia de las transformaciones generadas, el “**delay**” en modo de **tabla** también nos da esa posibilidad. Ejemplo:

- **time** = 500
- **delay** = {{100, 25}, 1}

Notamos que el primer elemento de la **tabla “delay”** es otra **tabla**, en donde el primer elemento de esta otra **tabla** (100) marca la **frecuencia**, y el segundo (25), marca la **duración** de las transformaciones:

1. **\t( 0, 25, 1, \...**      ← Duración = 25
2. **\t( 100, 125, 1, \...**    ← Duración = 25
3. **\t( 200, 225, 1, \...**    ← Duración = 25
4. **\t( 300, 325, 1, \...**    ← Duración = 25

## Kara Effector - Effector Book [Tomo XI]:

5. \t( 400, 425, 1, \...      ←Duración = 25

Y en las cinco transformaciones vemos que cada una de ellas empieza a 100 ms después de haber iniciado la transformación inmediatamente anterior (dado que 100 ms es la frecuencia asignada) y, la duración total de cada transformación es de 25 ms.

En resumen, el parámetro “**delay**” en la función **tag.oscill** tiene los siguientes cuatro modos:

1. **delay** = dur
2. **delay** = { dur, accel }
3. **delay** = { dur, accel, dilatation }
4. **delay** = { { frequency, dur }, accel, dilatation }

Los valores por default de las anteriores variables son de la siguiente forma:

Modo	frequency	accel	dilatation
1	dur	1	0
2	dur		0
3	dur		0
4		1	0

Lo que en resumen sería:

**delay** = dur = { { dur, dur }, 1, 0 }

El tercer parámetro de la función **tag.oscill** pone ( ... ), ya sabemos que los tres puntos seguidos hacen referencia a una cantidad indeterminada de parámetros, que para este caso son los tags que necesitamos que se **alternen** en las transformaciones.

Como el objetivo de la función **tag.oscill** es alternar tags, se debería usar por lo menos con dos de ellos. De ahí en adelante podemos usar la cantidad de tags que deseemos. Ejemplo:

**tag.oscill( 1000, 200, "\\\blur1", "\\\blur3", "\\\blur5" )**

Obtendríamos las siguientes transformaciones:

1. \t( 0, 200, \blur1 )      ←Duración = 200
2. \t( 200, 400, \blur3 )      ←Duración = 200
3. \t( 400, 600, \blur5 )      ←Duración = 200
4. \t( 600, 800, \blur1 )      ←Duración = 200
5. \t( 800, 1000, \blur3 )      ←Duración = 200

Los tres tags ingresados en la función se alternaron en las transformaciones, a manera de ciclo. Este procedimiento será el mismo sin importar la cantidad de tags, lo que hace que esta función tenga muchas utilidades, como alternar cambios de tamaños, de colores, de blur, de ángulos o lo que nos podamos imaginar.

Los tags a alternar se pueden ingresar en la función como lo hecho en el anterior ejemplo, pero también hay otra forma de hacerlo, y es en forma de **tabla**. Ejemplo:

Definimos nuestra **tabla** con los tags a alternar, que para este ejemplo, es una tabla de tres colores primarios:

Variables:  
colores = {"\\1c&H1D1EF0&", "\\1c&HCB8422&", "\\1c&HD803F3&"}

Y en **Add Tags** ponemos:

Add Tags:      Add Tags Language:      Lua  
tag.oscill(fx.dur, 300, colores)

Lo que haría que esos tres colores primarios se alternen cada 300 ms durante la duración total de cada línea de fx.

La última opción que nos da la función **tag.oscill** es poder decidir cuál será el primer elemento de la **tabla** de tags, con el que empezará la primera transformación entre todas las retornadas. Ejemplo:

Primero declaramos la siguiente **tabla**, con un Template Type: **Translation Word**

Variables:  
colores = table.make("color", word.n, 30, 90, "\\1c")

Recordemos que la función **table.make** crea una **tabla**, en este caso de colores, de word.n de tamaño, con colores equidistantes entre los ángulos 30° y 90° con un tag \1c.

## Kara Effector - Effector Book [Tomo XI]:

Ahora usamos la **tabla** creada en la función **tag.oscill** de la siguiente manera:

```
Add Tags:          Add Tags Language: Lua
tag.oscill(fx.dur, 240, colores)
```

Así la primera transformación de todas las generadas en cada palabra (**word**) empezarán con el mismo color:

**Kodoku na hoho wo nurasu nurasu keto**  
こどくな ほほ を ぬらす ぬらす けど

Y la opción que ya había mencionado, de decidir con cuál tag empezarán las transformaciones es:

```
Add Tags:          Add Tags Language: Lua
tag.oscill({0, fx.dur, word.i}, 240, colores)
```

- **time = { 0, fx.dur, word.i }**

El tercer elemento de la **tabla “time”** es el que nos ayuda a decidir el primer tag con el que empezará la primera transformación. Para este ejemplo es **word.i**

**Kodoku na hoho wo nurasu nurasu keto**  
こどくな ほほ を ぬらす ぬらす けど

Hecho esto, el tag de la primera transformación para la primera palabra, será el pimer color de la **tabla “colores”** ya que **word.i = 1**; para la segunda palabra será el color 2 de la **tabla**, porque **word.i = 2**; y así sucesivamente con las demás palabras.

**tag.ipol(valor\_i, valor\_f, index\_ipol )**: esta función interpola los parámetros **“valor\_i”** y **“valor\_f”** teniendo como referencia al parámetro interpolador **“index\_ipol”**. Retorna el valor que deseemos entre todos los que existan entre **“valor\_i”** y **“valor\_f”**, inclusive ellos mismos.

Los parámetros **“valor\_i”** y **“valor\_f”** pueden ser colores, transparencias (alpha) o números reales y ambos deben ser del mismo tipo. Y el parámetro **“index\_ipol”** es un número real entre 0 y 1, ya que si es menor que 0 la función lo tomará como 0, si es mayor que 1, la función lo tomará como 1, y su valor por default es 0.5 para que retorne el valor promedio entre **“valor\_i”** y **“valor\_f”**.

Ejemplos:

- **tag.ipol( "&HFFFFFF&", "&H000000&", 0.7 )**
- **tag.ipol( "&HFF&", "&HAA&", j/maxj )**
- **tag.ipol( 200, 100, char.i/char.n )**
- **tag.ipol( text.color3, "&H00FFFF&" )**

Y con la función **tag.ipol** se da por terminada la librería **“tag”** y seguimos avanzando en el estudio de recursos del **Kara Effector** y profundizando cada vez más en el mundo de los Karaokes.

Intenten poner en práctica todos los ejemplos vistos en este **Tomo** 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, exponer alguna duda o hacer alguna sugerencia. También pueden visitarnos y dejar su comentario en nuestra página de **Facebook**: [www.facebook.com/karaeffector](http://www.facebook.com/karaeffector)

# Kara Effector 3.2: Effector Book Vol. I [Tomo XIII]

---

# Kara Effector 3.2:

El **Tomo XII** arrancará con las librerías “color” y “alpha” del **Kara Effector**, ya que sus funciones son muy similares entre sí.

## Librerías Color y Alpha (KE):

Son dos librerías con funciones dedicadas a los colores y a las transparencias (alpha). Algunas de ellas nos servirán para hacer Efectos directamente y otras como apoyo para los mismos.

**color.to\_RGB( color\_or\_table )**: retorna los valores de la cantidad de Rojo, Verde y Azul (Red, Green and Blue) en base decimal, de un color asignado. El formato del color ingresado puede ser .ass o **HTML**

La función retorna los valores en una **tabla**. Ejemplo:

```
color.to_RGB("&HF4E67A") = { 122, 230, 244 }
```

- **122** (Rojo “7A”)
- **230** (Verde “E6”)
- **244** (Azul “F4”)

Y para el caso que le ingresemos una **tabla** de colores, la función retornará una **tabla**, en donde cada uno de sus elementos son a su vez otra **tabla**, que contienen los valores de Rojo, Verde y Azul de cada uno de los colores:

```
colores = { "&HF108B4&", "&H30F304&" }  
color.to_RGB( colores )  
= { {180, 8, 241}, {4, 243, 48} }
```

---

**color.to\_HVS( color\_or\_table )**: esta función es similar a **color.to\_RGB**, pero con la diferencia que retorna los valores en base decimal del Tono, la Saturación y del Valor (Hue, Saturation and Value). La forma de usar esta función es la misma que la de **color.to\_RGB**

---

## Kara Effector - Effector Book [Tomo XII]:

**color.vc( color\_or\_table ):** convierte a un color o a cada uno de los colores de una **tabla**, en formato “**vc**”, que es el formato de los colores en el **VSFilterMod**.

Ejemplo 1:

- **color.vc(“&HFF00A3&”)**
- **= (FF00A3, FF00A3, FF00A3, FF00A3)**

Ejemplo 2:

- **colores = { “&HFFFFFF&”, “H000000&” }**
- **color.vc( colores )**
- **= {FFFFF,FFFFF,FFFFF,FFFFF},  
(000000,000000,000000,000000) }**

Es decir, que si ingresamos en la función un solo color, se retornará ese mismo color en formato “**vc**”. Si ingresamos una **tabla** de colores, retornará una **tabla** con cada uno de los colores en formato “**vc**”.

**alpha.va( alpha\_or\_table ):** hace exactamente lo mismo que la función **color.vc**, pero con las transparencias (alpha). Ejemplo:

- **alpha.va(“&HFF&”) = (FF, FF, FF, FF)**

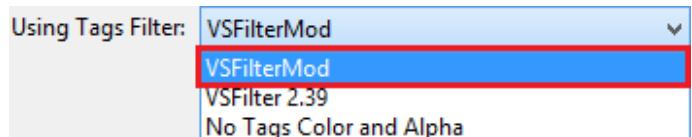
**color.r( ):** retorna un color al azar en formato .ass entre todos los del espectro, incluidos el Blanco, el Negro y todos los matices entre ellos y los demás colores.

**alpha.r( ):** retorna una transparencia (alpha) al azar en formato .ass, entre 0 (totalmente visible) y 255 (invisible).

**color.rc( ):** retorna un color al azar en formato “**vc**” entre todos los del espectro. El random se ejecuta de manera diferente para cada uno de los cuatro colores que conformarán al color final. Ejemplo:

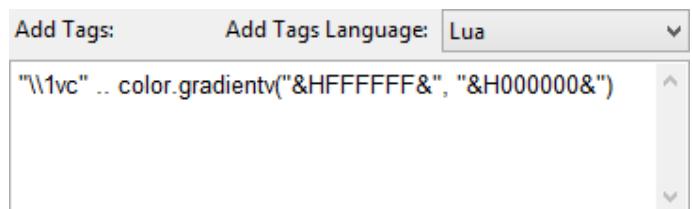


**color.gradientv( color\_top, color\_bottom):** esta función es de uso exclusivo del filtro **VSFilterMod**, ya que retorna un tag “**vc**” (vector color), y para ello es necesario que la opción de dicho filtro esté seleccionada en la primera ventana del **Kara Effector**:

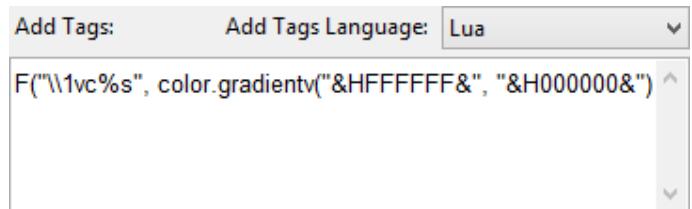


Esta función hace un **gradiente** (degradado) vertical entre dos colores asignados, **color\_top** y **color\_bottom**.

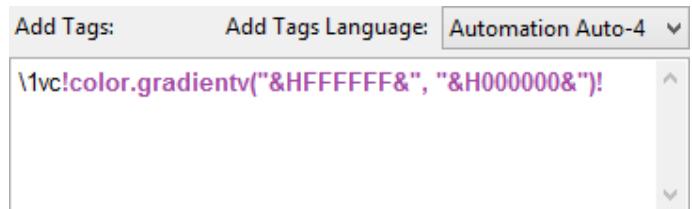
Un ejemplo simple en lenguaje **LUA** sería:



La anterior imagen muestra el método de concatenado en lenguaje **LUA**, pero recordemos que también lo podemos hacer usando la función **string.format** vista muchas veces anteriormente:



Y este mismo ejemplo en lenguaje **Automation-auto4** es:

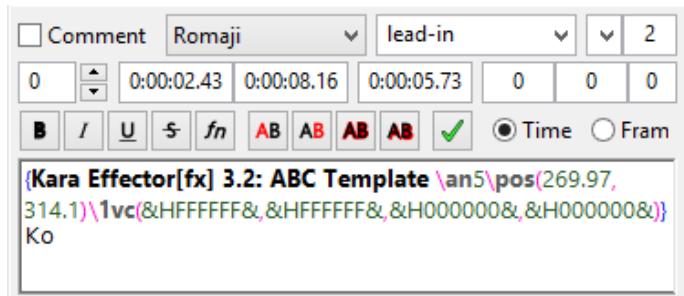


Cualquiera de los tres anteriores métodos es válido para hacer Efectos en el **Kara Effector**. Del anterior ejemplo, en pantalla veremos el **gradiente** entre el Blanco y el Negro:



## Kara Effector - Effector Book [Tomo XII]:

En la siguiente imagen vemos el tag “vc” que hace posible el **gradiente** (\1vc):



En el ejemplo anterior vimos cómo usar la función con ambos parámetros “string” (en este caso, colores), pero ambos parámetros también pueden ser **tablas** o en caso de ser necesario, también combinaciones entre un **string** y una **tabla**:

color.gradientv( color_top, color_bottom )			
Caso	color_top	color_bottom	return
1	string	string	string
2	string	tabla	tabla
3	tabla	string	tabla
4	tabla	tabla	tabla

**Caso 1:** al usar la función con string y string, entonces se retorna un **string** correspondiente al gradiente vertical de los dos strings ingresados.

**Caso 2:** retorna una **tabla** de gradientes entre el string del primer parámetro y cada uno de los elementos de la tabla ingresada.

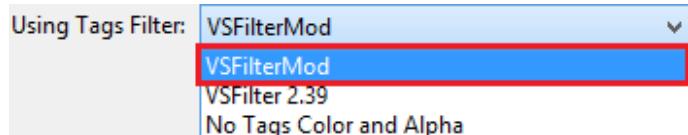
**Caso 3:** retorna una **tabla** de gradientes entre cada uno de los elementos de la tabla ingresada y el string del segundo parámetro.

**Caso 4:** retorna una **tabla** de gradientes entre las posibles combinaciones de los productos cartesianos entre los elementos de ambas tablas ingresadas.

**alpha.gradientv( alpha\_top, alpha\_bottom ):** hace exactamente lo mismo que la función **color.gradientv**, pero con las transparencias (alpha). Ejemplo:

- **alpha.gradientv(“&HFF&”, “&H44&”)**  
= (&HFF&, &HFF&, &H44&, &H44&)

**color.gradienth( color\_left, color\_right ):** esta función es de uso exclusivo del filtro **VSFilterMod**, ya que retorna un tag “vc” (vector color), y para ello es necesario que la opción de dicho filtro esté seleccionada en la ventana de inicio del **Kara Effector**:

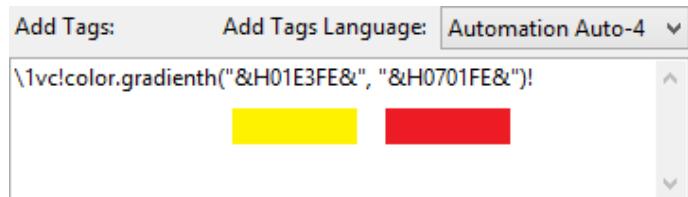


Esta función hace un **gradiente** (degradado) horizontal entre dos colores asignados, **color\_left** y **color\_right** (color izquierdo y color derecho).

Las opciones de uso de esta función son los mismos que los de la función **color.gradientv**:

color.gradienth( color_left, color_right )			
Caso	color_left	color_right	return
1	string	string	string
2	string	tabla	tabla
3	tabla	string	tabla
4	tabla	tabla	tabla

Veamos un ejemplo de cómo usar esta función:



**Yoake no kehai ga shizuka ni michite**  
よあけ の けはい が しづか に みちて

**alpha.gradienth( alpha\_left, alpha\_right ):** hace exactamente lo mismo que la función **color.gradienth**, pero con las transparencias (alpha). Ejemplo:

- **alpha.gradienth(“&HAA&”, “&H00&”)**  
= (&HAA&, &H00&, &HAA&, &H00&)

Las anteriores cuatro funciones guardan relación cercana con los gradientes, las dos primeras con los verticales y las dos siguientes con los horizontales.

## Kara Effector - Effector Book [Tomo XII]:

**color.vc\_to\_c( color\_vc )**: esta función convierte al color ingresado, de formato “vc” (**VSFilterMod**) a formato “c” (**VSFilter 2.39**). El parámetro **color\_vc** puede ser un **string** o una **tabla de string**, de manera que si es un **string**, la función retornará a ese color cambiado de un formato al otro, y si es una **tabla**, retornará una **tabla** con cada uno de los colores de la misma, convertidos en formato “c”.

color.vc_to_c( color_vc )		
Caso	color_vc	return
1	string	string
2	tabla	tabla

**alpha.va\_to\_a( alpha\_va )**: hace lo mismo que la función **color.vc\_to\_c**, pero con las transparencias (alpha).

**color.c\_to\_vc( color\_c )**: es la función opuesta a **color\_c\_to\_vc**. Convierte al color ingresado, de formato “c” (**VSFilter 2.39**) a formato “vc” (**VSFilterMod**).

Los modos son los mismos que los de su función opuesta:

color.c_to_vc( color_c )		
Caso	color_c	return
1	string	string
2	tabla	tabla

**alpha.a\_to\_va( alpha\_a )**: hace lo mismo que la función **color.c\_to\_vc**, pero con las transparencias (alpha).

**color.interpolate( color1, color2, index\_ipol )**: es similar a la función **\_G.interpolate\_color** del **Automation Auto-4**, pero con la diferencia que tanto **color1** como **color2** pueden ser o un **string** o una **tabla**.

El parámetro **index\_ipol** es opcional y es un número real entre 0 y 1 inclusive. Su valor por default es 0.5 con el fin de que la función retorne el color intermedio entre **color1** y **color2**. Si **index\_ipol** es cero o cercano a él, la función retorna un color cercano a **color1**, y si es 1 o cercano a 1,

retornará un color cercano al de parámetro **color2**, o sea que el color que se retornará dependerá únicamente de este valor (**index\_ipol**).

Los modos de usar esta función son los siguientes:

color.interpolate( color1, color2, index_ipol )			
Caso	color1	color2	return
1	string	string	string
2	string	tabla	tabla
3	tabla	string	tabla
4	tabla	tabla	tabla

**Caso 1**: al usar la función con **string** y **string**, entonces se retorna un **string** correspondiente a la interpolación entre los dos colores, dependiendo del valor de **index\_ipol**, o justo el color intermedio, en el caso de que no exista dicho parámetro.

**Caso 2**: retorna una **tabla** con las interpolaciones entre el color del primer parámetro y cada uno de los colores de la tabla ingresada en el segundo parámetro.

**Caso 3**: retorna una **tabla** con las interpolaciones entre cada uno de los colores de la tabla ingresada en el primer parámetro y el color del segundo parámetro.

**Caso 4**: retorna una **tabla** con las interpolaciones entre las posibles combinaciones de los productos cartesianos entre los colores de ambas tablas ingresadas.

**alpha.interpolate( alpha1, alpha2, index\_ipol )**: similar a la función **\_G.interpolate\_alpha** del **Automation Auto-4**. Hace lo mismo que la función **color.interpolate**, pero con las transparencias (alpha).

**color.delay( time\_i, delay, color\_i, color\_f, ... )**: es una función que convierte progresivamente al parámetro **color\_i** (color inicial) al parámetro **color\_f** (color final). Es exclusiva del filtro **VSFilterMod** ya que retorna un color en formato “vc” seguido de cuatro transformaciones que contienen colores también en formato “vc”. Las tres primeras transformaciones son colores al azar entre **color\_i** y **color\_f** y la última contiene a **color\_f**.

## Kara Effector - Effector Book [Tomo XII]:

El parámetro **time\_i** es el tiempo relativo al tiempo de inicio de la línea fx y corresponde al momento en que iniciarán las transformaciones.

El parámetro **delay** es la duración total de las cuatro transformaciones que retorna la función.

El parámetro ( ... ) hace referencia a uno o más tags de colores "vc" que eventualmente retornará la función. Las posibilidades serían:

- "\\\1vc"
- "\\\3vc"
- "\\\4vc"
- "\\\1vc", "\\\3vc"
- "\\\1vc", "\\\4vc"
- "\\\3vc", "\\\4vc"
- "\\\1vc", "\\\3vc", "\\\4vc"

Ejemplos:

- **color.delay**(0, 600, "&HFFFFFF&", "&H000000&", "\\\1vc")
- **color.delay**(1200, 2000, "&HFF00FFF&", "&H00FF00&", "\\\1vc", "\\\3vc")
- **color.delay**(500, 1000, "&HFF0000&", "&H00FFFF&", "\\\3vc")

```
Add Tags:          Add Tags Language: Lua
color.delay(0, 400, "&H0000FF&", "&HFFFFFF&", "\\\1vc")
```

**alpha.delay( time\_i, delay, alpha\_i, alpha\_f, ... )**: hace lo mismo que la función **color.delay**, pero con las transparencias (alpha).

**color.movedelay( dur, delay, mode, ... )**: es similar a la función **color.delay**, pero con la diferencia que la cantidad de transformaciones que retorna depende del parámetro **dur**, que es la duración total de la función. La otra diferencia es que en esta función se pueden ingresar la cantidad de colores que deseemos o también podemos poner como cuarto parámetro a una tabla de colores.

El parámetro **dur** es la duración total de todas las transformaciones que retorna la función.

El parámetro **delay** es la duración que tendrá cada una de las transformaciones retornadas por la función.

El parámetro **mode** es un número entero que indica a cuál o a cuáles tags de colores "vc" se aplicará la función. Pueden ser:

- 1 = \\\1vc
- 3 = \\\3vc
- 4 = \\\4vc
- 13 = \\\1vc\\\3vc
- 14 = \\\1vc\\\4vc
- 34 = \\\3vc\\\4vc
- 134 = \\\1vc\\\3vc\\\4vc

Si se quiere, el parámetro **mode** también se puede usar como un **string** que indique el o los tags a los que queremos que se aplique la función, ejemplo:

**mode** = "\\\1vc\\\3vc"

El parámetro ( ... ) hace referencia a los colores ingresados. Puede ser:

- Un solo color
- Una serie de colores (separados por comas):

```
Add Tags:          Add Tags Language: Lua
color.movedelay(fx.dur, 200, 1, "&H0000FF&", "&HFFFFFF&", "&H000000&", "&H00FFFF&")
```

- Una tabla de colores:

```
Variables:
mis_colores = {"&H0000FF&", "&H00FFFF&", "&HFF00FF&"}
```

```
Add Tags:          Add Tags Language: Lua
color.movedelay(fx.dur, 500, 3, mis_colores)
```

## Kara Effector - Effector Book [Tomo XII]:

**color.set( times, colors, ... )**: esta función retorna una o más transformaciones de colores de la tabla **colors** con respecto a los tiempos de la tabla **times**.

Todos los tiempos de la tabla **times** están medidos con referencia al cero absoluto del vídeo y a cada uno de los tiempos de esta tabla le debe corresponder un color de la tabla **colors**.

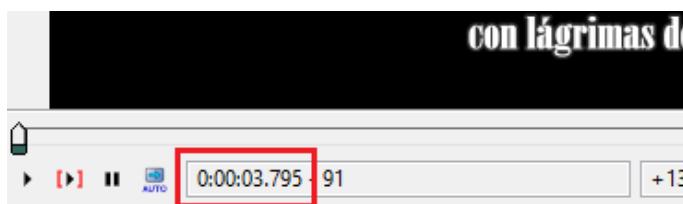
El parámetro ( ...) hace referencia al tag o tags de colores a los que afectarán las transformaciones que retornará la función. Las posibilidades serían:

- "\\\1vc"
- "\\\3vc"
- "\\\4vc"
- "\\\1vc", "\\\3vc"
- "\\\1vc", "\\\4vc"
- "\\\3vc", "\\\4vc"
- "\\\1vc", "\\\3vc", "\\\4vc"

Dado que esta función no es exclusiva de ninguno de los filtros que se pueden usar en el **Aegisub**, entonces las posibilidades de extenderían a:

- "\\\1c"
- "\\\3c"
- "\\\4c"
- "\\\1c", "\\\3c"
- "\\\1c", "\\\4c"
- "\\\3c", "\\\4c"
- "\\\1c", "\\\3c", "\\\4c"

Cada uno de los tiempos de la tabla del parámetro **times** deben ser copiados de la parte inferior izquierda del vídeo en el **Aegisub**. Dichos tiempos están en formato **HMSms** (Horas, Minutos, Segundo y milisegundos). Ejemplo:



Los tiempos deben ser pegados en la tabla **times** usando comillas simples o dobles para que el **Kara Effector** pueda reconocerlos a cada uno de ellos como un **string**.

Esta sería una opción para definir los dos parámetros en la celda de texto “**Variables**”:

Variables:

```
times = {"0:00:03.795", "0:00:10.552"};
colors = {"&H00FFFF&", "&HFF00FF"}
```

Notamos cómo ambas tablas tienen la misma cantidad de elementos. La cantidad total de transformaciones que la función retornará depende de cada uno de nosotros, puede ser mínimo un elemento y máximo hasta donde lleguemos a necesitar.

Al definir las tablas **times** y **colors** en “**Variables**” (el nombre con que se definen las tablas es decisión de cada uno), usaremos la función en **Add Tags**, de alguna de las siguientes dos maneras:

1. Indicando el tipo de variable que será cada una de las dos tablas:

Add Tags: Add Tags Language: Lua

```
color.set( var.line.times, var.line.colors, "\\\1c" )
```

2. Como usamos punto y coma ( ; ) para separar las tablas al definirlas en “**Variables**”, entonces las podemos llamar con tan solo escribir el nombre con que las definimos:

Add Tags: Add Tags Language: Lua

```
color.set( times, colors, "\\\1c" )
```

Otra forma de usar la función es ingresando directamente las tablas dentro de ella, sin necesidad de definirla:

Add Tags: Add Tags Language: Lua

```
color.set( {"0:00:03.795", "0:00:10.552"}, {"&H00FFFF&", "&HFF00FF"}, "\\\3c" )
```

## Kara Effector - Effector Book [Tomo XII]:

Cuando algún tiempo de la tabla **times** está definido como un **string** (como en los ejemplos anteriores), entonces la transformación correspondiente a ese tiempo tendrá su duración por default que es de 1 ms. Es decir que el cambio de un color a otro será instantáneo.

Si queremos que alguna transformación tenga una duración superior a la que tiene por default (1 ms), hay dos opciones posibles para poderlo hacer.

1. Convertir a dicho tiempo en otra **tabla** con dos elementos, y dentro de ella poner los tiempos de inicio y final de la transformación, ejemplo:

### Variables:

```
times = {"0:00:03.795", {"0:00:10.552", "0:00:11.052"}};
colors = {"&H00FFFF&", "&HFF00FF"}
```

En este caso, la transformación tendrá una duración de 500 ms que es la diferencia entre los dos tiempos de la tabla del segundo elemento de la tabla **times**. Esta opción tiene la ventaja de poner los dos tiempos sin la necesidad de calcular previamente la duración de la transformación.

2. Convertir a dicho tiempo en otra **tabla** con dos elementos, y dentro de ella poner el tiempo de inicio de la transformación y la duración en ms de la misma, ejemplo:

### Variables:

```
times = {"0:00:03.795", {"0:00:10.552", 360}};
colors = {"&H00FFFF&", "&HFF00FF"}
```

Haciéndolo de este modo, la transformación tendrá una duración de 360 ms, que es el valor que pone el segundo elemento de la tabla correspondiente al segundo tiempo de la tabla **times**. La ventaja de esta opción es poner la duración de la transformación sin poner el tiempo final.

Los anteriores dos métodos son válidos para cualquiera o para todos los elementos de la tabla **times**, y como ya lo había mencionado, son usados para modificar la duración por default de las transformaciones que se retornarán al usar la función.

