
# DreamBytes Adventures:
# Ad(M)inistrador (U)nificado de (S)on(I)do en Glul(X)

# « Damusix »


*DocV 3.2*

## Prólogo

Esta es la documentación de la extensión **Damusix**, creada por Eliuk Blau, para la programación de juegos de Ficción Interactiva en Inform 6. **Damusix** proporciona un *Administrador Unificado de Sonido en Glulx* capaz de gestionar todas las funciones necesarias para la reproducción de música y efectos de sonido en esta grandiosa máquina virtual. El Gestor Avanzado de **Damusix** encapsula todo el código necesario para el trabajo con el audio, permitiendo al programador hacer uso de estas funcionalidades mediante una interface de manejo sencilla.

## Introducción

La extensión **Damusix** puede considerarse, a grandes rasgos, como un completísimo "envoltorio" para todo el código que permite que la máquina virtual Glulx pueda reproducir audio.

Este "envoltorio" llamado **Damusix** encapsula en un sólo *objeto-gestor* todo el código y las llamadas a `glk()` pertinentes en la reproducción de música de fondo y/o sonidos sampleados; y divide este código en rutinas menores, de uso sencillo, que el programador puede utilizar en su juego. **Damusix** abstrae todo ese código casi críptico con `glk()` y le proporciona al programador una interface de uso bastante amigable para trabajar con el audio.

Es mi deseo que esta "intención de facilidad de uso" sea efectiva para quien se decida a utilizar mi extensión **Damusix**. =D

# Contenido

# Antes de Comenzar

La extensión **Damusix** requiere los siguientes componentes para funcionar:

- Compilador Inform v6.30 o superior
- Librería Inform 6/11
- Librería "`infglk.h`"

# Usando Damusix

En primer lugar, hay que dejar muy claro que **DAMUSIX NO ES BIPLATAFORMA**. Así que deberemos envolver cualquier código que le haga referencia con los habituales comandos de "compilación condicional" `#ifdef TARGET_GLULX` si queremos que nuestro juego compile también para la Máquina-Z. Por supuesto, si compilamos para Glulx notaremos los evidentes beneficios que supone tener un poco de musiquita de fondo y alguno que otro sonido por aquí y por allá que sorprenda al jugador cuando le dedique tiempo a nuestro juego.

El siguiente es un pequeño resumen técnico de lo que **Damusix** puede hacer. Se copia aquí un extracto del comentario en el propio código fuente de la extensión:

> El Gestor Avanzado de Audio de **Damusix** implementa 10 canales "normales" para la reproducción de sonidos con CONTROL TOTAL (tocarlos, detenerlos, volumen individual, repetición, etc.); 10 canales "virtuales" para una reproducción de sonidos más limitada, pero con la ventaja de no tener que "asignar un canal" previamente a un sonido concreto, permitiendo así que varios sonidos puedan tocarse al mismo tiempo sin interrumpirse mutuamente ni tener que "asignarlos" cada vez a un canal "normal" (el Gestor asignará los canales automáticamente); y 1 canal especialmente dedicado al trabajo con la "lista de reproducción de sonidos" que facilita **Damusix**. Esta "lista" tiene espacio suficiente para 10 sonidos que luego tocará en el orden en que hayan sido agregados, uno por uno. La "lista" puede ser útil para reproducir cadenas de sonidos.
>
> Adicionalmente, los sonidos asignados a canales "normales" pueden generar notificaciones del tipo evtype_SoundNotify que el programador podrá capturar en la rutina HandleGlkEvent() de la librería Inform.
>
> La extensión **Damusix** también permite realizar efectos de FadeIn y FadeOut con los sonidos asignados a un canal "normal". Estos efectos de Fade pueden ser o no en "tiempo-real". Un Fade en "tiempo-real" transcurrirá de fondo mientras el juego corre normalmente. Un Fade en "tiempo-no-real" hará primero el efecto y una vez haya finalizado, recién entonces devolverá el control de la ejecución al código a continuación. Los Fades en "tiempo-real" pueden ser abortados en cualquier momento si el programador lo necesita.

Resumen de Características de la extensión **Damusix**:

```
* 10 Canales “Normales” con Control Total del Audio
* 10 Canales “Virtuales” para sonidos sin “canal asignado”
* Lista de Reproducción de Sonidos (con espacio para 10 ítems)
* Control de Volumen Global del Gestor
* Control de Volumen Individual de cada Sonido
* Utilización con Abstracción por Sonidos o por Canales
* Efectos de FadeIn y FadeOut en “tiempo-real/tiempo-no-real”
* Activar/Desactivar el Audio limpiamente (sin cambiar volumen)
* Comprobación Automática de Soporte de Audio a nivel de Glk
* Mecanismo de “Protección de Sonidos” ante UNDO/RESTORE
* Muchas características más... =D
```

## Incluyendo la Extensión en el Juego

Para usar **Damusix** debemos hacer un `Include “Damusix”` justo después del `Include “Parser”` en el código principal de nuestro juego. En realidad, la inclusión de **Damusix** puede ir en cualquier parte del código, siempre que sea *después de incluir el Parser de Inform, pero antes del archivo de Gramática*. Entonces, en nuestro juego, la inclusión de **Damusix** debería quedar más o menos de la siguiente manera:

```
Constant Story "El Título de tu Juego";
Constant Headline "^El Subtítulo de tu Juego^";

Include "Parser";

! DreamBytes Adventures:
! Ad(M)inistrador (U)nificado de (S)on(I)do en Glul(X)
Include "Damusix";

! El archivo de recursos multimedia (los sonidos)
Include ">sonidos.bli";

Include "VerbLib";

[ Initialise ;
    ! Las instrucciones habituales en esta rutina, etc.
    ! ...
];

! Declaraciones de objetos, Rutinas, Clases, otras instrucciones, etc.
! ...

Include "SpanishG"; ! Libreria INFSP6: Gramatica Española

! Verbos del juego, Declaraciones de Gramatica, Rutinas de Acciones, etc.
! ...
```

Con esto ya estará incluida la extensión **Damusix** en el juego y *el Gestor de Audio se inicializará automáticamente*. No necesitamos escribir nada más. Por supuesto, ahora tenemos que aprender cómo manejar este Gestor. Lo trataremos en el apartado a continuación.

## Entendiendo el Concepto de “Canal de Audio”

Todo sonido que vaya a reproducirse mediante la API Glk de Glulx debe “sonar” por algún canal de audio. Un canal puede compararse con un “altavoz”: cada sonido que se reproduzca debe “sonar” por algún “altavoz”; es decir, todo sonido debe siempre tocarse en algún “canal de audio” previamente abierto para ello.

**Damusix** abre y administra automáticamente 21 canales de audio. De todos éstos, los primeros 10 canales son considerados por el Gestor como “canales normales” y permitirá realizar en ellos todo tipo de trabajos posibles con el audio que la API Glk implementa. Los siguientes 10 canales son considerados por el Gestor como “canales virtuales” y tienen una utilidad muy específica y más limitada que los canales normales (más adelante se explicará esta diferencia). El último canal gestionado por **Damusix**, el 21, existe solamente para la funcionalidad de la “lista de reproducción de sonidos” (también se explicará su utilidad más adelante).

Para manejar el Gestor de Audio de **Damusix** utilizaremos las rutinas específicas que éste implementa. El Gestor se ha programado como un objeto de Inform para facilitar la idea de “unificación” del sistema. Este *objeto-gestor* lo contiene todo: desde los estados de reproducción del *kernel* de **Damusix**, hasta cada una de las rutinas con las cuales trabajaremos.

## Comprobando si el Intérprete Soporta Audio

**Damusix** comprueba automáticamente si el intérprete en el que se ejecuta nuestro juego tiene **soporte completo de audio** o si no lo tiene. *“Soporte completo de audio” significa para **Damusix** “la posibilidad de reproducir archivos de audio en formato MOD y en formato sampleado AIFF u Ogg Vorbis”.*

AIFF es un formato de audio creado por Apple, muy usado en el mundo de Mac OSX (es equivalente al formato WAV en los sistemas Windows y Linux). Este formato guarda las ondas de sonido como datos sin ninguna compresión, por lo que un archivo de 1 minuto de duración puede llegar a tener un tamaño de 10 Mbytes.

Ogg Vorbis es un formato moderno y de uso libre que “comprime” la información de audio para que ocupe menos tamaño. Sus tazas de compresión son variables, por lo que podemos “comprimir más” o “comprimir menos” un archivo, sacrificando con ello la calidad del audio resultante. Mientras más se comprima la información, mayor será la pérdida de calidad. Como sea, Ogg Vorbis es una “obra de arte” entre los formatos de compresión de audio con pérdida (muchísimo mejor que MP3) y los resultados son siempre de excelente calidad, aun cuando el audio haya sido altamente comprimido. Un archivo Ogg Vorbis de 1 minuto de duración en calidad normal (-q3) tendrá un tamaño relativo a 1 Mbyte, lo cual puede ser muy bueno si deseamos poner en nuestro juego auténticas “bandas sonoras”, jeje! =P

Los archivos de audio en formato MOD son los llamados “módulos musicales”. Se crearon antiguamente para el sistema Amiga, cuando la capacidad de almacenamiento y la potencia de cálculo de las computadoras no eran suficientes para guardar y reproducir audio “real” como se hace actualmente. Un módulo musical es parecido, en esencia, a un archivo MIDI: describe notas musicales y como deben ser tocadas. Pero también hace un agregado especial: asocia “samples” (efectos de sonido reales pregrabados) a cada nota. De esta manera, los sonidos usados en cada nota musical pueden ser muy variados, permitiendo una “instrumentalización” sin límites. El “sample” asociado a cada nota es matemáticamente modificado para sonar en un determinado tono. Por ejemplo, teniendo exclusivamente el “sample” de una tecla de piano en DO, se pueden obtener las notas de piano para la escala musical completa en todas las octavas. Como no se usa una “grabación completa”, sino que los “samples” son simplemente sonidos cortos, los archivos MOD pueden reproducir una melodía increíblemente larga (normalmente están hechos para tocarse infinitamente) teniendo tamaños muy pequeños... generalmente menores a los 300 Kbytes.

Debido a su reducido tamaño y a la capacidad de repetirse infinitamente “sin saltos”, se recomienda el uso de archivos MOD para la música de fondo de los juegos. Existen muchas variantes del formato MOD, pero la API Glk acepta actualmente las variantes más extendidas: el formato “ProTracker” (.mod), el primero y padre de todos los demás; el formato “FastTracker 2” (.xm), una versión muy mejorada del anterior y bastante usada porque permite tener mucha *polifonía* sin aumentar demasiado el tamaño del archivo; el formato “ScreamTracker 3” (.s3m), que es simplemente otra variante más; y el formato “Impulse Tracker” (.it), el más moderno y con mejor sonido de todos los mencionados, por lo que es habitual que tenga un tamaño algo superior que el resto.

No todos los intérpretes Glulx son capaces de reproducir cada una de las variantes de MOD indicadas, pero a la fecha *Windows Glulxe* y *Gargoyle* lo hacen correctamente. Además, ambos intérpretes tienen soporte completo de audio, por lo que se recomienda su uso para juegos programados usando **Damusix** (sobre todo *Gargoyle*, que se puede ejecutar tanto en Windows como en Linux).

Si la comprobación automática de **Damusix** indica que el intérprete **no tiene soporte completo de audio**, el Gestor **inhabilitará el funcionamiento** de todas sus rutinas. Lo anterior quiere decir que el juego funcionará normalmente, pero sin reproducir sonidos. Esto se ha programado así para darle al Gestor un comportamiento “inteligente”: si el audio no está del todo disponible, sus funciones simplemente no harán nada y ya (pero no provocarán un error).

Con independencia de la comprobación automática de **Damusix**, nosotros también podemos hacer una comprobación “aparte” si la necesitamos para nuestros propios fines (por ejemplo: para mostrar alguna advertencia si el audio no está soportado). El Gestor facilita una rutina utilitaria para este propósito: `Damusix.TestAudio()`. Esta rutina devolverá el valor `1` si el intérprete tiene soporte completo de audio y el valor `0` si no lo tiene.

## Trabajando con Sonidos “Gestionados”

La API Glk permite crear muchos canales de audio, pero el trabajo con ellos no va más allá de las funcionalidades absolutamente necesarias. Glk implementa rutinas para reproducir sonidos, cambiar su volumen y detenerlos. Sin embargo, cosas tan básicas como, por ejemplo, *averiguar el volumen de reproducción* de un sonido o *determinar si está sonando de fondo en algún momento concreto*, simplemente no son abarcadas por la API.

Pero **Damusix sí permite hacer todo esto** y una buena cantidad de cosas más. Para lograr estos objetivos, el Gestor de Audio posee un *kernel* (un núcleo) que actúa según los distintos *estados y valores asociados* de cada sonido que esté siendo “gestionado”. Por ejemplo, el *kernel* puede tocar o detener sonidos que hayan cambiado su “estado de reproducción”, o puede informarnos sobre el volumen y el *modo de repetición* actuales de un determinado sonido.

Por otro lado, cuando usamos las rutinas básicas de la API Glk para dotar de audio a nuestro juego, evidentemente tendremos que ser nosotros mismos los que nos encarguemos, por ejemplo, de cuidar que los sonidos se toquen cuando corresponda, que se recuperen correctamente luego de que el jugador use comandos como UNDO, etc. Podemos hacernos una idea de lo complicado que esto puede llegar a ser porque, al fin y al cabo, tendremos que “gestionar” nosotros el apropiado funcionamiento de todos los sonidos y de la característica de audio que le hemos dado al juego.

Con **Damusix**, es el propio Gestor el que se encarga de cuidar de la *estabilidad del Sistema de Audio*. Así, el Gestor hará todo el “trabajo feo”, por ejemplo: actualizar los sonidos luego de un comando UNDO, detener los sonidos actuales al momento de cargar una partida y comenzar a tocar la música que estaba oyéndose en el instante en que la partida fue guardada, junto con recuperar los distintos valores de “volumen” y “repetición” de todos los sonidos gestionados... y así otros trabajos por el estilo. **Damusix** pone a nuestra disposición todos estos beneficios.

Para que **Damusix** pueda gestionar un sonido, éste debe ser “asignado” a uno de los 10 canales normales que el Gestor habilita para ello. Llamamos a estos canales como “normales” por ningún motivo especial más que para *diferenciarlos* de los canales “virtuales”. Podemos trabajar directamente sólo con los canales normales, así que de ahora en adelante nos referiremos a ellos simplemente como “los canales”.

