Sintaxis de configuración

Sintaxis básica

Como es tradicional en los ficheros INI, toda la información de configuración es expresada usando parejas de llave/valor.
Estas parejas están organizadas en secciones.

Una sección es declarada con corchetes ([] - ES-ES). Todas las parejas de llave/valor definidas debajo serán parte de esta sección.
Esta sección termina cuando otra comienza.

```
[Section]
```

Llaves y valores son delimitados con signo de igual (=).

```
Key = Value
```

Punto y coma (;) indica el comienzo de un comentario. El comentario continua hasta el final de la linea.

```
; Esto es un comentario y será ignorado por el módulo de configuración
```

Las secciones pueden ser definidas en más de un lugar. Incluso pueden abarcar varios ficheros de configuración.

He aquí un ejemplo.

```
; Aquí tenemos un ejemplo de una sintaxis de configuración de orx
[MySection] ; Esto define el inicio de 'MiSección'('MySection')
MyKey      = MyValue; Aquí damos un valor a 'MiLlave'('MyKey')
MyOtherKey = MyOtherValue; Y aquí damos uno para 'MiOtraLlave('MyOtherKey')

[MyOtherSection]; Aquí termina 'MySection' y estamos ahora en
'MiOtraSección'('MyOtherSection')
AKey = Otro valor

[MySection]; Estamos de regreso a 'MySection'
MyLastKey = MyLastValue; Añadiendo otra pareja llave/valor a 'MySection'
```

PD: Los espacios alrededor del operador de asignación ('=') son recortados y simplemente ignorados por la configuración del sistema.

Si quieres usar un ';' como parte de un valor no-númerico, necesitas usar una asignación en bloque. Los bloques son delimitados con comillas dobles . Los bloques también son la única manera de tener valores cubriendo varias lineas.

```
MyKey      = "MyValuePart1 ; MyValuePart2"
```

```
MyOtherKey = "Este valor
se extiende
en múltiples lineas"
```

Si doblas las primeras , la cadena no será considerada como un bloque pero si como un valor normal y tendremos un como inicio del valor.

```
MyKey = ""MyQuotedValue"
```

Aquí la cadena MiValorCitado(MyQuotedValued) (incluyendo las comillas dobles), serán almacenadas como un valor para la llave 'MyKey'.

Herencia

El sistema de herencia esta basado en la misma idea que la herencia en programación orientada a objetos.

La idea básica es que todas las llaves definidas en una sección pueden ser heredadas por cualquier otra sección[1)].
La sección hereditaria(acá, la sección hija) puede entonces añadir nuevas llaves o sobreescribir cualquier llave definida en la sección padre.

Para hacer esto, la arroba('@' - ES-ES) es usada como marcador de herencia.

```
[Parent]
MyKey1 = MyValue1
MyKey2 = MyValue2

[Child@Parent]; <= La sección 'Hijo'('Child') contiene ahora todas las
parejas llave/valor definidas en la sección 'Padre'('Parent')
```

Si no quieres heredar una sección entera, puedes también usar herencia directa de una sola llave.
Si la llave padre no tiene el mismo nombre del hijo, puedes especificar su nombre completo usando el separador punto('.') además del marcador de herencia.

```
[Parent]
MyKey      = MyValue
MyOtherKey = MyOtherValue

[Child]
MyKey     = @Parent; <= El valor de 'MyKey' será heredado de uno definido en
la sección 'Parent', con el mismo nombre de llave.
MyLastKey = @Parent.MyKey; <= El valor para 'MiÚltimaLlave'('MyLastKey')
será heredado de la llave 'MyKey' definida en la sección 'Parent'.
```

En el ejemplo anterior, vimos que ambas 'MyKey' y 'MyLastKey' de la sección 'Child' heredan de 'MyKey' de la sección 'Parent'.
En este ejemplo, la llave 'MyOtherKey' no será heredada en la sección 'Child'.

Cuando usamos herencia, cuando el valor de las llaves padres cambian, incluso en tiempo de ejecución, el valor hijo será cambiado.
Valores heredados pueden ser cambiados como vemos en el siguiente ejemplo.

```
[GrandParent]
MyKey      = MyValue
MyOtherKey = MyOtherValue

[Parent]
MyKey = @GrandParent

[Child@Parent]
```

En la sección 'Child', solo una llave llamada 'MyKey' es definida. Su valor es heredado de la sección 'Parent' quien lo heredó del 'Abuelo'('GrandParent'). En otras palabras, cuando `GrandParent.MyKey` cambia, ambas `Parent.MyKey` y `Child.MyKey` serán afectadas.