Los colores de la tabla **colors** pueden estar todos en formato del filtro **VSFilter 2.39**, del **VSFilterMod** o una combinación de ambos, si así se quiere. Ejemplo:

### Variables:

```
times = {"0:00:03.795", "0:00:10.552"};
colors = { ("&H00FFFF&,&H00FFFF&,&HAA00D8&,
&HAA00D8&)", "&HFF00FF&" };
```

Del ejemplo anterior vemos cómo el primer color está en formato “vc” y el segundo está en formato normal. Es obvio que para que los colores salgan tal cual como los pusimos en formato “vc”, la opción del filtro **VSFilterMod** debe estar seleccionada en la ventana de inicio del **Kara Effector**.

Un tercer formato que puede tener uno o más colores de la tabla **colors** es el de una **tabla** de colores. Ejemplo:

### Variables:

```
times = { "0:00:03.795", "0:00:10.552" };
colors = { {"&H00FFFF&", "&HF5900F&", "&HA62090&"},
           "&HFF00FF&" }
```

En el anterior ejemplo, el primer elemento de la tabla **colors** es una tabla de 3 colores (amarillo, azul y morado), pero la cantidad puede ser la que nosotros convengamos. La función la usaríamos en **Add Tags** del mismo modo ya aprendido hasta este momento:

Add Tags: Add Tags Language: Lua  
color.set(times, colors, "\\"1c")

En la siguiente imagen vemos el texto justo antes de que ocurra la primera transformación:



Y lo que sucederá en la primera transformación, dado que el primer elemento de la tabla **colors** es una tabla de colores y que el ejemplo se está haciendo en el modo

## Kara Effector - Effector Book [Tomo XII]:

**Template Type: Syl**, es que a cada sílaba le corresponderá un color de esa tabla; pero como dicha tabla solo tiene 3 colores, entonces a la cuarta sílaba se le volverá a asignar el color 1 de la tabla, lo mismo que a las séptima sílaba y así sucesivamente hasta completar todas las sílabas:



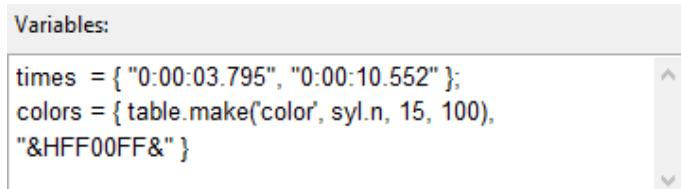
En la próxima imagen veremos el texto justo antes de la tercera transformación:



Y así se verá el texto justo después del cambio de color, ya que el segundo elemento de la tabla **colors** era el color magenta (&HFF00FF&):



Esta tercera habilidad que tienen los elementos de la tabla de **colors** de poder ser una tabla de colores nos da muchas posibilidades, como hacer “**máscaras**” o **degradaciones** en una o más de las transformaciones que retorna la función. Ejemplo:



El primer elemento de la tabla **colors** está determinado por la función **table.make** así:

- El modo “**color**” creará una tabla de colores.
- **syl.n** determinará el tamaño de dicha tabla.
- 15 y 100 serán los límites inferior y superior de la degradación con respecto al círculo cromático de la teoría del color **HSV**. Los 15° corresponden un tono naranja rojizo de dicho círculo, y los 100° a un tono verde limón.

De este modo, la función hará que el texto cambie a la degradación anteriormente descrita:



Y para la segunda transformación el texto pasará al color que se haya asignado como segundo elemento en la tabla **colors** (magenta = &HFF00FF&):



Todo es cuestión de poner a volar un poco la imaginación y empezar a experimentar con las distintas opciones que nos ofrece esta función, y las posibilidades son casi que infinitas al igual que los resultados.

**alpha.set( times, alphas, ... )**: esta función hace lo exactamente lo mismo que la función **color.set**, pero usa las transparencias (alpha) en vez de colores.

Es todo por ahora, pero las librerías **color** y **alpha** aún continuarán en el **Tomo XIII**. Intenten poner en práctica todos los ejemplos vistos en este **Tomo** 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, exponer alguna duda o hacer alguna sugerencia. También pueden visitarnos y dejar su comentario en nuestra página de **Facebook**:

[www.facebook.com/karaeffector](http://www.facebook.com/karaeffector)

Kara Effector - Effector Book [Tomo XIII]:

# Kara Effector 3.2: Effector Book Vol. I [Tomo XIII]

# Kara Effector 3.2:

El **Tomo XIII** es la continuación de las librerías “color” y “alpha” del **Kara Effector** que se iniciaron en el **Tomo** anterior, en el cual ya vimos numerosas funciones y aún restan muchas más por ver.

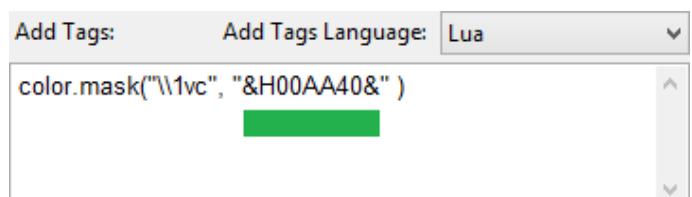
## Librerías Color y Alpha (KE):

**color.mask( mode, color, color\_mask )**: esta función retorna un color para cada uno de los elementos de una línea de texto (word, syl, char y demás) a manera de “máscara”.

El parámetro **mode** hace referencia al único tag de color en formato “vc” en el que se realizará la “máscara”, es decir:

- “\\1vc”
- “\\3vc”
- “\\4vc”

El parámetro **color** puede ser, tanto un **string** de color como una **tabla** de colores. Ejemplo:



```
color.mask("\\1vc", "&H00AA40&" )
```

En este ejemplo, el parámetro **color\_mask** no está ya que es opcional, y el parámetro **color** es un **string**, un color en tono de verde.

## Kara Effector - Effector Book [Tomo XIII]:

El texto se vería de la siguiente forma:

**Surechigau ishiki te ga fureta yo ne**  
すれちがう いしき てが ふれた よね

Este ejemplo está hecho con un **Template Type: Syl**, y podemos notar cómo cada una de las sílabas tiene el mismo todo de verde, pero mezclado en ciertas partes con tonos entre el blanco y el negro.

Si remplazamos a cada sílaba por un rectángulo con sus mismas dimensiones, se podrá apreciar más claramente el efecto de la “máscara”:



La función **color.mask** hace por default la “máscara” con el color ingresado en el parámetro **color** y los tonos al azar entre el blanco y el negro.

En el caso en que el parámetro **color** sea una **tabla** de colores, la función retornará una **tabla** con la “máscara” de cada uno de los colores de dicha tabla.

El parámetro **color\_mask**, al igual que el parámetro **color**, también puede ser tanto un color **string** como una **tabla** de colores. Ejemplos:

- **color\_mask: string**

Add Tags: Add Tags Language: Lua  
`color.mask("\\1vc", "&H00AA40&", "&H00FFFF&" )`

De este modo, la función ya no hace la “máscara” con los tonos por default entre el blanco y el negro, sino que la hace entre el parámetro **color** y el color ingresado en el parámetro **color\_mask** (verde y amarillo):

**Yoake no kehai ga shizuka ni michite**  
よあけの けはい が しずか に みちて

Así que de esta forma uno decide un solo color con el que el parámetro **color** hará la “máscara”.

- **color\_mask: table**

Add Tags: Add Tags Language: Lua  
`color.mask("\\1vc", "&H00AA40&", { "&H00FFFF&", "&H0000FF&" })`

Si empleamos al parámetro **mask\_color** como una **tabla** (en este ejemplo, una de dos colores: amarillo y rojo), la función hará la “máscara” entre el color principal que es el parámetro **color**, y tonos al azar entre todos los colores de la tabla de colores **mask\_color**:

**Mayoi nagara mo kimi wo sagasu tabi**  
まよい ながら も きみ を さがす たび

Resumiendo los modos, tenemos:

color.mask ( mode, color, color_mask )			
Caso	color	color_mask	return
1	string	string	string
2	string	tabla	string
3	tabla	string	tabla
4	tabla	tabla	tabla

---

**alpha.mask( mode, alpha )**: es similar a la función **color.mask**, con la diferencia que esta función carece del tercer parámetro, y que trabaja con las transparencias (alpha) en lugar de colores.

---

**color.movemask( dur, delay, mode, color )**: esta función es similar a **color.mask**, ya que también genera una “máscara”, pero ésta da la sensación de movimiento gracias a una serie de transformaciones. Dicho efecto de movimiento es realizado de derecha a izquierda.

El parámetro **dur** es la duración total de la función, o sea la duración total de todas las transformaciones. Puede ser un **número** o una **tabla** con dos números, el primero de ellos indicaría el inicio de la función y el segundo el tiempo final, ambos tiempos son relativos al tiempo de inicio de cada una de las líneas fx generadas.

## Kara Effector - Effector Book [Tomo XIII]:

Las opciones del parámetro **dur** son:

1. dur
2. {time1, time2}

El parámetro **delay** cumple con la misma tarea que su parámetro homónimo en la función **tag.oscill**, y estas son sus opciones:

1. delay
2. {{delay, dur\_delay}}
3. {delay, accel}
4. {{delay, dur\_delay}, accel}
5. {delay, accel, dilat}
6. {{delay, dur\_delay}, accel, dilat}

Todas las anteriores variables mencionadas para este parámetro hacen referencia a valores numéricos, es decir, a números.

- **delay**: valor numérico que indica cada cuánto suceden las transformaciones generadas.
- **dur\_delay**: valor numérico que indica la duración de cada una de las transformaciones generadas. Su valor por default es **delay**.
- **accel**: valor numérico que indica la aceleración de las transformaciones generadas. Su valor por default es 1.
- **dilat**: valor numérico que indica el tiempo que se va extendiendo cada una de las transformaciones con respecto a la anterior, o sea que extiende la duración. Su valor por default es 0.

El parámetro **mode** hace referencia al tag de color en formato “**vc**” que se retornará en las transformaciones generadas por la función:

1. “\\1vc”
2. “\\3vc”
3. “\\4vc”

El parámetro **color** hace referencia a un **string** de color. A diferencia de la función **color.mask**, acá este parámetro ya no puede ser una **tabla**. Ejemplo:

```
Add Tags:          Add Tags Language: Lua
color.movemask( fx.dur, 200, "\\1vc", "&HFF8D00&" )
```

Del anterior ejemplo tenemos:

- **dur** = fx.dur
- **delay** = 200
- **mode** = “\\1vc”
- **color** = “&HFF8D00”



Las “máscara” se genera igual que con **color.mask**, pero ahora se mueve de derecha a izquierda con la duración de cada transformación en 200 ms.

**alpha.movemask( dur, delay, mode, alpha )**: es similar a la función **color.movemask**, con la diferencia que trabaja con las transparencias (alpha) en lugar de colores.

**color.setmovemask( delay, mode, times, colors )**: esta función es la combinación de la función **color.set** y la función **color.movemask**, sus parámetros son:

- **delay**: es un valor numérico que hace referencia a la duración de cada una de las transformaciones, al igual, también indica cada cuánto suceden las mismas.
- **mode**: hace referencia al tag de color en formato “**vc**” al que afectará la función:
  1. “\\1vc”
  2. “\\3vc”
  3. “\\4vc”
- **times**: hace referencia a la **tabla** con los tiempos copiados de la parte inferior izquierda del vídeo en el **Aegisub**, pero todos los elementos de la tabla deben ser un string, ya no pueden ser una tabla para modificar el tiempo del cambio de un color a otro.
- **colors**: hace referencia a la **tabla** de colores que hacen pareja con cada uno de los tiempos de la tabla **times**, pero esta vez solo pueden un **string** de color, ya no pueden ser tablas.

Entonces es fácil deducir lo que esta función hace.

## Kara Effector - Effector Book [Tomo XIII]:

Ejemplo:

Variables:

```
times = { "0:00:03.795", "0:00:10.552" };
colors = { "&H00FFFF&", "&HA62090&" }
```

Y en Add Tags:

Add Tags:      Add Tags Language: **Lua**

```
color.setmovemask( 200, "\\\1vc", times, colors )
```

Entonces, justo antes de la primera transformación se verá algo como esto:



Luego la primera transformación:



**color.movemaskv( dur, delay, mode, color, color\_mask ):** similar a la función **color.movemask**, pero ahora la “máscara” no se moverá de derecha a izquierda, sino de forma vertical, de abajo a arriba. La otra diferencia está en que el parámetro **color** ya no puede ser una **tabla**, solo está habilitado para ser un **string** de color.

El resto de las cualidades y características son todas las mismas, y el uso entre una función y otra, solo depende de los resultados que queremos obtener.

**alpha.movemaskv( dur, delay, mode, alpha ):** similar a la función **color.movemaskv**, pero carece del quinto parámetro y que trabaja con las transparencias (alpha) en vez de colores.

**color.masktable( color ):** esta función crea una **tabla** con los colores necesarios para hacer una “máscara” dependiendo del **Template Type**, ya que éste determina el tamaño de la **tabla**.

El parámetro **color** puede ser tanto un **string** de color, como una **tabla** de colores. Para el primer caso, retorna una **tabla** con la “máscara” de dicho color y para el caso en que el parámetro **color** sea una **tabla**, la función retorna una tabla de tablas, es decir, una **tabla** en donde cada uno de sus elementos es otra tabla, y cada una de ellas contiene la “máscara” correspondiente a cada color de la **tabla** ingresada.

**alpha.masktable( alpha ):** similar a la función **color.masktable**, pero con la diferencia que trabaja con las transparencias (alpha) en vez de colores.

**color.module( color1, color2 ):** esta función retorna la interpolación completa entre los colores de los parámetros **color1** y **color2**, con respecto al **loop**.

Los parámetros **color1** y **color2** pueden ser tanto strings de colores, como tablas de colores:

color.module ( color1, color2 )			
Caso	color1	color2	return
1	string	string	string
2	string	tabla	tabla
3	tabla	string	tabla
4	tabla	tabla	tabla

**Caso 1:** al usar la función con ambos parámetros **strings**, entonces se retorna un **string** de color correspondiente a la interpolación entre los dos colores. Se puede decir que es la forma más convencional de usar esta función.

**Caso 2:** retorna una **tabla** con las interpolaciones entre el color del primer parámetro y cada uno de los colores de la tabla ingresada en el segundo parámetro.

**Caso 3:** retorna una **tabla** con las interpolaciones entre cada uno de los colores de la tabla ingresada en el primer parámetro y el color del segundo parámetro.

## Kara Effector - Effector Book [Tomo XIII]:

Caso 4: retorna una **tabla** con las interpolaciones entre las posibles combinaciones de los productos cartesianos entre los colores de ambas tablas ingresadas.

Recordemos que todas las interpolaciones en esta función dependen únicamente del **Loop (maxj)**. Ejemplo:

- **Template Type: Line**

Template Type [fx]: Line

Center in 'X' = line.center

Center in 'Y' = line.middle

- Configuramos a **Return [fx]**, **loop** y **Size**

Return [fx]: shape.circle

loop = 32

Size = 20

- En **Pos in "X"** y **Pos in "Y"** usaremos la función **math.polar** para que los 32 círculos se ubiquen equidistantemente respecto al centro de cada línea karaoke, con un radio de 120 pixeles:

Pos in 'X' = fx.pos\_x + math.polar(360\*j/maxj, 120, 'x')

Pos in 'Y' = fx.pos\_y + math.polar(360\*j/maxj, 120, 'y')

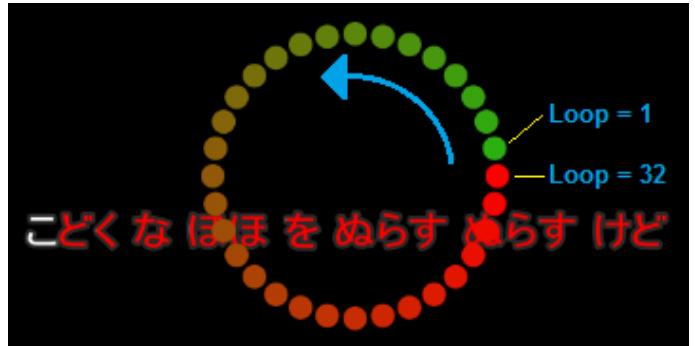
- Por último, en **Add Tags** llamamos a la función de la siguiente forma (a esta altura ya deben estar familiarizados con los métodos de concatenación entre dos **string**):

Add Tags: Add Tags Language: Lua

"\\1vc" .. color.module("&H12B02D&", "&H0000FF&")

Entonces la función asignará a cada uno de los **loops** un único color correspondiente a la interpolación entre los dos colores asignados en los parámetros **color1** y **color2**.

Se pueden notar los círculos y el color primario ("\\1vc") asignado a cada uno de ellos:



La función **color.module** se puede usar con los tags de colores de ambos filtros, o sea:

1. \\1c
2. \\3c
3. \\4c
4. \\1vc
5. \\3vc
6. \\4vc

La función **color.module** recibe su nombre gracias a la variable **module** del **Kara Effector**, que recordemos hace referencia a la interpolación de todos los números entre cero y uno, con respecto al **Loop**.

---

**alpha.module( alpha1, alpha2 )**: similar a la función **color.module**, pero con la diferencia que trabaja con las transparencias (alpha) en vez de colores.

---

**color.module1( color1, color2 )**: esta función es similar a **color.module**, pero retorna la interpolación completa entre los colores de los parámetros **color1** y **color2**, con respecto al **Temaplate Type**.

Como su nombre lo indica, genera la interpolación por medio de la variable **module1**, que hace referencia a la interpolación entre cero y uno sin importar el Loop y teniendo en cuenta al **Template Type** (ejemplo: desde 1 hasta **syl.n**, o desde 1 hasta **word.n**).

Se podría decir que **color.module** interpola a todos los elementos de cada una de las partes de línea karaoke, por

## Kara Effector - Effector Book [Tomo XIII]:

---

otra parte, la función **color.module1** interpola a la línea de karaoke de principio a fin sin importar el **Loop** que tenga cada una de las partes de la misma.

---

**alpha.module1( alpha1, alpha2 ):** similar a la función **color.module1**, pero con la diferencia que trabaja con las transparencias (alpha) en vez de colores.

---

**color.module2( color1, color2 ):** esta función es similar a **color.module** y a **color.module1**, pero retorna la interpolación completa entre los colores **color1** y **color2**, con respecto a todas las líneas a las que se le aplica un efecto. Es decir que genera la interpolación desde el primer elemento (char, syl, word y demás) de la primera línea a la que se le aplique un efecto, hasta el último elemento de la última línea a la que se le haya aplicado un efecto.

Esta función hereda su nombre de la variable **module2** del **Kara Effector**, que hace una interpolación de los números entre cero y uno con respecto a la cantidad de líneas a las que se les aplique un efecto.

---

**alpha.module2( alpha1, alpha2 ):** similar a la función **color.module2**, pero con la diferencia que trabaja con las transparencias (alpha) en vez de colores.

---

Con el final de este **Tomo XIII** se dan por concluidas las librerías **color** y **alpha**. Intenten poner en práctica todos los ejemplos vistos en este **Tomo** 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, exponer alguna duda o hacer alguna sugerencia. Pueden visitarnos y dejar su comentario en nuestra página de **Facebook**:

[www.facebook.com/karaeffectort](http://www.facebook.com/karaeffectort)

---

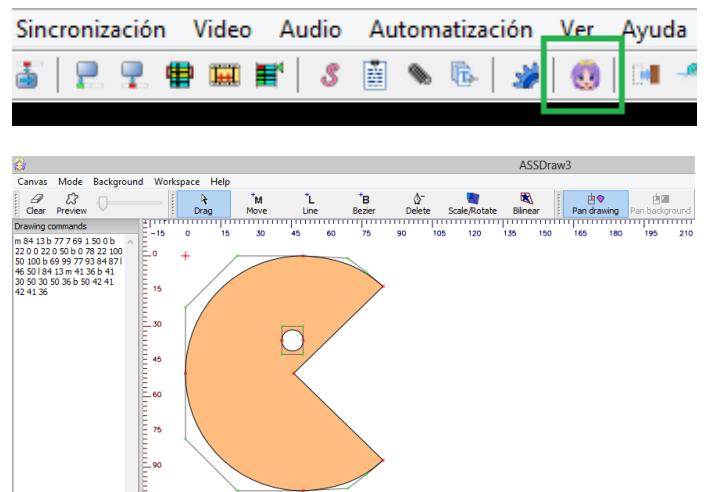
# Kara Effector 3.2: Effector Book Vol. I [Tomo XIV]

## Kara Effector 3.2:

El **Tomo XIV** es el comienzo de una nueva librería y cada vez estamos más cerca de terminarlas todas y así ampliar aún más las herramientas a nuestra disposición a la hora de desarrollar un **Efecto Karaoke**, un **Logo**, un **Cartel** y todo aquello que nos dispongamos a hacer para nuestros proyectos en el **Aegisub** y el **Kara Effector**.

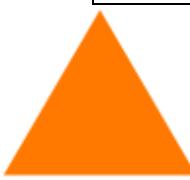
### Librería Shape [KE]:

Se conoce como **shape** a un dibujo hecho por vectores en el **AssDraw3** que viene por default en el **Aegisub**, y que nos sirven de apoyo como una herramienta más a la hora de desarrollar un Efecto.



La Librería **shape** hace referencia a dichos dibujos y a las funciones relativas a ellos. Aparte de las funciones en esta Librería, el **Kara Effector** consta de un amplio listado de **shapes** (dibujos) prediseñadas para hacer uso de ellas y a continuación veremos el nombre, su tamaño en pixeles y una pre-visualización de las mismas.

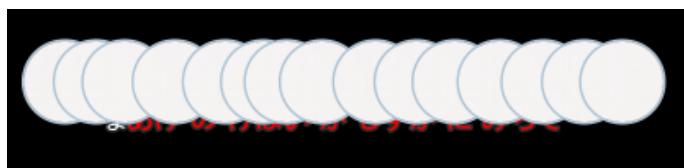
## Kara Effector - Effector Book [Tomo XIV]:

Shapes Prediseñadas del Kara Effector	
shape.circle	shape.triangle
 100 x 100 px	 100 x 106 px

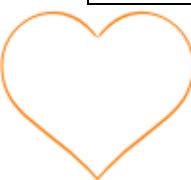
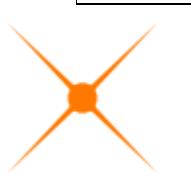
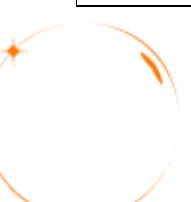
Ejemplo:



En dicho caso se retornaría un **círculo** en el lugar en donde antes estaban las sílabas de la línea karaoke:



Shapes Prediseñadas del Kara Effector	
shape.rectangle	shape.pentagon
 100 x 100 px	 100 x 95 px
shape.hexagon	shape.octagon
 100 x 116 px	 100 x 100 px
shape.heart	shape.heart2t
 100 x 106 px	 100 x 106 px

Shapes Prediseñadas del Kara Effector	
shape.heart_b	shape.shine1t
 100 x 106 px	 100 x 100 px
shape.shine2t	shape.shine3t
 100 x 100 px	 100 x 100 px
shape.shine4t	shape.trebol
 100 x 100 px	 100 x 106 px
shape.feather	shape.diamond
 100 x 100 px	 100 x 100 px
shape.gear	shape.bubble
 100 x 100 px	 100 x 100 px
shape.note1t	shape.note2t
 100 x 56 px	 100 x 100 px

## Kara Effector - Effector Book [Tomo XIV]:

Shapes Prediseñadas del Kara Effector	
shape.note3t 	shape.note4t 
shape.star 	shape.star1t 
shape.star2t 	shape.star3t 
shape.star4t 	shape.star5t 
shape.star6t 	shape.star7t 
shape.star8t 	shape.star9t 

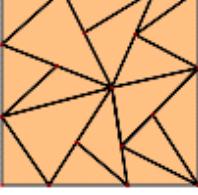
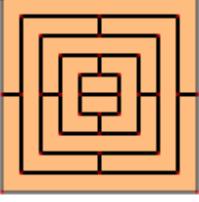
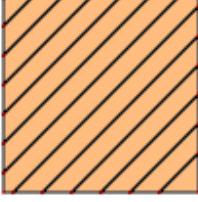
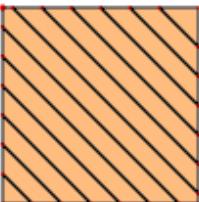
Shapes Prediseñadas del Kara Effector	
shape.star10t 	shape.sakura 
shape.sakura1t 	shape.sakura2t 
shape.sakura3t 	shape.sakura4t 
shape.sakura5t 	shape.sakura6t 
shape.sakura7t 	shape.snow1t 
shape.snow2t 	shape.snow3t 

## Kara Effector - Effector Book [Tomo XIV]:

Shapes Prediseñadas del Kara Effector	
shape.flower1t 	shape.flower2t 
100 x 95 px	100 x 95 px
shape.flower3t 	shape.flower4t 
100 x 95 px	100 x 95 px
shape.flower5t 	shape.flower6t 
100 x 95 px	100 x 95 px
shape.flower7t 	shape.flower8t 
100 x 95 px	100 x 95 px
shape.flower9t 	shape.flower10t 
100 x 95 px	100 x 95 px
shape.flower11t 	shape.flower12t 
100 x 116 px	100 x 116 px

Shapes Prediseñadas del Kara Effector	
shape.flower13t 	shape.flower14t 
100 x 116 px	100 x 130 px
shape.flower15t 	shape.flower16t 
100 x 116 px	100 x 116 px
shape.flower17t 	shape.flower18t 
100 x 116 px	100 x 116 px
shape.flower19t 	shape.flower20t 
100 x 116 px	100 x 116 px
shape.flower21t 	shape.flower22t 
100 x 116 px	100 x 116 px
shape.flower23t 	shape.flower24t 
100 x 116 px	100 x 116 px

## Kara Effector - Effector Book [Tomo XIV]:

Shapes Prediseñadas del Kara Effector	
<b>shape.flower25t</b>  100 x 116 px	<b>shape.flower26t</b>  100 x 130 px
<b>shape.flower27t</b>  100 x 116 px	<b>shape.flower28t</b>  100 x 116 px
<b>shape.flower29t</b>  100 x 116 px	<b>shape.cristal17</b>  100 x 100 px
<b>shape.geometric10</b>  100 x 100 px	<b>shape.diagonal13r</b>  100 x 100 px
<b>shape.diagonal13l</b>  100 x 100 px	

Como pueden ver, son muchas las opciones de Shapes a escoger entre todas las Shapes que vienen por default en el **Kara Effector**, todo dependerá del efecto a realizar y de los resultados que queremos obtener.

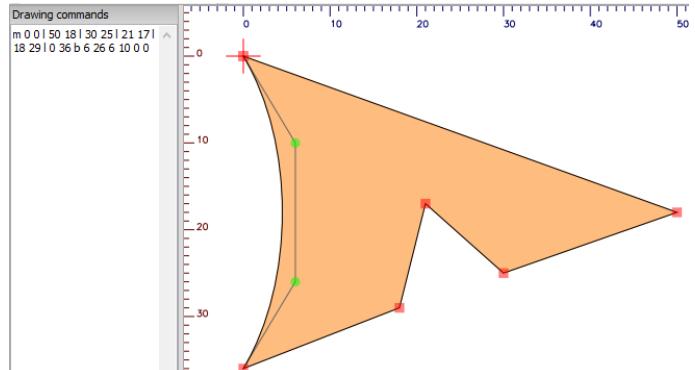
Con el anterior listado de Shapes prediseñadas del **Kara Effector**, comenzamos con la descripción de las funciones de la Librería **shape**, en donde algunas de ellas nos sirven para modificar y crear nuestras propias Shapes o, generar efectos directamente.

**shape.rotate( Shape, Angle, x, y )**: rota a la shape en un ángulo dado respecto al punto **P = (x, y)**.

Los parámetros **x** e **y** son opcionales al tiempo, sus valores por default son las coordenadas del punto **P = (0, 0)**. El parámetro **Angle** hace referencia a un valor numérico de un ángulo entre 0° y 360°.

Para este y los próximos ejemplos usaremos un **Template Type: Line**, con el fin de generar una única línea por cada línea karaoke y así, sea más sencillo visualizar el resultado.