Cuando se asigna un sonido a un canal **se asocian a este sonido una serie de datos que describen algunos aspectos de su comportamiento**. Estos datos, como se mencionó antes, pueden ser su volumen inicial, las repeticiones, su estado de reproducción, etcétera. De esta manera, el Gestor de Audio “reconocerá” el sonido asignado y sabrá cómo debe trabajar con éste.

Si queremos ocupar tal o cual sonido, no es necesario que esté asignado a un canal en todo momento. Simplemente debe estar asignado ANTES que el Gestor necesite trabajar con aquel sonido. Así, por ejemplo, si tenemos 10 sonidos no es necesario que les asignemos canales a todos desde el principio. Sólo debemos procurar que antes de comenzar a trabajar con cualquiera de ellos, nuestro sonido elegido tenga previamente un canal asignado.

Los canales pueden asignarse en cualquier instante, así que si tenemos pocos sonidos podemos asignarles un canal a cada uno desde el mismo inicio del juego, por ejemplo en la rutina `Initialise()`. Y si, por el contrario, tenemos muchos sonidos, quizás lo mejor sea irles asignando un canal a medida que lo vayan necesitando.

*Para los siguientes ejemplos se asume que el programador sabe escribir archivos de recursos multimedia para su juego y utilizar algún empaquetador Blorb, por lo que esto no se explicará en la presente documentación.*

La rutina del Gestor para asignar canales se usa de la siguiente manera:

```
Damusix.AsignarCanal(SONIDO,CANAL,VOLUMEN,REPETICIONES)
```

Analicemos los argumentos pasados a esta rutina, uno a uno:

1. **`SONIDO`:** Es el sonido al que le vamos a asignar un canal. Usualmente corresponderá al nombre de algún recurso de sonido que hemos declarado en el archivo ".res" para Blorb (*en este ejemplo, usando IBLORB de L. Ross Raszewski*).

2. **`CANAL`:** Es el número del canal que vamos a asignar. Los canales son 10, pero siempre se cuentan del `0` al `9`.

3. **`VOLUMEN`:** Es un valor numérico que representa el porcentaje de volumen que se le asociará inicialmente al sonido. Aquí tenemos que parar un poco y examinar este dato con más detalle. En **Damusix** el volumen de los sonidos siempre se mide en porcentajes. El valor mínimo es `0%` y el máximo `100%`. Cada sonido puede tener su propio volumen, pero el Gestor también posee un "*Volumen Global*" que usará, si así lo deseamos, para todos los sonidos gestionados. Si cambiamos el *Volumen Global*, el Gestor también cambiará el volumen de todos los sonidos al mismo valor que tiene el nuevo *Volumen Global*. Esto es muy práctico si necesitamos modificar de pronto el volumen de todos los sonidos. Por otro lado, el *Volumen Global* también se usa como *valor por defecto* y algunas rutinas pueden ocuparlo para establecer el volumen de un sonido si no se les pasa un porcentaje válido. Cuando necesitemos que cualquier rutina que trabaje con volumen use el porcentaje de *Volumen Global* actual, lo indicaremos siempre con el valor `-1`. En el caso específico de la rutina `Damusix.AsignarCanal()`, si indicamos este valor como volumen, entonces se usará el *Volumen Global* actual como volumen inicial para el sonido asignado.

(**NOTA:** El *Volumen Global* siempre vale `100%` al comienzo de la ejecución del juego. Luego este valor puede ser cambiado. Si una rutina del Gestor utiliza el *Volumen Global*, siempre tomará su valor establecido actualmente al momento de la llamada.)

4. `REPETICIONES`**:** Es un valor numérico correspondiente a la cantidad de veces que deberá repetirse el sonido cuando se reproduzca. Éste es el único argumento opcional de la rutina `Damusix.AsignarCanal()` y si lo omitimos tomará el valor por defecto de `1` repetición (cuando tocamos un sonido lo normal es que suene al menos una vez). Por otro lado, tal vez necesitemos que el sonido se repita infinitamente, sobre todo si lo estamos ocupando como *música de fondo*. Para ello, simplemente debemos indicar el valor especial `-1` que significa que el sonido deberá repetirse una y otra vez, sin parar, hasta que nosotros mismos lo detengamos o hasta que lo haga alguna otra rutina del Gestor, si lo considera apropiado.

Además de todo lo anterior, hay algo muy importante que debemos tener presente: **CADA SONIDO SÓLO PUEDE TENER 1 CANAL ASIGNADO**. La extensión **Damusix *no funcionará correctamente*** si un sonido tiene más de un canal asignado. El Gestor "buscará el canal" del sonido con el que se debe trabajar y si se encuentra en la situación de que un sonido tiene más de un canal asignado, *se confundirá y no sabrá qué canal le corresponde realmente a dicho sonido*. Así que en **Damusix** cada sonido deberá estar asignado exclusivamente a un único canal al momento de trabajar con él.

Por supuesto, hay una forma un poco "sucia" de conseguir trabajar con un sonido que estuviera (*¡Dios no quiera! =D*) asignado a más de un sólo canal: se trata de recurrir a las llamadas "rutinas técnicas" del Gestor que trabajan directamente con los canales (no siguen la filosofía de "abstracción por sonidos"). Pero esto será explicado más adelante ya que corresponde a funcionamientos avanzados de **Damusix**. Con todo, si realmente necesitamos que un sonido se reproduzca varias veces al mismo tiempo... lo mejor será ocupar la funcionalidad de "reproducción virtual" que implementa el Gestor y que trataremos en breve.

La explicación sobre la asignación de canales puede provocar la falsa impresión de complejidad. Realmente el proceso de asignación es muy sencillo y trivial. Pero era necesario explicar con detalle cada uno de los argumentos de la rutina `Damusix.AsignarCanal()` para que se entienda correctamente la forma en que el Gestor maneja cada sonido junto con sus datos asociados (volumen y repeticiones).

Como se mencionó anteriormente, una vez que un sonido tiene su canal asignado, entonces ya podemos comenzar a trabajar con él. El trabajo más "evidente" es, por supuesto, reproducir el sonido. Vamos a examinar ahora un ejemplo para que pongamos en práctica, de una buena vez, todo lo que se ha explicado hasta ahora. =P

En un juego Glulx cargado a la multimedia, lo habitual será tener unas cuantas *músicas de fondo* (para las localidades, sean melodías o sonidos ambientales) y otra buena cantidad de *efectos de sonido* (que se toquen ante una determinada acción o al examinar algún objeto).

Supongamos que tenemos en nuestro juego una *música de fondo* llamada `BGMUSIC` y un *efecto de sonido* llamado `FXSOUND`. Como son apenas dos, les asignaremos inmediatamente sus correspondientes canales en la rutina `Initialise()` para ahorrarnos trabajo. Además, lanzaremos la reproducción de la *música de fondo* también al comienzo del juego.

El código para conseguir todo lo expuesto sería el siguiente:

```
[ Initialise ;
  ! Todo el codigo que necesitemos poner aqui para nuestro juego
  ! ...

  ! Asignamos los canales para nuestros distintos sonidos
  Damusix.AsignarCanal(BGMUSIC,0,-1,-1); ! asignar música-fondo al canal 0
  Damusix.AsignarCanal(FXSOUND,1,100);   ! asignar efecto-sonido al canal 1

  ! Comenzamos a reproducir ahora la musica de fondo
  Damusix.Tocar(BGMUSIC);

  ! El resto de otras cosas que queramos hacer en esta rutina
  ! ...
];
```

Revisemos lo que hace cada una de las tres llamadas al Gestor de **Damusix**:

`Damusix.AsignarCanal(BGMUSIC,0,-1,-1)` asignará el sonido `BGMUSIC` al canal número `0`, aplicándole el porcentaje de *Volumen Global* (`-1`) actual y *repeticiones infinitas* (`-1`).

`Damusix.AsignarCanal(FXSOUND,1,100)` hará lo mismo que la línea anterior, salvo que aquí el sonido es `FXSOUND` y se asignará al canal número `1`, aplicándole un porcentaje específico de volumen del `100%`. Además, por omisión del cuarto argumento, se asume 1 repetición para este sonido (se tocará una sola vez).

En este punto sería interesante hacer una nueva observación especial sobre la **utilidad del *Volumen Global***:

En la asignación de `BGMUSIC` usamos el *Volumen Global* actual. En cambio, en la asignación de `FXSOUND` usamos un porcentaje específico de volumen, concretamente `100%`. Al momento de la ejecución del juego ambos sonidos tendrán `100%` de volumen. ***¿Cuál es entonces la diferencia?...***

Es muy sencillo: ya sabemos que el *Volumen Global* siempre vale `100%` al inicio de la ejecución del juego, por lo tanto asignar `BGMUSIC` con volumen `-1` significa que el Gestor asociará a este sonido un volumen de `100%` (porque **estará tomando el valor actual del *Volumen Global* al momento de la asignación**, y éste vale `100%` ya que hasta ahora no lo hemos cambiado). Todo esto sería muy distinto si, por ejemplo, antes de asignar `BGMUSIC` hubiéramos cambiado el valor del *Volumen Global* a `50%` (más adelante veremos cómo hacer esto), entonces el volumen asociado a `BGMUSIC` ya no sería `100%`, sino `50%`. En cambio, para `FXSOUND` el volumen asociado siempre será `100%`, incluso si el *Volumen Global* ha sido modificado antes. Siempre se le asociará un valor para el volumen del `100%` porque **así lo hemos indicado explícitamente** en la asignación de aquel sonido.

`Damusix.Tocar(BGMUSIC)` se encargará de iniciar la reproducción de la *música de fondo*. Como ya se mencionó, una vez que hemos asignado los sonidos ya no tenemos que preocuparnos de nada más. Para nuestros fines, esta rutina simplemente comenzará a tocar el sonido (previamente asignado) que le pasemos como argumento. Y será el propio Gestor de **Damusix** el que se encargará de "consultar" todos los datos "asociados" al sonido `BGMUSIC` y ocuparlos para establecer su volumen de reproducción y la cantidad de veces que se repetirá. Ya podemos darnos cuenta de la ventaja que supone "asignar" los sonidos, porque una vez que lo hemos hecho ya no tenemos que preocuparnos de nada más: simplemente de trabajar con éstos. =D

La *llamada completa* a la rutina de reproducción tiene la siguiente forma:

```
Damusix.Tocar(SONIDO,FLAG_SOUND_NOTIFY)
```

Usualmente sólo necesitaremos especificar el argumento `SONIDO`. El segundo argumento, `FLAG_SOUND_NOTIFY`, es opcional y lo omitiremos salvo en circunstancias muy especiales. La utilidad de este segundo argumento es indicarle a la rutina de reproducción *si acaso se debe generar un evento Glk de notificación* cuando el sonido termine de reproducirse naturalmente (es decir: cuando llega efectivamente a su final, no cuando es detenido por el programador). Este segundo argumento puede tomar dos valores: `0` ó `1`. Si omitimos el argumento, entonces se asume por defecto que vale `0` y **no se generarán notificaciones**. Si el valor del argumento es `1`, entonces **sí se generarán notificaciones** del tipo `evtype_SoundNotify` que podremos capturar en el *Punto de Entrada Glk* `HandleGlkEvent()`, proporcionado por la librería Inform. Veremos más información sobre la utilización de esta característica en la sección ***"Sobre los Puntos de Entrada Glk"***, puesto que corresponde a temas ya un poco más avanzados.

El Gestor de **Damusix** utiliza una "abstracción por sonidos" como método preferencial de manejo, antes que la habitual "abstracción por canales". En **Damusix** lo normal es usar el nombre del recurso de sonido para todas las rutinas del Gestor, mientras que en otras librerías o extensiones, y en la propia API Glk, lo usual es usar la referencia interna del canal de sonido (su ID)... sin embargo esta forma puede confundir, eventualmente, al programador. La filosofía de "abstracción por sonidos" de **Damusix** es un tanto más beneficiosa en varios sentidos.

La extensión **Damusix** ha sido concebida para tener rutinas minimalistas. ¿Qué quiere decir esto? Pues que las rutinas que proporciona el Gestor están programadas para hacer solamente lo que deben hacer y nada más. Si queremos cambiar el volumen, hay una rutina que hace solamente eso; si queremos cambiar el *modo de repetición*, otra rutina hace lo justo; si queremos asignar un sonido a un canal, hay una rutina que hace eso y nada más; si queremos parar un sonido, hay otra rutina que detiene la reproducción de los sonidos y no hace nada más. El motivo por el que se ha elegido esta forma de implementación es facilitar que la extensión pueda ser ampliada por el propio programador en el mismo código del juego. Por ejemplo: si deseamos tocar un sonido en un canal que previamente ya fue asignado a otro sonido, lo que tenemos que hacer es **"re-asignar"** dicho canal al nuevo sonido que queremos tocar. Si vamos a hacer este cambio muchas veces, lo mejor será crear una rutina, en el propio código de nuestro juego, que maneje los cambios. Por ejemplo, observemos como sería el código de una rutina proporcionada por el programador para cambiar la *música de fondo* del juego en cada localidad:

```
[ PonerMusica mus;
    Damusix.AsignarCanal(mus,0,-1,-1); ! canal 0, vol.global, rep.infinitas
    Damusix.Tocar(mus);                ! tocamos la musica recien asignada
];
```

Y luego, para cambiar la música en las distintas localidades (por ejemplo en "La Caverna"), ejecutaríamos `PonerMusica(MUS_CAVERNA)` cuando el jugador llegara a esa localidad.

En el código de ejemplo para `PonerMusica()` no hay ninguna rutina que detenga la posible reproducción del sonido que se estuviera tocando actualmente en el canal `0`, sencillamente se comenzará a tocar el nuevo sonido asignado sin más. Si ya existe un sonido que esté reproduciéndose en el canal `0`, ¿no deberíamos primero detenerlo antes de intentar tocar uno nuevo asignado a ese mismo canal `0`? En realidad, cada vez que llamamos a la rutina `Damusix.AsignarCanal()` siempre se detendrá la reproducción de cualquier sonido que estuviera tocándose actualmente en el canal que se va a ser asignado. En el código de ejemplo para `PonerMusica()`, lo que hacemos es asignar nuevos sonidos siempre al mismo canal (y con toda seguridad ya estará en reproducción un sonido anterior en aquel canal), por lo tanto la línea `Damusix.AsignarCanal(mus,0,-1,-1)` lo que hará será detener el sonido que actualmente está sonando en el canal `0` para asignar un nuevo sonido a ese canal y luego `Damusix.Tocar(mus)` tocará el sonido `mus` al que justamente ya le hemos asignado el canal `0`. Así que, para todos los fines, no necesitamos detener la reproducción del sonido anterior en ese canal, pues la propia rutina de asignación de canales ya lo ha hecho. =D *(Además, la propia `Damusix.Tocar()` detiene siempre el sonido [previamente asignado] que se le pasa como argumento, antes de comenzar a tocarlo efectivamente, jejeje! =D)*

Ahora, si efectivamente quisiéramos *detener un sonido que actualmente se está reproduciendo*, no es nada más simple que llamar a la rutina `Damusix.Parar()`, con el sonido que quieres que se detenga como argumento de la llamada. Por ejemplo: `Damusix.Parar(FXSOUND)`.

