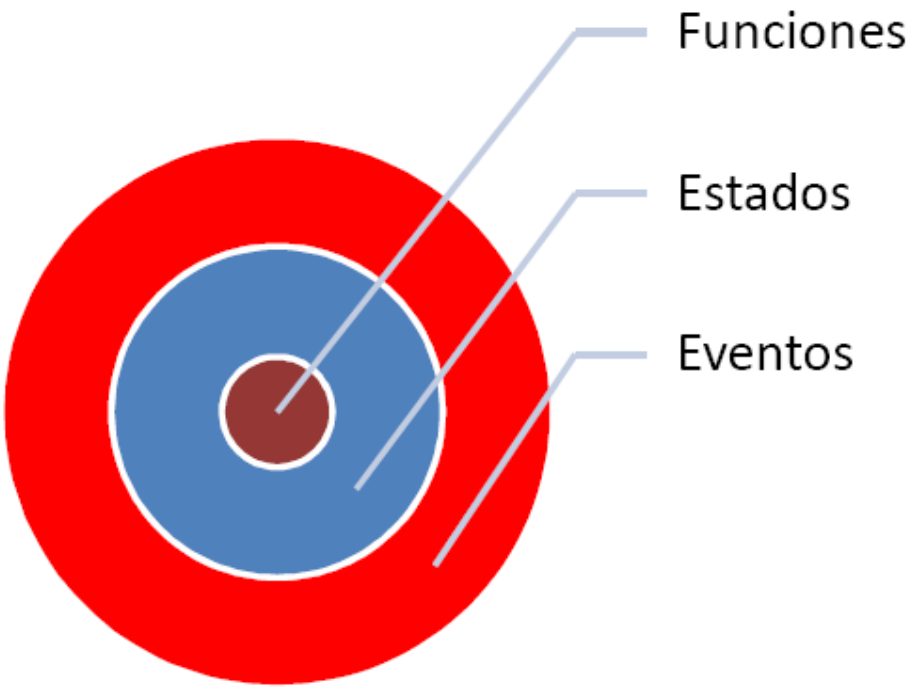
Guía de iniciación Linden Scripting Language

Contenido

¿Qué es Linden Scripting Language?

Linden Scripting Language (LSL) es un lenguaje de programación orientado a eventos, similar a C y java. Con este lenguaje de programación podremos crear scripts para controlar el comportamiento de los objetos en el mundo Vleaf.

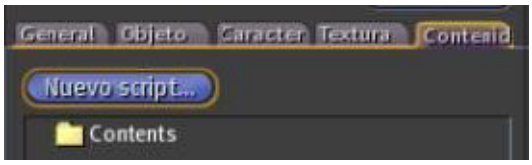
Los scripts están compuesto de uno o varios estados, dentro de cada estado hay uno o varios eventos y dentro de estos una o varias funciones.



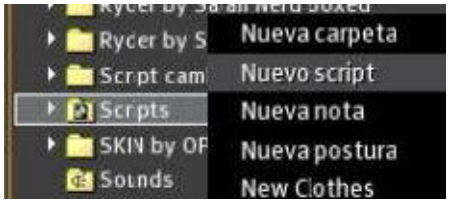
¿Cómo crear un script?

Primer de todo tenemos que crear un objeto. Para ello pulsamos en el botón construir en la parte baja de la pantalla (al lado del botón de “Buscar” y “Minimapa”) o bien pulsamos el botón derecho en el terreno y seleccionar la opción “Crear”. Aparecerá una ventana con las formas primitivas (Prims) disponibles, si desplazamos el ratón fuera de esta ventana vemos como el cursor se transforma en una varita mágica, pulsamos con botón izquierdo del ratón sobre cualquier punto del terreno y aparecerá la forma seleccionada.

Los script se asocian a objetos que creamos. Podemos crear un script directamente dentro del objeto, pulsando en “Nuevo script” desde la pestaña “Contenido” de la ventana de edición del objeto.