La **shape** que usaré en estos ejemplos será una muy simple, pero con un diseño en particular que permita ver la rotación de la misma:



Ejemplo:

Por lo general, siempre suelo declarar a las Shapes en la celda de texto “**Variables**”, con el fin de hacer un poco más cómodo el uso de la misma, pero también se pueden usar directamente, entre comillas, dentro de las funciones que requieran una **shape** como alguno de los parámetros a usar dentro de ella:

```
Variables:  
mi_shape = "m 0 0 l 50 18 l 30 25 l 21 17 l 18 29 l 0 36  
b 6 26 6 10 0 0 "
```

## Kara Effector - Effector Book [Tomo XIV]:

Las dos formas de usar la **shape** son válidos. Con el fin de que se visualice en pantalla, usaremos la función en la celda **Return [fx]**.

Opción 1:

Return [fx]:

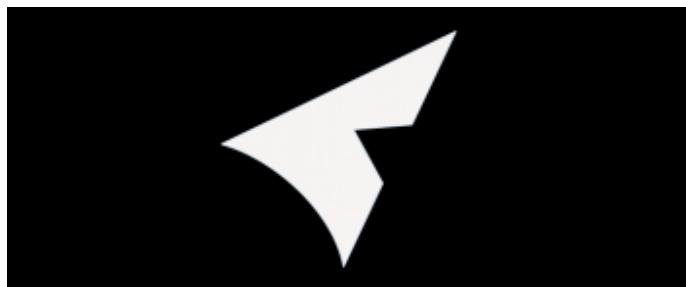
```
shape.rotate( mi_shape, 45 )
```

Opción 2:

Return [fx]:

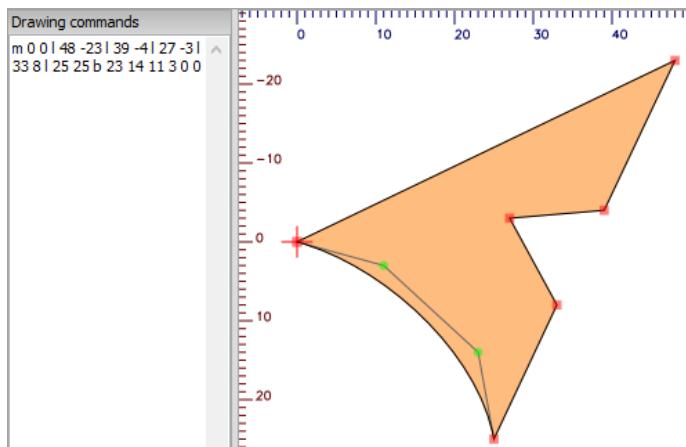
```
shape.rotate( "m 0 0 | 50 18 | 30 25 | 21 17 | 18 29 | 0 36 ^  
b 6 26 6 10 0 0 ", 45 )
```

En vídeo:



lead-in | Effector [Fx] \*m 0 0 | 48 -23 | 39 -41 27 -31 33 8 | 25 25 b 23 14 11 3 0 0

En el AssDraw3:



La **shape** ha rotado 45° respecto al punto P = (0, 0), ya que en el ejemplo anterior se omitieron los parámetros **x** e **y**.

Hecho este ejemplo, la pregunta pareciera caer por su propio peso, ¿por qué no simplemente usar el tag \frz para rotar la **shape** en vez de esta función?

Cuando la shape rotada se va a usar como una simple **shape**, entonces sí es lo mismo usar el tag \frz para rotarla y la función **shape.rotate**, pero cuando se va a usar una shape para generar un \clip o un \iclip, entonces ningún tag puede afectar a las Shapes que se encuentren dentro de ellos. Ejemplo:

```
\clip( m 0 0 | 0 50 | 0 0 )
```

Como la **shape** “m 0 0 | 0 50 | 0 0” está dentro del \clip, no hay forma de modificar las características de la misma, a menos que usemos las funciones de la Librería **shape**. Si quisieramos rotar la **shape** dentro del \clip, lo haríamos de la siguiente forma. Ejemplo:

Add Tags: Add Tags Language: Automation Auto-4

```
\clip( !shape.rotate( "m 0 0 | 0 50 | 0 0", 30 )! )
```

Entonces se usará la **shape** rotada 30° dentro del \clip.

Nos resta ver ejemplos de cómo usar la función con los parámetros **x** e **y** incluidos. Ejemplos:

- **shape.rotate( mi\_shape, 100, 0, 20 )**
- **shape.rotate( mi\_shape, 150, -30, 0 )**
- **shape.rotate( mi\_shape, 30, 45, -20 )**
- **shape.rotate( mi\_shape, 210, -15, -50 )**

**shape.reflect( Shape, Axis )**: refleja a la shape respecto al eje asignado en el parámetro **Axis**. Ejemplo:

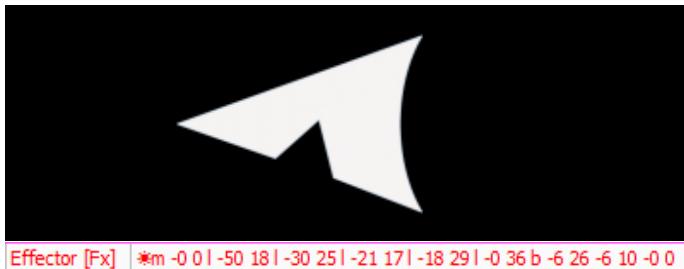
Return [fx]:

```
shape.reflect( mi_shape, 'y' )
```

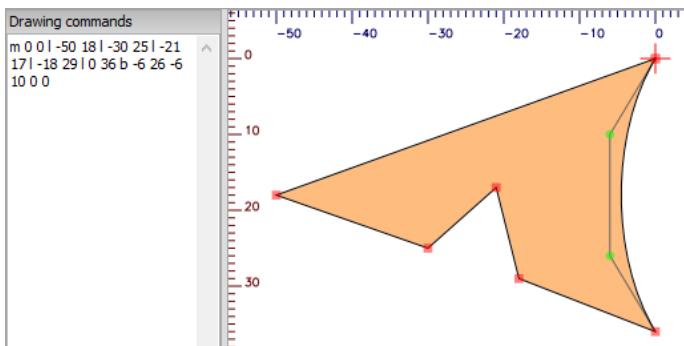
El parámetro **Axis** puede ser “x” para hacer referencia al eje “x”, o “y” para hacer referencia al eje “y”, como se acaba de usar en el ejemplo inmediatamente anterior.

## Kara Effector - Effector Book [Tomo XIV]:

Lo que resultaría así:



Y en el **AssDraw3** se verá así:



Entonces decimos que si **Axis** es “x”, la **shape** se reflejará respecto al eje “x”; si es “y” se reflejará en dicho eje, pero hay una tercera opción que hace que la **shape** se refleje respecto a ambos ejes al mismo tiempo, así:

```
shape.reflect( mi_shape, "xy" )
```

Cuando **Axis** es “xy” la **shape** se refleja respecto a los ejes “x” e “y” de manera simultánea. Se obtiene el mismo resultado si solo no ponemos el parámetro **Axis**:

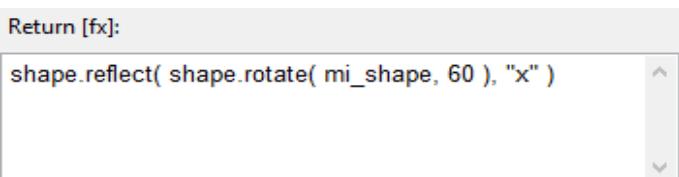
```
shape.reflect( mi_shape )
```

En ambos casos la **shape** será reflejada respecto a los dos ejes del plano.

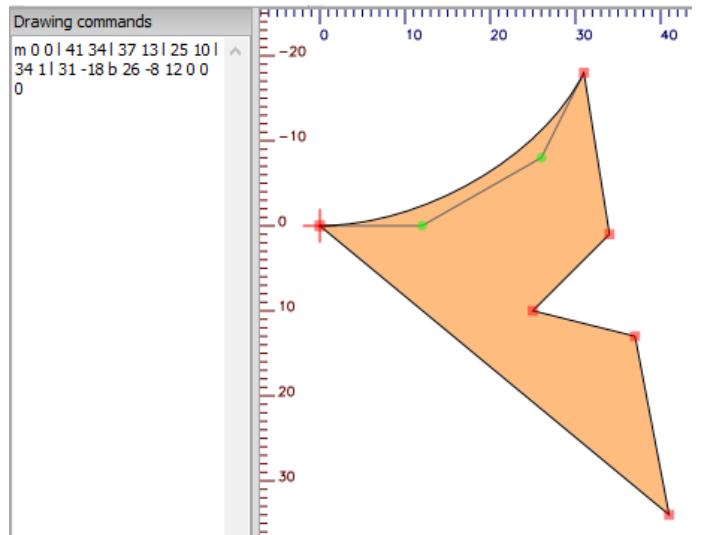
Veamos un ejemplo combinando las dos funciones hasta acá vistas de la Librería **shape**:

```
shape.reflect( shape.rotate( mi_shape, 60 ), "x" )
```

O sea que la **shape** primero se rota 60° y luego se refleja respecto al eje “x”:



Y en el **AssDraw3** podemos ver el suceso anteriormente descrito:



Son muchas las posibilidades al combinar tan solo estas dos funciones. Espero que puedan inventar sus propios ejemplos y que se puedan sorprender con los diferentes resultados.

Este **Tomo XIV** es solo la introducción de la Librería **shape**, ya que aún nos resta por ver muchas más funciones de la misma y todas las posibilidades que ellas nos ofrecen. Intenten poner en práctica todos los ejemplos vistos en este **Tomo** 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, exponer alguna duda o hacer alguna sugerencia. Pueden visitarnos y dejar su comentario en nuestra página de **Facebook**:

[www.facebook.com/karaeffector](http://www.facebook.com/karaeffector)

Kara Effector - Effector Book [Tomo XV]:

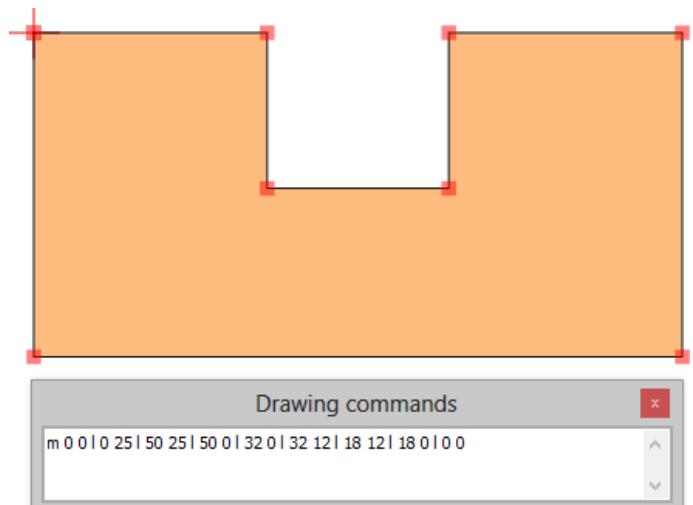
## Kara Effector 3.2: Effector Book Vol. I [Tomo XV]

## Kara Effector 3.2:

El **Tomo XV** es la continuación de la librería **shape**, ya que en el **Tomo** anterior vimos el listado de las Shapes que trae por default el **Kara Effector**, más un par de funciones de la Librería.

### Librería Shape [KE]:

Para los siguientes ejemplos en las definiciones de las funciones de esta Librería, usará esta simple **shape** que mide 50 X 25 px, pero ustedes pueden usar la que quieran y aun así los resultados deben ser visibles:



Y para mayor comodidad, la declaro en “**Variables**”:

```
Variables:  
mi_shape = "m 0 0 | 0 25 | 50 25 | 50 0 | 32 0 | 32 12 | 18  
12 | 18 0 | 0 0 "
```

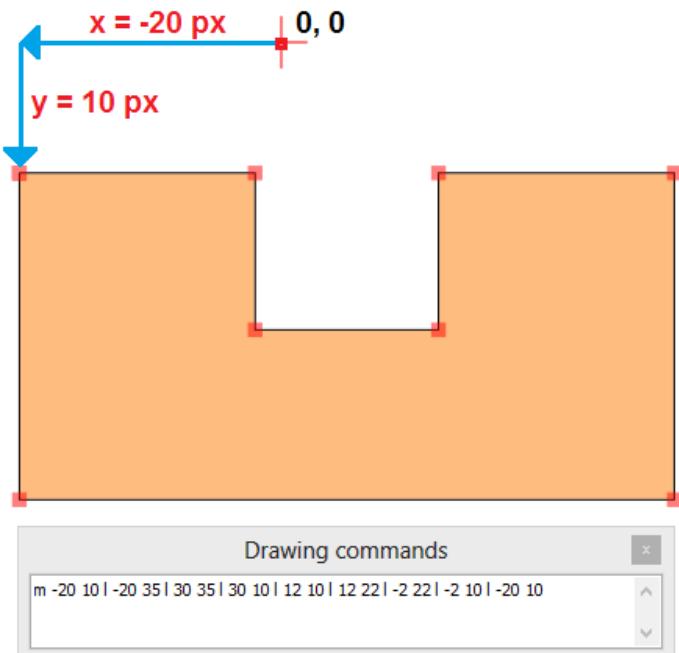
## Kara Effector - Effector Book [Tomo XV]:

**shape.displace( shape, x, y ):** desplaza la **shape** tantos pixeles respecto a ambos ejes cartesianos, como le indiquemos en los parámetros **x** e **y**. Ejemplo:

Return [fx]:

```
shape.displace( mi_shape, -20, 10 )
```

Retorna la misma **shape**, pero desplazada 20 pixeles a la izquierda ( $x = -20$ ) y 10 pixeles hacia abajo ( $y = 10$ ):



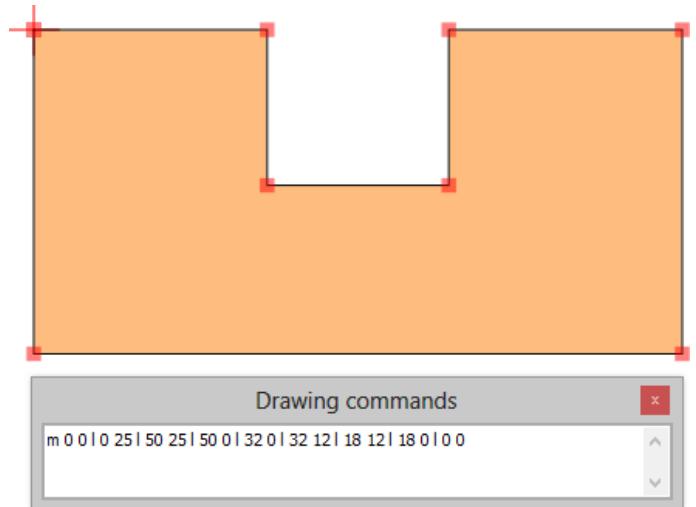
Recordemos que en el **AssDraw3** y en el formato .ass, el eje positivo de "y" es hacia abajo del eje "x" y el negativo, hacia arriba del mismo.

**shape.origin( shape ):** desplaza la **shape** tantos pixeles respecto a ambos ejes cartesianos, hasta ubicarla en el **Cuadrante IV** del plano en el **AssDraw3**, respecto al punto de origen **P = (0, 0)**. Para el siguiente ejemplo usaré la **shape** desplazada del ejemplo anterior:

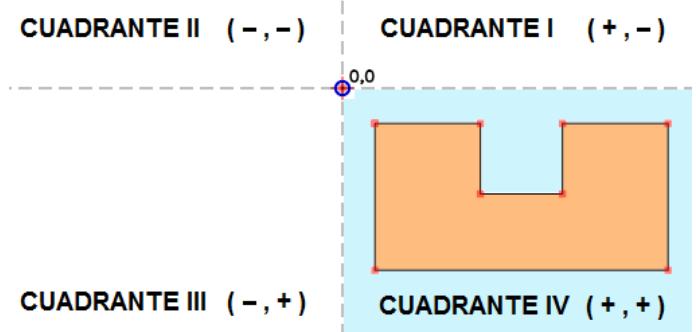
Return [fx]:

```
shape.origin( "m -20 10 | -20 35 | 30 35 | 30 10 | 12 10 | 12 22 | -2 22 | -2 10 | -20 10" )
```

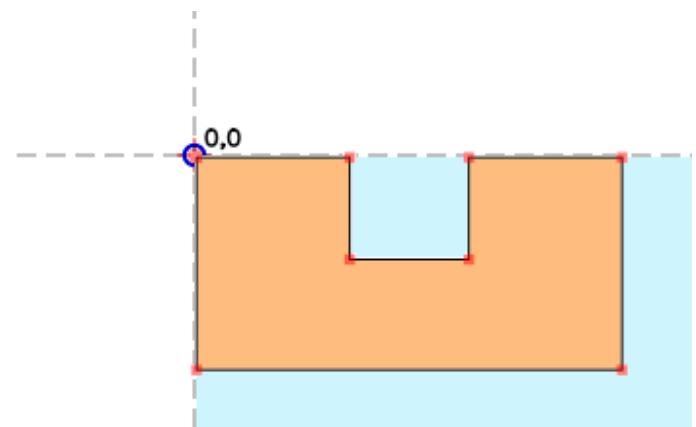
Entonces la función la ubicará en el **Cuadrante IV**:



En la siguiente imagen se muestran los **Cuadrantes** del **AssDraw3** y los signos que tienen ambas coordenadas en dichos Cuadrantes:



Y la función **shape.origin** desplaza la **shape**, en donde quiera que esté en el plano, al origen del **Cuadrante IV**, que es el Cuadrante en donde ambas coordenadas son positivas. Cuando una **shape** está ubicada en el origen del **Cuadrante IV**, se hace más sencillo aplicarle los tags de modificación y los resultados serán los esperados:



## Kara Effector - Effector Book [Tomo XV]:

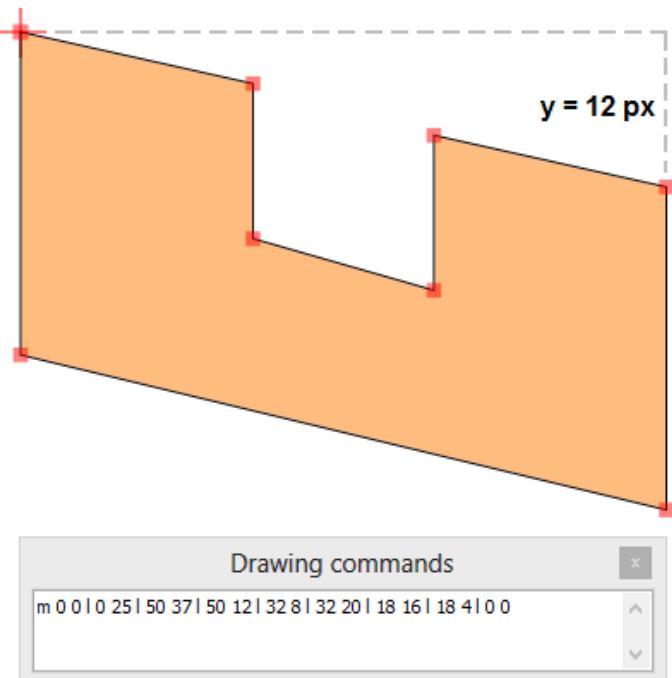
**shape.oblique( shape, Pixels, Axis ):** deforma la **shape** de manera oblicua, en tantos pixeles indicados en el parámetro **Pixels**, respecto al eje asignado **Axis**.

- Ejemplo 1:

Return [fx]:

```
shape.oblique( mi_shape, 12, "y" )
```

Entonces la función deforma la **shape** 12 pixeles positivos respecto al eje “y”:



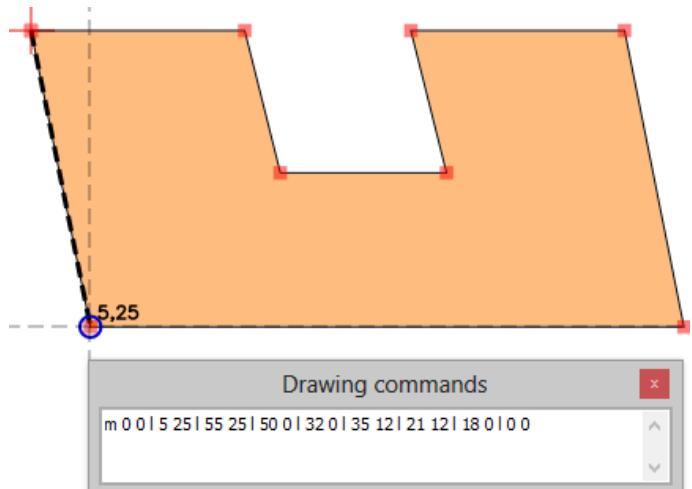
Todas las coordenadas en “x” se conservan, y las de “y” son desplazadas de forma progresiva hasta que aquellas que acompañan a las coordenadas “x” más alejadas del origen, se desplacen la cantidad de pixeles asignados en el parámetro **Pixels** (12 px). **Pixels** también puede ser un valor negativo, lo que deformaría la **shape** hacia arriba.

- Ejemplo 2:

Return [fx]:

```
shape.oblique( mi_shape, 5, "x" )
```

En este caso, la **shape** se deformará 5 pixeles hacia la derecha, dado que **Pixels** es positivo:



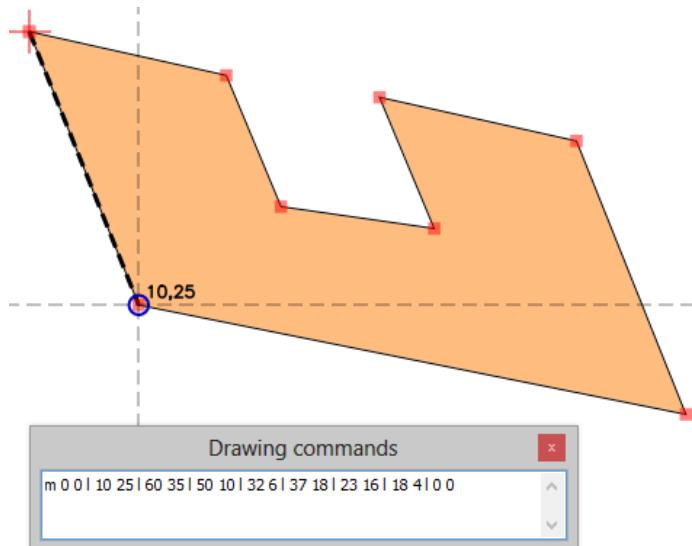
En el anterior ejemplo, las coordenadas que se conservan intactas en las **shape** son las de “y”, y las coordenadas de “x” se deforman de forma progresiva.

- Ejemplo 3:

Return [fx]:

```
shape.oblique( mi_shape, 10 )
```

Ahora, al no poner el parámetro **Axis**, entonces la función deforma la **shape** en ambos ejes, en igual cantidad de pixeles (10 px); 10 px hacia la derecha ( $x = 10\text{px}$ ) y 10 px hacia abajo ( $y = 10\text{px}$ ).



## Kara Effector - Effector Book [Tomo XV]:

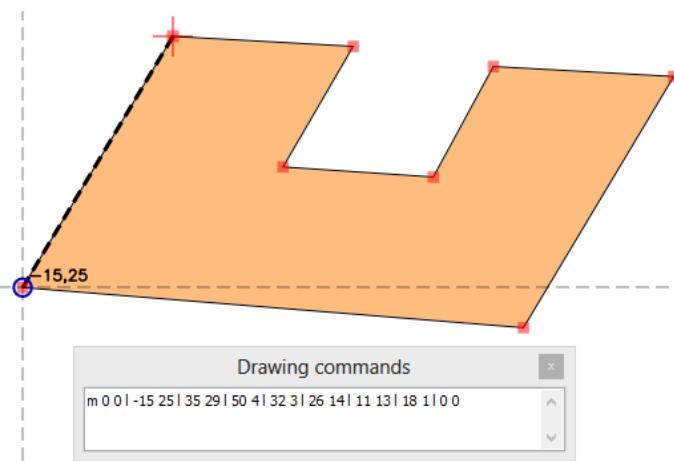
- Ejemplo 4:

Con este método podemos decidir la cantidad de pixeles en que se deformará la **shape** en ambos ejes:

Return [fx]:

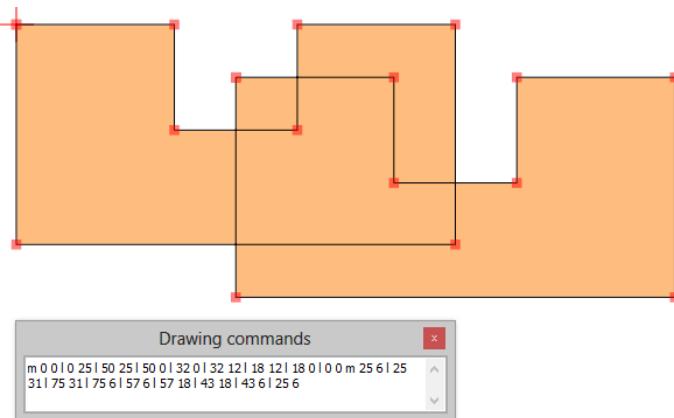
```
shape.oblique( mi_shape, { -15, 4 } )
```

Vemos cómo la **shape** se deformó 15 pixeles a la izquierda ( $x = -15$  px) y 4 pixeles hacia abajo ( $y = 4$  px):



**shape.reverse( shape )**: esta función reescribe la **shape** de manera que quede exactamente igual, pero dibujada a la inversa para que pueda ser sustraída de otra.

Para el siguiente ejemplo, dupliqué la misma **shape** y la desplacé varios pixeles respecto a la original, de manera que queden superpuestas como podemos ver en la siguiente imagen:

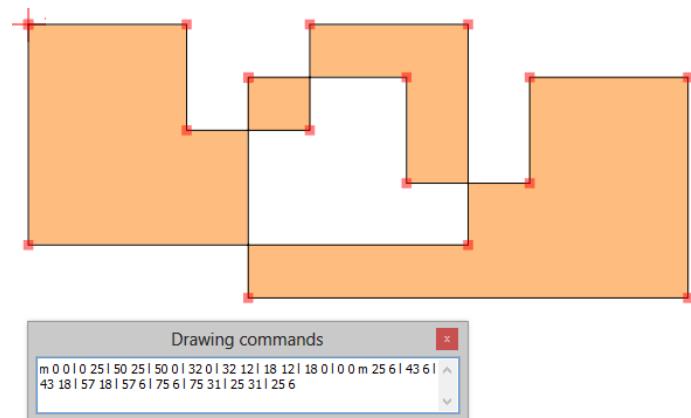


Ahora, aplicaremos la función a esa segunda **shape** para que sea redibujada de manera inversa:

Return [fx]:

```
shape.reverse( "m 25 6 | 25 31 | 75 31 | 75 6 | 57 6 | 57 18 | 43 18 | 43 6 | 25 6" )
```

Las Shapes siguen siendo las mismas, pero el área que coincide entre ambas es sustraída, ya que una **shape** está dibujada en un sentido (sentido anti horario) y la otra a la inversa (sentido horario):



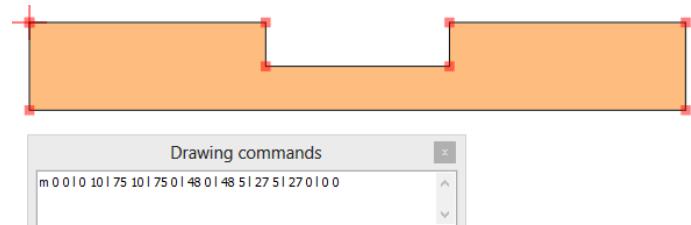
**shape.ratio( shape, ratio\_x, ratio\_y )**: esta función redimensiona la **shape** en una proporción equivalente a **ratio\_x** y **ratio\_y**.