El Gestor de **Damusix** facilita un par de rutinas para modificar los "datos asociados" más importantes de cualquier sonido asignado: su *número de repeticiones* y su *porcentaje de volumen*...

`Damusix.Repeticion(SONIDO,REPETICIONES)` cambiará el *modo de repetición* del `SONIDO`, estableciéndolo en el número de `REPETICIONES` que se le ha indicado. La cantidad de repeticiones debe ser siempre un valor numérico **mayor a cero**, con excepción del valor especial `-1` que indica *repeticiones infinitas*; si se le pasa cualquier otro valor distinto, la rutina asumirá un valor por defecto de `1` repetición. El cambio en las repeticiones se aplicará la siguiente vez que se reproduzca el sonido.

`Damusix.Volumen(SONIDO,VOLUMEN)` cambiará el *porcentaje de volumen* del `SONIDO`, estableciéndolo en la cantidad de `VOLUMEN` que se le ha indicado. Este segundo argumento debe ser siempre un valor numérico entre `0` y `100`, con excepción del valor especial `-1` que indica que se use el porcentaje de *Volumen Global* actual del Gestor. El cambio en el volumen se aplicará inmediatamente si el sonido está en reproducción; en caso contrario, el cambio se reflejará la siguiente vez que se reproduzca el sonido.

Y el Gestor también dispone de una rutina especial que **en algunas determinadas circunstancias** puede resultarnos útil:

`Damusix.LiberarCanal(SONIDO)` liberará el canal que se le ha asignado previamente al `SONIDO`. Esto involucra eliminar la asignación entre el sonido y su canal, y detener dicho sonido. Esta rutina del Gestor no debería necesitarse casi nunca, pero se proporciona como contraparte de la rutina que asigna canales. *Liberar un canal* puede sernos útil en el siguiente caso: por ejemplo, si intentamos tocar un sonido que todavía no está asignado a ningún canal (no está *gestionado*), dicho sonido no podrá ser reproducido, pero **Damusix** tampoco generará un error. Entonces, ésta es una manera simple de hacer que en un determinado momento un sonido concreto ya no se pueda reproducir más, incluso si en el código fuente aún lo referenciamos o tratamos de tocarlo. *(Por ejemplo, imaginemos que tenemos un sonido de "bip" que se toca en cada turno de juego. Si en algún momento deseamos que el "bip" ya no suene más, pero no queremos liarnos haciendo comprobaciones condicionales y por el estilo, pues simplemente* ***liberaremos el canal*** *asignado a `BIP` y listo. Las siguientes llamadas a `Damusix.Tocar(BIP)` no tendrán ningún efecto puesto que el sonido `BIP` ya no está siendo gestionado por* ***Damusix*** *y, en consecuencia, el Gestor simplemente omitirá cualquier acción sobre aquel sonido.* ***Damusix siempre ignora cualquier trabajo con sonidos "no gestionados". Las únicas DOS EXCEPCIONES a esto son 1) la rutina de "reproducción virtual" y 2) el par de rutinas relacionadas con la "lista de reproducción de sonidos". Ambas excepciones las estudiaremos a continuación.****)*

## “Reproducción Virtual” de Sonidos

Ocasionalmente necesitaremos tocar un sonido incidental o de poca importancia como para asignarle un canal propio. Para estos casos es notablemente útil la rutina `Damusix.TocarV()` que reproduce sonidos en los “canales virtuales” de **Damusix**. El programador nunca podrá trabajar directamente con los canales virtuales; estos canales existen exclusivamente para permitir la ***reproducción virtual*** de sonidos, en otras palabras: tocar sonidos con la libertad de “no tener que asignarles previamente ningún canal”. Los sonidos lanzados en los canales virtuales **no se interrumpirán unos a otros**, así que podemos tocar un mismo sonido varias veces al mismo tiempo, provocando que **“se mezcle consigo mismo”**. Hay un límite, por supuesto, para esta “polifonía virtual”: existen 10 canales virtuales, por lo que podrán sonar 10 sonidos “mezclados entre sí” al mismo tiempo. **Los canales virtuales se asignan automáticamente y de forma rotativa**: esto quiere decir que si se llega al último de los 10 canales, el siguiente sonido lanzado será asignado al primer canal virtual, deteniendo cualquier sonido previamente asignado a ese canal si aún estuviera reproduciéndose. La rutina de *reproducción virtual* se usa de la siguiente manera:

`Damusix.TocarV(SONIDO,VOLUMEN)` lanzará la reproducción del `SONIDO` en alguno de los canales virtuales (si el `SONIDO` tiene o no tiene canal normal asignado no es relevante para esta rutina). El segundo argumento, `VOLUMEN`, es completamente opcional, normalmente no necesitaremos usarlo, e indica el porcentaje de volumen (un valor numérico entre `1` y `100`) con el que deberá reproducirse el `SONIDO`. **Si omitimos este segundo argumento de volumen (valor cero)** se usará el *porcentaje de volumen común actual para todos los canales virtuales*, que inicialmente siempre tiene un valor del `100%`, pero que se puede cambiar, como lo veremos a continuación. (Excepcionalmente, si el argumento `VOLUMEN` tiene el valor especial `-1`, esta rutina usará el porcentaje de *Volumen Global* actual del Gestor.)

El Gestor de **Damusix** no puede “gestionar” todos los aspectos de un sonido lanzado en un canal virtual, así que existen unas cuantas restricciones en su uso: 1) los sonidos sólo se tocarán **1 sola vez**; 2) los sonidos **no podrán generar eventos** `SoundNotify` cuando hayan acabado de reproducirse; 3) las rutinas de trabajo con los canales normales **no pueden ser aplicadas** en una “reproducción virtual”.

***UNA EXPLICACIÓN TÉCNICA:*** *reproducir “varias instancias” de un mismo sonido (es decir, que suene al mismo tiempo más de una vez, “mezclándose consigo mismo”) no se puede lograr con los sonidos asignados a los canales normales. Esto es así porque para lograr aquella “polifonía” el mismo sonido debería, básicamente, estar asignado a varios canales: sólo de esa forma podría reproducirse al mismo tiempo muchas veces. Pero, como ya sabemos, un sonido concreto* ***jamás debe tener asignado más de un canal****, porque de otro modo el Gestor de* ***Damusix*** *se equivocaría al tratar de “buscar el canal” de aquel sonido. Para salvar este “impedimento” se puede usar la rutina de “reproducción virtual” recién explicada.*

La siguiente rutina es la **única que se puede aplicar a los canales virtuales** y nos permitirá ajustar el volumen por defecto de los sonidos *reproducidos virtualmente*:

`Damusix.VolumenV(VOLUMEN)` cambiará el ***porcentaje de volumen común para todos los canales virtuales*** de acuerdo a la cantidad indicada en el argumento `VOLUMEN` que siempre debe ser un valor numérico correcto entre `0` y `100`, con excepción del valor especial `-1` que indica que se use el porcentaje de *Volumen Global* actual del Gestor. Esta rutina se encargará de establecer un *nuevo volumen común para cada uno de los 10 canales virtuales* que implementa **Damusix**. Al principio de la ejecución del juego el *volumen común actual para todos los canales virtuales* siempre vale `100%`. Esto se comprenderá mejor con la ayuda de un sencillo ejemplo:

*Imaginemos que tenemos el sonido de maullido de un minino. Lo lanzaremos en una "reproducción virtual" al comienzo del juego. Así, `Damusix.TocarV(MIAU)` tocará el maullido con `100%` de volumen, que **corresponde al volumen común para todos los canales virtuales de aquel momento**. Si posteriormente nosotros hacemos la llamada a `Damusix.VolumenV(50)` y nuevamente ejecutamos `Damusix.TocarV(MIAU)`, pues ahora el maullido se tocará con un volumen del `50%`. Y si ahora volvemos a tocar el maullido, pero esta vez con `Damusix.TocarV(MIAU,20)` el sonido ahora se tocará con `20%` de volumen, **ignorando el volumen común actual para los canales virtuales**. Ya podemos hacernos una idea general sobre el funcionamiento de este sistema de volúmenes... y quizás de cómo podríamos sacar ventajas de esta característica. Cada programador encontrará formas de sacarle el mejor partido.*

**NOTA TÉCNICA:** El *volumen común para los canales virtuales* es **independiente** del *Volumen Global* del Gestor. **Usualmente ambos valores estarán sincronizados**, salvo cuando el programador cambie directamente el *volumen para los canales virtuales* con la rutina `Damusix.VolumenV()`. Si posteriormente se cambia también el porcentaje de *Volumen Global* del Gestor (con la rutina `Damusix.VolumenGlobal()` que trataremos en breve), el nuevo valor de volumen será aplicado también para los canales virtuales.

## Usando la Lista de Reproducción de Sonidos

Otra funcionalidad interesante de **Damusix** es su "lista de reproducción de sonidos". Imaginemos que hemos juntado algunos sonidos para representar el ataque con cañón a un barco pirata y su posterior hundimiento. Así, por ejemplo, tendríamos un sonido para la "mecha encendida consumiéndose", luego otro para el "estallido del cañón" y uno más para el "hundimiento del barco enemigo". Si quisiéramos tocar todos estos sonidos en unidad, lo más natural sería considerarlos como una "lista de sonidos" que deben reproducirse en un orden específico, uno a uno.

Para estos fines, **Damusix** implementa una **"lista de reproducción de sonidos"** (un *playlist*, para los que sepan de inglés =D). Esta lista de reproducción es muy fácil de utilizar. Además, no es relevante que los sonidos agregados a la lista estén siendo *gestionados* por **Damusix** (es decir, no importa si tienen o no canal asignado)**.**

Vamos con el ejemplo del barquito zozobrado para mostrar el modo en que se usa la lista:

```
Damusix.CrearLista(MECHA_CANYON,5000);
Damusix.CrearLista(ESTALLIDO_CANYON);
Damusix.CrearLista(ESTALLIDO_CANYON);
Damusix.CrearLista(HUNDIMIENTO_BARCO,3500);
```

Cada una de las cuatro líneas de arriba **agregará un sonido a la lista de reproducción**. En la lista, el orden de dichos sonidos es el mismo en el que van siendo agregados a ella y se tocarán posteriormente en aquel mismo orden.

Revisemos, entonces, las líneas del ejemplo con más detalle:

`Damusix.CrearLista(MECHA_CANYON,5000)` agregará el sonido de la "mecha del cañón" a la lista de reproducción. El sonido, como puede deducirse, corresponde al primer argumento de la llamada a la rutina. El segundo argumento indica la **duración del sonido, expresada en milisegundos**. Sabemos que el sonido de la "mecha" dura 5 segundos, por lo que debemos informarle a esta rutina sobre esa duración. **Indicar la duración del sonido es obligatorio y necesario para que otra rutina (la que se encarga de "reproducir la lista") sepa cuánto tiempo debe esperar antes de comenzar a tocar el siguiente sonido de la lista.** Con todo, este segundo argumento es "opcional" en el sentido de que si se omite, se asumirán `1000` milisegundos (1 segundo) de duración para el sonido que se está agregando. Evidentemente, **lo mejor siempre será que indiquemos nosotros mismos la duración exacta del sonido.**

`Damusix.CrearLista(ESTALLIDO_CANYON)` agregará el sonido del "estallido del cañón" a la lista de reproducción. Ahora este sonido se tocará después del sonido de la "mecha". Como no se indica la **duración del sonido**, se asume que dura 1 segundo.

`Damusix.CrearLista(ESTALLIDO_CANYON)` hará exactamente lo mismo que la línea anterior, agregando por una segunda vez el "estallido del cañón". Un mismo sonido puede ser agregado varias veces a la lista de reproducción sin ningún problema.

`Damusix.CrearLista(HUNDIMIENTO_BARCO,3500)` agregará finalmente el sonido del "hundimiento del barco enemigo" a la lista de reproducción. Este sonido se tocará después de los otros tres sonidos previos. Además, su duración será de 3,5 segundos.

Ahora ya hemos "creado la lista de reproducción" con un total de cuatro sonidos. **Hay espacio para 10 elementos en la lista de reproducción: 10 sonidos.** Si la lista está completa e intentamos agregar más sonidos, no pasará nada: la rutina no fallará, pero simplemente no agregará el nuevo sonido y ya (compilando en modo DEBUG se nos informará con un mensaje apropiado sobre esta situación, si ocurriera).

Ahora que ya tenemos nuestra lista de reproducción creada, vamos a querer "reproducirla", por supuesto, en el momento oportuno. Para ello utilizaremos la rutina `Damusix.TocarLista()`. Esta rutina del Gestor tocará la lista de reproducción, previamente creada, reproduciendo los sonidos en ella uno a uno, en el orden en que fueron agregados y esperando el tiempo de duración indicado para cada uno antes de tocar el siguiente sonido de la lista.