Includes

Un fichero de configuración puede incluir cualquier número de ficheros. Esto ayuda a mantener cortos los ficheros de configuración y concentrarnos en otro aspecto de nuestro juego.
Por ejemplo, puedes definir todos las propiedades de objetos/configuraciones de tu UI en un fichero llamado ui.ini y etc...
Aquí está la sintaxis.

```
@path/to/MyIncludedFile@
```

PD: El camino usado es relativo al directorio de trabajo correspondiente y no para el camino del fichero de configuración que contiene!

Por favor, nota también que el include será hecho en el lugar. Esto significa que cualquier valor definido en el fichero include puede sobreescribir cualquier valor previamente definido. De la misma manera, cualquier valor definido en los ficheros include pueden entonces reemplazarse después.

Cuando escribimos cualquier pareja llave/valor después del include sin redefinir una nueva sección, la última sección definida antes del include será usada.

```
[MySection]
Key1 = Var1

@IncludeFile@

Key2 = Var2; <= esto todavía se añadirá a la sección 'MySection'
```

Valores numéricos

Tipos básicos

Los valores de punto flotante (float) son expresados usando un separador decimal ('.').

```
MyFloat = 3.5
```

Los enteros pueden ser expresado en las bases mas comunes usando prefijos:

- Decimal, sin usar prefijo

```
DecimalValue = 16
```

- Hexadecimal, prefijo '0x'

```
HexadecimalValue = 0x10
```

- Octal, prefijo '0'

```
OctalValue = 020
```

- Binario, prefijo '0b'

```
ValorBinario = 0b10000
```

Vector

Los vectores son siempre definidos usando tres componentes, separados por comas (',') y encerrados por tanto entre parentesis ('()') o llaves ('{}'), sin distinción.

```
MyVector      = (1.0, 2.0, 3.0)
MyOtherVector = {4, 5, 6}
```

Los vectores son usados, la mayoría del tiempo, para representar coordenadas o componentes de color (r, g, b).

Aleatorio

Cuando valores numéricos son usados [2)], un valor generado aleatoriamente pueden ser obtenido usando el caracter tilde ('~') para separar los límites bajo y alto.

```
RandomInt    = 1 ~ 10
RandomFloat  = 0.5 ~ 1.0
RandomVector = (0.0, 0.0, 0.0) ~ (1.0, 1.0, 1.0)
```

Cada vez que se consulta el valor de una clave aleatoria en el código, un nuevo valor al azar entre los límites especificados se generará.

Listas

Listas de valores para una sola llave es también soportado. Todos los elementos son separados usando el caracter número ('#'). Las listas pueden contener valores aleatorios como elementos.

```
ListValue = Val1 # Val2 # RandVal3 ~ RandVal4 #Val5
```

Si hay espacios definidos alrededor de los separadores de la lista ('#') serán ignorados.

PD: Espacios alrededor de los delimitadores ('#') son recortados y simplemente ignorados por la configuración del sistema.
PD: Cuando pedimos listas usando la configuración de la API, por favor revise las cunciones que contengan la palabra 'Lista'('List') en su nombre. Si usted usa una función no-lista para pedir una llave que contenga un valor lista, cualquier valor de la lista será aleatoriamente retornado.

Pero visalicemos esto con un ejemplo. 
Código

```
orxFLOAT fMyFloat = orxConfig_GetFloat("MyFloat");
```

INI

```
MyFloat = 1.0 # 2.0 # 3.0
```

La función retornará aleatoriamente entonces 1.0, 2.0 o 3.0.
Si el valor fue definido usando un separador de aleatoriedad ('~'), ningún valor flotante entre 1.0 y 3.0 puede ser devuelto.
Note, por favor que aleatorios controlados usando listas funcionan también con valores no-numéricos.

[1]
herencia cíclica **no** está soportada
[2]
Enteros, Flotantes y Vectores

From:
https://www.orx-project.org/wiki/ - **Orx Learning**

Permanent link:
https://www.orx-project.org/wiki/es/orx/config/syntax?rev=1331144792

Last update: **2025/09/30 17:26 (14 months ago)**