- Ejemplo 1:

Return [fx]:

```
shape.ratio( mi_shape, 1.5, 0.4 )
```

**ratio\_x** = 1.5, es decir que la **shape** ahora es 1.5 veces más ancha de lo que era originalmente (150 %):



## Kara Effector - Effector Book [Tomo XV]:

**ratio\_y** = 0.4, es decir que la altura de la **shape** solo el 40% de la altura original.

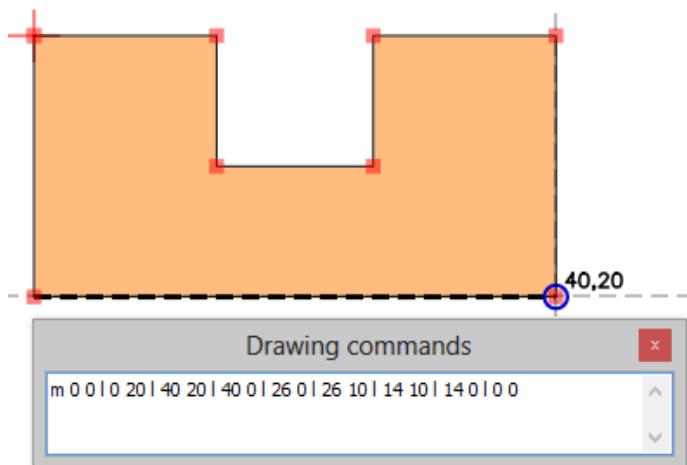
- Ejemplo 2:

En este modo se omite el parámetro **ratio\_y**, entonces la función asume que **ratio\_x** es la proporción del tamaño final, respecto a ambos ejes:

Return [fx]:

```
shape.ratio( mi_shape, 0.8 )
```

La **shape** que retorna es casi la misma, solo que su tamaño es un 80% (**ratio\_x** = 0.8) del tamaño de la **shape** original:



**shape.size( shape, size\_x, size\_y )**: esta función es similar a la función **shape.ratio**, pero con la diferencia que redimensiona la **shape** de tal manera que el ancho de la misma determinado según el parámetro **size\_x** en pixeles, y su altura será determinada por el parámetro **size\_y**, también en pixeles.

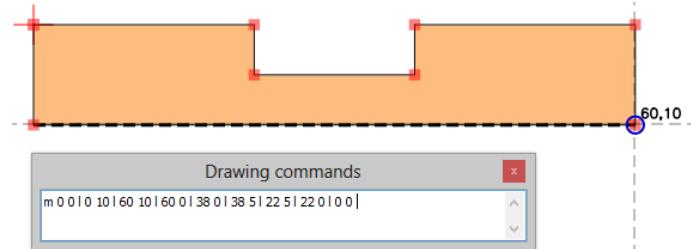
- Ejemplo 1:

De este modo, la **shape** quedará midiendo 60 X 10 px:

Return [fx]:

```
shape.size( mi_shape, 60, 10 )
```

Como se evidencia, tanto en la imagen de la **shape** como en el código de la misma, las dimensiones de la **shape** son las ingresadas en la función (60 X 10 px):



- Ejemplo 2:

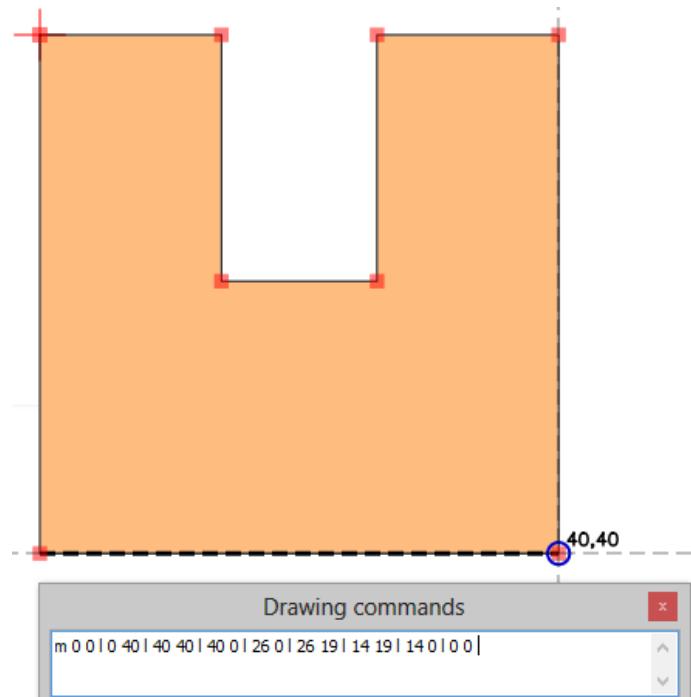
Se omite el parámetro **size\_y**, de modo que la función asume que tanto el ancho como el alto de la **shape** serán el mismo, o sea **size\_x**:

Return [fx]:

```
shape.size( mi_shape, 40 )
```

Usada la función de este modo, la **shape** queda con las medidas en pixeles que hayamos ingresado en la función, en este caso 40 px.

- Ancho = 40 px
- Alto = 40 px



## Kara Effector - Effector Book [Tomo XV]:

**shape.info( shape ):** brinda información primaria básica de la **shape**. Dicha información está dada en seis variables:

1. **minx**
2. **maxx**
3. **miny**
4. **maxy**
5. **w\_shape**
6. **h\_shape**

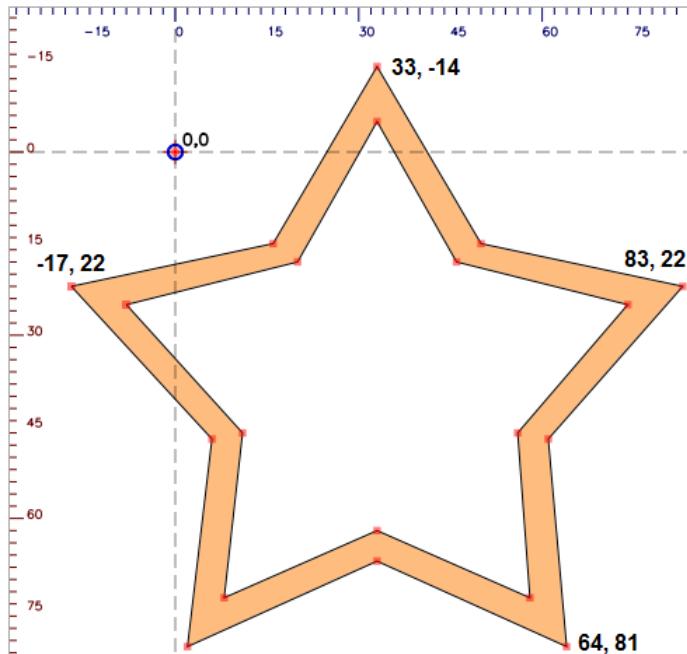
Ejemplo:

Declaramos una variable con el nombre que queramos y la igualamos a la función con una **shape**:

Variables:

```
Shape_info = shap.info( "m -17 22 | 6 47 | 2 81 | 33 67 | 64  
81 | 61 47 | 83 22 | 50 15 | 33 -14 | 16 15 | -17 22 m -8 25 |  
20 18 | 33 -5 | 46 18 | 74 25 | 56 46 | 58 73 | 33 62 | 8 73 |  
11 46 | -8 " )
```

Esta es la shape que corresponde al código anterior:



Al llevar a cabo este procedimiento en la celda de texto “Variables”, ya podemos usar las anteriores seis variables mencionadas, con los siguientes valores:

- **minx** = -17, que es el mínimo valor respecto a “x”
- **maxx** = 83, máximo valor en “x”
- **miny** = -14, mínimo valor en “y”

- **maxy** = 81, máximo valor en “y”
- **w\_shape** = **maxx** – **minx** = 83 – (-17) = 100 px, corresponde al ancho de la shape ingresada.
- **h\_shape** = **maxy** – **miny** = 81 – (-14) = 95 px, corresponde al alto de la shape ingresada.

Las anteriores seis variables ya pueden ser usadas como valores numéricos en cualquier otra celda de texto de la ventana de modificación del **Kara Effector**. Ejemplo:

```
Add Tags: Add Tags Language: Lua  
"\fscy" .. h_shape
```

Que a la postre retornará: \fscy85, dado que la altura de la **shape** ingresada era de 85 px.

Es todo por ahora para este **Tomo XV**, pero las funciones de la Librería **shape** aún no llegan a su fin. En el próximo **Tomo** continuaremos profundizando en el mundo de las Shapes y las posibilidades que nos ofrecen. Intenten poner en práctica todos los ejemplos vistos en este **Tomo** 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, exponer alguna duda o hacer alguna sugerencia. Pueden visitarnos y dejar su comentario en nuestra página de **Facebook**:

[www.facebook.com/karaeffectort](http://www.facebook.com/karaeffectort)

**Kara Effector - Effector Book [Tomo XVI]:**

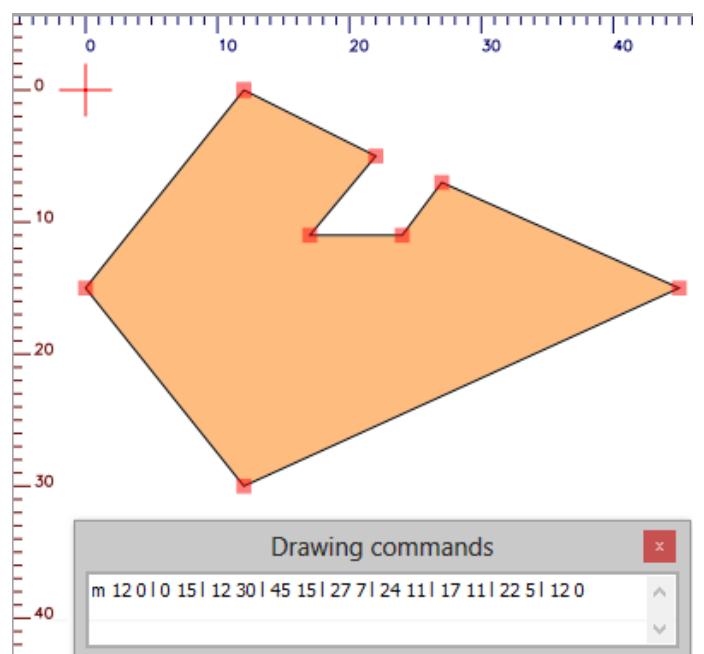
# **Kara Effector 3.2: Effector Book Vol. I [Tomo XVI]**

# **Kara Effector 3.2:**

En el **Tomo XVI** seguimos con la funciones de la Librería **shape**, pero con la certeza que en este **Tomo** no terminará la misma, ya que es muy extensa y completa con el fin que tengamos a nuestra plena disposición la mayor cantidad de herramientas posibles a la hora de desarrollar nuestros proyectos.

## **Librería Shape [KE]:**

Para los siguientes ejemplos en las definiciones de las funciones, usaremos esta simple **shape** que de 45 X 30 px:



Y como ya es costumbre, declaramos una variable con nuestra **shape** (el nombre es a gusto de cada uno):

```
Variables:  
mi_shape = "m 12 0 | 0 15 | 12 30 | 45 15 | 27 7 | 24 11 |  
17 11 | 22 5 | 12 0 "
```

## Kara Effector - Effector Book [Tomo XVI]:

`shape.array( shape, loops, A_mode, Dxy )`: hace un Arreglo o Matriz (duplicaciones) de la **shape**, una cierta cantidad de veces (**loops**), con diversas características y modalidades dependiendo de los otros dos parámetros (**A\_mode** y **Dxy**).

Esta función tiene tres modalidades distintas, y cada una tiene una serie de opciones que veremos a continuación en los siguientes ejemplos:

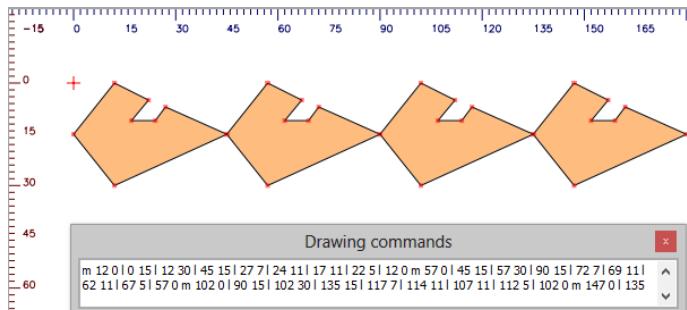
### Modo Lineal

- **Ejemplo 1.** Ingresamos la **shape**, el número de repeticiones, que para este ejemplo es 4, y no se pone ni **A\_mode** ni **Dxy**:

Return [fx]:

```
shape.array( mi_shape, 4 )
```

Entonces la **shape** se duplicará una a la derecha de la otra, cuatro veces, en forma lineal. El duplicar la **shape** una a la derecha de la otra es equivalente a un Arreglo Lineal con un ángulo de 0°:



- **Ejemplo 2.** Aparte de la **shape** y la cantidad de repeticiones, el parámetro **A\_mode** es 15, que es equivalente a un ángulo de 15°.

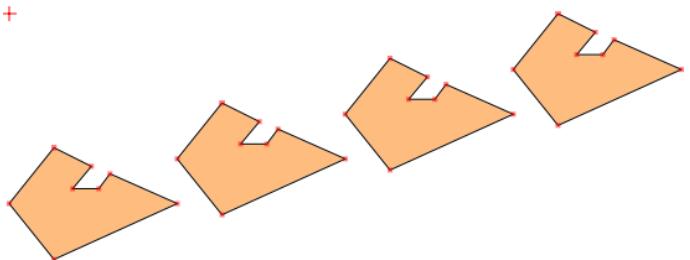
Cuando **A\_mode** es un valor numérico, la función asume que dicho valor es el ángulo que tendrá el Arreglo Lineal de la shape resultante:

Return [fx]:

```
shape.array( mi_shape, 4, 15 )
```

El Arreglo es Lineal y con un ángulo de 15°:

+



Drawing commands

m 12 36   0 51   12 66   45 51   27 43   24 47   17 47   22 41   12 36 m 57 24   45 39   57 54   90 39   72 31   69 35   62 35   67 29   57 24 m 102 12   90 27   102 42   135 27   117 19   114 23   107 23   112 17   102 12   147 0   135 15   147 30   180 15   162 7   159 11   152 11   157 5   147 0
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

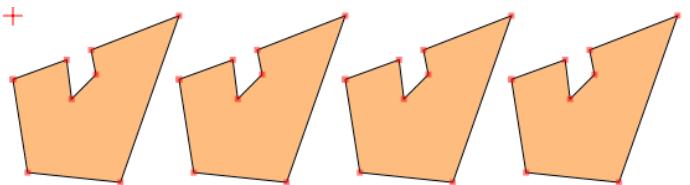
- **Ejemplo 3.** **A\_mode** ahora es una **tabla** con dos valores numéricos, el primero indica el ángulo del arreglo lineal y el segundo, la rotación respecto al centro de la **shape**:

Return [fx]:

```
shape.array( mi_shape, 4, {0, 45} )
```

Entonces el ángulo del arreglo es 0 y la **shape**, antes de ser duplicada, se rotó un ángulo de 45°:

+



Drawing commands

m 0 13   3 32   22 34   34 0   16 7   17 12   12 17   11 9   0 13 m 34 13   37 32   56 34   68 0   50 7   51 12   46   17 14 9   34 13 m 68 13   71 32   90 34   102 0   84 7   85 12   80 17   79 9   68 13 m 102 13   105 32   124   34   136 0   118 7   119 12   114 17   113 9   102 13
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

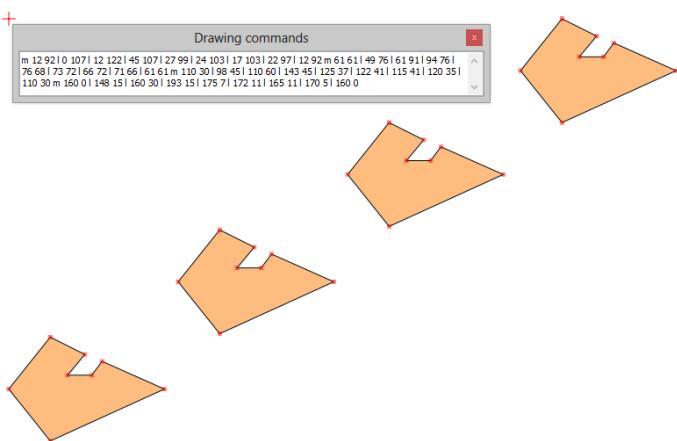
- **Ejemplo 4.** Incluimos al parámetro **Dxy** como valor numérico, que hace referencia a la distancia en px que separará a la **shape** dentro del arreglo:
- loops : 4
- Ángulo del Arreglo: 32°
- Distancia Separadora: 5 px

Return [fx]:

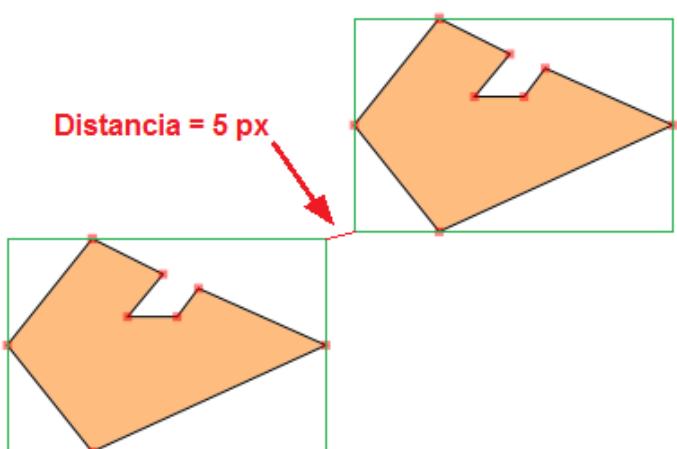
```
shape.array( mi_shape, 4, 32 , 5)
```

## Kara Effector - Effector Book [Tomo XVI]:

La distancia entre las Shapes del Arreglo ahora ya no es cero, que es su valor por default, sino 5 px:



En la anterior imagen queda la sensación de que las Shapes están separadas por una distancia mayor a 5 px, pero es porque la forma de la shape no es totalmente rectangular. Si tomamos dos de ellas y las enmarcamos en rectángulos, su pueden ver los 5 px de separación:

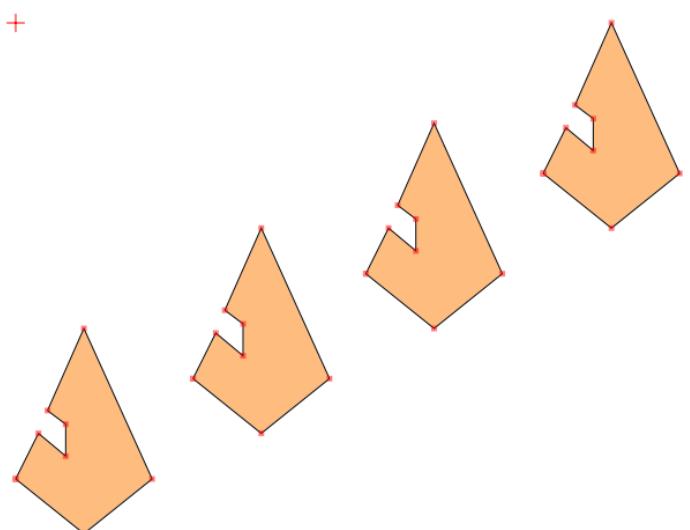


La distancia de separación también puede ser un valor numérico negativo, y lo que hará que el Arreglo Lineal quede más compacto y se acerquen tanto, una shape a la otra, hasta que se superpongan, si eso quisieramos.

- **Ejemplo 5.** Es una combinación de los ejemplos 3 y 4. Ahora podemos decidir la separación, el ángulo del Arreglo y el ángulo de rotación:

```
Return [fx]:  
shape.array( mi_shape, 4, {30, 90}, 10 )
```

- loops: 4
- Ángulo del Arreglo: 30°
- Ángulo de Rotación de la **shape**: 90°
- Distancia Separadora: 10 px



Los anteriores ejemplos nos muestran las cinco opciones de Arreglos Lineales que podemos hacer con la función. El **Modo Lineal** es la forma más simple de usar esta función, pero a continuación veremos el próximo modo de uso y sus diversas opciones:

### Modo Radial

- **Ejemplo 1.** Ingresamos la **shape**, el número de repeticiones del Arreglo, en **A\_mode** escribimos entre comillas la palabra “radial”. **Dxy** equivale al radio del arco:

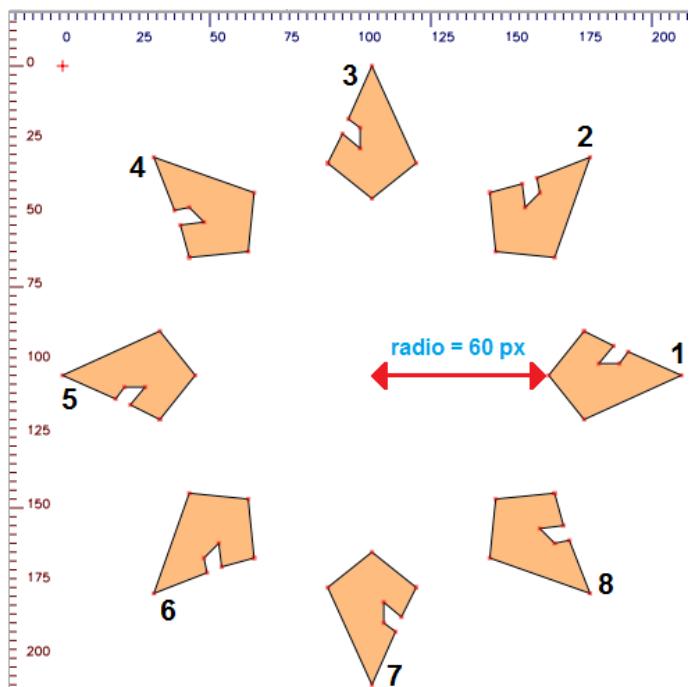
```
Return [fx]:  
shape.array( mi_shape, 8, "radial", 60 )
```



## Kara Effector - Effector Book [Tomo XVI]:

Vemos la **shape** repetida ocho veces. El ángulo del arco es  $360^\circ$  por default, y es por eso que la **shape** se repite en un Arreglo Radial, equidistantemente en esos  $360^\circ$ :

- loops: 8
- Modo: "radial"
- Radio: 60 px



El radio es constante para cada una de las repeticiones del Arreglo (60 px), y cada una de las Shapes está rotada de tal manera que quedan orientadas en dirección del centro imaginario del Arreglo.

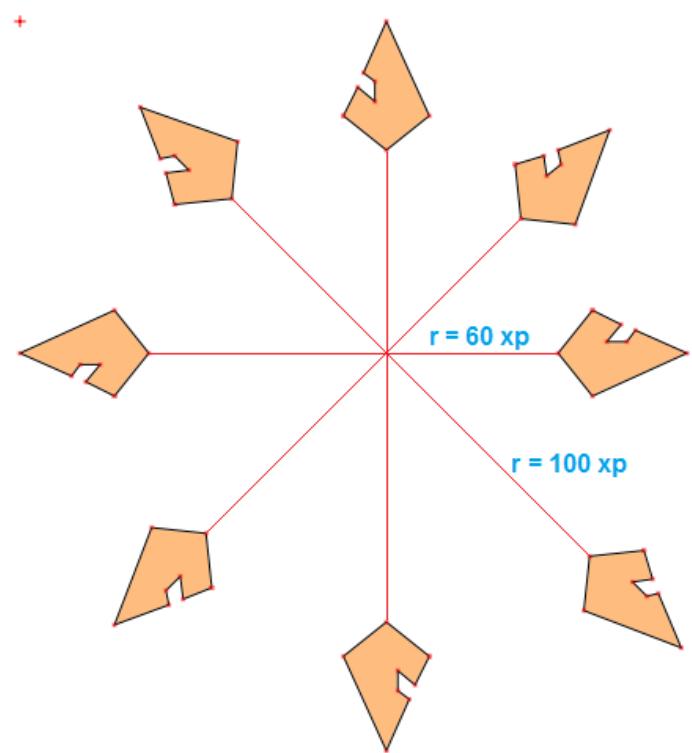
- **Ejemplo 2.** Ingresamos la **shape**, el número de repeticiones del Arreglo, en **A\_mode** escribimos entre comillas la palabra "**radial**". Pero ahora **Dxy** es una **tabla** que contiene dos valores numéricos, el primero equivale al radio inicial en pixeles del Arreglo y el segundo al radio final:

- loops: 8
- Modo: "radial"
- Radio Inical: 60 px
- Radio Final: 100 px

Return [fx]:

```
shape.array( mi_shape, 8, "radial", {60, 100} )
```

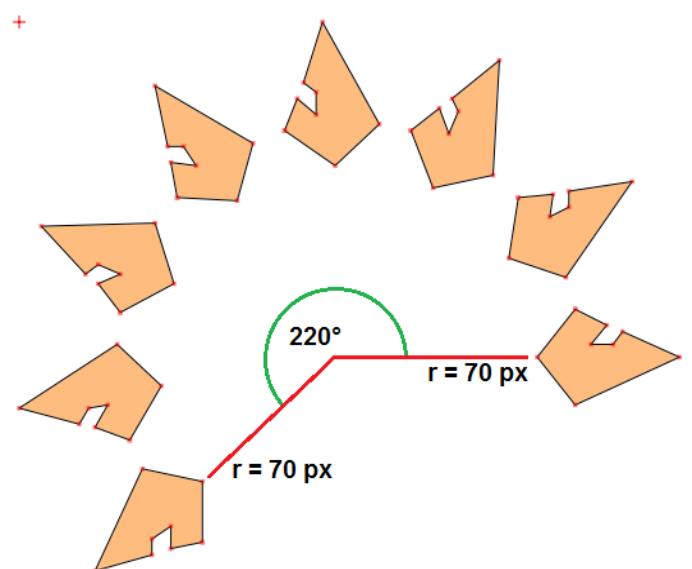
Los radios del Arreglo van aumentando progresivamente desde el Radio Inicial (60 px) hasta el Radio Final (100 px):



- **Ejemplo 3.** Agregamos un tercer valor en la **tabla Dxy**, equivalente al ángulo del arco del Arreglo, que por default era  $360^\circ$ :

Return [fx]:

```
shape.array( mi_shape, 8, "radial", {70, 70, 220} )
```



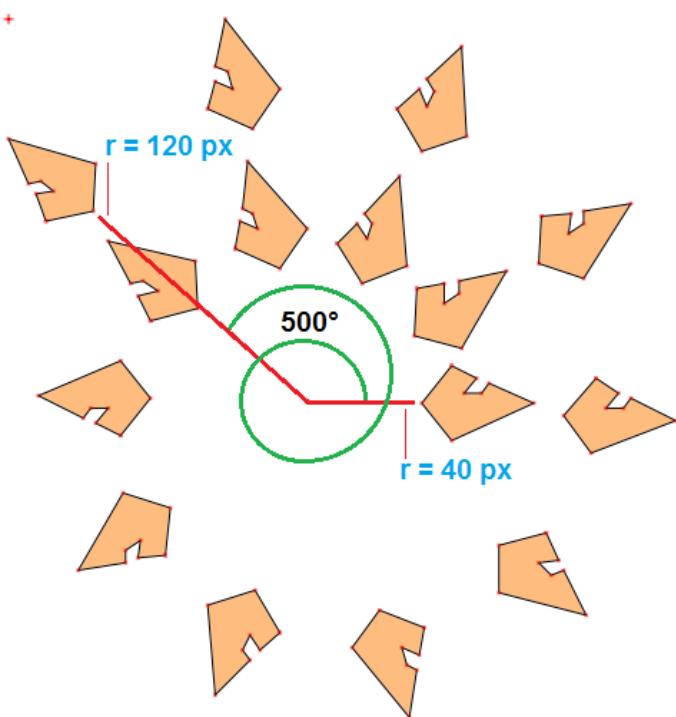
## Kara Effector - Effector Book [Tomo XVI]:

- **Ejemplo 3.1.** Aumentamos el valor del ángulo del Arreglo de tal manera que exceda los 360° de la circunferencia y, ponemos al Radio Inicial y al Radio Final con diferentes valores para evitar que las Shapes se superpongan:

Return [fx]:

```
shape.array( mi_shape, 15, "radial", {40, 120, 500} )
```

- loops: 15
- Modo: "radial"
- Radio Inical: 40 px
- Radio Final: 120 px
- Ángulo del Arco: 500°