Cuando la rutina `Damusix.TocarLista()` acabe de tocar todos los sonidos, se encargará automáticamente de "limpiar el contenido de la lista", dejándola vacía y preparada para que se puedan agregar nuevos sonidos más adelante. También si llamamos a esta rutina y en ese momento la lista de reproducción de sonidos está vacía, entonces no ocurrirá nada. No se producirá un error, simplemente no se tocará ningún sonido. La *llamada completa* a la rutina de reproducción de la *lista de sonidos* tiene la siguiente forma:

`Damusix.TocarLista(VOLUMEN)`, siendo `VOLUMEN` un argumento completamente opcional y poco probable que necesitemos usarlo, e indica el porcentaje de volumen (un valor numérico entre `1` y `100`) con el que deberá reproducirse cada sonido de la lista (afecta a la lista completa). **Si omitimos este argumento de volumen (valor cero)** se usará el *porcentaje de volumen común actual para la lista de reproducción de sonidos*, que inicialmente siempre tiene un valor del `100%`, pero que se puede cambiar, como lo veremos a continuación. (Excepcionalmente, si el argumento `VOLUMEN` tiene el valor especial `-1`, esta rutina usará el porcentaje de *Volumen Global* actual del Gestor.)
***AVISO:*** *No se puede tocar la lista de reproducción si un Fade en "tiempo-real" está activo en ese momento (lo trataremos en breve) porque ambas rutinas hacen uso del Timer Glk. En tal situación,* `Damusix.TocarLista()` *se negará a funcionar y mostrará un mensaje apropiado si estamos compilando en modo DEBUG.*

Existen dos rutinas más que se pueden aplicar a la *lista de reproducción de sonidos*:

`Damusix.LimpiarLista()` limpiará el contenido actual de la lista. Esto es útil, por ejemplo, si agregamos unos cuantos sonidos a la lista y luego nos arrepentimos de tocarla. Si queremos agregar un nuevo conjunto de sonidos, tendremos primero que "limpiar el contenido previo" de la lista. De otro modo, los nuevos sonidos se agregarán después de los primeros, que nunca tocamos. Esto sucede porque la lista sólo se limpia una vez que es tocada y si no la tocamos, tendremos que limpiarla nosotros mismos para obtener una nueva lista vacía, si la necesitáramos.

La siguiente rutina nos permitirá ajustar el volumen por defecto con el que se tocarán todos los sonidos que la lista tiene agregados. Revisémosla detalladamente...

`Damusix.VolumenLista(VOLUMEN)` cambiará el ***porcentaje de volumen común con el que se tocarán todos los sonidos de la lista de reproducción***, de acuerdo a la cantidad indicada en el argumento `VOLUMEN` que siempre debe ser un valor numérico correcto entre `0` y `100`, con excepción del valor especial `-1` que indica que se use el porcentaje de *Volumen Global* actual del Gestor. Esta rutina se encargará de establecer un *nuevo volumen común* con el que se tocará cada sonido que esté presente en la lista de reproducción que implementa **Damusix**. Al principio de la ejecución del juego el *volumen común actual para la lista de reproducción de sonidos* siempre vale `100%`. Nuevamente usaremos el ejemplo del gatito para explicar aquí de manera más sencilla cómo funciona este asunto del volumen:

*Imaginemos que tenemos el sonido de maullido de un minino. Crearemos la lista de reproducción al comienzo del juego agregándole el maullido del gatito tres veces. Así,* `Damusix.TocarLista()` *tocará los tres maullidos con* `100%` *de volumen,* ***que corresponde al volumen común para la lista de reproducción de sonidos*** **en** ***aquel momento****. Si luego nosotros hacemos la llamada a* `Damusix.VolumenLista(50)` *y nuevamente creamos la lista y ejecutamos* `Damusix.TocarLista()`*, pues ahora los maullidos se tocarán con un volumen del* `50%`*. Y si ahora volvemos a crear por tercera vez la lista, pero esta vez la tocamos con* `Damusix.TocarLista(20)` *los maullidos ahora se tocarán con* `20%` *de volumen,* ***ignorando el volumen común actual para la lista de reproducción de sonidos****. Eso es todo. =D*

**NOTA TÉCNICA:** El *volumen común para la lista de reproducción* es **independiente** del *Volumen Global* del Gestor. **Usualmente ambos valores estarán sincronizados**, salvo cuando el programador cambie directamente el *volumen para la lista de reproducción* con la rutina `Damusix.VolumenLista()`. Si posteriormente se cambia también el porcentaje de *Volumen Global* del Gestor (con la rutina `Damusix.VolumenGlobal()` que trataremos en breve), el nuevo valor de volumen será aplicado también para la lista de reproducción de sonidos.

## Activando y Desactivando la Salida de Audio

La mayoría de las extensiones o librerías que permiten trabajar con el Audio en Glk traen funciones de “desactivación del audio” que se limitan básicamente a bajar el volumen de todos los sonidos hasta el silencio. **El Gestor de Damusix no hace eso, sino que activa o desactiva la Salida de Audio de manera “limpia”**. Esto quiere decir sencillamente que el Gestor *no modifica los valores de volumen* de los sonidos gestionados. No cambia de ninguna forma estos valores. Más bien, lo que hace es cambiar el “modo de salida de audio” de **Damusix**. El *kernel* tomará cuenta del cambio y si la salida de audio está “activada”, permitirá la reproducción normal de los sonidos gestionados; por el contrario, si la salida de audio está desactivada, el *kernel* impedirá de manera “limpia” la reproducción de cualquier sonido gestionado.

Si el audio está desactivado, todas las rutinas del Gestor de **Damusix** seguirán funcionando normalmente, pero no reproducirán sonidos. Todo funcionará tal y como si el audio estuviera activado, pero ningún sonido gestionado se tocará realmente.

Este sistema es muy interesante porque permite que cualquier cambio que se haga en los sonidos gestionados por **Damusix** sea efectivo, incluso si no escuchamos nada. Por ejemplo, imaginemos que tenemos una *música de fondo* que actualmente está sonando. Si desactivamos el audio, la música dejará de sonar; sin embargo, para el *kernel* de **Damusix**, la música sigue en reproducción, solamente que el audio está desactivado. Cuando activemos el audio de nuevo, la música volverá a sonar sin que tengamos que hacer nada más. Otro ejemplo: la música de fondo está sonando con volumen del 100% y entonces desactivamos el audio. Ahora cambiamos el volumen de la música al 50% y activamos nuevamente la salida de audio. La música volverá a sonar, pero esta vez con 50% de volumen...

Es más, incluso podríamos estar aplicando un efecto de *Fade en "tiempo-real"* con el audio desactivado y el efecto **realmente estaría ejecutándose aunque no escuchemos nada**. Si activamos el audio a la mitad del Fade, podríamos oír el efecto desde la mitad hasta el final. =D (Trataremos los efectos de *Fade en "tiempo-real" y "tiempo-no-real"* más adelante.)

Cuando el audio de **Damusix** está desactivado, todo funciona normalmente, sólo que los sonidos gestionados ya no sonarán (esto sin cambiar sus volúmenes). Lo anterior es realmente muy útil, puesto que nos permite prescindir de líneas de código de comprobación para saber si acaso debemos tocar un sonido dependiendo de si el audio está activado o no en nuestro juego. Si está desactivado, una llamada como `Damusix.Tocar(SONIDO)`, por ejemplo, funcionará normalmente pero no tocará nada.

Estas son las rutinas para controlar la salida de audio de **Damusix**:

`Damusix.ActivarAudio()` activará la salida de audio del Gestor.

`Damusix.DesactivarAudio()` desactivará la salida de audio del Gestor.

Una aplicación práctica para estas dos rutinas podría ser, por ejemplo, darle la posibilidad al jugador de tener audio a elección mientras se divierte con nuestro juego.

Además, también podemos comprobar el estado actual de la salida de audio de **Damusix** mediante la rutina `Damusix.HayAudio()`, que devolverá `1` si el audio está activado y `0` si el audio está desactivado.

## Otras Rutinas Útiles del Gestor

`Damusix.VolumenGlobal(VOLUMEN)`
Cambia el porcentaje de *Volumen Global* del Gestor de **Damusix**. Este cambio se aplicará a *todos los sonidos gestionados*, al *volumen común de los canales virtuales* y al *volumen común de la lista de reproducción de sonidos*; vale decir, absolutamente todos los volúmenes serán establecidos al valor del nuevo *Volumen Global*. El argumento `VOLUMEN` siempre debe ser un valor numérico correcto entre `0` y `100`. Aquí no se permite el valor especial `-1` (no tiene sentido en esta rutina).

`Damusix.PararTodo()`

**Detiene absolutamente todos los canales inicializados por Damusix**. Esto quiere decir que **todos los sonidos serán parados sin excepción**: todos los *sonidos gestionados*, todos los *sonidos en reproducción virtual* y, por seguridad, también se detendrá el canal de la *lista de reproducción de sonidos*.

`Damusix.PararCanalesExtra()`

Detiene la reproducción en todos los canales virtuales (*sonidos en reproducción virtual*) y, por seguridad, también se detendrá el canal de la *lista de reproducción de sonidos*. Esta rutina es para usos más técnicos y normalmente no necesitaremos ocuparla.

`Damusix.SonandoDeFondo(SONIDO)`

Comprueba si el sonido que se le pasa como argumento actualmente está *sonando de fondo* (es decir, **en reproducción con "repeticiones infinitas"**). La rutina devolverá `1` si actualmente el sonido está *sonando de fondo* y `0` si no lo está. Esta rutina es muy útil, por ejemplo, para comprobar si en un momento determinado está sonando alguna *música de fondo* en nuestro juego. Imaginemos, por ejemplo, que cada localidad de nuestro juego tiene una distinta *música de fondo*. Pero también tenemos unas cuantas localidades conectadas entre sí que comparten la misma música. No se produciría un efecto muy agradable si el jugador cambia de localidad entre ellas y se "re-lanza" la música por cada localidad, desde el principio cada vez, siendo que todas ellas tienen la misma música. Para estos casos sería muy útil comprobar si esa música ya está "sonando de fondo" actualmente y sólo en caso negativo lanzar su reproducción. Con un código de ejemplo todo esto quedará más claro...

*Supongamos que cada localidad tiene una propiedad llamada `musica` que indica su música de fondo. Ocuparemos el punto de entrada `NewRoom()` provisto por la librería Inform para lanzar la reproducción de la música de fondo que corresponda al llegar a cada localidad:*

```
1  [ NewRoom ;
2      if (location provides musica) {
3        if (Damusix.SonandoDeFondo(location.musica) == 0) {
4          Damusix.AsignarCanal(location.musica,0,-1,-1);
5          Damusix.Tocar(location.musica);
6        }
7      }
8  ];
```

*Este código se ejecutará cada vez que se llegue a una localidad y hará lo siguiente:*
*(Línea 2) comprobará que la localidad tenga una propiedad `música`, de ser así...*
*(Línea 3) comprobará que la música de la loc. no esté sonando de fondo (`0`), de ser así...*
*(Líneas 4 y 5) asignará el canal para la música de la localidad y comenzará a tocarla.*

*Podemos darnos cuenta de que la música de la localidad **sólo comenzará a tocarse si no está sonando todavía**. Si actualmente ya está en reproducción, no se intentará volverla a tocar. Todo esto gracias a `Damusix.SonandoDeFondo()`.*

## Efectos de FadeIn y FadeOut en *"tiempo-real"* (*fades complejos*)

**Damusix** es capaz de realizar efectos de audio del tipo *FadeIn y FadeOut en "tiempo-real"* a cualquier sonido gestionado que esté actualmente en reproducción. Un efecto de ***FadeIn*** **subirá gradualmente el volumen** de un sonido desde su valor actual hasta llegar a un máximo de volumen determinado, mientras que un efecto de ***FadeOut*** **bajará gradualmente el volumen** de un sonido desde su valor actual hasta llegar a un mínimo de volumen determinado.

Como estos efectos son de *"tiempo-real"*, **el juego continuará su ejecución mientras el trabajo de Fade se está llevando a cabo**, y el jugador podrá seguir introduciendo órdenes y avanzando en el juego.

Los efectos de *Fade en "tiempo-real"* **funcionan perfectamente**, incluso si el jugador utiliza los comandos UNDO/RESTORE/RESTART. Si se guarda la partida mientras un Fade está en progreso, su estado actual también será guardado; entonces cuando el jugador cargue esa partida, el efecto de Fade será continuado correctamente desde el punto en que había quedado cuando se guardó la partida. Esto funciona igualmente en el caso de usar el comando UNDO.

El único comportamiento extraño puede darse si se aplica un *Fade en "tiempo-real"* a un sonido que actualmente no está *"sonando de fondo"* (es decir, en reproducción *sin repeticiones infinitas*). En tal caso, el Fade será **intencionalmente no recuperado**, porque luego de un UNDO/RESTORE **Damusix** jamás re-lanza sonidos que no estén *"sonando de fondo"* (esto tiene mucho sentido, puesto que no es correcto volver a reproducir todos los sonidos que simplemente se debían tocar un sola vez y ya). Por lo anterior, se recomienda **no aplicar** un efecto de Fade a sonidos que **no están *"sonando de fondo"***.

Con independencia de lo anterior, los efectos de *Fade en "tiempo-real"* **funcionan perfectamente** y están exhaustivamente testeados. Podemos usarlos sin temor de colgar el juego ni de provocar errores irrecuperables. Ante cualquier situación imprevista, el Fade será abortado por el *kernel* de **Damusix**. =D

Antes de usar los efectos de *Fade en "tiempo-real"* debemos dejar dos asuntos muy en claro: **1) sólo se puede realizar 1 efecto de Fade por vez; 2) el intérprete Glulx en el que se esté ejecutando el juego debe tener soporte de tiempo real (soporte para el *Timer Glk*).** Si el intérprete no es capaz de manejar el *Timer Glk*, los efectos de Fade no se realizarán. No se provocará ningún error en el juego, pues **Damusix** anticipa una situación de este tipo. En lugar de ocurrir un error, el efecto simplemente no se producirá, es todo.