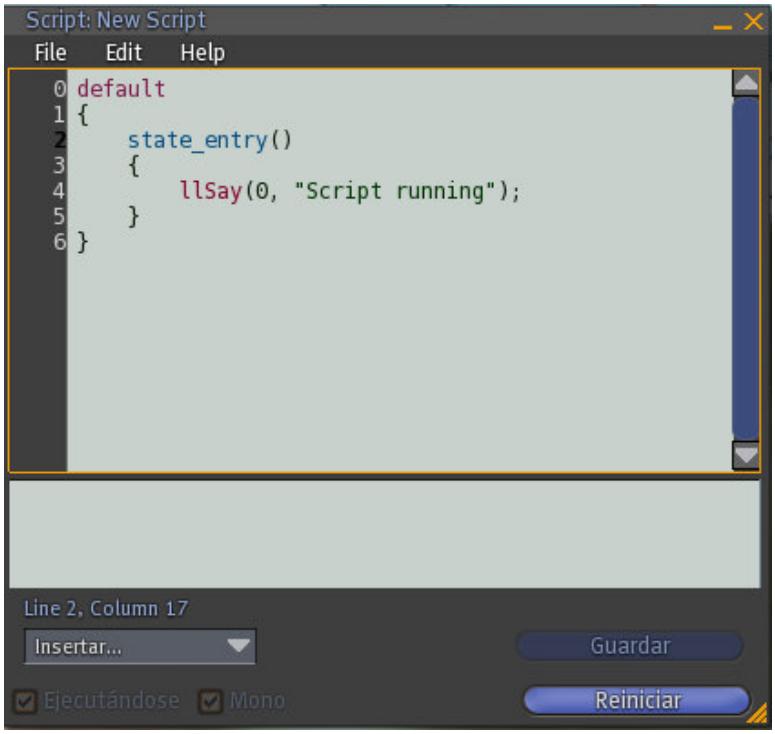


O directamente en el inventario para más tarde arrastrarlo al inventario del objeto en la pestaña “Contenido”.



Creando el primer script:

Una vez creado un script, al hacer doble clic en él nos aparecerá una ventana llamada editor de script desde la cual podremos modificar nuestro script.



Vemos que el script comienza con el estado default, todos los scripts deben tener definido obligatoriamente el default que será el primero que se ejecute, el texto azul corresponde a los estados, el texto rojo a las funciones

Borramos el contenido del script y escribimos lo siguiente:

```
default
{       //con esta llave abrimos el estado default
        touch_start (integer total_number)          //Este es el evento touch_start ()
        {       //abrimos el evento touch_start ()
                llSay (0, "Me has tocado");         //esta es la función llSay ()
        }       // Cerramos el evento touch_start ()

        collision_start (integer num)
        {       //abrimos el evento collision_start ()
                llSay (0, "Te has chocado");
        }       //cerramos el evento collision_start ()
}       // Cerramos el estado default
```

Este script tiene dentro del estado default dos eventos:

1. touch_start (integer total_number)

   Cuando toquemos el objeto se ejecutaran las funciones del interior de este evento, en ese caso la función es llSay (0, " Me has tocado ");

   Con la función llSay () hacemos que el objeto diga algo por un canal, como vemos la función llSay () tiene dos parámetros dentro del paréntesis separados por una coma (0, " Me has tocado ") el primero indica el canal y el segundo es el texto que queremos que diga el objeto que debe ir siempre entre comillas.

   En este caso al tocar el objeto este nos dirá “Me has tocado” por el canal 0 que es del chat y por lo tanto lo escucharían todos los avatares que estuviesen a menos de 20 metros del objeto.

2. collision_start (integer num)

   Las funciones del interior de este evento se ejecutaran al chocarnos con el objeto, en este caso al chocarnos el objeto nos dirá “Te has chocado”, por el canal 0

Como vemos las funciones siempre tienen que ir dentro de las llaves que abren y cierran un eventos y terminadas en”;” y los eventos siempre tienen que ir dentro de las llaves que aben y cierran los estados

Con // podemos escribir anotaciones dentro de los script y que no interfieran en ellos

Poner texto sobre un objeto:

Para poner un texto sobre un objeto se usa la función llSetText (), por ejemplo:

llSetText (“Hola”, <1.0, 0.0, 0.0>, 1);

Esta función tiene 3 parámetros separados por comas:

1. El primero es el texto el cual queremos poner encima del objeto que debe ir siempre entre comillas.
2. El segundo es un vector que nos indica el color del texto. Una forma de representar el color en la informática es el RGB (red, green, blue). La intensidad de un color se tendría que definir con un número que va del 0.0 al 1.0, por ejemplo:

   <1.0, 0.0, 0.0> = rojo

   <0.0, 1.0, 0.0> = verde

   <0.0 ,0.0, 0.0> = blanco

   <1.0, 1.0, 0.0> = amarillo

3. El tercer parámetro indica la transparencia del texto siendo 0 completamente transparente y 1 completamente opaco.

```
default
{       // Abrimos default
        state_entry ()          //este es el evento state_entry ()
        {       //abrimos evento
                llSetText ("Hola ", <1.0, 0.0, 0.0>, 1);
        }       // Cerramos evento
} //cerramos default
```

state_entry () es un evento que se ejecuta al ejecutarse el estado al que pertenece, como al ejecutarse el script lo primero que se ejecuta es el estado default, lo primero que se ejecutara serán las funciones que haya dentro del state_entry (). En este caso *llSetText (“Hola”, <1.0, 0.0, 0.0>, 1);*