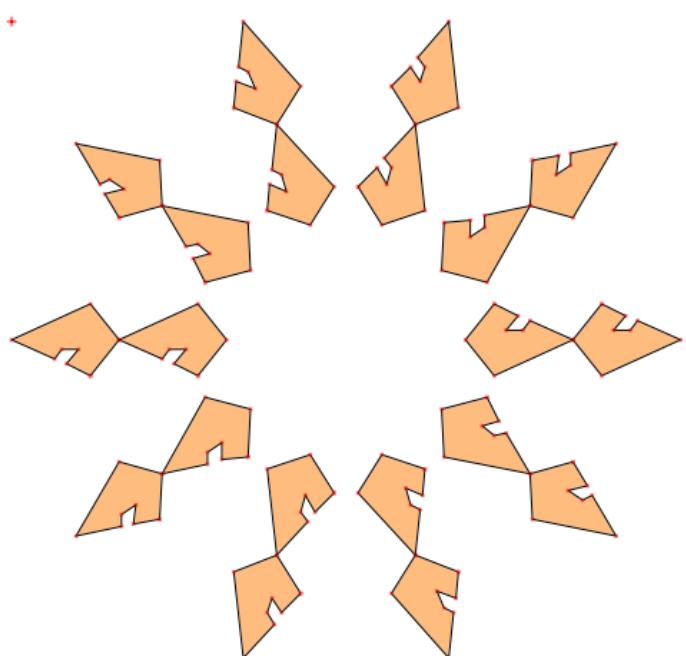
- **Ejemplo 4.** El parámetro **loops** es una **tabla** con dos valores numéricos, el primero indica las veces que se repite la **shape** en el Arreglo Radial, y el segundo indica la veces que se repite el Arreglo de forma y tamaño mayormente progresivo:

Return [fx]:

```
shape.array( mi_shape, {10, 2}, "radial", 50 )
```

- loops del Arreglo: 10
- Repeticiones: 2
- loops Total:  $10 \times 2 = 20$
- Modo: "radial"
- Radio Interior del primer Arreglo: 50 px

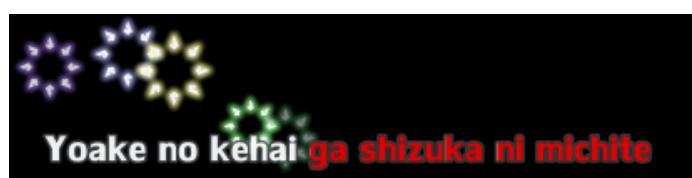
Al usar la función de esta forma, la separación entre cada uno de los Arreglos Radiales es por default 0, es decir que el radio interior de cada Arreglo Radial será exactamente del mismo tamaño en pixeles del radio exterior del Arreglo inmediatamente anterior:



La **shape** resultante es cada vez más compleja y resultaría muy laborioso dibujarla manualmente en el **AssDraw3**, además del hecho de no poder hacerlo con tanta precisión y velocidad. Del anterior ejemplo resulta una **shape** muy interesante para hacer un Efecto:



Con la **shape** del Ejemplo 1:



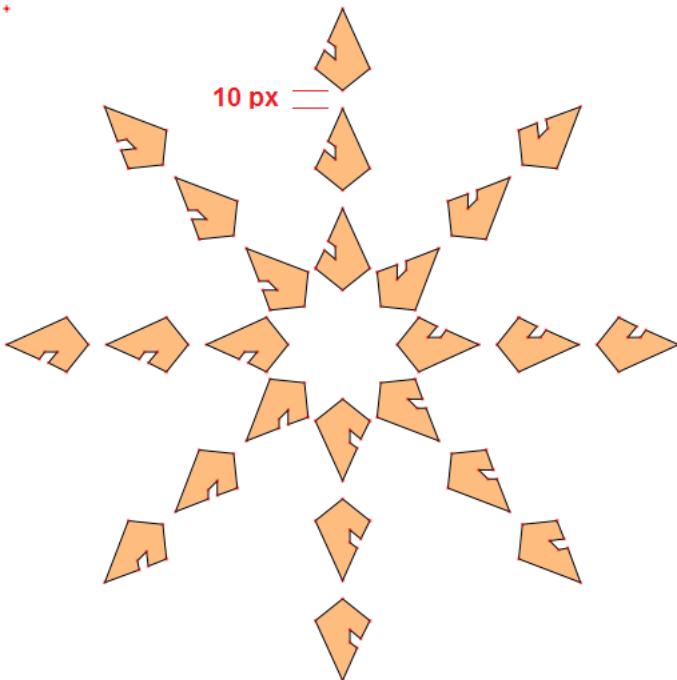
## Kara Effector - Effector Book [Tomo XVI]:

- **Ejemplo 5.** El parámetro **Dxy** es ahora una **tabla** con dos valores numéricos, el primero equivale al radio interior del primer Arreglo y el segundo, a la distancia en pixeles que separará a cada uno de los Arreglos:

Return [fx]:

```
shape.array( mi_shape, {8, 3}, "radial", {30, 10} )
```

- loops del Arreglo: 8
- Repeticiones: 3
- loops Total:  $8 \times 3 = 24$
- Modo: "radial"
- Radio Interior del primer Arreglo: 30 px
- Distancia Separadora entre Arreglos: 10 px



Para los siguientes ejemplos convertiremos a la variable **mi\_shape** en una **tabla**, en donde el primer elemento será la shape que inicialmente ya teníamos y el segundo será un círculo de 24 px de diámetro:

Variables:

```
mi_shape = { "m 12 0 | 0 15 | 12 30 | 45 15 | 27 7 | 24 11 | 17 11 | 22 5 | 12 0 ", shape.size(shape.circle, 24) }
```

Círculo de 24 px

La **tabla mi\_shape** de ser completamente de Shapes para poder ser usada dentro la función, así que pueden usar las de estos ejemplos o las que ustedes quieran. La cantidad de Shapes de la **tabla** es ilimitada.

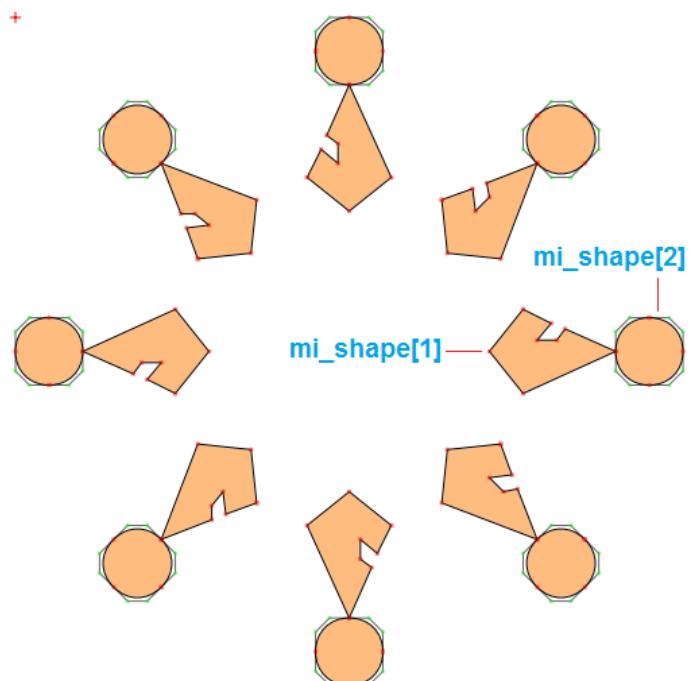
- **Ejemplo 6.** **mi\_shape** es una **tabla** de Shapes:

- loops del Arreglo: 8
- Repeticiones: 2
- loops Total:  $8 \times 2 = 16$
- Modo: "radial"
- Radio Interior del primer Arreglo: 50 px

Return [fx]:

```
shape.array( mi_shape, {8, 2}, "radial", 50 )
```

La función toma a la primera **shape** de la **tabla** y la repite en el primer Arreglo, luego toma la segunda para el segundo Arreglo, y así de manera sucesiva para el caso en que la **tabla mi\_shape** tenga todavía más elementos.



En la anterior imagen notamos cómo el primer Arreglo está hecho con las repeticiones de **mi\_shape[1]** (o sea, el primer elemento de la **tabla mi\_shape**) y para el segundo Arreglo se usó **mi\_shape[2]**.

La distancia que separa un Arreglo respecto a otro es 0 px, por default, pero también la podemos modificar.

## Kara Effector - Effector Book [Tomo XVI]:

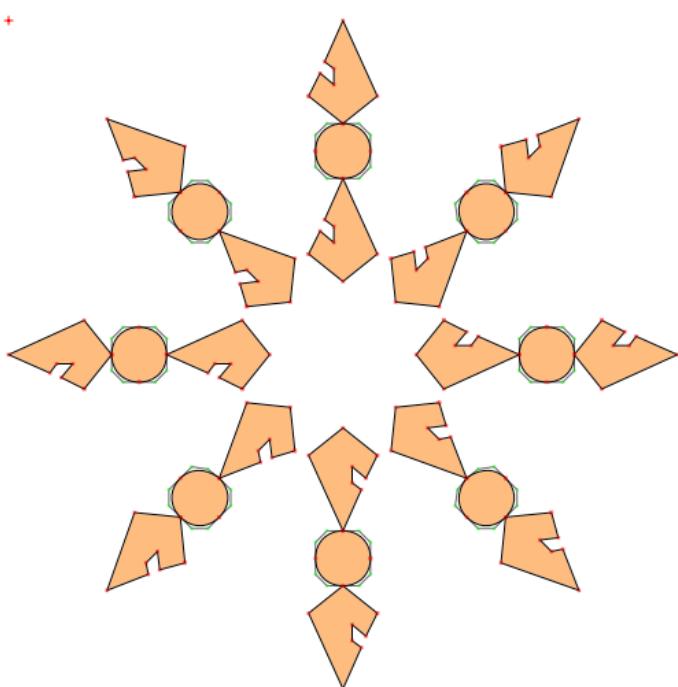
- Ejemplo 6.1. **mi\_shape** es una **tabla** de Shapes:

- loops del Arreglo: 8
- Repeticiones: 3
- loops Total:  $8 \times 3 = 24$
- Modo: "radial"
- Radio Interior del primer Arreglo: 40 px

Return [fx]:

```
shape.array( mi_shape, {8, 3}, "radial", 32 )
```

Este ejemplo está hecho para mostrar que no importa que la cantidad de repeticiones de los Arreglos asignada en la tabla **loops** (o sea 3) exceda a la cantidad total de Shapes en la tabla **mi\_shape** (`#mi_shape = 2`). La función tomará para el tercer Arreglo nuevamente a la primera **shape** de la tabla:

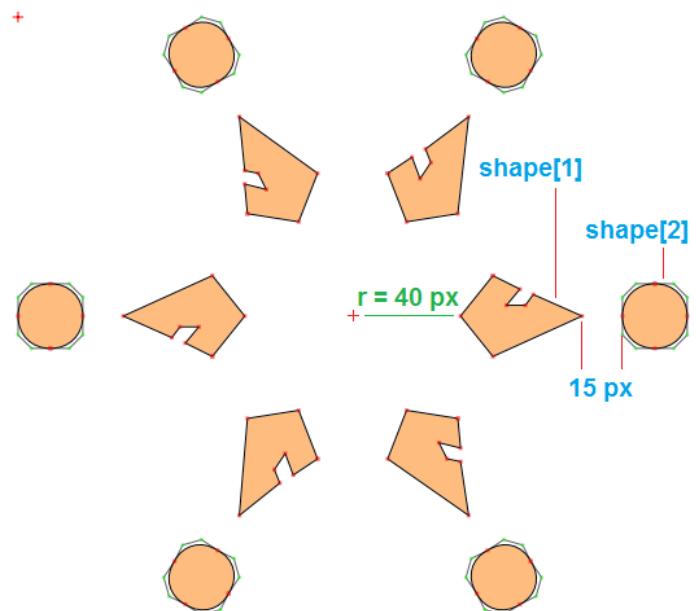


- Ejemplo 7. **mi\_shape** es una **tabla** de Shapes y en **Dxy**, en forma de **tabla**, modificamos la distancia que separará a cada uno de los Arreglos:

Return [fx]:

```
shape.array( mi_shape, {6, 2}, "radial", {40, 15} )
```

- loops del Arreglo: 6
- Repeticiones: 2
- loops Total:  $6 \times 2 = 12$
- Modo: "radial"
- Radio Interior del primer Arreglo: 40 px
- Distancia Separadora: 15 px



Acá es el final del **Tomo XVI**, pero la función **shape.array** aún tiene muchos más secretos que revelarnos y quedan muchos ejemplos para poner en práctica. Literalmente las opciones son infinitas.

En el **Tomo XVII** continuaremos con el tercer **Modo** de uso de la función **shape.array** y con las demás funciones de la Librería **shape**. Intenten poner en práctica todos los ejemplos vistos en este **Tomo** 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, exponer alguna duda o hacer alguna sugerencia. Pueden visitarnos y dejar su comentario en nuestra página de **Facebook**:

[www.facebook.com/karaeffector](http://www.facebook.com/karaeffector)

# Kara Effector 3.2: Effector Book Vol. I [Tomo XVIII]

---

## Kara Effector 3.2:

En el **Tomo XVII** seguiremos con los Ejemplos del Modo “radial” de la función **shape.array** de la Librería **shape**, y con el inicio del último Modo de dicha función.

### Librería Shape [KE]:

El **tomo XVI** había terminado con el Ejemplo 7 del Modo “radial”, en donde **mi\_shape** es una **tabla** en vez de una simple **shape**:

Variables:

```
mi_shape = {"m 12 0 | 0 15 | 12 30 | 45 15 | 27 7 | 24 11 |  
17 11 | 22 5 | 12 0 ", shape.size(shape.circle, 24)}
```

**shape.array( shape, loops, A\_mode, Dxy ):**

#### Modo Radial

- **Ejemplo 8.** Ingresamos la **tabla mi\_shape**, el **loops** vuelve a ser un valor numérico, que para este ejemplo es 6:

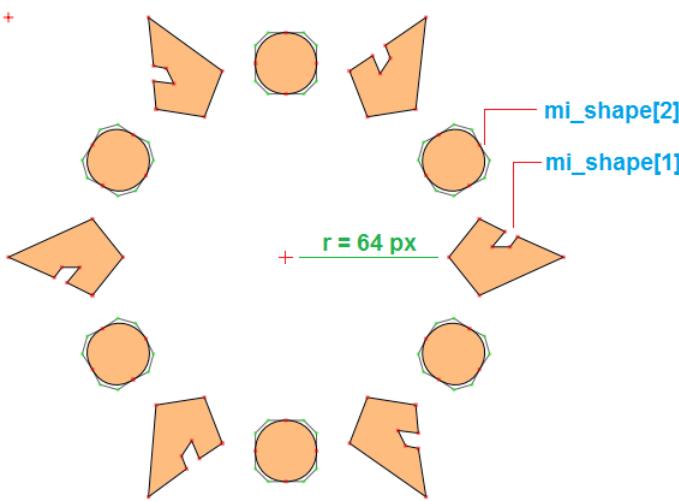
Return [fx]:

```
shape.array( mi_shape, 6, "radial", 64 )
```

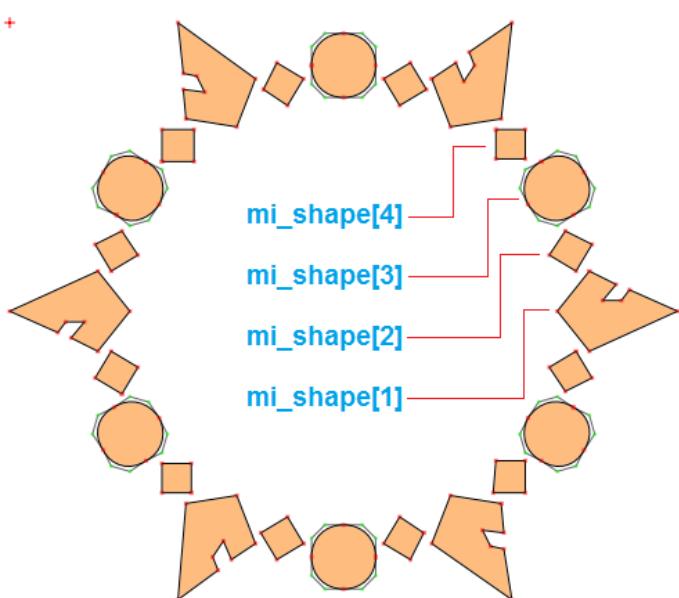
- loops de cada **shape** del Arreglo: 6
- Tamaño de la **tabla**: 2
- loops Total:  $6 \times 2 = 12$
- Modo: “radial”
- Radio Interior del Arreglo: 64 px

## Kara Effector - Effector Book [Tomo XVII]:

Entonces la función hace el **Arreglo Radial** alternando seis veces a cada uno, a los elementos de la **tabla mi\_shape**, a un radio de 64 px:



Acá otro ejemplo con una tabla de cuatro Shapes, en la que convenientemente las Shapes 2 y 4 son las mismas:



### Modo Matricial

Este modo consiste en hacer los **Arreglos** a modo de una **Matriz** rectangular con una cierta cantidad de repeticiones de la **shape** en ambas direcciones.

**Ejemplo 1.** Nuevamente iniciamos con **mi\_shape** como una **shape**, declarada en forma de variable. El parámetro **loops** siempre debe ser una **tabla** con dos valores numéricos en donde se dan la cantidad de repeticiones, el primer número indica

las repeticiones de la **shape** respecto al eje "x" (a lo ancho) y el segundo número indica la cantidad de repeticiones respeto al eje "y" (a lo alto). Por último, en **A\_mode** ponemos la palabra "**array**":

Variables:

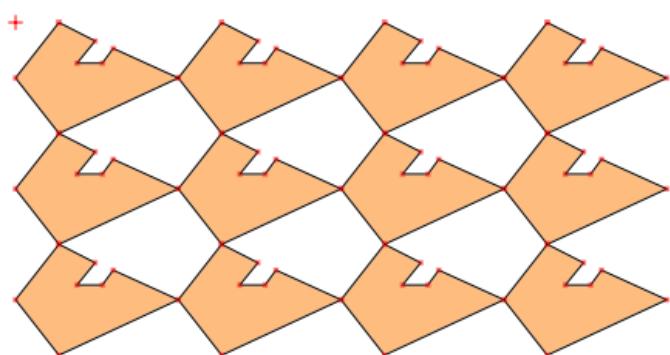
```
mi_shape = "m 12 0 | 0 15 | 12 30 | 45 15 | 27 7 | 24 11 |  
17 11 | 22 5 | 12 0 "
```

Y en **Return [fx]** ponemos así:

Return [fx]:

```
shape.array( mi_shape, {4, 3}, "array" )
```

Se generará el siguiente arreglo:



- loop Horizontal: 4
- loop Vertical: 3
- loops Total:  $4 \times 3 = 12$
- Modo: "array"
- Distancia Separadora: 0 px (por default)

**Ejemplo 2.** Usamos las mismas configuraciones del ejemplo anterior, pero ahora incluimos a **Dxy** en forma de valor numérico:

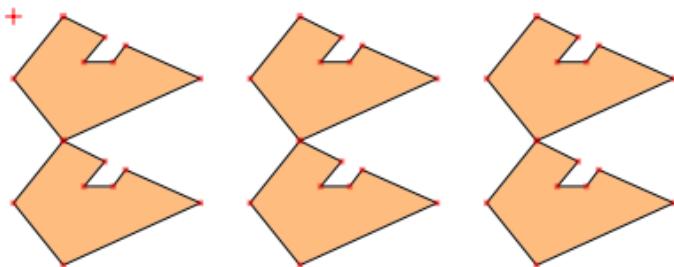
Return [fx]:

```
shape.array( mi_shape, {3, 2}, "array", 12 )
```

- loop Horizontal: 3
- loop Vertical: 2

## Kara Effector - Effector Book [Tomo XVII]:

- loops Total:  $3 \times 2 = 6$
- Modo: "array"
- Distancia Separadora Horizontal: 12 px
- Distancia Separadora Vertical: 0 px (default)

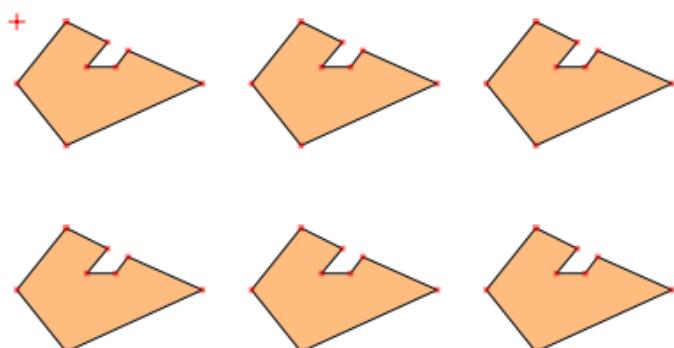


**Ejemplo 3.** Modificamos a **Dxy** a una **tabla** con dos valores numéricos, el primero indicará la distancia separadora horizontal y el segundo la vertical:

Return [fx]:

```
shape.array( mi_shape, {3, 2}, "array", {12, 20} )
```

- loop Horizontal: 3
- loop Vertical: 2
- loops Total:  $3 \times 2 = 6$
- Modo: "array"
- Distancia Separadora Horizontal: 12 px
- Distancia Separadora Vertical: 20 px

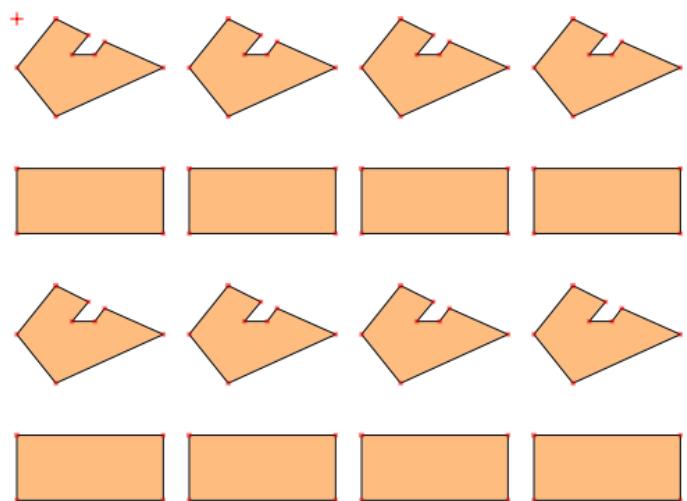


**Ejemplo 4.** **mi\_shape** la usamos ahora como una **tabla** de Shapes, la cantidad de elementos de dicha **tabla** es decidida por cada quien:

Return [fx]:

```
shape.array( mi_shape, {4, 4}, "array", {8, 16} )
```

Entonces las Shapes de la **tabla mi\_shape** se alternarán en el **Arreglo**, similar como pasaba en el **Arreglo Radial**:

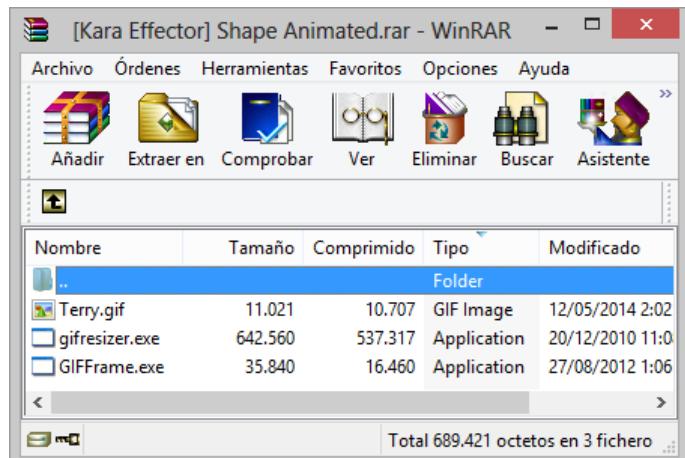


**shape.animated( dur, frame\_dur, frames, Sx, Sy):**  
retorna Shapes rectangulares que contienen imágenes en formato **PNG** para hacer efectos de animaciones.

Esta función es de uso exclusivo para el filtro **VSFilterMod**, ya que está basado en el tag **\1img**, que es el que hace posible el poder insertar imágenes en formato **PNG** en un archivo **.ass** y por ello no es posible poder visualizar los resultados si solo usamos el **VSFilter 2.39**.

Para el siguiente ejemplo he subido un **.rar** que contiene tres archivos, un archivo **GIF** y dos aplicaciones que no requieren de instalación, que nos ayudarán a manipular y modificar los archivos **GIF**:

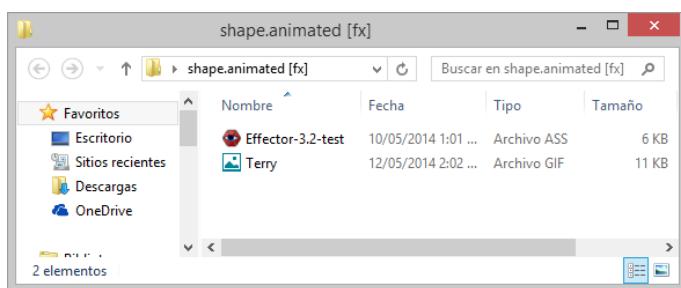
### [Kara Effector] Shape Animated



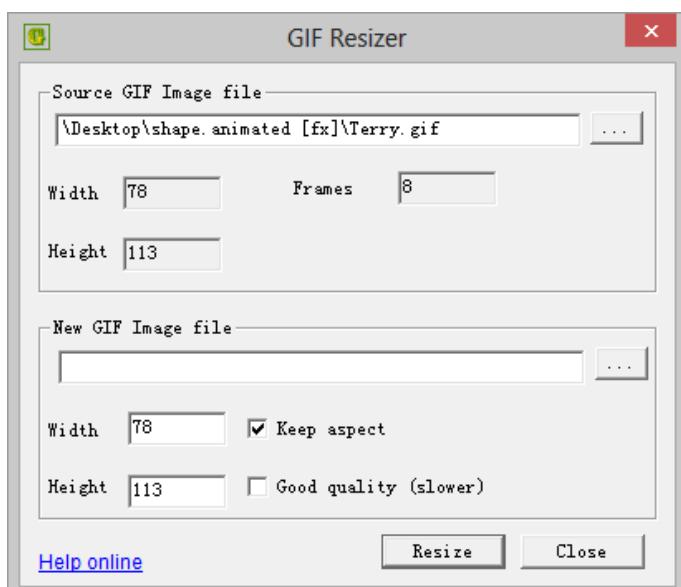
## Kara Effector - Effector Book [Tomo XVII]:

1. **Terry.gif:** es un ejemplo de imágenes animadas que pueden usar para un efecto en esta función.
2. **Gifresizer:** es una aplicación que nos permite dar las dimensiones a las imágenes que necesitemos, para que la animación se ajuste a las proporciones de las líneas karaoke y del vídeo usado.
3. **GIFFrame:** es una aplicación que extrae cada uno de los **frames** de un archivo **GIF**, en formato **PNG**, para poder insertarlas en nuestros karaokes.