**Las rutinas de *Fade en "tiempo-real"* sólo funcionarán correctamente con sonidos GESTIONADOS y que estén actualmente EN REPRODUCCIÓN.** Por ejemplo:

```
Damusix.Tocar(FXSOUND);          ! comienza a tocar el sonido FXSOUND
Damusix.FadeOut(FXSOUND,5000);   ! hace FadeOut en "tiempo-real" al sonido
```

Vamos a revisar ahora detenidamente las dos rutinas de Fade en *"tiempo-real"*:

`Damusix.FadeIn(SONIDO,DURACION,VOL_FINAL)`
**Subirá gradualmente el volumen** del `SONIDO` desde su *valor actual* hasta llegar a un máximo igual al *volumen final* `VOL_FINAL` indicado (este 3er argumento es **completamente opcional** y debe tener un valor numérico correcto entre 1-100; **si se omite o vale cero**, se ocupará un valor por defecto correspondiente al *Volumen Global* actual establecido en el Gestor). El argumento `DURACION` **indica la cantidad de milisegundos que deberá durar en total el efecto de FadeIn**; debe ser siempre un valor igual o superior a `100`. Si este segundo argumento se omite o si tiene un valor inferior a `100`, entonces se asumirán por defecto `100` milisegundos de duración total para el *FadeIn*.

```
Damusix.Volumen(FXSOUND,0)    ! ponemos a cero el volumen de FXSOUND
Damusix.Tocar(FXSOUND);       ! comenzamos a tocar el sonido FXSOUND
Damusix.FadeIn(FXSOUND,5000); ! FadeIn en "tiempo-real" a FXSOUND por 5seg.
```

*El código anterior es el ejemplo más sencillo del uso de un FadeIn en "tiempo-real". Subirá el volumen de `FXSOUND` desde 0% hasta un máximo igual al valor actual del Volumen Global (100% en este caso, porque hasta ahora no lo hemos cambiado). Aquí se asume como "volumen final" el valor actual del Volumen Global, simplemente por la omisión del 3er argumento, que es completamente opcional.*

`Damusix.FadeOut(SONIDO,DURACION,VOL_FINAL,PFADE_SND,PFADE_SND_NOTIFY)`
**Bajará gradualmente el volumen** del `SONIDO` desde su valor actual hasta llegar a un mínimo igual al *volumen final* `VOL_FINAL` indicado (este 3er argumento es **completamente opcional** y debe tener un valor numérico correcto entre 1-100; **si se omite o vale cero**, se ocupará un valor por defecto correspondiente al mínimo de volumen, es decir 0%). El argumento `DURACION` **indica la cantidad de milisegundos que deberá durar en total el efecto de FadeOut**; debe ser siempre un valor igual o superior a `100`. Si este segundo argumento se omite o si tiene un valor inferior a `100`, entonces se asumirán por defecto `100` milisegundos de duración total para el *FadeOut*. Si el *volumen final* es igual a `0%`, entonces cuando el *FadeOut* termine se **detendrá *artificialmente* la reproducción del `SONIDO`** (no tiene sentido seguirlo tocando con `0%` de volumen) y luego se restablecerá su volumen, usando para ello el ***valor de volumen que tenía originalmente*** antes de realizar el *FadeOut*.

```
Damusix.Volumen(FXSOUND,100)   ! ponemos a 100% el volumen de FXSOUND
Damusix.Tocar(FXSOUND);        ! comenzamos a tocar el sonido FXSOUND
Damusix.FadeOut(FXSOUND,5000); ! FadeOut en "tiempo-real" a FXSOUND por 5seg.
```

*El código anterior es el ejemplo más sencillo del uso de un FadeOut en "tiempo-real". Bajará el volumen de `FXSOUND` desde su porcentaje de volumen actual (100% según el ejemplo) hasta un mínimo igual a 0%. Aquí se asume como "volumen final" un valor de 0%, simplemente por la omisión del 3er argumento, que es completamente opcional.*

Los últimos dos argumentos de `Damusix.FadeOut()` **son opcionales también y sólo tendrán sentido si el *volumen final* indicado para el *FadeOut* es igual a 0%**. Esto requiere una explicación más profunda... Vamos a suponer que comenzamos a tocar un sonido más o menos largo, con el propósito de aplicarle un *Fade en "tiempo-real"* mientras se está reproduciendo. Nuestro sonido **se tocará 1 sola vez y además generará `SoundNotify`**, porque nos interesa que al terminar de sonar se ejecute un código que lanzará la reproducción de otro sonido cualquiera. Para un *FadeIn* esto se lograría así:

```
Damusix.Volumen(FXSOUND,0);   ! pone vol. 0% para hacer FadeIn desde silencio
Damusix.Tocar(FXSOUND,1);     ! toca FXSOUND con vol. 0%, hace notificaciones
Damusix.FadeIn(FXSOUND,5000); ! FadeIn FXSOUND hasta Vol.Global durante 5seg.
```

El código anterior producirá un *FadeIn* para `FXSOUND` muy bonito, que partirá desde el silencio hasta el *Volumen Global* actual del Gestor. Como le hemos avisado a la rutina `Damusix.Tocar()` que genere notificaciones de sonido, entonces cuando `FXSOUND` termine de sonar se generará un evento de notificación que será capturado por la rutina `HandleGlkEvent()` (*esto lo trataremos en la sección "Sobre los Puntos de Entrada Glk"*) en la cual tendremos programado un código que lanzará cualquier otro sonido, tal como lo deseábamos.

Para un *FadeOut*, el mismo código produce un resultado **completamente distinto**:

```
Damusix.Volumen(FXSOUND,100);  ! pone 100% para hacer FadeOut desde vol. max.
Damusix.Tocar(FXSOUND,1);      ! toca FXSOUND con vol. 100% y notificaciones
Damusix.FadeOut(FXSOUND,5000); ! FadeOut FXSOUND hasta vol. 0% durante 5seg.
```

En este caso, `FXSOUND` **nunca generará notificaciones**. ¿Por qué? La explicación es muy sencilla...

*Cuando estudiamos la rutina `Damusix.Tocar()` habíamos mencionado que los eventos del tipo `evtype_SoundNotify` solamente eran generados cuando el sonido **terminaba su reproducción naturalmente**, no cuando nosotros lo deteníamos. Como ya sabemos, cuando un FadeOut termina, si el "volumen final" indicado era igual a `0%`, entonces el sonido que estaba en FadeOut será detenido efectivamente (no tiene sentido que se siga en reproducción si no vamos a poder escucharlo, ¿verdad? =D). En conclusión: **puesto que un sonido en FadeOut es detenido artificialmente si el "volumen final" es igual a 0%, las notificaciones para dicho sonido nunca llegan a ser generadas**.*

Entonces, es evidente que para el caso de *FadeOut* no podremos capturar ningún evento de sonido en la rutina `HandleGlkEvent()`. Y esto puede ser un problema, sobre todo porque el uso más frecuente para `SoundNotify` sea quizás el de permitir ejecutar un código que lance la reproducción de algún sonido cuando uno previo haya terminado de sonar.

Para **salvar** de alguna manera este problema, **existen los dos últimos argumentos opcionales** en la llamada a `Damusix.FadeOut()`. Estos argumentos son completamente opcionales y **sólo se utilizarán si el *volumen final* indicado para el *FadeOut* es igual a 0%**, *en caso contrario simplemente serán omitidos*. Eso es todo.

Vamos a recordar la sintaxis completa de la rutina de *FadeOut*:

```
Damusix.FadeOut(SONIDO,DURACION,VOL_FINAL,PFADE_SND,PFADE_SND_NOTIFY)
```

El cuarto argumento `PFADE_SND` debe ser algún **sonido gestionado por Damusix** *(obligatoriamente debe **estar asignado** en el Gestor con anterioridad y **no puede ser "liberado"** antes de que el FadeOut termine, porque en tal caso dicho sonido no podrá ser reproducido, ya que la rutina `Damusix.Tocar()` no puede tocar sonidos que no están siendo "gestionados")*. Llamaremos a este sonido *Post-FadeOut*, porque será el sonido que se reproducirá *una vez que el efecto de FadeOut termine*.

El quinto argumento `PFADE_SND_NOTIFY` indica si acaso el sonido *Post-FadeOut* debe o no generar eventos `SoundNotify` (internamente, `Damusix.FadeOut()` llama a `Damusix.Tocar()` para reproducir el sonido *Post-FadeOut*, así que también tenemos la opción de poder *generar notificaciones para aquel sonido si lo deseamos*). Si no necesitamos generar eventos `SoundNotify`, simplemente omitiremos este argumento o le daremos un valor cero.

Recordemos que nuestra intención en el ejemplo inicial era *tocar un sonido, hacerle un Fade, y cuando el sonido terminara debería lanzarse la reproducción de algún otro sonido*. Con *FadeIn* funcionaba a la perfección usando un `SoundNotify`. Con *FadeOut* no podemos usar el `SoundNotify` porque el sonido en *FadeOut* es detenido artificialmente cuando el efecto acaba (evidentemente, esto se aplica solamente en el caso de hacer un *FadeOut* con un *volumen final* de 0%). Si usamos los dos últimos argumentos de la llamada a `Damusix.FadeOut()` podremos lograr el objetivo que planteábamos en el ejemplo. =D

```
Damusix.Volumen(FXSOUND,100); ! pone 100% para hacer FadeOut desde vol. max.
Damusix.Tocar(FXSOUND);       ! toca FXSOUND con vol. 100%, no notificaciones

! ahora hace FadeOut en "tiempo-real" a FXSOUND con un "volumen final"
! de 0%  (valor asumido, por poner cero como valor en el 3er argumento)
! durante 5 segundos, luego reproduce OTROFX (4to) y (opcionalmente)
! genera notificaciones cuando OTROFX termine de sonar. En este caso
! no se generarán notificaciones puesto que el 5to argumento se omite,
! y por defecto vale 0 (no notificar), aunque no existe ningun problema
! en que pongamos el valor cero explícitamente... simple economía =P
Damusix.FadeOut(FXSOUND,5000,0,OTROFX);
```

El código anterior hará un *FadeOut* en *"tiempo-real"* a `FXSOUND` desde 100% de volumen hasta 0% de *volumen final*; y una vez que el efecto de Fade termine, entonces se tocará el sonido `OTROFX`. Y todo sin usar para nada `HandleGlkEvent()` (jeje! =D). Como la reproducción de `OTROFX` se realiza mediante una llamada implícita a `Damusix.Tocar()`, pues este sonido podría generar notificaciones `SoundNotify` si así lo quisiéramos, tal como cualquier otro sonido gestionado. En el ejemplo le hemos dicho que no lo haga. Pero si realmente necesitamos que el sonido *Post-FadeOut* genere una notificación, simplemente habría que poner `1` como quinto argumento de la llamada a `Damusix.FadeOut()`.

Si todo esto parece complicado, esperemos que quede más claro en la sección ***"Sobre los Puntos de Entrada Glk"***. Debemos tener en cuenta, por cierto, que los efectos de Fades en *"tiempo-real"* hacen uso de conceptos y métodos de programación, tanto de Glulx como de la librería Inform, un poco más avanzados de lo que suele ser habitual.

Vamos a conocer ahora dos rutinas que nos resultarán realmente muy útiles cuando usemos los Fades en *"tiempo-real"* de **Damusix**...

Los efectos de Fade en *"tiempo-real" ab-usan* (jeje! =P) del *Timer Glk*. Lo ocupan de manera exhaustiva y fallarán, por supuesto, si el *Timer* es ***reprogramado*** por algún código externo a **Damusix** mientras el efecto aún está en progreso. Existen cientos de extensiones/librerías que usan el *Timer* para sus propios fines; hasta la propia librería Inform proporciona rutinas que trabajan con el *Timer* para un uso privado. Incluso el mismo programador puede reprogramar el *Timer* en el código del juego. Como es evidente, **Damusix** no puede prever todas estas situaciones, pero sí **facilita un método para "controlarlas"**...

`Damusix.EnFade()` es una rutina de consulta que comprueba si actualmente **está en proceso un efecto de Fade en *"tiempo-real"***. Devolverá `1` si se está haciendo un Fade actualmente *(FadeIn/FadeOut)* o devolverá `0` si no hay ningún trabajo de Fade activo al momento. *Esta rutina nos resultará útil cuando queramos que cierto código se ejecute o no, dependiendo de si está en proceso un Fade en "tiempo-real". Si tenemos un código que reprograma el Timer Glk, lo mejor será hacer una comprobación con esta rutina para saber si hay o no un Fade activo en ese momento y si podemos ejecutar el código implicado sin riesgo de estropear nuestro Fade.*

`Damusix.AbortarFade()` **abortará cualquier efecto de *Fade en "tiempo-real"* que pudiera estar en proceso**. Si se aborta un *FadeIn*, el sonido seguirá en reproducción y se le aplicará el *volumen final* indicado cuando se lanzó el efecto. Si se aborta un *FadeOut*, existen dos posible comportamientos: 1) si el *volumen final* es mayor a 0%, entonces simplemente se le aplicará al sonido dicho *volumen final* y finalmente se terminará el Fade; 2) si el *volumen final* es igual a 0%, entonces se detendrá el sonido, luego se le aplicará un *volumen final igual al volumen que tenía originalmente antes de realizar el efecto* y finalmente se terminará el Fade. Si no hay ningún efecto de Fade activo en el momento de la llamada, esta rutina simplemente no hará nada.

***NOTA: Las rutinas del Gestor de Damusix son "inteligentes" y si se aplican a cualquier sonido que tenga activo un Fade en "tiempo-real", primero abortarán dicho efecto si lo consideran apropiado y luego seguirán su ejecución normal.***

## Efectos de FadeIn y FadeOut en *"tiempo-no-real"* (*fades simples*)

**Damusix** proporciona una alternativa a los *Fades en "tiempo-real"*, que podemos usar si no queremos liarnos coordinando el *Timer Glk* de nuestro propio código temporizado con el de los efectos de *Fade en "tiempo-real"* de **Damusix**. Así evitamos tener que hacer comprobaciones del tipo `Damusix.EnFade()` para averiguar si hay un *Fade en "tiempo-real"* activo y, en consecuencia, impedir que se ejecute alguna rutina que reprograme el *Timer* y "mate" dicho efecto.

Esta "alternativa" son los ***Fades en "tiempo-no-real"*** y son mucho más simples y sencillos de usar que los en *"tiempo-real"* explicados anteriormente.

Un *Fade en "tiempo-no-real"* **primero realiza el efecto y sólo cuando ha terminado devuelve el control de la ejecución a la siguiente línea en el código fuente**. Por este motivo su uso será más práctico con tiempos de duración muy reducidos. De hecho, estos Fades **pueden tener tiempos de duración inferiores a los 100 milisegundos**, lo cual es ideal para producir transiciones bonitas y elegantes entre distintos sonidos.

Como estos efectos de Fade no son en *"tiempo-real"*, no pueden ser abortados. Tampoco es necesario tener una rutina para comprobar si están activos. Simplemente se limitan a dos rutinas elementales, que son las siguientes:

```
Damusix.SimpleFadeIn(SONIDO,DURACION,VOL_FINAL)
```

```
Damusix.SimpleFadeOut(SONIDO,DURACION,VOL_FINAL)
```

Estas dos rutinas funcionan exactamente igual que sus homólogas en *"tiempo-real"*. La única salvedad es `Damusix.SimpleFadeOut()`, que por ser en *"tiempo-no-real"* **no requiere tener argumentos para tocar un sonido *Post-FadeOut* ni tampoco detiene la reproducción del sonido si el *volumen final* indicado es igual a `0%`**. Como la ejecución del código *continuará en la siguiente línea una vez que el efecto de SimpleFadeOut acabe*, podemos detener nosotros mismos el sonido que tiene `0%` de volumen si lo consideramos apropiado o podemos lanzar la reproducción de otro sonido, etcétera.

