# Tutorial de Independiente

## Sumario

Este es nuestro primer tutorial básico de C++. También muestra como escribir un ejecutable independiente usando orx y como usar el módulo de localización (`orxLOCALE`).

Como **NO** estamos usando el ejecutable por defecto para este tutorial, su código será directamente compilado en un ejecutable y no dentro de una librería externa.

Esto implica que **NO** tendremos el comportamiento codificado por defecto que tuvimos en los tutoriales anteriores:

- F11 no afectará el cambiador de sincronia vertical.
- Escape no saldrá de la aplicación automáticamente.
- F12 no captura una imagen.
- Backspace no recarga ficheros de configuración.
- La sección [Main] en el fichero de configuración no será usada para cargar un plugin (llave `GameFile`).

Un programa basado directamente en orx [1)], por defecto, **NO** saldrá de la aplicación si recibe el evento orxSYSTEM_EVENT_CLOSE.
Para hacer esto, o tendriamos que usar la función auxiliar `orx_Execute()` (ver siguiente) o manejarlo por nuestra cuenta.

Ver los anteriores tutoriales básicos para más información sobre la creación básica de objetos, manejo del reloj, jerarquía de fotogramas, animaciones, cámaras & vistas, música & sonido, efectos(FXs), física y desplazamiento.

Como estamos por nuestra cuenta aquí, necesitamos escribir la función principal e inicializarla manualmente con orx.
La parte buena es que podemos entonces especificar que módulo queremos usar, y desactivar la pantalla o cualquier otro módulo a voluntad, si fuera necesario.

Si quisieramos mantener una semi automática inicialización de orx, podemos usar la función `orx_Execute()`.
Este tutorial cubrirá el uso de orx con su función auxiliar, pero puedes decidir si no la usas su su comportamiento no sirve para tus necesidades.

Esta función auxiliar tendrá cuidado de inicializar todo correctamente y salir adecuadamente.
Estará también segura que el módulo del reloj está marcando constantemente (como parte del núcleo de orx) y que podamos salir si el evento orxSYSTEM_EVENT_CLOSE fue enviado.
Este evento es enviado cuando cerramos una ventana, por ejemplo, pero puede ser enviado por criterio propio (la tecla escape es presionado, por ejemplo).

Este código es un ejemplo básico de C++ para mostrar como usar orx sin tener que escribir código de C.
Este tutorial pudo haber estado estructurado de una mejor manera (cortandolo en piezas con encabezados de ficheros, por ejemplo) pero queremos mantener un solo fichero por tutorial *básico*.

Este ejecutable independiente también crea una consola (como hace el ejecutable de orx por defecto), pero tu puedes tener tu propio programa sin consola si así lo deseas.
A fin de lograr eso, solo necesitas proveer un listado de argumentos que contenga el nombre del ejecutable.
Si no, el fichero cargado por defecto será orx.ini en vez del que está basado en el nombre de nuestro ejecutable (ej. 10_StandAlone.ini).

Los usuarios(windows) de Visual Studio, fácilmente pueden lograr esto escribiendo una función `WinMain()` en vez de `main()`, y obteniendo el nombre del ejecutable (o hacerlo a mano, como se hace sin pudor en este tutorial 😀)

Este tutorial simplemente muestra el logo de orx y una leyenda localizada. Presione espacio o el botón click izquierdo para pasar por todas las lenguas disponibles para la leyenda del texto.

Algunas explicaciones acerca de elementos del núcleo puedes encontrarlas en este tutorial:

- `Función correr`(`Run function`): No ponga *ningún* código lógico aquí, es solo en la columna vertebral donde puedes manejar por defecto los comportamientos del núcleo(rastreando la salida o cambiando la localización, por ejemplo) o perfilar algunas cosas. Como esto es llamado llamado directamente desde un ciclo principal y no como parte del reloj del sistema, la consistencia en el tiempo no se puede imponer. Para todas las ejecuciones principales de tu juego, por favor crea un reloj(o usa uno existente) y registra tu llamada de retorno(callback) a el.
- `Controladores de Evento`(`EventHandlers`): Cuando un controlador de evento retorna orxSTATUS_SUCCESS, ningún otro controlador será llamado después de el por el mismo evento. En la otra mano, si orxSTATUS_FAILURE, es retornado, el procesamiento de eventos continuará durante ese evento si los controladores de otros están escuchando este tipo de evento. Vamos a supervisar los eventos de localización para actualizar el texto de nuestra leyenda, cuando el idioma seleccionado se cambia.
- `orx_Execute()`: Inicia y ejecuta orx usando nuestra propia función definida(Init, Run y Exit). Podemos, claro, no usar este auxiliar y controlar todo manualmente, si este comportamiento no suple nuestras necesidades. Puedes echar un vistazo al contenido de `orx_Execute()` [2] para tener una mejor idea de como se hace esto.

# Detalles

# Recursos

[1]
ej. sin la ayuda del lanzador orx
[2]
que está implementado en `orx.h`