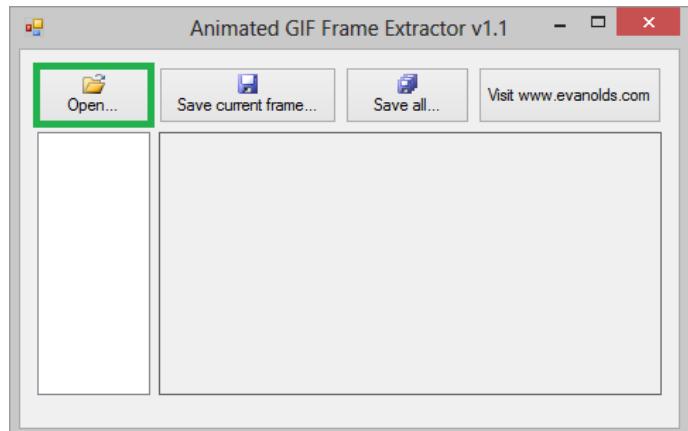
El primer paso para usar la función **shape.animated** es tener el archivo **.ass** y el archivo **GIF** en la misma carpeta. No importa el nombre ni la ubicación de la carpeta, lo que importa es que ambos archivos estén en la misma:



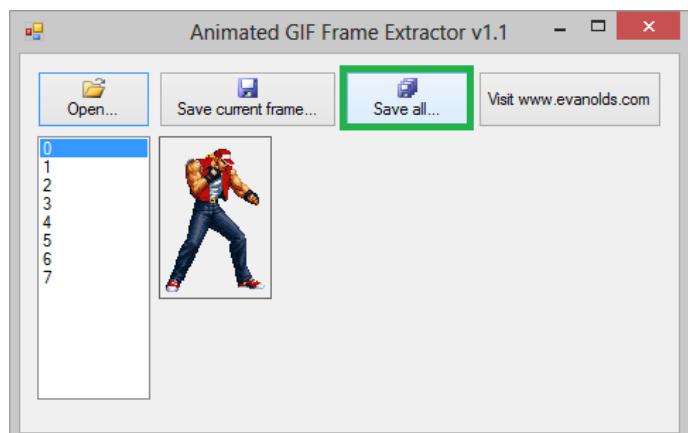
En el caso que nuestra **GIF** tenga un tamaño superior o inferior al que necesitamos (aunque no es recomendable ampliar un **GIF**, ya que la imagen pixela y empieza a perder calidad), abrimos la aplicación **GIFResizer** y en la parte inferior le damos las dimensiones en pixeles que se ajuste a nuestras necesidades. Esta aplicación crea un nuevo archivo, en tal caso se debe guardar en la misma carpeta del archivo **.ass** y del **GIF** original:



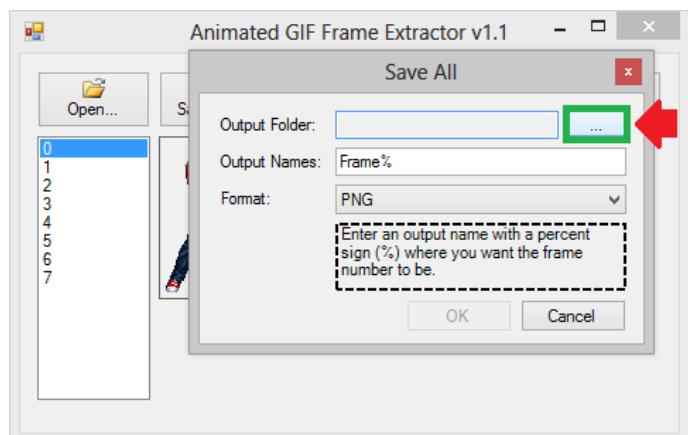
Para este ejemplo, no hubo la necesidad de cambiar el **GIF** de tamaño, pero en cualquiera de los dos casos, el tercer paso es abrir la aplicación **GIFFrame** y le damos al botón que pone “**Open...**”:



Ubicamos el **GIF** que vamos a utilizar, y pulsamos el botón “**Save All...**”:

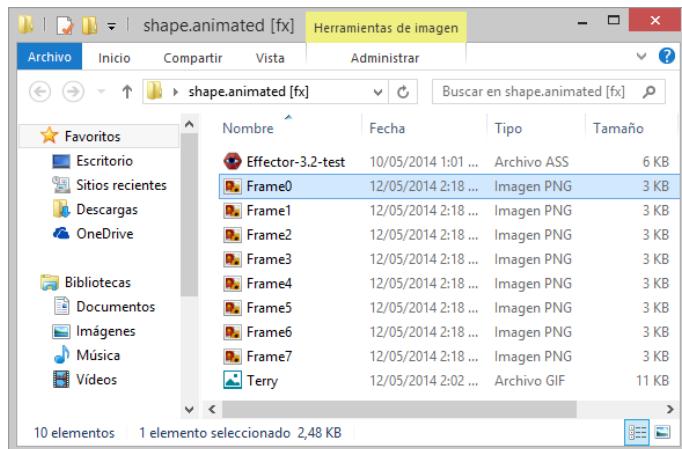


Y en donde pone “**Output Folder**” ubicamos la carpeta en donde están el archivo **.ass** que usaremos y el **GIF**, y en donde pone “**Format**” siempre debe poner “**PNG**”:

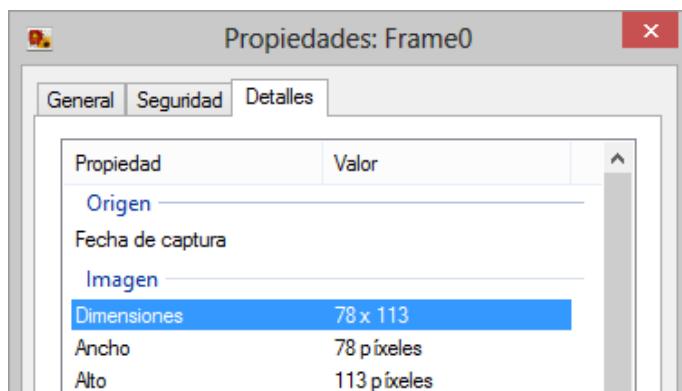


## Kara Effector - Effector Book [Tomo XVII]:

Entonces la aplicación **GIFFrame** extrae las imágenes **PNG** que componen al **GIF**, y éstas quedan en la carpeta en donde estaban el archivo **.ass** y el **GIF**:



Para confirmar las dimensiones de los archivos **PNG**, las podemos ver en sus propiedades (78 X 113 px):



Hasta este punto, podemos decir que hemos hechos todos los pasos preliminares para usar la función y generar un efecto de animación. Los pasos siguientes ya los debemos realizar desde el **Kara Effector** directamente, y uno de ellos es crear una tabla en la celda de texto “**Variables**” que contenga, entre comillas, el nombre de cada una de las imágenes y su extensión (**.png**):

```
Variables:  
  
frames = { "Frame0.png", "Frame1.png", "Frame2.png",  
          "Frame3.png", "Frame4.png", "Frame5.png",  
          "Frame6.png", "Frame7.png" };
```

Para este ejemplo, tan solo son ocho imágenes, desde la 0 hasta la 7, y en algunos casos hay archivos **GIF** que están compuestos de mucho más.

Y el segundo paso en la ventana de modificación del **Kara Effector** es llamar a la función **shape.animated** en la celda de texto **Return [fx]**:



1. En el primer parámetro de la función ponemos el tiempo total de duración de la animación, para este ejemplo he usado la variable **fx.dur**, que ya sabemos que es el tiempo de cada una de las líneas de efecto generadas.
2. En el segundo parámetro debemos poner la duración en milisegundos que tendrá cada uno de los **frames** de la animación de nuestro efecto. Para este ejemplo, **100** ms. Son recomendables valores entre 40 y 300 ms.
3. Para el tercer parámetro ponemos el nombre de la **tabla** declarada en “**Variables**”, que para este ejemplo se llama **frames**.
4. Y los parámetros cuatro y cinco son el ancho y el alto en pixeles de cada una de las imágenes **PNG**.

El resto de las configuraciones dependerá del efecto que queremos hacer, configuraciones como el **Template Type**, las posiciones y los tiempos. Para este ejemplo usé un **Template Type: Line**, con posición izquierda de la línea y con los tiempos por default de la misma:

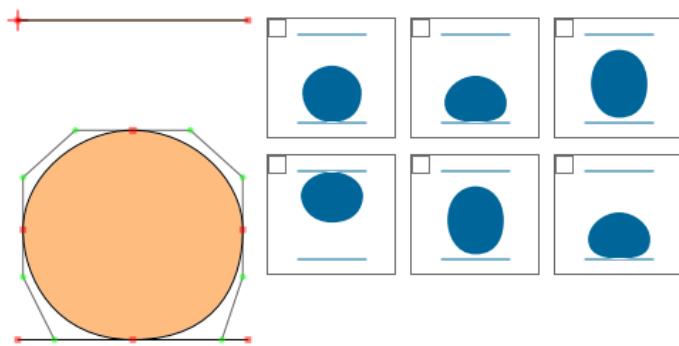


Para el caso en que a la derecha o en la parte superior de las imágenes de la animación se vea alguna parte de otra imagen, debemos reducir las medidas ingresadas, si por ejemplo se ve a la derecha, reducimos un poco al ancho.

## Kara Effector - Effector Book [Tomo XVII]:

**shape.animated2( dur, frame\_dur, shapes):** es similar a **shape.animated**, pero con la diferencia que no retorna imágenes **PNG** sino **Shapes**. Otra de las diferencias con su función homónima, es que solo son necesarios tres parámetros para que haga la animación: la duración total, la duración de cada **frame** y la tabla de **Shapes**.

El primer paso es dibujar la secuencia de **Shapes** que harán el efecto de animación. Para este ejemplo hice seis **Shapes** que simulan una pelota rebotando:



Las dos líneas horizontales paralelas extras se dibujan para delimitar la animación, es decir que el ancho de las líneas hacen referencia al ancho total y están a una distancia una de la otra, de tal manera que estas dos líneas delimitan el alto de la animación. En este ejemplo las dos líneas ayudan a dar la referencia del suelo, para la línea inferior, y del techo para la línea superior.

El segundo paso es copiar el código de las **Shapes** y hacer una **tabla** en la celda de texto “**Variables**” con ellas. Recordemos que las **Shapes** la debemos poner entre comillas, ya sean dobles o sencillas, y el nombre de la **tabla** es a gusto de cada quien:

```
Variables:  
shapes = {"m 0 0 144 0 m 0 61 144 61 m 22 21 b 11 21 1  
30 1 40 b 1 49 7 61 22 61 b 39 61 43 49 43 40 b 43 30 33  
21 22 21 ", "m 0 0 144 0 m 0 61 144 61 m 22 28 b 11 28 0  
37 0 47 b 0 56 7 61 22 61 b 39 61 44 56 44 47 b 44 37 33  
28 22 28 ", "m 0 0 144 0 m 0 61 144 61 m 22 10 b 11 10 2  
19 2 34 b 2 43 7 58 22 58 b 39 58 42 43 42 34 b 42 19 33  
10 22 10 ", "m 0 0 144 0 m 0 61 144 61 m 22 0 b 11 0 0 7  
0 17 b 0 25 7 36 22 36 b 39 36 44 25 44 17 b 44 7 33 0 22  
0 ", "m 0 0 144 0 m 0 61 144 61 m 22 10 b 11 10 2 19 2  
34 b 2 43 7 58 22 58 b 39 58 42 43 42 34 b 42 19 33 10  
22 10 ", "m 0 0 144 0 m 0 61 144 61 m 22 28 b 11 28 0 37  
0 47 b 0 56 7 61 22 61 b 39 61 44 56 44 47 b 44 37 33 28  
22 28 "}
```

Llamamos en “**Return [fx]**” a la función y como tercer parámetro ponemos a la tabla declarada en “**Variables**” que contiene a todas las **Shapes** de la animación:

```
Return [fx]:  
shape.animated2(fx.dur, 120, shapes)
```

Y la función generará la animación:



La ventaja de **shape.animation2** es poder hacer modificar fácilmente el tamaño de la misma ya que está hecha con **Shapes**, y éstas se modifican con los tags **\fscx** y **\fscy**.

Ya hemos avanzado mucho en la Librería **shape** y cada vez resta menos para terminar de ver todas sus funciones. En el **Tomo XVIII** continuaremos con el resto de las funciones hasta que las hayamos visto todas y consideraremos que ya dominamos un poco más el mundo de los efectos hechos con **Shapes**. Intenten poner en práctica todos los ejemplos vistos en este **Tomo** 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, exponer alguna duda o hacer alguna sugerencia. Pueden visitarnos y dejar su comentario en nuestra página de **Facebook**:

[www.facebook.com/karaeffector](http://www.facebook.com/karaeffector)

# Kara Effector 3.2: Effector Book Vol. I [Tomo XVIII]

# Kara Effector 3.2:

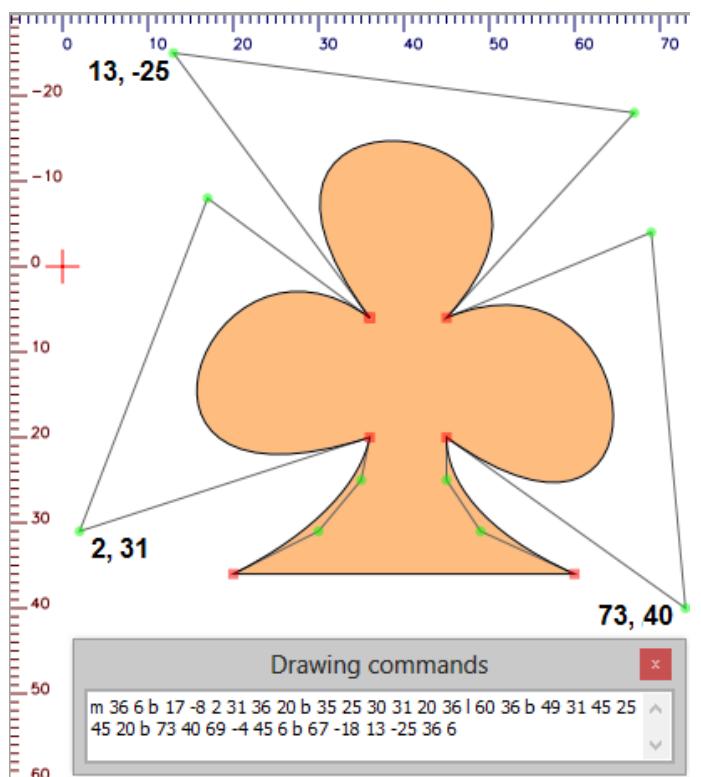
En el **Tomo XVIII** seguimos viendo el resto de las funciones de la Librería **shape** que hemos venido viendo desde hace ya varios Tomos. Habrán notado el gran tamaño de esta Librería y la importancia de la misma, ya que es una parte muy importante de todo lo que se debe saber para hacer efectos de alta calidad.

## Librería Shape [KE]:

**shape.config( Shape, Return, Ratio )**: esta función es similar a **shape.info**, pero con la ventaja que retorna mucha más información de la **shape** ingresada.

El parámetro **Ratio** es opcional y hace referencia a un valor por el cual se multiplicarán todos los puntos de la **shape** ingresada.

El parámetro **Return** es el que decide lo que retornará la función y tiene múltiples opciones:



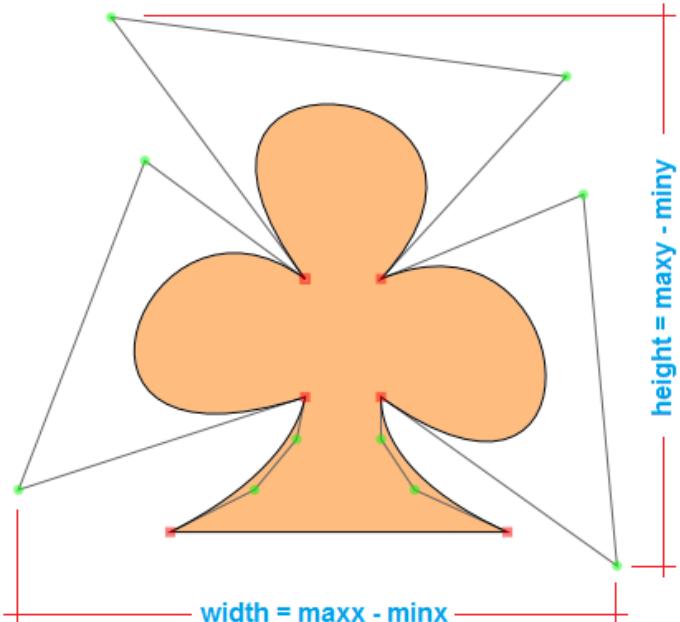
## Kara Effector - Effector Book [Tomo XVIII]:

Usaremos la anterior **shape** para los siguientes ejemplos, y como siempre (aunque no es obligatorio), la declararé a modo de variable en la celda de texto “**Variables**” para que sea un poco más simple su uso:

### Variables:

```
mi_shape = "m 36 6 b 17 -8 2 31 36 20 b 35 25 30 31  
20 36 l 60 36 b 49 31 45 25 45 20 b 73 40  
69 -4 45 6 b 67 -18 13 -25 36 6 "
```

- **shape.config( mi\_shape, “minx” )**: este modo retorna el mínimo valor de las coordenadas “x”. como se puede ver en la imagen, este valor es 2.
- **shape.config( mi\_shape, “maxx” )**: este modo retorna el máximo valor de las coordenadas “x”, que para esta **shape** es 73.
- **shape.config( mi\_shape, “miny” )**: este modo retorna el mínimo valor de las coordenadas “y”, que para esta **shape** es -25.
- **shape.config( mi\_shape, “maxy” )**: este modo retorna el máximo valor de las coordenadas “y”, que para esta **shape** es 40.
- **shape.config( mi\_shape, “width” )**: este modo retorna el ancho de la **shape** calculado como la diferencia entre **maxx** y **minx**:  $73 - 2 = 71 \text{ px}$

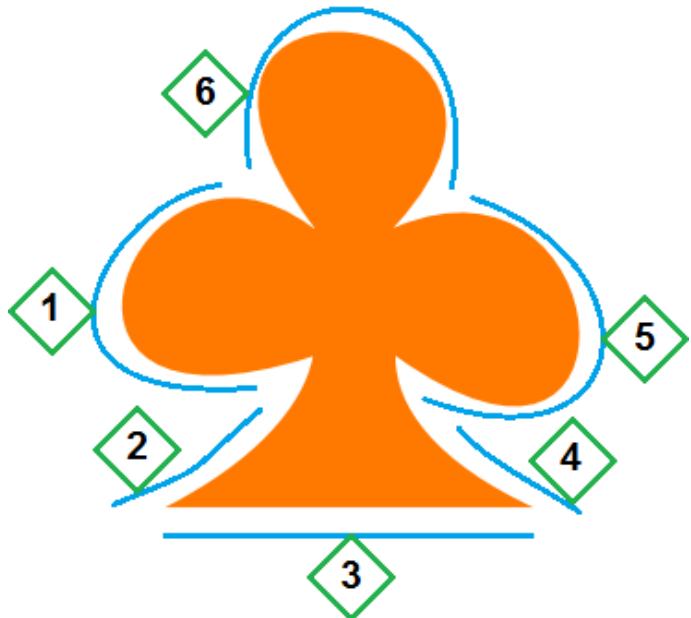


- **shape.config( mi\_shape, “height” )**: este modo retorna el alto de la **shape** calculado como la diferencia entre **maxy** y **miny**:  $40 - (-25) = 65 \text{ px}$

- **shape.config( mi\_shape, “length” )**: este modo retorna la medida de la longitud de la **shape**, es decir la medida de su perímetro:



- **shape.config( mi\_shape, “segments” )**: este modo retorna una **tabla** que contiene las coordenadas de cada uno de los segmentos de la **shape**. Dichas coordenadas están a su vez dentro de una **tabla**:



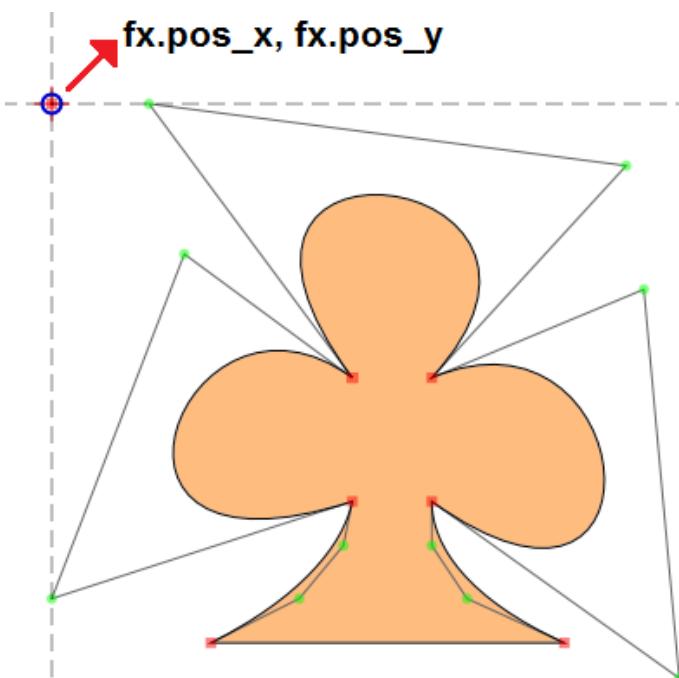
```
Segmentos = {  
[1] = {36, 6, 17, -8, 2, 31, 36, 20},  
[2] = {36, 20, 35, 25, 30, 31, 20, 36},  
[3] = {20, 36, 60, 36, },  
[4] = {60, 36, 49, 31, 45, 25, 45, 20},  
[5] = {45, 20, 73, 40, 69, -4, 45, 6 },  
[6] = {45, 6, 67, -18, 13, -25, 36, 6 }  
}
```

## Kara Effector - Effector Book [Tomo XVIII]:

- **shape.config( mi\_shape, "move" )**: este modo usa toda la información concerniente a la **shape** para generar una secuencia de movimientos a modo de efecto, que hacen que el objeto karaoke se mueva siguiendo la trayectoria de la **shape** ingresada. Por ejemplo, en un **Template Type: Line** usamos la función y hacemos algo como esto:

```
Add Tags: Add Tags Language: Lua
shape.config( mi_shape, "move" )
```

Lo que internamente hace la función es desplazar a la **shape** a un origen relativo, que ya no es el punto (0, 0) sino el centro en el vídeo del objeto karaoke:



Luego la función creará tantos **loops** de una misma línea, como segmentos tenga la **shape** ingresada en la función, que para este ejemplo son 6, como ya lo habíamos visto en el modo “**segments**”:

24	0	0:00:45.92	0:00:54.56	Dos corazones que se buscan conforman este sueño
25	0	0:00:02.43	0:00:08.16	*Mis mejillas se manchan con lágrimas de soledad
26	0	0:00:02.43	0:00:08.16	*Mis mejillas se manchan con lágrimas de soledad
27	0	0:00:02.43	0:00:08.16	*Mis mejillas se manchan con lágrimas de soledad
28	0	0:00:02.43	0:00:08.16	*Mis mejillas se manchan con lágrimas de soledad
29	0	0:00:02.43	0:00:08.16	*Mis mejillas se manchan con lágrimas de soledad
30	0	0:00:02.43	0:00:08.16	*Mis mejillas se manchan con lágrimas de soledad
31	0	0:00:08.33	0:00:13.19	*Pero puedo sentir la llegada del amanecer

De la anterior imagen se pueden apreciar los seis **loops** por cada línea de fx generada, y que todas ellas tienen los mismos tiempos de inicio y final, pero que gracias a una serie de transformaciones, éstas generan el efecto de movimiento continuo armónico a través del perímetro de la **shape**:

```
Comentar English lead-in 47
0 0:00:02.43 0:00:08.16 0:00:05.73 0 0 0
B I U S fn AB AB AB AB ✓ Tiempo C
[Kara Effector[fx] 3.2: ABC Template \an5\moves4(674, 669, 655, 655, 640, 694, 674, 683, 0, 1264.01)\alpha&HFF&\t(0, 1, 1a&H00&\3a&H00&\4a&H00&\t(1264.01, 1265.01)\alpha&HFF&) Mis mejillas se manchan con lágrimas de soledad
```

El tag **\alpha&HFF&** genera la invisibilidad, y luego los tags **\1a**, **\3a** y **\4a** dan la transparencia por default del objeto karaoke. Esta característica imposibilita que usemos estos mismos tags para cualquier otra cosa dentro del mismo efecto, porque pueden afectar las transformaciones.

Por cada segmento de la **shape** que esté conformado por una **curva Bezier**, la función retornará un tag **\moves4** para poder moverse a través de ella. Si el segmento de la **shape** es una recta, entonces la función retorna un tag **\move** para moverse de un punto a otro.

Como ya lo había mencionado anteriormente, la función genera el **loop** de forma automática dependiendo de los segmentos que contenga la **shape**, así que para generar un **loop** independiente de ése, debemos hacerlo de la siguiente manera:

```
loop = 1, 3
```

Sabemos que el 1 en este caso se pasa por alto, ya que la función generó un **loop** 6, y el 3 hace referencia ahora a la cantidad de repeticiones de esos 6 **loops**. O sea que el **loop** total será de  $6 \times 3 = 18$ , pero en pantalla, modificando un poco los tiempos de inicio y final, solo veremos los 3 que hemos asignado previamente en la celda de texto “**loop**”:

Mis mejillas se manchan con lágrimas de soledad

## Kara Effector - Effector Book [Tomo XVIII]:

Es algo complicado intentar explicar con palabras, incluso con imágenes, lo que esta función hace, ya que el efecto se basa en movimientos y en la sincronía de los mismos. Lo que recomiendo es que pongan la función en práctica con varias Shapes para poder notar las diferencias entre los resultados. Ejemplos

- `shape.config( shape.rectangle, "move" )`
- `shape.config( shape.triangle, "move" )`
- `shape.config( shape.circle, "move" )`

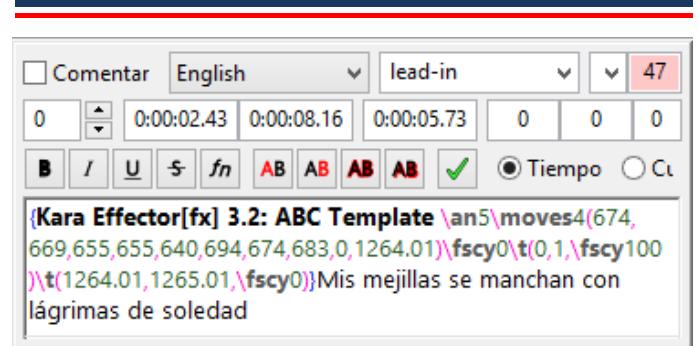
Este último ejemplo haría que el objeto karaoke se mueva siguiendo como trayectoria al perímetro de `shape.circle`, que tiene 100 px de diámetro, que es la medida de esta **shape predeterminada del Kara Effector**. Si quisieramos que un objeto karaoke se moviera siguiendo la trayectoria de un círculo más grande o más pequeño, tenemos las dos siguientes opciones. Ejemplos:

1. `shape.config( shape.circle, "move" , 1.5 )`: este modo hace que el **Ratio** (1.5) aumente el tamaño de la `shape` ingresada en un 150%, entonces el objeto karaoke se moverá siguiendo la trayectoria de un círculo de 150 px de diámetro.
2. `shape.config( shape.size(shape.circle, 72), "move" )`: la función `shape.size` redefine el tamaño del círculo a 72 px, que será el diámetro del círculo por el cual se moverá el objeto karaoke.

Entonces, podemos usar el tercer parámetro de la función `shape.config (Ratio)` o usar la función `shape.size`, para modificar las dimensiones de la `shape` ingresada y así redefinir las trayectorias de los desplazamientos.

Como les mencioné antes, el modo “**move**” usa los tags de transparencia para generar el efecto de fluidez en el movimiento del objeto karaoke, lo que imposibilitaba el volver usar dichos tags nuevamente en el mismo efecto. Si inevitablemente tuviéramos que usar estos tags, tenemos un modo más que lo hace posible:

- `shape.config( mi_shape, "move2" )`: este modo es similar al modo “**move**”, ya que hacen lo mismo, pero no usa los tags de transparencia en sus transformaciones. El tag que el modo “**move2**” usa para generar el efecto de movimiento fluido es el tag `\fscy`:



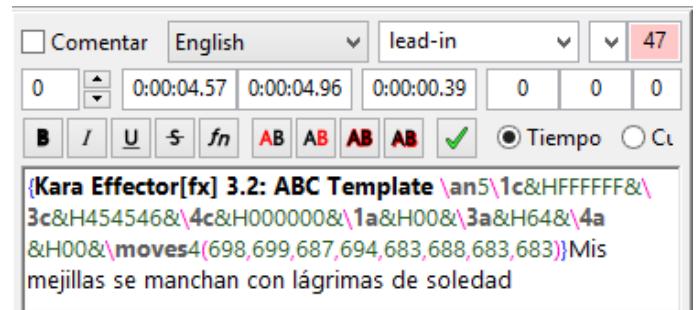
Entonces en el modo “**move2**” el tag que no se puede usar en el resto del efecto es `\fscy`, ya que es el que hace que cada uno de los loops desaparezca en el momento justo y dé la sensación de fluidez en el movimiento.

- `shape.config( mi_shape, "move3" )`: este modo es similar a los modos “**move**” y “**move2**”, genera el mismo efecto de movimiento, pero sin usar los tags de transparencias ni tampoco el tag `\fscy` en sus transformaciones. El modo “**move3**” genera líneas independientes respecto al tiempo, o sea que no necesita de un tag para hacer desaparecer al objeto karaoke que se mueve3 en determinado segmento.