Vamos a ver un ejemplo de utilización de estos Fades "simples"...

```
! [...antes ya hemos asignado el sonido MUSICA y lo estamos reproduciendo...]
Damusix.Volumen(MUSICA,100);   ! 100% para hacer SimpleFadeOut desde vol.max.

print "Realizando efecto SimpleFadeOut^";
Damusix.SimpleFadeOut(MUSICA,500,50); ! medio segundo (500ms) hasta vol. 50%

print "Realizando efecto SimpleFadeIn^";
Damusix.SimpleFadeIn(MUSICA,500);     ! medio segundo (500ms) hasta vol. 100%

print "Realizando efecto SimpleFadeOut^";
Damusix.SimpleFadeOut(MUSICA,1000);   ! 1 segundo (1000ms) hasta vol. 0%

! como MUSICA ahora tiene vol. 0%, detenemos su reproduccion
! (si no podemos escucharla, no tiene sentido que siga sonando)
Damusix.Parar(MUSICA);
```

*El código anterior es un sencillo ejemplo de aplicación de los efectos de Fade en "tiempo-no-real". Primero se establece para la `MUSICA` un volumen del `100%`; luego se le aplica un **SimpleFadeOut** de medio segundo con un "volumen final" de 50% (es decir, bajará el volumen desde 100% hasta 50%); cuando acaba, se le aplica un **SimpleFadeIn** de medio segundo con un "volumen final" del 100%, valor que se asume, ya que no se ha indicado explícitamente el 3er argumento **[ver la nota más abajo]** (es decir, subirá el volumen desde 50% hasta 100%); cuando el efecto anterior termina, se le aplica al sonido un **SimpleFadeOut** de 1 segundo con un "volumen final" del 0%, valor que también se asume, por omisión del 3er argumento **[ver la nota más abajo]** (es decir, bajará el volumen desde 100% hasta 0%); finalmente, como la `MUSICA` tiene ahora `0%` de volumen, detenemos su reproducción (de otra manera seguiría sonando, sin que pudiéramos escucharla). Se sugiere revisar el código de **"Damusix Demo"**, incluido en el paquete de la extensión **Damusix**, para ver un ejemplo más completo.*

***NOTA: Cuando el tercer argumento de cualquier rutina de Fade (sea o no sea en "tiempo-real") vale cero, es exactamente lo mismo que si no se escribiera y siempre se asumirá un valor por defecto. Para los [Simple]FadeIns, este valor siempre será el porcentaje de Volumen Global actual; para los [Simple]FadeOuts, siempre será el 0%.***

## Rutinas Técnicas y Rutinas de Consulta del Gestor

El Gestor de **Damusix** facilita varias rutinas más, aparte de las ya descritas, bastante útiles en situaciones de programación más avanzada.

Por un lado tenemos las llamadas **"Rutinas Técnicas"**. Estas rutinas realizan las mismas funcionalidades que las rutinas que trabajan con los canales normales de **Damusix**, pero no usan la "abstracción por sonidos". En lugar de indicar el sonido con el que deseamos trabajar, tendremos que indicar el **"número del canal"** en el que nos interesa que se realice el trabajo. Por convención, estas rutinas tendrán el mismo nombre que el de sus homólogas "no-técnicas", pero seguido de la palabra "Canal".

Así, para `Damusix.Tocar()` tenemos la rutina `Damusix.TocarCanal()`, por ejemplo. A la primera le pasaremos como argumento el sonido que queremos reproducir mientras que a la segunda, el número del canal que deseamos que comience a tocar.

Hay tantas rutinas *técnicas* como *no-técnicas* tiene el Gestor de **Damusix**. No se explicarán aquí. En el ***"Apéndice de Rutinas del Gestor"*** se detalla cada una de éstas.

Por otro lado tenemos las llamadas **"Rutinas de Consulta"**. Estas rutinas simplemente *"averiguan" algún dato sobre el estado del kernel* de **Damusix** y devuelven un valor en consecuencia.

Por ejemplo, hay una rutina de consulta para averiguar el volumen actual de un sonido: `Damusix.QueVolumen()`. Algunas rutinas de consulta también tienen equivalentes "técnicas". Por ejemplo, para la rutina anterior existe una homóloga llamada `Damusix.QueVolumenCanal()`. Jeje! =P

Las rutinas de consulta no se explicarán aquí. Nuevamente, en el ***"Apéndice de Rutinas del Gestor"*** se detalla cada una de éstas.

## Mecanismo de "Protección de Sonidos" ante UNDO/RESTORE

**Damusix** incluye "de fábrica" un novedoso mecanismo de "protección de sonidos" ante los comandos UNDO/RESTORE. Por motivos técnicos de Glulx, siempre es necesario recuperar las referencias de los canales de audio Glk cuando el *estado del juego* cambia de alguna forma. Este cambio puede producirse, justamente, cuando el jugador usa los comandos UNDO/RESTORE/RESTART. La recuperación de las referencias de los canales de audio implica que es necesario siempre actualizar la reproducción en dichos canales, porque de otro modo estos podrían seguir sonando sin ningún control o podrían no reproducir los sonidos que corresponden con el nuevo *estado del juego*; siendo todas éstas, circunstancias no deseadas e irregulares. Por ejemplo, si grabamos una partida en la que estaba sonando una música de fondo, al cargarla en otro momento debería siempre "re-lanzarse" la reproducción de esa música de fondo. Lo mismo pasa ante UNDO.

Éste es el comportamiento deseado normalmente, pero no el ideal. Imaginemos que estamos jugando con cierta música *"sonando de fondo"* y luego hacemos UNDO. La música volverá a ser lanzada, desde el principio, aunque sea exactamente la misma que se estaba tocando antes del UNDO. Si hacemos UNDO muchas veces seguidas, este comportamiento "natural" podría llegar a ser muy molesto. Si la música actual es la misma que la música que estaba sonando cuando se grabó la partida, de todas formas se "volverá a tocar" desde el principio, produciendo el mismo feo efecto.

**Damusix** es la primera extensión para el manejo de audio en Glulx que incorpora un mecanismo para evitar este "comportamiento natural" al cambiar los estados del juego. El *kernel* de **Damusix** implementa un complejo sistema de comprobaciones para determinar si es preciso "re-lanzar" un sonido que estuviera *"sonando de fondo"* al momento de hacer UNDO o al recuperar una partida previamente grabada. Si el nuevo sonido que está *"sonando de fondo" actualmente* es el mismo que el sonido que estaba *"sonando de fondo" previo al cambio de estado*, entonces no es necesario volver a lanzar la música porque es evidente que *"ya está sonando de fondo"*. ***("Sonando de fondo" quiere decir para Damusix un sonido que ya está en reproducción con repeticiones infinitas.)***

El mecanismo de "protección de sonidos" utiliza para sus propósitos una llamada estándar en ensamblador de Glulx: el opcode `@protect`. ***Este opcode debe estar correctamente soportado por el intérprete***; de otro modo, el mecanismo de "protección de sonidos" podría fallar o producir comportamientos inesperados.

Actualmente, el mecanismo de "protección de sonidos" de **Damusix** funciona perfectamente en **cualquier implementación del intérprete "Glulxe"** [sea *Windows Glulxe* o la versión de *Glulxe de Gargoyle*; también va muy bien en *ZAG* para Java y en *Spatterlight* para el sistema Mac OSX] y ***en cualquier implementación del intérprete "Git" *versión 1.2.3 o superior**** [sea *Windows Git* o la versión de *Git de Gargoyle*]. (*Versiones de "Git" anteriores no soportan este mecanismo especial de* ***Damusix***.)

En el muy especial caso de que necesitáramos que **Damusix** se ejecutara en intérpretes de Glulx que no soporten apropiadamente el opcode `@protect` (definitivamente NO recomendado), existe una **constante de compilación condicional para que la extensión compile *sin usar el mecanismo de "protección de sonidos"***. De esta manera, los sonidos siempre serán recuperados correctamente ante UNDO/RESTORE, pero ya no se comprobará si están actualmente "sonando de fondo", por lo que todo sonido recuperado siempre será "re-lanzado" (el comportamiento tradicional).

La constante se llama `DAMUSIX_NO_PROTEGER_SONIDOS` y debemos declararla al ***principio del código de nuestro juego, antes de incluir Damusix***. Con ello ya no se ocupará el mecanismo de "protección de sonidos".

**Con todo, se invita a los autores a no prescindir de este mecanismo**. Así podremos de alguna manera "animar amablemente" a los programadores de intérpretes a ir mejorándolos cada vez más y a incorporar las funcionalidades del estándar de Glulx que sus intérpretes todavía no soportan. *(Un poquito de "presión amigable", jejeje! =D)*

# Sobre los Puntos de Entrada Glk

Los *Puntos de Entrada Glk* son unas **rutinas especializadas** que la librería Inform dispone para que los programadores puedan "personalizar y controlar" los distintos comportamientos y funcionalidades presentes en la API Glk. De esta manera, el programador puede establecer cómo se debe comportar el juego cuando el jugador, por ejemplo, carga una partida grabada (mostrar imágenes previas, sonidos, etc.); o cuando se produce algún *evento Glk* (por ejemplo, cuando se redimensiona la ventana del juego o cuando un sonido produce una notificación de finalización).

*Para aprender más sobre los Puntos de Entrada Glk se puede recurrir a manuales de referencia como "Gull" (de Adam Cadre) o "La Guía para Programadores de Glulx Inform" (de Andrew Plotkin), ambos excelentemente ilustrativos.*

**Damusix** utiliza los siguientes Puntos de Entrada Glk para funcionar correctamente: `IdentifyGlkObject()` y `HandleGlkEvent()`. Estas dos rutinas son implementadas por **Damusix** de manera automática mediante la extensión de soporte **Dainunek**. Esta extensión ha sido creada con la finalidad de permitir que varias extensiones/librerías que ocupen los Puntos de Entrada Glk puedan usarse en conjunto, sin que el programador tenga que re-escribir dichos Puntos de Entrada combinando el código que cada extensión utiliza para sus propios fines. **Dainunek** se encarga de "colectar" todo ese código de los Puntos de Entrada Glk que cada extensión implementa y los ejecuta de manera unificada. Algo en verdad muy práctico.

En determinadas circunstancias puede ser que necesitemos programar nuestra propia versión del Punto de Entrada `IdentifyGlkObject()`. No tenemos más que escribirla siempre antes del `Include "Damusix"`. De esta manera, la extensión **Dainunek** reconocerá que nosotros mismos estamos implementando el Punto de Entrada Glk y ya no tratará de hacerlo automáticamente.

Si escribimos nuestra propia versión de `IdentifyGlkObject()` hay algo **MUY IMPORTANTE que no debemos olvidar**: dentro de aquel Punto de Entrada tenemos obligatoriamente que hacer una llamada a la *rutina-gancho* especial que habilita **Damusix** para estos casos: la rutina `Damusix.IdentificarSonidos()`. La llamada a esta *rutina-gancho* es **ABSOLUTAMENTE NECESARIA**. De otra manera, **Damusix** ni siquiera intentará funcionar.

El código de implementación de la rutina `IdentifyGlkObject()` de nuestro juego debería tener, más o menos, la siguiente forma:

```
[ IdentifyGlkObject fase tipo ref rock;
    ! Identificar todos los sonidos gestionados por Damusix
    Damusix.IdentificarSonidos(fase); ! argumento 'fase' es muy importante!!

    ! el resto de codigo que necesitemos poner aqui...
];
```

La *rutina-gancho* debe ser llamada **SIEMPRE** con el argumento `fase` (o como sea que se llame el *primer argumento* de tu `IdentifyGlkObject()` personalizada). Esto es **MUY IMPORTANTE** porque ese argumento le indica a **Damusix** en que "fase" de la identificación de objetos de sonido se encuentra (un asunto técnico de la API Glk).

Eso es todo lo que necesitamos hacer para que **Damusix** pueda funcionar correctamente si hemos escrito nuestra propia rutina `IdentifyGlkObject()`.

Un caso parecido ocurre con la rutina `HandleGlkEvent()`. **Damusix** la utiliza para el trabajo relacionado con los *Fades en "tiempo-real"*. Si queremos escribir nuestra propia versión de este Punto de Entrada Glk, tenemos que hacerlo siempre antes del `Include "Damusix"`, tal como en la explicación anterior. **Dainunek** tomará cuenta de nuestra versión "personalizada" y evitará implementar automáticamente dicha rutina.

Si escribimos nuestra propia versión de `HandleGlkEvent()`, entonces tenemos que llamar dentro de ésta a la *rutina-gancho* especial: `Damusix.NotificarFade()`. **Si no vamos a utilizar los *Fades en "tiempo-real"* no es obligatorio que llamemos a esta rutina-gancho.** El siguiente ejemplo muestra como implementar nuestra propia versión de este Punto de Entrada Glk para que los *Fades en "tiempo-real"* de **Damusix** sigan funcionando correctamente:

```
[ HandleGlkEvent ev contexto abortres;
    ! Notificar Efecto de Fade (In, Out) en "tiempo-real" de Damusix
    Damusix.NotificarFade(ev); ! argumento 'ev' es muy importante!!

    ! el resto de codigo que necesitemos poner aqui...
];
```

Nuevamente hay que observar que **SIEMPRE** debemos llamar a la *rutina-gancho* con el argumento `ev` (o como sea que se llame el *primer argumento* de tu `HandleGlkEvent()` personalizada). Esto es **MUY IMPORTANTE** porque ese argumento le indica a **Damusix** qué tipo de "evento" se ha generado en un determinado momento (y los *Fades en "tiempo-real"* usan los eventos "tick" del Timer Glk).

Ya que estamos tratando el tema, veamos una utilidad práctica para cuando llamamos a la rutina `Damusix.Tocar()` con el segundo argumento en valor `1`. Como ya lo explicamos, el primer argumento es el sonido que queremos tocar; el segundo argumento, si vale `1`, le indica a la rutina que el sonido que se va a tocar debe generar un evento de notificación del tipo `evtype_SoundNotify` que podemos capturar en la rutina `HandleGlkEvent()` escrita por nosotros mismos.

Entonces, ejecutar `Damusix.Tocar(FXSOUND,1)` hará que en el momento en que `FXSOUND` llegue al final de su reproducción se genere una notificación informando sobre este hecho.