25	0	0:00:02.43	0:00:03.52	English	lead-in	Effector [Fx]	*Mis mejillas se
26	0	0:00:03.52	0:00:03.91	English	lead-in	Effector [Fx]	*Mis mejillas se
27	0	0:00:03.91	0:00:04.57	English	lead-in	Effector [Fx]	*Mis mejillas se
28	0	0:00:04.57	0:00:04.96	English	lead-in	Effector [Fx]	*Mis mejillas se
29	0	0:00:04.96	0:00:06.09	English	lead-in	Effector [Fx]	*Mis mejillas se
30	0	0:00:06.09	0:00:07.39	English	lead-in	Effector [Fx]	*Mis mejillas se
31	0	0:00:07.39	0:00:08.16	English	lead-in	Effector [Fx]	*Mis mejillas se

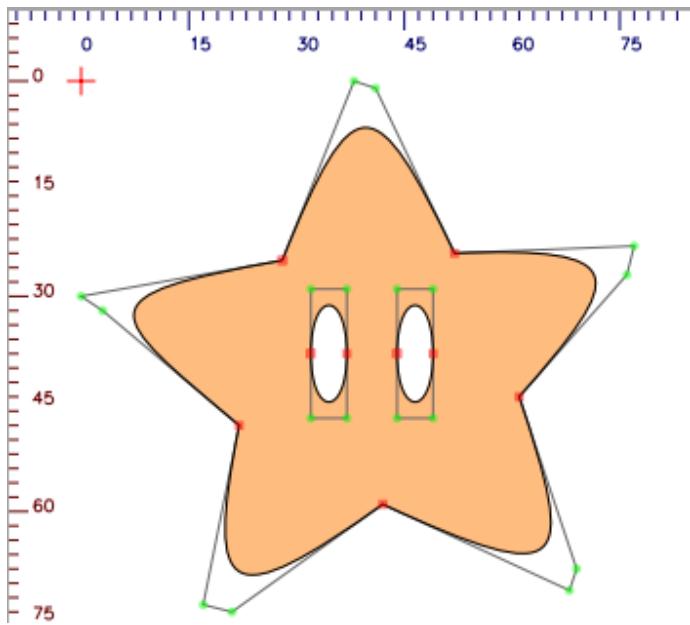
El modo “**move3**” estará disponible a partir de la **Versión 3.2.7** del **Kara Effector**, para versiones anteriores solo es posible usar los modos “**move**” y “**move2**”.

Al ver en detalle una de las líneas de fx generadas en este modo, notamos que el único tag que retorna es el de movimiento, `\move` para las restas de la `shape` y `\moves4` para las curvas `Bezier`, como en este ejemplo:



## Kara Effector - Effector Book [Tomo XVIII]:

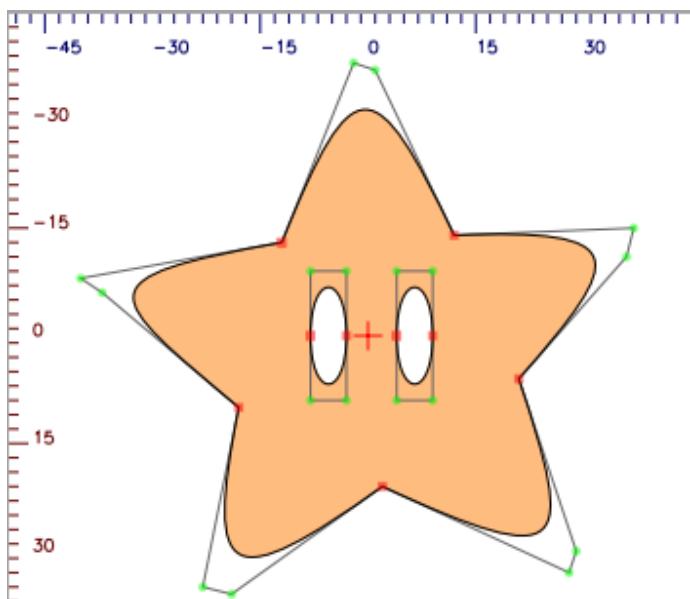
**shape.incenter( Shape )**: esta función desplaza a la **shape** ingresada en el centro de las coordenadas del **AssDraw3**, con referencia al punto P = (0, 0). Ejemplo:



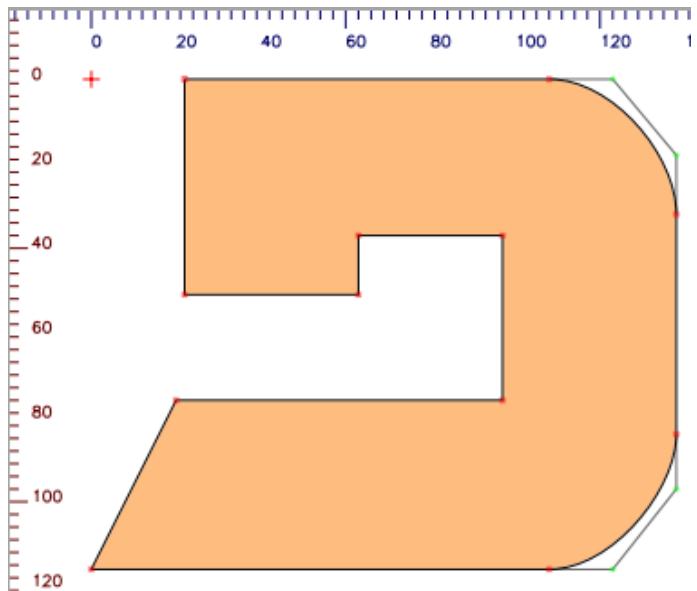
La anterior **shape** está ubicada en el **Cuadrante IV** del **AssDraw3**. Usamos la función con esta **shape**, por ejemplo en la celda de texto **Return [fx]**:

```
Return [fx]:  
shape.incenter( mi_shape )
```

Y al copiarla la **shape** generada y pegarla en el **AssDraw3**:



**shape.centerpos( Shape, Dx, Dy )**: desplaza a la **shape** ingresada con referencia a su centro, al punto con coordenadas P = (**Dx**, **Dy**). Ejemplo:

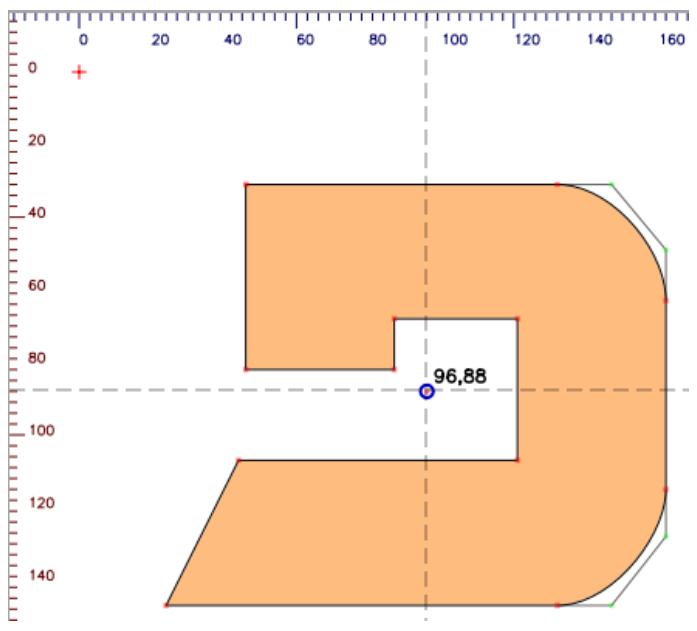


Ingresamos los dos valores que necesitamos desplazar la **shape** respecto a su centro, por ejemplo:

```
Return [fx]:  
shape.centerpos( mi_shape, 96, 88 )
```

**Dx**      **Dy**

Ahora la nueva ubicación del centro de la **shape** es el que se haya especificado en la función (96, 88):



## Kara Effector - Effector Book [Tomo XVIII]:

**shape.trajectory( loop, D\_nim, D\_max ):** crea una **shape** con una definida cantidad de segmentos (**loop**), que distan entre sí dependiendo de los parámetros **D\_min** y **D\_max** (Distancia mínima y Distancia máxima).

Cada uno de los segmentos es creado de forma aleatoria y dibujados con una **Curva Bezier** de tal manera que, entre todos los segmentos formen una sola trayectoria fluida con cada una de las curvas.

Los parámetros **D\_min** y **D\_max** son opcionales. En el caso de no ponerlos en la función, éstos tienen sus respectivos valores por default:

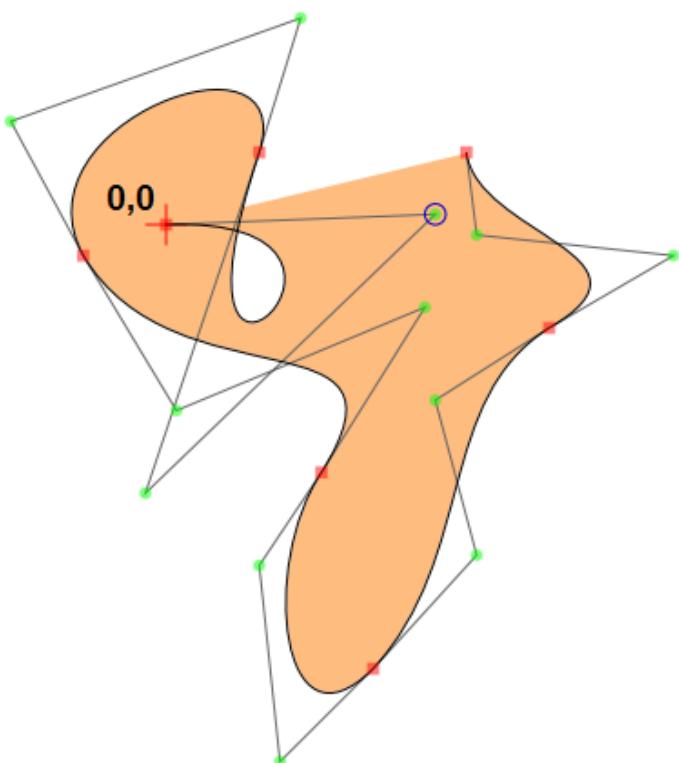
- **D\_min** = 10\*ratio
- **D\_max** = 20\*ratio

Ejemplo:

Return [fx]:

```
shape.trajectory( 5 )
```

Se genera una curva fluida con 5 segmentos de **Curvas Bezier**, y cada una de ellas separada de la otra a una distancia aleatoria entre 10 y 20 px:



Las curvas fluidas que genera esta función se pueden usar como una trayectoria de movimiento dentro de la función **shape.config** en los modos “**move**”, “**move2**” y “**move3**”. Ejemplos:

```
Add Tags: Add Tags Language: Lua
shape.config( shape.trajectory( 6, 25, 50 ), "move" )
```

```
Add Tags: Add Tags Language: Lua
shape.config( shape.trajectory( R(4,8) ), "move2" )
```

```
Add Tags: Add Tags Language: Lua
shape.config( shape.trajectory( 5 ), "move3" )
```

Y para cada uno de los ejemplos anteriores, la función **shape.config** hará que el objeto karaoke de la línea de fx se mueva siguiendo como trayectoria a la curva generada por la función **shape.trajectory**.

Las curvas fluidas que genera la función **shape.trajectory** tienen muchas más aplicaciones y opciones de uso. En los próximos tomos, de a poco iremos viendo cómo sacarle el máximo provecho a esta y otras funciones.

Es todo por ahora. En el **Tomo XIX** continuaremos viendo más de las funciones de la librería **shape**. Intenten poner en práctica todos los ejemplos vistos en este **Tomo** 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, exponer alguna duda o hacer alguna sugerencia. Pueden visitarnos y dejar su comentario en nuestra página de **Facebook**:

[www.facebook.com/karaeffector](http://www.facebook.com/karaeffector)

Kara Effector - Effector Book [Tomo XIX]:

## Kara Effector 3.2: Effector Book Vol. I [Tomo XIX]

## Kara Effector 3.2:

El **Tomo XIX** es otro más dedicado a la librería **shape**, que como ya habrán notado, es la más extensa hasta ahora vista en el **Kara Effector**. El tamaño de esta librería nos da una idea de la importancia de las Shapes en un efecto karaoke, y es por ello que debemos tomarnos un tiempo en ver y conocer a cada una de las funciones y recursos disponibles para poder dominarlas.

### Librería Shape [KE]:

**shape.Ltrajectory( length\_t, length\_c, height\_c )**: es una función similar a **shape.trajectory** con la diferencia de que crea la trayectoria en una sola dirección y con las siguientes especificaciones:

- **length\_t**: es la longitud lineal total de la trayectoria medida en pixeles. Su valor por default equivale a la diferencia entre **xres** y **fx.move\_x1**.
- **length\_c**: es la longitud lineal de cada uno de los segmentos de las **Curvas Bezier** que conforman a la trayectoria. Su valor por default es **xres/4**.
- **height\_c**: equivale a la mitad de la altura promedio máxima que tendrá cada una de las curvas de los segmentos. Su valor por default es **40\*ratio**.

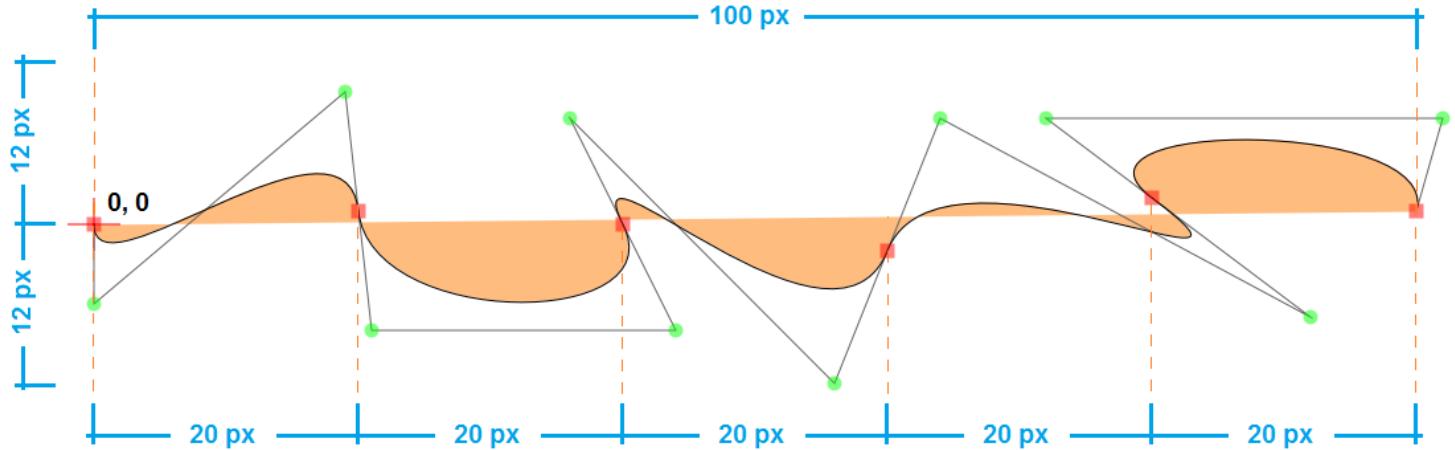
La función crea la trayectoria a partir del punto P = (0,0) y a 0° de dirección, o sea, hacia la derecha de dicho punto.

Ejemplo:

Return [fx]:

```
shape.Ltrajectory( 100, 20, 12 )
```

## Kara Effector - Effector Book [Tomo XIX]:



En la anterior imagen podemos ver una de las trayectorias creadas aleatoriamente, es fluida y sigue las condiciones de los parámetros ingresados en la función:

- Longitud lineal total: **100 px**
- Longitud lineal de los segmentos: **20 px**
- Máximo ascenso y descenso: **12 px**

Las ventajas que tiene cualquier trayectoria creada por una **shape**, es que las podemos modificar usando las funciones de dicha librería. Ejemplos:

- Modificar el ángulo:  
`shape.rotate( shape.Ltrajectory( 100, 20, 12 ), 60 )`
- Modificar el orden del trazado:  
`shape.reverse( shape.Ltrajectory( 100, 20, 12 ) )`
- Modificar el “Ratio” de alguna de las dimensiones:  
`shape.ratio(shape.Ltrajectory(100, 20, 12), 1, 0.5 )`

En fin, las opciones son muchas ya que también podemos combinar dos o más funciones de la librería **shape** para obtener nuevos resultados.

Ejemplo para poner en práctica:

```
Variables:
Trj = shape.reflect( shape.Ltrajectory(100,20,12), "y" )

Add Tags: Add Tags Language: Lua
shape.config( Trj, "move" )
```

**shape.Ctrajectory( Loop, r\_min, r\_max )**: crea una trayectoria con centro en el punto P = (0,0) y sin exceder como máximo al radio **r\_max**, ni como mínimo al radio **r\_min**, en donde ambos radios son ingresados en pixeles.

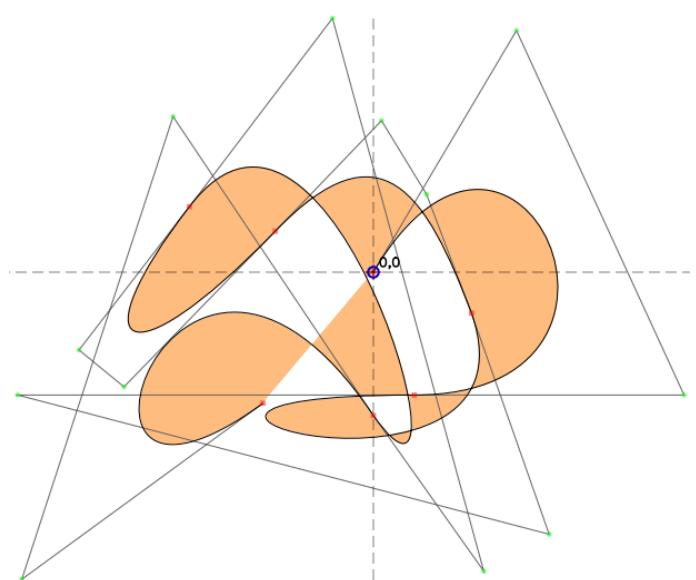
El parámetro **Loop** indica la cantidad de segmentos que tendrá la trayectoria final retornada y su valor por default es `line.duration/720`.

El valor por default de **r\_min** es `xres/40` y el de **r\_max** es `xres/25`, o sea que ambos pueden ser opcionales.

Ejemplo:

```
Return [fx]:
shape.Ctrajectory( 5, 20, 50 )
```

Veamos una de las trayectorias fluidas generadas:



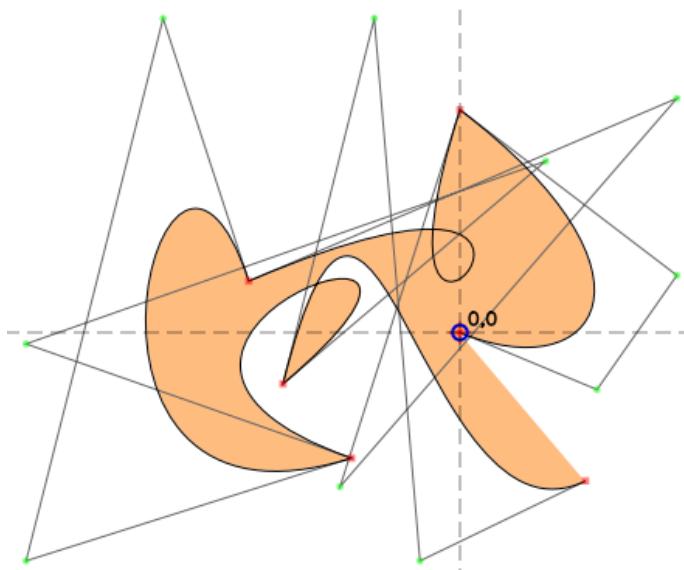
## Kara Effector - Effector Book [Tomo XIX]:

**shape.Rtrajectory( Loop, r\_min, r\_max ):** es una función similar a **shape.Ctrajectory**, pero la trayectoria que genera ya no es fluida sino totalmente aleatoria (random). Esta función crea una trayectoria con centro en el punto  $P = (0,0)$  y sin exceder como máximo al radio **r\_max**, ni como mínimo al radio **r\_min**, con ambos radios son ingresados en pixeles.

El parámetro **Loop** indica la cantidad de segmentos que tendrá la trayectoria final retornada y su valor por default es **line.duration/720**. El valor por default de **r\_min** es **xres/40** y el de **r\_max** es **xres/25**. Ejemplo:

```
Return [fx]:  
shape.Rtrajectory( 5, 30, 40 )
```

Y notamos que la trayectoria esta vez ya no es fluida:



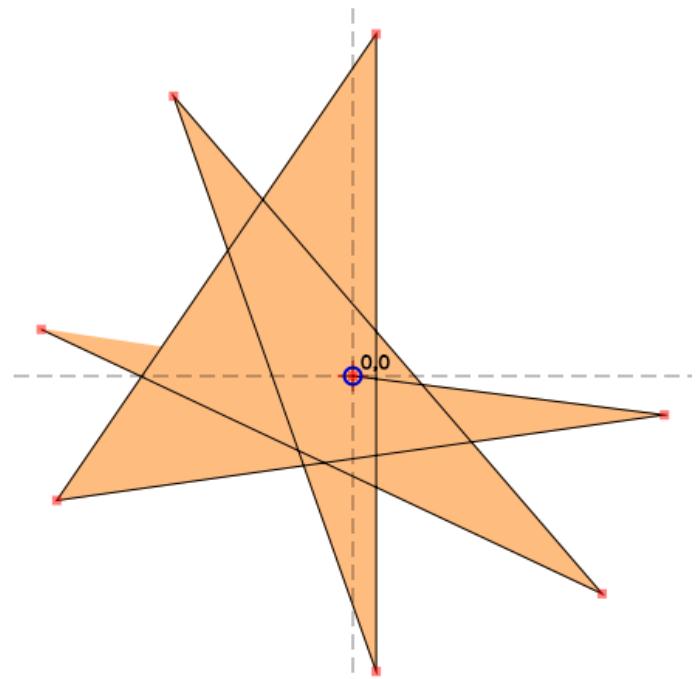
**shape.Strajectory( Loop, Radius ):** esta función es similar a **shape.Rtrajectory**, pero crea la trayectoria con segmentos lineales de forma aleatoria, en vez de usar las **Curvas Bezier** como en las cuatro anteriores funciones.

El parámetro **Loop** indica la cantidad total de segmentos lineales que conformarán la trayectoria y su valor por default es **line.duration/820**. El parámetro **Radius** indica la distancia a partir del punto  $P = (0,0)$ , de los extremos de los segmentos. Su valor por default es **0.75\*line.height**.

Ejemplo:

```
Return [fx]:  
shape.Strajectory( 7, 45 )
```

En la siguiente gráfica vemos cómo la trayectoria está conformada por siete segmentos rectos:



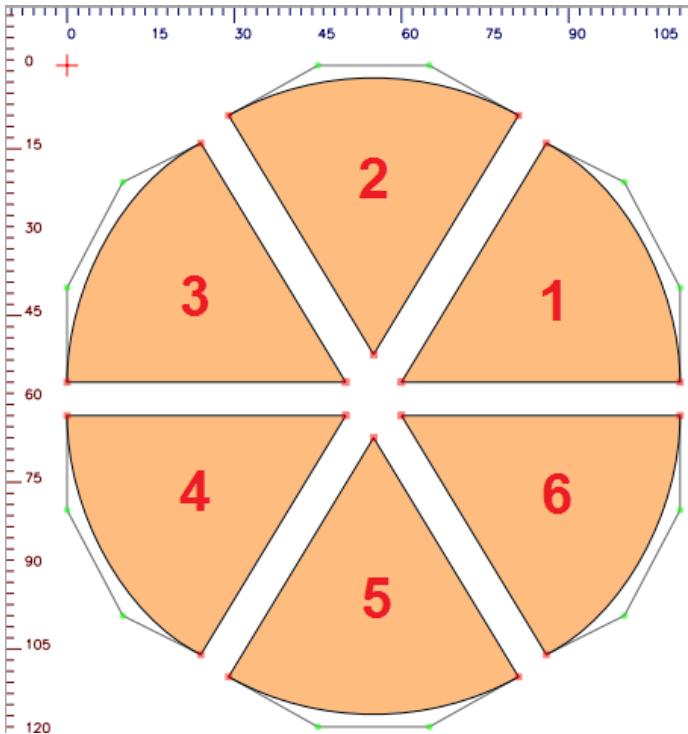
**shape.movevc( Shape, Rtn, width, height, x, y, Dx, Dy, t\_i, t\_f ):** es similar a la función **tag.movevc**, pero con la diferencia que a esta función se le ingresa una **shape** conformada por dos o más Shapes para ser usadas dentro de un tag **\clip**, que posteriormente serán manipuladas de forma individual por el tag **\movevc**.

- **Shape:** es la shape que está conformada por dos o más Shapes individuales.
- **Rtn:** este parámetro decide qué va a retornar la función y tiene tres opciones:
  - “**shape**”: retorna individualmente a cada una de la Shapes que conforma a la shape ingresada en la función.
  - “**loops**”: retorna en número equivalente a la cantidad de Shapes que conforman a la shape ingresada.

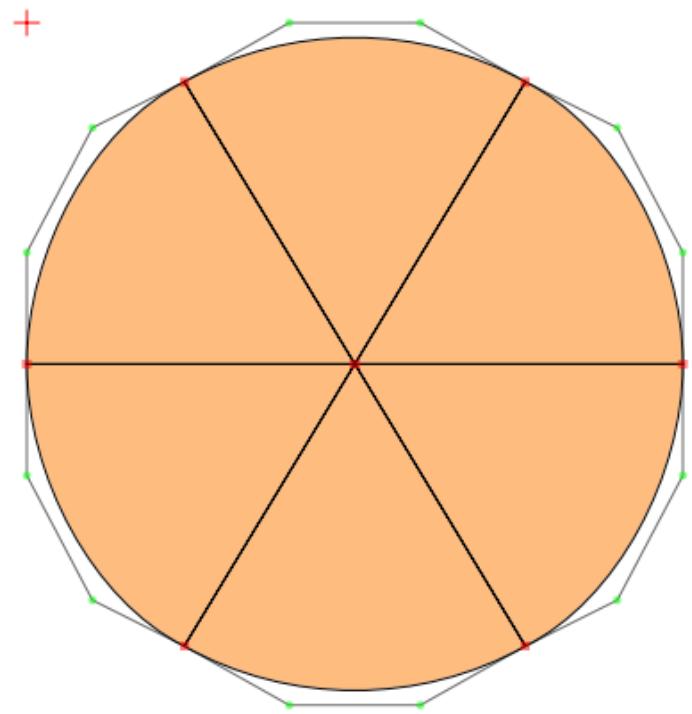
## Kara Effector - Effector Book [Tomo XIX]:

- “tag”: retorna una serie de tags equivalente al efecto generado por la función.
- **width:** es el ancho total que abarcarán todos los clip's que genere la función y su valor por default es **val\_width** (ancho del objeto karaoke: syl, word, char, line y demás. Depende del **Template Type**).
- **height:** es el alto total que abarcarán todos los clip's que genere la función y su valor por default es **val\_height** (alto del objeto karaoke: syl, word, char, line y demás. Depende del **Template Type**).
- **x, y:** son las coordenadas que ubicarán el centro de la **shape** ingresada respecto al vídeo. Sus valores por default son:
  - **x = fx.move\_x1**
  - **y = fx.move\_y1.**
- **Dx, Dy:** son las distancias medida en pixeles en las que se moverán cada uno de los clip's generados, respecto a ambos ejes. Sus valores por default son:
  - **Dx = fx.move\_x2 – fx.move\_x1**
  - **Dy = fx.move\_y2 – fx.move\_y1**
- **t\_i, t\_f:** son los tiempos de inicio y final de los movimientos de cada uno de los clip's. sus valores por default son:
  - **t\_i = fx.movet\_i**
  - **t\_f = fx.movet\_f**

Para el ejemplo, usaré el siguiente grupo de Shapes:



Son seis Shapes individuales en total, pero las he juntado de manera que aparenten ser una sola:



A continuación, usaremos el código de la anterior **shape** del **ASSDraw3**, para declarar una variable en la celda de texto “Variables”:

Variables:

```
Shapes = "m 50 52 | 100 52 b 100 35 90 16 76 9 | 50 52  
m 50 52 | 76 9 b 60 0 40 0 24 9 | 50 52 m 50 52 | 24 9 b  
10 16 0 35 0 52 | 50 52 m 50 52 | 0 52 b 0 69 10 88 24  
95 | 50 52 m 50 52 | 24 95 b 40 104 60 104 76 95 | 50  
52 m 50 52 | 76 95 b 90 88 100 69 100 52 | 50 52 "
```

He resaltado las letras “m” del código de la **shape** con el fin de poder identificar fácilmente a las seis Shapes individuales que conforman a toda la **shape**.