Si capturamos ese evento de notificación podemos hacer que nuestro juego tenga un comportamiento especial cuando el evento se genere. Un ejemplo de uso práctico para todo esto podría ser: *¿cómo hacemos que cuando un sonido llegue a su fin, el juego comience a tocar otro sonido? ¡Pues con* `HandleGlkEvent()`*! =D*

Imaginemos que tenemos el siguiente código:

```
Damusix.Tocar(FXSOUND,1); ! tocar FXSOUND y notificar cuando termine de sonar
```

... esto hará que se reproduzca el sonido `FXSOUND` y que cuando éste finalice se genere una notificación de ello. Si nosotros capturamos esa notificación, pues podemos hacer que en aquel momento se toque un nuevo sonido. Para lograrlo, programaremos `HandleGlkEvent()` de la siguiente manera:

```
! lo usamos para que cuando se notifique "fin de la reproducción" de algún
! sonido  (FXSOUND en nuestro ejemplo), se comience a tocar el sonido BGMUSIC
[ HandleGlkEvent ev contexto abortres;
    contexto = abortres; ! para evitar warnings de variables no usadas

    ! Notificar Efecto de Fade (In, Out) en "tiempo-real" de Damusix
    Damusix.NotificarFade(ev); ! argumento 'ev' es muy importante!!

    ! Aquí capturamos los eventos que se generen
    switch (ev-->0) {
      evtype_SoundNotify: ! evento de notificacion de fin de sonido
        Damusix.Tocar(BGMUSIC); ! ahora tocamos la musiquita de fondo
    }
];
```

El código anterior se ejecutará **cada vez que se produzcan notificaciones de sonido**. En nuestro caso, como previamente hemos llamado a la rutina `Damusix.Tocar()` con el segundo argumento con valor `1`, efectivamente se enviará una notificación cuando `FXSOUND` termine de sonar, y en ese momento el código del ejemplo se encargará de iniciar la reproducción de la melodía de fondo `BGMUSIC`.

---

*Si has llegado hasta aquí, terminaste con éxito la audaz y laboriosa aventura de leer completamente esta extensa documentación.* ****** Has Ganado ******  *=D Jejeje!*

**Eso es todo. ¡Ahora tener sonido en tu juego en Glulx no podría ser más simple!**

---

# Epílogo

Si has decidido ocupar la extensión **Damusix** en tu juego, recibe mi agradecimiento. Mientras más personas la usen y revisen, más crecerá. Si detectas un BUG, por favor repórtalo a mi e-Mail. Y no olvides estar atento a las nuevas versiones de la extensión.

***¡Gracias por leer hasta aquí y por promover la Ficción Interactiva!***

Eliuk Blau

***eliukblau (AT) gmail.com***

# Agradecimientos

**Damusix** es mi creación. Sin embargo, la extensión no sería más que simplemente una buena idea si no fuera por la colaboración de varias personas que me brindaron su apoyo tanto moral como técnico durante cada paso del desarrollo...

**Muchas gracias Sarganar (Sebastián Arg)** por crear la librería INFSP que hace posible que el engranaje de **Damusix** se mueva; por la enorme paciencia que siempre me has tenido; por tu gran interés en mi extensión y por los ánimos constantes que me has dado desde mis inicios con Inform, cuando no sabía nada. Muchas gracias por tu increíble compañerismo y por permitirme aportar mi granito de arena en algo tan bueno como es tu proyecto INFSP.

**Muchas gracias Urbatain** por el ánimo que me diste en los comienzos de **Damusix**; por ser el primero en pensar que sería útil para su aventura y por tus testeos iniciales y sugerencias endiabladas sobre la necesidad de una "cola de sonidos", el volumen global y la minuciosidad en la documentación.

**Muchas gracias Jarel** por ser el estoico receptor de mis continuas consultas sobre decisiones técnicas importantes ("el de la opinión canónica", según Sarganar =P) y no huir despavorido ante tanta pregunta y verborrea informática. Tus recomendaciones han influido poderosa y positivamente en la implementación actual de **Damusix**.

**Muchas gracias Jenesis** por "invitarme amablemente" (¡amenazas y collejas! =D) a publicar un tutorial básico sobre la utilización de **Damusix**. El CAAD tendría un sabor muy distinto si no estuvieras con nosotros, "mami Jen". =P

**Muchas gracias Mel Hython** por seguir con tanto interés el desarrollo de **Damusix** (aunque sea por razones meramente personales, jeje! =P) y por tus significativos comentarios de "usuario novato en multimedia" que han permitido mejorar cosas que de otro modo no se me hubieran ocurrido.

**Muchas gracias Presi** por la paciencia que tuviste al ayudarme con el tema de la licencia LGPL. Tu experiencia me sirvió para aplicar la licencia de la manera correcta en la versión final de **Damusix**, a diferencia del descalabro con que la licencié en su versión temprana.

Y muchas gracias también a **Al-Khwarizmi**, **JB** y **Aaron A. Reed** por sus aportaciones en **Damusix para Inform7** (es *off-topic*, pero bueno... =P).

---

# Licencia



---

# Apéndice de Rutinas del Gestor

***Se usará la siguiente notación:***

- ✓ Para los argumentos **obligatorios**.
- ✗ Para los argumentos **opcionales que pueden ser omitidos**.

## Rutinas relacionadas con la Reproducción de Audio

**`Damusix.AsignarCanal(SONIDO,CANAL,VOLUMEN,REPETICIONES)`**
Asigna un sonido al canal indicado, asociándole un porcentaje de volumen y una cantidad de repeticiones. Las repeticiones son opcionales y si se omiten (cero), se asumirá 1 repetición. Si el sonido es incorrecto, devuelve `DAMUSIX_ERROR_SND` y termina. Si el canal indicado es incorrecto, devuelve `DAMUSIX_ERROR_CNL` y termina.

- ✓ `SONIDO` → Constante del recurso de sonido.
- ✓ `CANAL` → Número del canal (*0-9*).
- ✓ `VOLUMEN` → Porcentaje de volumen (*0-100; -1 = Vol. Global*).
- ✗ `REPETICIONES` → Número de repeticiones (*-1 = Repeticiones Infinitas*).

**`Damusix.LiberarCanal(SONIDO)`**
Libera el canal que se le ha asignado previamente a un sonido. Si el sonido no tiene canal asignado, devuelve `DAMUSIX_ERROR_SND` y termina.

- ✓ `SONIDO` → Constante del recurso de sonido.

**`Damusix.ActivarAudio()`**
Activa la salida de audio. El Gestor reproducirá todos los sonidos normalmente.

**`Damusix.DesactivarAudio()`**
Desactiva la salida de audio. El Gestor impedirá limpiamente cualquier intento de reproducción de sonidos. Al ejecutarse, detiene todos los *sonidos gestionados*, todas las *reproducciones virtuales* y también el *canal de lista de reproducción*.

**`Damusix.Tocar(SONIDO,FLAG_SOUND_NOTIFY)`**
Reproduce un sonido previamente asignado a un canal. Opcionalmente generará un evento `SoundNotify` cuando el sonido termine de sonar naturalmente. Si el sonido no tiene canal asignado, devuelve `DAMUSIX_ERROR_SND` y termina.

- ✓ `SONIDO` → Constante del recurso de sonido.
- ✗ `FLAG_SOUND_NOTIFY` → ¿Generar `SoundNotify`? (*0 = No, 1 = Sí*).

**`Damusix.Parar(SONIDO)`**
Detiene la reproducción de un sonido previamente asignado a un canal. Si el sonido no tiene canal asignado, devuelve `DAMUSIX_ERROR_SND` y termina.

- ✓ `SONIDO` → Constante del recurso de sonido.

**`Damusix.TocarV(SONIDO,VOLUMEN)`**
Reproduce "virtualmente" un sonido (lo toca en alguno de los canales virtuales). Si el sonido está o no asignado a un canal no es relevante. Opcionalmente se puede indicar un volumen personalizado para el sonido; si se omite (cero), se usará el valor actual del porcentaje de *Volumen Común para los canales virtuales*. Si el sonido es incorrecto, devuelve `DAMUSIX_ERROR_SND` y termina.

- ✓ `SONIDO` → Constante del recurso de sonido.
- × `VOLUMEN` → Porcentaje de volumen (*1-100; 0 = Vol. Común; -1 = Vol. Global*).

**`Damusix.CrearLista(SONIDO,DURACION)`**
Agrega un sonido a la lista de reproducción. Si el sonido está o no asignado a un canal no es relevante. El sonido ocupará el siguiente slot disponible de la lista. Se debe indicar la duración del sonido en milisegundos; si la duración se omite, se asumirán 1000ms (1 segundo). Si la lista está llena (los 10 slots están ocupados), la rutina muestra un mensaje de aviso (sólo en modo DEBUG) y termina. Si el sonido es incorrecto, devuelve `DAMUSIX_ERROR_SND` y termina.

- ✓ `SONIDO` → Constante del recurso de sonido.
- × `DURACION` → Duración del sonido en milisegundos (*por omisión = 1000ms*).

**`Damusix.LimpiarLista()`**
Limpia el contenido de la lista de reproducción, dejándola vacía y preparada para que se puedan agregar nuevos sonidos posteriormente.

**`Damusix.TocarLista(VOLUMEN)`**
Toca la lista de reproducción, previamente creada, reproduciendo los sonidos en ella uno a uno, en el orden en que fueron agregados y esperando el tiempo de duración indicado para cada uno antes de tocar el siguiente sonido de la lista. Cuando acaba de tocar todos los sonidos, limpia automáticamente el contenido de la lista, dejándola vacía y preparada para que se puedan agregar nuevos sonidos posteriormente. Opcionalmente se puede indicar un volumen personalizado para los sonidos de la lista; si se omite (cero), se usará el valor actual del porcentaje de *Volumen Común para la lista de reproducción de sonidos*. Si la lista de reproducción está vacía, se termina la ejecución sin hacer nada. **ESTA RUTINA USA EL TIMER GLK (no puede ejecutarse mientras esté activo un *Fade en "tiempo-real"*).**

- × `VOLUMEN` → Porcentaje de volumen (*1-100; 0 = Vol. Común; -1 = Vol. Global*).

**`Damusix.Repeticion(SONIDO,REPETICIONES)`**

Cambia el *modo de repetición* de un sonido; es decir, la cantidad de veces que un sonido se repetirá cuando esté en reproducción. Debe ser siempre un valor correcto mayor que cero. El valor especial -1 indica "repeticiones infinitas". Si el sonido no tiene canal asignado, devuelve `DAMUSIX_ERROR_SND` y termina.

- ✓ `SONIDO` → Constante del recurso de sonido.
- ✓ `REPETICIONES` → Número de repeticiones (*-1 = Repeticiones Infinitas*).

**`Damusix.Volumen(SONIDO,VOLUMEN)`**

Cambia el volumen de un sonido. El volumen se expresa en porcentaje y debe ser siempre un valor correcto entre 0-100. El valor especial -1 indica que debe ocuparse el porcentaje de *Volumen Global* actual. Si el sonido no tiene canal asignado, devuelve `DAMUSIX_ERROR_SND` y termina.

- ✓ `SONIDO` → Constante del recurso de sonido.
- ✓ `VOLUMEN` → Porcentaje de volumen (*0-100; -1 = Vol. Global*).

**`Damusix.VolumenV(VOLUMEN)`**

Cambia el porcentaje de *Volumen Común para todos los canales virtuales* (los que están implicados en la "reproducción virtual" de sonidos). El volumen se expresa en porcentaje y debe ser siempre un valor correcto entre 0-100. El valor especial -1 indica que debe ocuparse el porcentaje de *Volumen Global* actual.

- ✓ `VOLUMEN` → Porcentaje de volumen (*0-100; -1 = Vol. Global*).

**`Damusix.VolumenLista(VOLUMEN)`**

Cambia el porcentaje de *Volumen Común para la lista de reproducción de sonidos* (es decir, el volumen por defecto con el que se tocará cada sonido de la lista). El volumen se expresa en porcentaje y debe ser siempre un valor correcto entre 0-100. El valor especial -1 indica que debe ocuparse el porcentaje de *Volumen Global* actual.

- ✓ `VOLUMEN` → Porcentaje de volumen (*0-100; -1 = Vol. Global*).

**`Damusix.VolumenGlobal(VOLUMEN)`**

Cambia el porcentaje de *Volumen Global* del Gestor. Este cambio se aplicará a *todos los sonidos gestionados*, al *volumen común de los canales virtuales* y al *volumen común de la lista de reproducción de sonidos*; vale decir, absolutamente todos los volúmenes serán establecidos al valor del nuevo *Volumen Global*. El volumen se expresa en porcentaje y debe ser siempre un valor correcto entre 0-100; no se permite el valor especial -1 ya que no tiene sentido en esta rutina.

- ✓ `VOLUMEN` → Porcentaje de volumen (*0-100*).

**`Damusix.FadeIn(SONIDO,DURACION,VOL_FINAL)`**

Aplica un efecto de *FadeIn en "tiempo-real"* (*fade complejo*) al sonido que se le indica (dicho sonido debe estar en reproducción actualmente). En segundo plano, subirá gradualmente el volumen del sonido desde su valor actual hasta llegar a un máximo igual al *volumen final* `VOL_FINAL` indicado (este 3er argumento es opcional y debe tener un valor numérico correcto entre 1-100; si se omite o vale cero, se ocupará un valor por defecto correspondiente al *Volumen Global* actual). Se debe indicar la duración total del efecto de *FadeIn* en milisegundos; si la duración se omite, se asumirán 100ms (que es la duración mínima para cualquier *Fade en "tiempo-real"*). No puede aplicarse un *FadeIn* si otro efecto de *Fade en "tiempo-real"* (sea *FadeIn* o *FadeOut*) ya está activo en ese momento; en tal caso, la rutina muestra un mensaje de aviso (sólo en modo DEBUG) y termina. Si el sonido no tiene canal asignado, devuelve `DAMUSIX_ERROR_SND` y termina. **ESTA RUTINA USA EL TIMER GLK.**

- ✓ `SONIDO` → Constante del recurso de sonido.
- ✖ `DURACION` → Duración total del *FadeIn* en milisegs. (*por omisión = 100ms*).
- ✖ `VOL_FINAL` → Porcentaje de *volumen final* para el efecto (*1-100*).