Para el ejemplo usaré un **Template Type: Syl** y la plantilla de efectos: **[001] ABC Template Hilight Syl**. Y en **Add Tags** llamaremos a la función usando casi todos sus parámetros por default, ya que todos ellos hacen referencia a valores ya ingresados, como las posiciones y los tiempos:

Add Tags:      Add Tags Language: **Lua**

```
shape.movevc( Shapes, "tag" )
```

## Kara Effector - Effector Book [Tomo XIX]:

Entonces la función generará automáticamente un loop equivalente a la cantidad total de Shapes individuales que conforman a la **shape** ingresada (o sea 6) y generará un clip por cada una de esas Shapes con las siguientes posiciones:



Es decir, como es un **Template Type: Syl**, generará seis líneas de fx por cada sílaba de cada línea a la que se le aplique el efecto:

24	0	0:00:45.92	0:00:54.56	English			Dos corazones
25	1	0:00:02.43	0:00:02.60	Romaji	hi-light	Effector [Fx]	*Ko
26	1	0:00:02.43	0:00:02.60	Romaji	hi-light	Effector [Fx]	*Ko
27	1	0:00:02.43	0:00:02.60	Romaji	hi-light	Effector [Fx]	*Ko
28	1	0:00:02.43	0:00:02.60	Romaji	hi-light	Effector [Fx]	*Ko
29	1	0:00:02.43	0:00:02.60	Romaji	hi-light	Effector [Fx]	*Ko
30	1	0:00:02.43	0:00:02.60	Romaji	hi-light	Effector [Fx]	*Ko
31	1	0:00:02.60	0:00:02.76	Romaji	hi-light	Effector [Fx]	*do

Pero como no le hemos ordenado ningún movimiento ni tampoco le hemos dado ubicaciones distintas a las que ya tiene por default, en el video veremos a cada sílaba de forma normal:



Pero si manualmente eliminamos a una de esas seis líneas, ya se verá la diferencia, ya que la sílaba que se ve en pantalla está formada por seis partes de la misma:



Al ampliar un segmento de los que conforman la sílaba, veremos algo como esto:



O sea que cada clip usa a una única **shape** para hacer visible una sección de la sílaba y el resto quedará invisible.

Ahora, para darle movimiento a los clip's, simplemente le damos movimiento al objeto karaoke, en este caso, a las sílabas:

Pos in 'X' =	fx.pos_x, fx.pos_x + 50
Pos in 'Y' =	fx.pos_y, fx.pos_y - 32
Times Move =	

Pero no tendría mucho sentido hacerlo de esta forma, ya que todos los clip's se moverán exactamente a la misma dirección y en el mismo tiempo. Entonces si queremos que los clip's se muevan a lugares distintos, debemos usar valores aleatorios (random) para dar la ilusión de que la sílaba se fragmenta en pedazos:

Pos in 'X' =	fx.pos_x, fx.pos_x + R(-40,40)
Pos in 'Y' =	fx.pos_y, fx.pos_y + R(-50,50)
Times Move =	

Así cada trozo se moverá a lugares distintos respecto a los otros, aunque aún lo seguirán haciendo al mismo tiempo, ya que los tiempos del movimiento se dejaron por default:



## Kara Effector - Effector Book [Tomo XIX]:

Siempre que queramos que los clip's se muevan al mismo lugar a dónde lo hará el objeto karaoke y al mismo tiempo que él, entonces lo que debemos hacer es usar la función solo con los dos primeros parámetros, como lo hicimos en el ejemplo anterior.

Es momento para recordarles la gran cantidad de recursos de la **Memoria RAM** que consumen los tags **\clip**, **\iclip** y **\movec**. Lo recomendable es no exceder un **loop** entre 20 o 25 en el efecto, es decir que la **shape** ingresada en la función esté conformada por, a lo máximo, 25 Shapes individuales.

Exceder las 25 Shapes individuales en la **shape** ingresada en la función hará que la computadora empiece a ponerse lenta a medida que se reproduce el efecto, también hará mucho más lento el proceso en encodeo.

---

**shape.movevci( Shape, Rtn, width, height, x, y, Dx, Dy, t\_i, t\_f )**: es similar a la función **shape.movevci**, pero con la diferencia que retorna iclip's en lugar de clip's como la anterior función.

---

**shape.multi1( Size, Px )**: crea una **shape** formada de múltiples Shapes cuadradas concéntricas para ser usada en las funciones **shape.movevc** y **shape.moveci**.

El parámetro **Size** indica las dimensiones máximas del cuadrado de mayor tamaño y su valor por default es equivalente a la mayor dimensión entre **val\_width** y **val\_height** del objeto karaoke, es decir:

- **Size = math.max( val\_width, val\_height )**

El parámetro **Px** equivale al ancho en pixeles de cada una de las Shapes cuadradas concéntricas que conforman a la **shape** que será retornada. Su valor por default es **4\*ratio**.

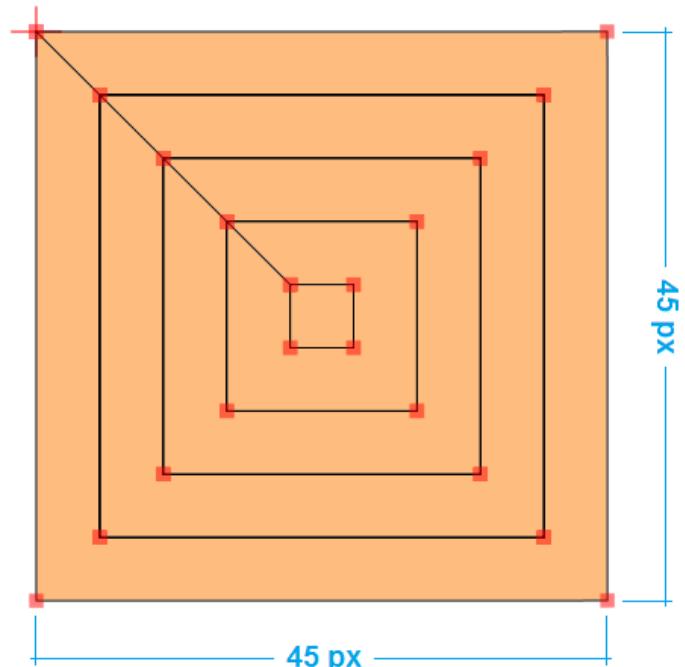
Ejemplo:

Return [fx]:

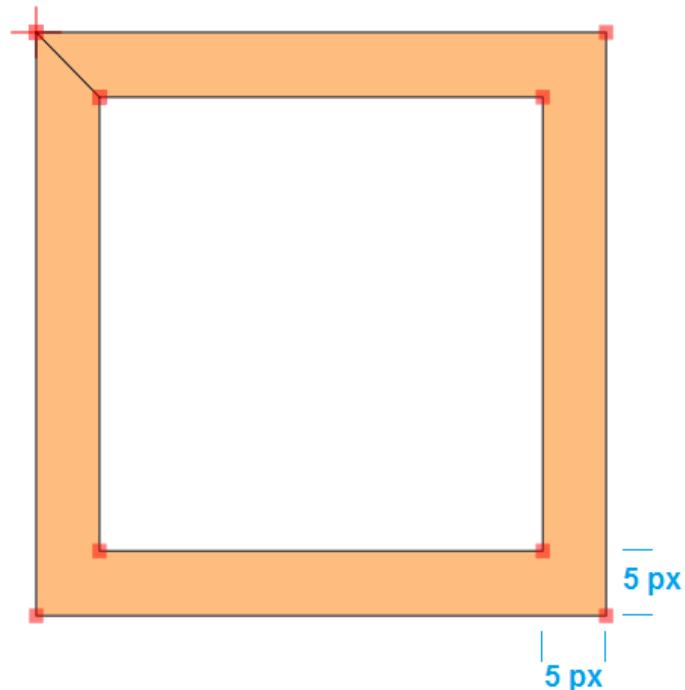
```
shape.multi1( 45, 5 )
```

---

Se generará la siguiente **shape**:



Vemos que las dimensiones de la **shape** son 45 X 45 px, y el ancho de cada una de las Shapes que la conforman es de 5 px:



Otra forma de acceder al valor por default del parámetro **Size** es escribiendo la palabra “**default**” en él, Ejemplos:

- **shape.multi1( “default”, 8 )**
- **shape.multi1( “default”, 2 )**

## Kara Effector - Effector Book [Tomo XIX]:

Entonces, para usar la función **shape.multi1** dentro de la función **shape.movevc** es recomendable usar el valor por default del parámetro **Size**, para que el objeto karaoke sea completamente visible en los clip's generados. Ejemplo:

```
Add Tags: Add Tags Language: Lua
shape.movevc( shape.multi1( "default", 4 ), "tag" )
```

Lo que generará los siguientes clip's:



La cantidad de clip's generados por la función dependerá de las dimensiones del objeto karaoke; entre más grande sea éste, mayor será la cantidad de clip's generados para poder abarcar toda la dimensión del objeto karaoke.

En algunos casos, el valor por default de **Size** no abarca completamente a las dimensiones del objeto karaoke, ya sea por un borde muy grueso, una sombra muy grande, un **blur** muy marcado u otros factores más; para estos casos, debemos sumar un valor extra que compense el tamaño total de la **shape** generada. Ejemplo:

```
Variables:
new_Size = math.max( val_width, val_height ) + 12
```

O sea que sumamos 12 px para compensar el tamaño de la **shape** generada. Luego usamos esta variable dentro de la función:

```
Add Tags: Add Tags Language: Lua
shape.movevc( shape.multi1( new_Size, 8 ), "tag" )
```

Usando un poco de imaginación, usamos las funciones de la librería **shape** para lograr nuevos resultados. Ejemplo:

```
Variables:
Shapes = shape.rotate( shape.multi1( ), 45 )
Add Tags: Add Tags Language: Lua
shape.movevc( Shapes, "tag" )
```



Es todo por ahora. En el **Tomo XX** continuaremos viendo más de las funciones de la librería **shape**. Intenten poner en práctica todos los ejemplos vistos en este **Tomo** 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, exponer alguna duda o hacer alguna sugerencia. Pueden visitarnos y dejar su comentario en nuestra página de **Facebook**:

[www.facebook.com/karaeffector](http://www.facebook.com/karaeffector)

# Kara Effector 3.2: Effector Book Vol. I [Tomo XX]

# Kara Effector 3.2:

El **Tomo XX** es otro más dedicado a la librería **shape**, que como ya habrán notado, es la más extensa hasta ahora vista en el **Kara Effector**. El tamaño de esta librería nos da una idea de la importancia de las Shapes en un efecto karaoke, y es por ello que debemos tomarnos un tiempo en ver y conocer a cada una de las funciones y recursos disponibles para poder dominarlas.

## Librería Shape [KE]:

**shape.multi2( width, height, Dxy )**: es una función similar a **shape.multi1**, ya que también genera una **shape** conformada por múltiples Shapes para ser usada en las funciones **shape.movevc** y **shape.movevci**.

Esta función genera Shapes diagonales con un ancho de **Dxy**, dentro del rectángulo de medidas **width X height**.

- **width**: ancho total de la shape generada.
- **height**: alto total de la shape generada.
- **Dxy**: ancho de las Shapes diagonales

El valor por default del parámetro **width** es **val\_width**, el de **height** es **val\_height**, y el de **Dxy** es **6\*ratio**:

- **width = val\_width**
- **height = val\_height**
- **Dxy = 6\*ratio**

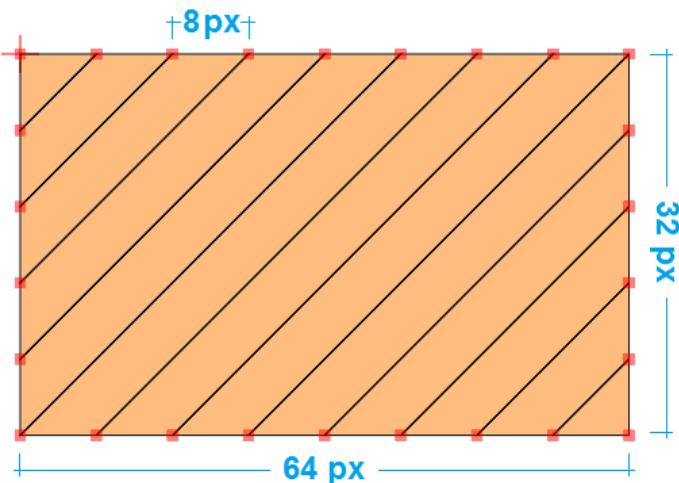
Ejemplo:

Return [fx]:

```
shape.multi2( 64, 32, 8 )
    Ancho   Alto   Grosor
```

## Kara Effector - Effector Book [Tomo XX]:

Lo que generará siguiente grupo de Shapes:



Para el próximo ejemplo usará un **Template Type: Word** y los siguientes parámetros en la función:

Variables:

```
Shapes = shape.multi2( word.width, word.height, 5 )
```

Y luego de declarar la anterior variable, la usamos en la función **shape.movevc**:

```
Add Tags: Add Tags Language: Lua
shape.movevc( Shapes, "tag" )
```

Lo que generará la siguiente serie de clip's:



Otra variante que podemos usar es:

```
Add Tags: Add Tags Language: Lua
shape.movevc( shape.reflect( Shapes, "y" ), "tag" )
```

Lo que invertiría el sentido de las diagonales:



El **loop** total de los clip's generados solo dependerá de las dimensiones del rectángulo, así como del ancho que le demos en la función a las diagonales:

Con el uso de más recursos de la librería **shape** se pueden lograr resultados más complejos como este:



**shape.multi3( Size, Dxy, Shape )**: retorna a una **shape** compuesta por Shapes concéntricas respecto a la **shape** ingresada (**Shape**) con un ancho de **Dxy**, y de un tamaño total **Size**.

- **Size**: tamaño total de la **shape** generada.
- **Dxy**: espesor de las Shapes concéntricas.
- **Shape**: **shape** ingresada.

El valor por default del parámetro **Size** equivale a la medida de la diagonal de un rectángulo de dimensiones **val\_width**, **val\_height**. El valor por default de **Dxy** es  $5 * \text{ratio}$  y el de **Shape** es **shape.circle**:

- **Size** = **math.distance( val\_width, val\_height )**



También se puede acceder a este valor si ponemos la palabra “**default**” en este parámetro.

## Kara Effector - Effector Book [Tomo XX]:

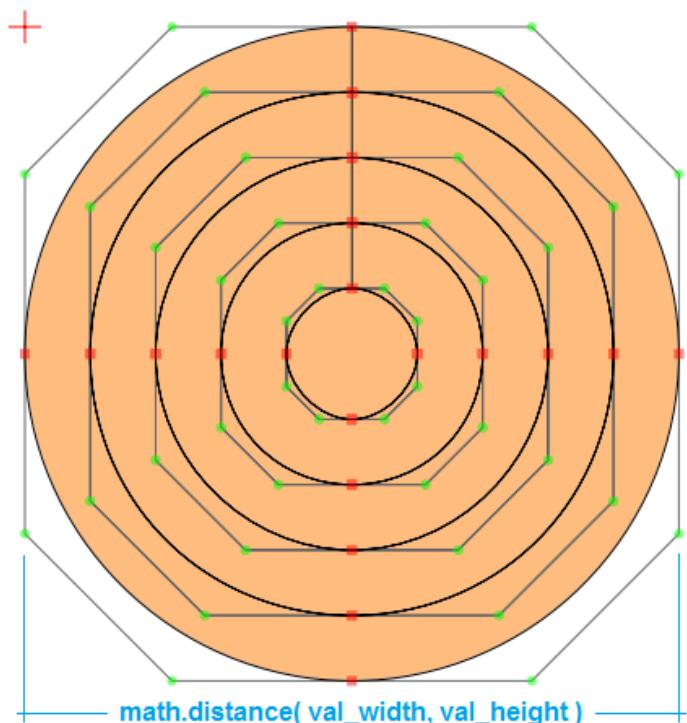
- **Dxy = 5\*ratio**
- **Shape = shape.circle**

Ejemplo 1:

Return [fx]:

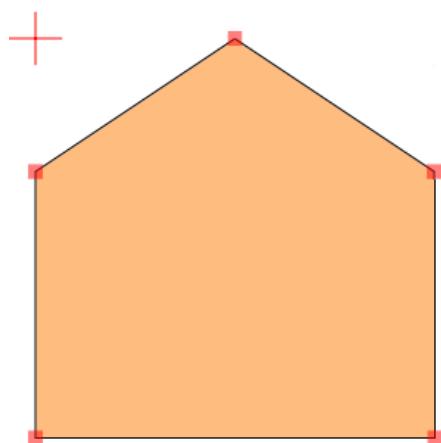
```
shape.multi3( "default", 8 )
```

Lo que generará círculos concéntricos de 8 px de espesor cada uno:



Ejemplo 2:

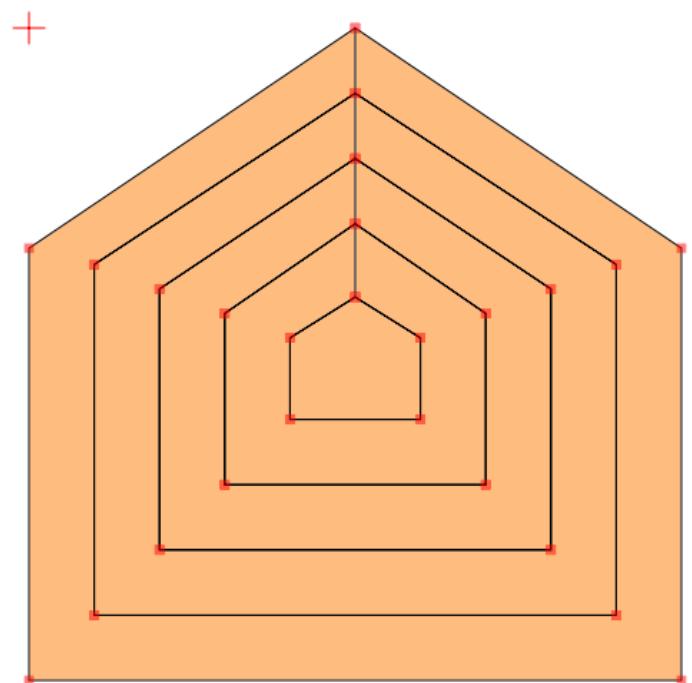
Para este ejemplo, usaremos la siguiente **shape**:



Con un espesor de 6 px:

Return [fx]:

```
shape.multi3( "default", 6, "m 15 0 1 0 10 1 0 30 1 30 30  
    | 30 10 1 15 0" )
```



O sea que las opciones son infinitas, ya que pueden duplicar de forma concéntrica a cualquier **shape** que se imagines. Como ya sabemos, el **loop** total depende de los valores ingresados en la función al igual que el tamaño del objeto karaoke. Una vez decididos los parámetros en la función, ya podemos usar la **shape** generada dentro de las funciones **shape.movevc** y/o **shape.movevci**.

---

**shape.multi4( Size, loop1, loop2 )**: esta función retorna un **Arreglo Radial** de tamaño total **Size** y con una cantidad de repeticiones, en principio determinada por el parámetro **loop1**.

Esta función solo está disponible para la versión **3.2.7** o superior del **Kara Effector**.

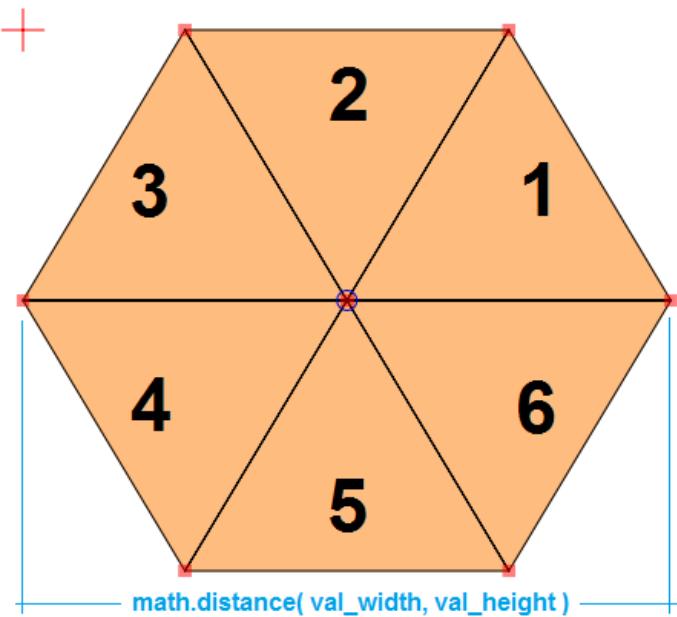
Sus valores por default son:

- **Size: math.distance( val\_width, val\_height )**
- **loop1: 6**
- **loop2: 1**

## Kara Effector - Effector Book [Tomo XX]:

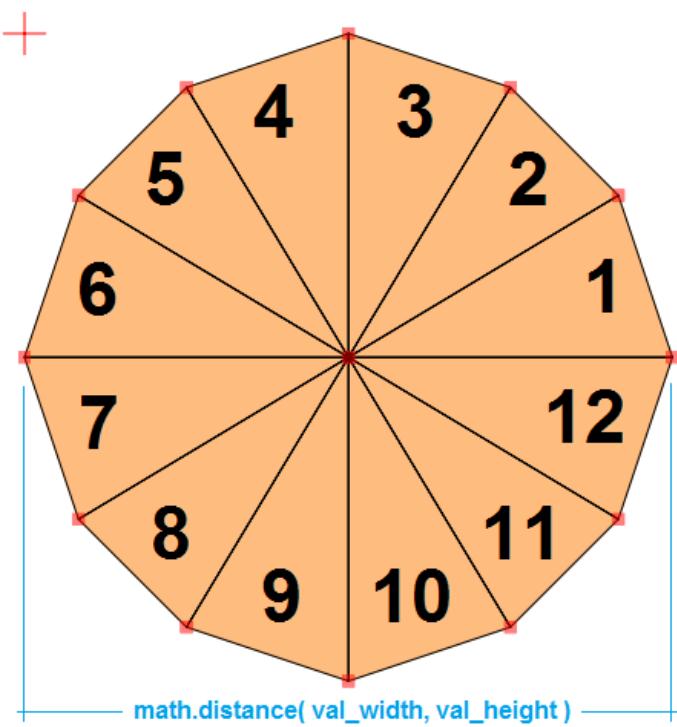
- **Ejemplo 1.** Todos los parámetros por default:

```
Return [fx]:  
shape.multi4()
```



- **Ejemplo 2.** Modificar a **loop1**:

```
Return [fx]:  
shape.multi4( "default", 12 )
```



Del Ejemplo 2 notamos cómo el ancho total de la **shape** es el asignado por default:

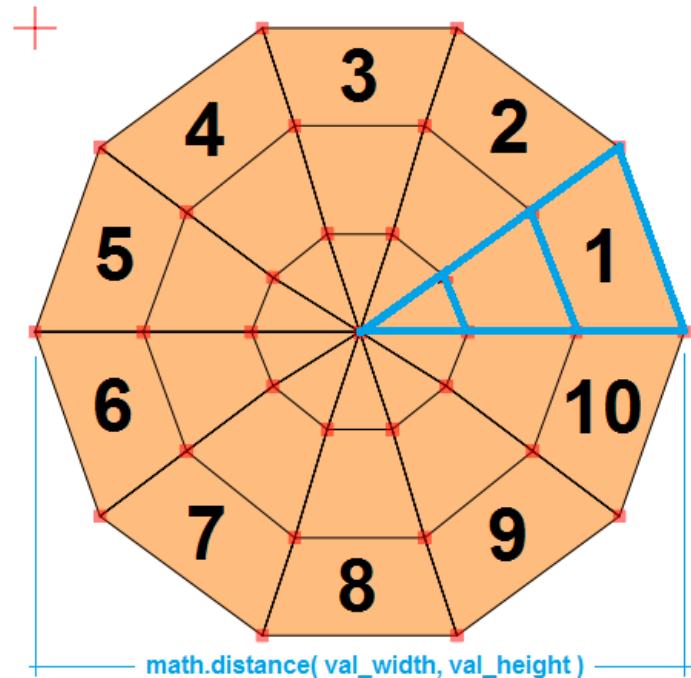
```
math.distance( val_width, val_height )
```

Observamos también que el **Arreglo Radial** está compuesto por doce Shapes individuales, y como **loop2** no está, toma su valor por default que es 1, así que el **loop** total sería de  $12 \times 1 = 12$ .

- **Ejemplo 3.** Modificar el parámetro **loop2**:

```
Return [fx]:  
shape.multi4( "default", 10, 3 )
```

Ahora el parámetro **loop2** es 3, lo que hace que el **Arreglo Radial** se multiplique tres veces:



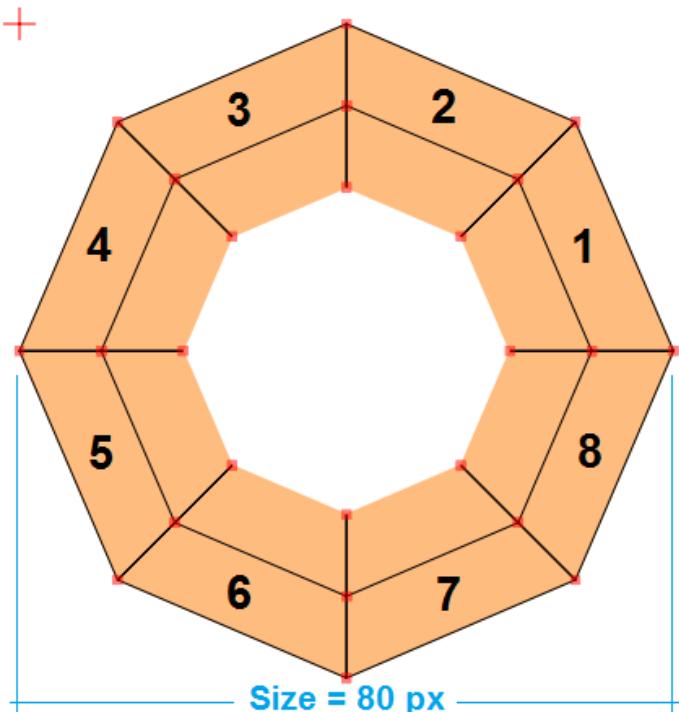
Como **loop1** es 10 en este ejemplo, entonces cada **Arreglo Radial** está compuesto por 10 Shapes individuales, que multiplicados por 3, de **loop2**, hace que el **loop** total sea de  $10 \times 3 = 30$ .

- **Ejemplo 4.** Usar un cuarto parámetro en la función que condiciona los anteriores resultados:

```
Return [fx]:  
shape.multi4( 80, 8, 4, 2 )
```

## Kara Effector - Effector Book [Tomo XX]:

- Size = 80 px
- loop1 = 8
- loop2 = 4
- Repeticiones a tener en cuenta: 2



Entonces, de las cuatro repeticiones del **Arreglo Radial**, solo se tendrán en cuenta las dos primeras, gracias al cuarto parámetro de la función **shape.multi4**.

Ya teniendo una mejor idea de cómo usar esta función, se nos hace un poco más simple poderla emplear dentro de la función **shape.movevc**. Ejemplo:

Variables:

```
Shape = shape.multi4( "default", 6, 2 )
shape.movevc( Shape, "tag" )
```

O sino, de forma directa:

```
shape.movevc( shape.multi4( "default", 6, 2 ), "tag" )
```

Crea un **Arreglo Radial** de 6 Shapes individuales y dicho Arreglo se repite 2 veces, para un total de 12 clip's:



Hecho este ejemplo, ya se podrán imaginar la gran cantidad de opciones que nos ofrece esta función. Todo es cuestión de ensayar y experimentar con las diversas combinaciones hasta que se familiaricen con esta y las demás funciones.

Es todo por ahora. En el **Tomo XXI** continuaremos viendo más de las funciones de la librería **shape**. Intenten poner en práctica todos los ejemplos vistos en este **Tomo** 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, exponer alguna duda o hacer alguna sugerencia. Pueden visitarnos y dejar su comentario en nuestra página de **Facebook**:

[www.facebook.com/karaeffector](http://www.facebook.com/karaeffector)