**`Damusix.FadeOut(SONIDO,DURACION,VOL_FINAL,PFADE_SND,PFADE_SND_NOTIFY)`**

Aplica un efecto de *FadeOut en "tiempo-real"* (*fade complejo*) al sonido que se le indica (dicho sonido debe estar en reproducción actualmente). En segundo plano, bajará gradualmente el volumen del sonido desde su valor actual hasta llegar a un mínimo igual al *volumen final* `VOL_FINAL` indicado (este 3er argumento es opcional y debe tener un valor numérico correcto entre 1-100; si se omite o vale cero, se ocupará un valor por defecto correspondiente al mínimo de volumen, es decir 0%). Se debe indicar la duración total del efecto de *FadeOut* en milisegundos; si la duración se omite, se asumirán 100ms (que es la duración mínima para cualquier *Fade en "tiempo-real"*). Si el *volumen final* es igual a 0%, entonces cuando el *FadeOut* termine se detendrá artificialmente la reproducción del sonido (no tiene sentido seguirlo tocando con 0% de volumen) y luego se restablecerá su volumen, usando para ello el *valor de volumen que tenía originalmente el sonido* antes de realizar el *FadeOut*. Opcionalmente, si el *volumen final* es igual a 0%, puede indicarse un sonido adicional *Post-FadeOut*; es decir, un sonido que se tocará una vez que el efecto de *FadeOut haya finalizado*. También opcionalmente se puede generar un evento `SoundNotify` cuando el sonido *Post-FadeOut* termine de sonar naturalmente. (*Si el "volumen final" es distinto de 0%, entonces estos dos últimos argumentos opcionales simplemente serán omitidos.*) No puede aplicarse un *FadeOut* si otro efecto de *Fade en "tiempo-real"* (sea *FadeIn* o *FadeOut*) ya está activo en ese momento; en tal caso, la rutina muestra un mensaje de aviso (sólo en modo DEBUG) y termina. Si el sonido no tiene canal asignado, devuelve `DAMUSIX_ERROR_SND` y termina. **ESTA RUTINA USA EL TIMER GLK.**

- ✓ `SONIDO` → Constante del recurso de sonido.
- ✖ `DURACION` → Duración total del *FadeIn* en milisegs. (*por omisión = 100ms*).
- ✖ `VOL_FINAL` → Porcentaje de *volumen final* para el efecto (*1-100*).
- ✖ `PFADE_SND` → Constante del recurso de sonido que se tocará *Post-FadeOut*.
- ✖ `PFADE_SND_NOTIFY` → ¿`SoundNotify` en sonido *Post-FadeOut*? (0 = No, 1 = Sí).

`Damusix.AbortarFade()`

Abortará cualquier efecto de *Fade en "tiempo-real"* que pudiera estar en proceso. Si se aborta un *FadeIn*, el sonido seguirá en reproducción y se le aplicará el *volumen final* indicado cuando se lanzó el efecto. Si se aborta un *FadeOut*, existen dos posible comportamientos: 1) si el *volumen final* es mayor a 0%, entonces simplemente se le aplicará al sonido dicho *volumen final* y finalmente se terminará el Fade; 2) si el *volumen final* es igual a 0%, entonces se detendrá efectivamente la reproducción del sonido, luego se le aplicará un *volumen final igual al volumen que tenía originalmente antes de realizar el efecto* y finalmente se terminará el Fade. Si no hay ningún efecto de Fade activo en el momento de la llamada, se termina la ejecución sin hacer nada. **NOTA ADICIONAL: cualquier rutina del Gestor que se aplique en un sonido que tenga activo un *Fade en "tiempo-real"*, primero abortará el efecto si lo considera apropiado y luego continuará con su comportamiento normal.**

`Damusix.SimpleFadeIn(SONIDO,DURACION,VOL_FINAL)`

Aplica un efecto de *FadeIn en "tiempo-no-real"* (*fade simple*) al sonido que se le indica. Subirá gradualmente el volumen del sonido desde su valor actual hasta llegar a un máximo igual al *volumen final* `VOL_FINAL` indicado (este 3er argumento es opcional y debe tener un valor numérico correcto entre 1-100; si se omite o vale cero, se ocupará un valor por defecto correspondiente al *Volumen Global* actual). Sólo cuando el efecto haya acabado se devolverá el control de la ejecución a la siguiente línea del programa. Se debe indicar la duración total del efecto de *SimpleFadeIn* en milisegundos, siendo siempre un valor mayor a cero; si la duración se omite, se asumirán 100ms. Como un *SimpleFadeIn* es en *"tiempo-no-real"*, no es necesario tener la opción de abortarlo ni de consultar si está activo. Si el sonido no tiene canal asignado, devuelve `DAMUSIX_ERROR_SND` y termina. **ESTA RUTINA USA EL TIMER GLK (no puede ejecutarse mientras esté activo un *Fade en "tiempo-real"*).**

- ✓ `SONIDO` → Constante del recurso de sonido.
- ✗ `DURACION` → Duración total del *SimpleFadeIn* en ms. (*por omisión = 100ms*).
- ✗ `VOL_FINAL` → Porcentaje de *volumen final* para el efecto (*1-100*).

**`Damusix.SimpleFadeOut(SONIDO,DURACION,VOL_FINAL)`**
Aplica un efecto de *FadeOut en "tiempo-no-real"* (*fade simple*) al sonido que se le indica. Bajará gradualmente el volumen del sonido desde su valor actual hasta llegar a un mínimo igual al *volumen final* `VOL_FINAL` indicado (este 3er argumento es opcional y debe tener un valor numérico correcto entre 1-100; si se omite o vale cero, se ocupará un valor por defecto correspondiente al mínimo de volumen, es decir 0%). Sólo cuando el efecto haya acabado se devolverá el control de la ejecución a la siguiente línea del programa. Se debe indicar la duración total del efecto de *SimpleFadeOut* en milisegundos, siendo siempre un valor mayor a cero; si la duración se omite, se asumirán 100ms. Como un *SimpleFadeOut* es en *"tiempo-no-real"*, no es necesario tener la opción de abortarlo ni de consultar si está activo. A diferencia de un *FadeOut en "tiempo-real"*, un *SimpleFadeOut* nunca detendrá el sonido una vez que haya acabado el efecto (si es que el *volumen final* indicado es igual a 0%), ya que el programador tiene la opción de detenerlo por sí mismo en la siguiente línea del código. Si el sonido no tiene canal asignado, devuelve `DAMUSIX_ERROR_SND` y termina. **ESTA RUTINA USA EL TIMER GLK (no puede ejecutarse mientras esté activo un *Fade en "tiempo-real"*).**

- ✓ `SONIDO` → Constante del recurso de sonido.
- × `DURACION` → Duración total del *SimpleFadeOut* en ms. (*por omisión = 100ms*).
- × `VOL_FINAL` → Porcentaje de *volumen final* para el efecto (*1-100*).

## Rutinas Técnicas relacionadas con la Reproducción de Audio

**`Damusix.TocarCanal(CANAL,FLAG_SOUND_NOTIFY)`**
Reproduce el sonido asignado al canal indicado. Opcionalmente generará un evento `SoundNotify` cuando el sonido termine de sonar naturalmente. Si el canal indicado es incorrecto, devuelve `DAMUSIX_ERROR_CNL` y termina.

- ✓ `CANAL` → Número del canal (*0-9*).
- × `FLAG_SOUND_NOTIFY` → ¿Generar `SoundNotify`? (*0 = No, 1 = Sí*).

**`Damusix.PararCanal(CANAL)`**
Detiene la reproducción del sonido asignado al canal indicado. Si el canal indicado es incorrecto, devuelve `DAMUSIX_ERROR_CNL` y termina.

- ✓ `CANAL` → Número del canal (*0-9*).

**`Damusix.PararCanalesExtra()`**
Detiene la reproducción en todos los *canales virtuales* (*sonidos en reproducción virtual*) y, por seguridad, en el canal de la *lista de reproducción de sonidos*.

`Damusix.PararTodo()`
Detiene absolutamente todos los canales inicializados por el Gestor. Esto quiere decir que serán detenidos sin excepción todos los *sonidos gestionados*, todos los *sonidos en reproducción virtual* y, por seguridad, el canal de la *lista de reproducción de sonidos*.

`Damusix.RepeticionCanal(CANAL,REPETICIONES)`
Cambia el *modo de repetición* del sonido asignado al canal indicado; es decir, la cantidad de veces que ese sonido se repetirá cuando esté en reproducción. Debe ser siempre un valor correcto mayor que cero. El valor especial -1 indica "repeticiones infinitas". Si el canal indicado es incorrecto, devuelve `DAMUSIX_ERROR_CNL` y termina.

- ✓ `CANAL` → Número del canal (*0-9*).
- ✓ `REPETICIONES` → Número de repeticiones (*-1 = Repeticiones Infinitas*).

`Damusix.VolumenCanal(CANAL,VOLUMEN)`
Cambia el volumen del sonido asignado al canal indicado. El volumen se expresa en porcentaje y debe ser siempre un valor correcto entre 0-100. El valor especial -1 indica que debe ocuparse el porcentaje de *Volumen Global* actual. Si el canal indicado es incorrecto, devuelve `DAMUSIX_ERROR_CNL` y termina.

- ✓ `CANAL` → Número del canal (*0-9*).
- ✓ `VOLUMEN` → Porcentaje de volumen (*0-100; -1 = Vol. Global*).

## Rutinas de Consulta que devuelven un Valor

`Damusix.BuscarCanal(SONIDO)`
**[TÉCNICA]** Averigua el canal que se le ha asignado a un sonido. Es quizá la rutina más técnica del Gestor y la más importante de **Damusix**. Sirve de base para toda la *"abstracción por sonidos"* de la extensión.

- ✓ `SONIDO` → Constante del recurso de sonido.

***Valores devueltos:***
- Número del canal asignado al sonido.
- `DAMUSIX_ERROR_SND` si el sonido no tiene canal asignado.

`Damusix.BuscarSonido(CANAL)`
**[TÉCNICA]** Averigua el sonido que se le ha asignado a un canal. Es la contraparte de la rutina `Damusix.BuscarCanal()`.

- ✓ `CANAL` → Número del canal (*0-9*).

***Valores devueltos:***

- Constante del recurso de sonido asignado al canal (*0 = ningún sonido asignado*)
- `DAMUSIX_ERROR_CNL` si el canal indicado es incorrecto.

`Damusix.TestAudio()`
**[UTILITARIA]** Comprueba si el intérprete actual tiene *soporte completo de audio*. Esto significa la posibilidad de poder reproducir archivos de audio en *formato MOD* y en *formato sampleado AIFF u Ogg Vorbis*.

***Valores devueltos:***

- 1 si el intérprete tiene soporte completo de audio.
- 0 si el intérprete no tiene soporte completo de audio.

`Damusix.HayAudio()`
Comprueba el estado actual de la *salida de audio* del Gestor; es decir, si la salida de audio está *activada* o *desactivada*.

***Valores devueltos:***

- 1 si la salida de audio está activada.
- 0 si la salida de audio está desactivada.

`Damusix.SonidosLista()`
Averigua la cantidad de sonidos agregados actualmente a la *lista de reproducción de sonidos*.

***Valores devueltos:***

- Número total de sonidos agregados actualmente a la *lista de reproducción*.

`Damusix.QueRepeticion(SONIDO)`
Averigua el *modo de repetición* actual de un sonido; es decir, la cantidad de veces que un sonido se repetirá cuando esté en reproducción.

- ✓ `SONIDO` → Constante del recurso de sonido.

***Valores devueltos:***

- Número de repeticiones para el sonido (*-1 = Repeticiones Infinitas*).
- `DAMUSIX_ERROR_SND` si el sonido no tiene canal asignado.

**`Damusix.QueVolumen(SONIDO)`**
Averigua el volumen actual de un sonido.

- ✓ `SONIDO` → Constante del recurso de sonido.

***Valores devueltos:***
- Porcentaje de volumen del sonido.
- `DAMUSIX_ERROR_SND` si el sonido no tiene canal asignado.

**`Damusix.QueVolumenV()`**
Averigua el porcentaje de *Volumen Común* actual para todos los *canales virtuales* (los que están implicados en la "reproducción virtual" de sonidos).

***Valores devueltos:***
- Porcentaje de *Volumen Común* actual para los *canales virtuales*.

**`Damusix.QueVolumenLista()`**
Averigua el porcentaje de *Volumen Común* actual para la *lista de reproducción de sonidos* (es decir, el volumen por defecto con el que se tocará cada sonido de la lista).

***Valores devueltos:***
- Porcentaje de *Volumen Común* actual para la *lista de reproducción de sonidos*.

**`Damusix.QueVolumenGlobal()`**
Averigua el porcentaje de *Volumen Global* actual del Gestor.

***Valores devueltos:***
- Porcentaje de *Volumen Global* actual del Gestor.

**`Damusix.SonandoDeFondo(SONIDO)`**
Comprueba si actualmente un sonido está *sonando de fondo* (es decir, *en reproducción con "repeticiones infinitas"*).

- ✓ `SONIDO` → Constante del recurso de sonido.

***Valores devueltos:***
- 1 si el sonido está actualmente *sonando de fondo*.
- 0 si actualmente no está *sonando de fondo* (o si no tiene canal asignado).

`Damusix.EnFade()`
Comprueba si actualmente está activo un *efecto de Fade en "tiempo-real"* (sea éste un *FadeIn* o *FadeOut*).

***Valores devueltos:***

- 1 si actualmente está activo un *Fade en "tiempo-real"*.
- 0 si actualmente no hay ningún trabajo de *Fade en "tiempo-real"* activo.

`Damusix.QueRepeticionCanal(CANAL)`
**[TÉCNICA]** Averigua el *modo de repetición* actual del sonido asignado al canal indicado; es decir, la cantidad de veces que ese sonido se repetirá cuando esté en reproducción.

- ✓ `CANAL` → Número del canal (*0-9*).

***Valores devueltos:***

- Número de rep. para el sonido asignado al canal indicado (-1 = *"rep. infinitas"*).
- `DAMUSIX_ERROR_CNL` si el canal indicado es incorrecto.

`Damusix.QueVolumenCanal(CANAL)`
**[TÉCNICA]** Averigua el volumen actual del sonido asignado al canal indicado.

- ✓ `CANAL` → Número del canal (*0-9*).

***Valores devueltos:***

- Porcentaje de volumen del sonido asignado al canal indicado.
- `DAMUSIX_ERROR_CNL` si el canal indicado es incorrecto.

`Damusix.SonandoDeFondoCanal(CANAL)`
**[TÉCNICA]** Comprueba si el sonido asignado al canal indicado está actualmente *sonando de fondo* (es decir, *en reproducción con "repeticiones infinitas"*).

- ✓ `CANAL` → Número del canal (*0-9*).

***Valores devueltos:***

- 1 si el sonido asignado al canal indicado está actualmente *sonando de fondo*.
- 0 si actualmente no está *sonando de fondo.*
- `DAMUSIX_ERROR_CNL` si el canal indicado es incorrecto.