Estados:

Además del estado default podemos tener más estados que definimos escribiendo "state" seguido del nombre del estado. Por ejemplo:

```
default
{       // Abrimos default
        state_entry ()
        {
                llSetText ("Estoy en el estado default", <1.0, 1.0, 1.0>, 1);
        }
        touch_start (integer total_number)
        {
                state dos;      // con ente comando hacemos que cambie al
                                //estado "dos"
        }
} //cerramos default

state dos
{       // Abrimos el estado dos
        state_entry ()
        {
                llSetText ("Estoy en el estado dos", <1.0, 1.0, 1.0>, 1);
        }
        touch_start (integer total_number)
        {
                state default; // con ente comando hacemos que cambie al
                                //estado "default"
        }
} //cerramos el estado dos
```

Al ejecutar este script se ejecutara el primero el estado default, por lo tanto se ejecutara primero la función llSetText ("Estoy en el estado default", <1.0, 1.0, 1.0>, 1); por encontrarse dentro del state_entry () del estado default. Al ejecutarse esta función aparecerá el texto "Estoy en el estado default" encima del objeto.

Cuando toquemos el objeto se ejecutara el comando state dos; el cual hará que se ejecute el estado dos y por lo tanto encima del objeto pasara a poner "Estoy en el estado dos" porque se ejecutara la función llSetText ("Estoy en el estado default",<1.0, 1.0, 1.0>, 1); por encontrarse dentro del state_entry () del estado dos.

Si volvemos a tocar el objeto volverá a cambiar de estado y de texto sucesivamente.

Variables:

Una variable es un identificador donde almacenar la información en un script. Una variable tiene un nombre, un tipo y un valor. El nombre debe empezar por una letra y las normas para nombrar una variable son similares a C o Java. Se diferencian mayúsculas de minúsculas.

Hay dos tipos de variables según la posición que ocupan, las variables locales que están definidas dentro de un estado y las variables globales que están definidas donde empieza el script. Las variables tienen que empezar por una letra y es recomendable que tengan un nombre descriptivo por si algún día quieres modificar ese script sea más fácil cambiar cosas.

Tipos de variables:

Los tipos de variables dependen de lo que estas puedan almacenar:

- Integer: Sirve para almacenar números enteros, como por ejemplo: 2.
- Float: Sirve para almacenar números reales, como por ejemplo: 2,35.
- String: Almacena texto, por ejemplo: “Hola”.
- key: Es un string especial usado para identificar algo propio de Vleaf, como por ejemplo: un avatar un objeto o una textura.
- Vector: Es una variable que está compuesta por tres números de tipo float, como por ejemplo: <1.0,1.0,1.0>
- Rotation: es una variable compuesta por 4 números de tipo float, que usaremos para dar movimiento a los objetos.

Declaración de variables:

Para declarar una variable hay que escribir primero el tipo y luego el nombre, por ejemplo: string miVariable

La variable miVariable es del tipo string lo que significa que en ella podemos almacenar cualquier texto.

Ejemplo de uso de variables:

```
string myVariable;
default
 {
        touch_start (integer total_number)
        {
                myVariable="Me has tocado";        //Almacenamos dentro de
                                                   //myVariable el valor Me has
tocado

                llSay (0, myVariable);          //al poner myVariable es igual que si
                                                //pusiésemos lo que lleva almacenada, es
                                                        //decir "Me has tocado";
        }

        collision_start (integer num)
        {
                myVariable="Me has tocado";        //Cambiamos el contenido de
                                                   //myVariable por Te has chocado

                llSay (0, myVariable);
        }
}
```

Operadores:

LSL, como el resto de lenguajes, tiene unos operadores que es necesario tener en cuenta:

- X = Y a X le asignamos el valor de Y
- X == Y verdadero si el valor de X es igual al Y
- X != Y verdadero si X es diferente a Y
- X < Y verdadero si X es menor que Y
- X > Y verdadero si X es mayor que Y
- X <= Y verdadero si X es menor o igual que Y
- X >= Y verdadero si X es mayor o igual que Y
- + mas(suma)
- - menos(resta)
- * multiplicación
- / división
- ++ incremento de 1 en 1
- -- descrecimiento de 1 en 1
- % resto de división
- && and(y)
- || or (o)

Operador cast:

El operador cast sirve para transformar un tipo de variable en otra, por ejemplo:

Con la función llSay (0, myVariable) la variable mi variable solo puede ser del tipo string si queremos que el objeto diga el valor de otra variable que no fuese string lo haríamos de la siguiente forma: llSay (0, (string) myVariable)

```
integer numero1=3;
integer numero2=2;
default
 {
        touch_start (integer total_number)
        {
                integer suma; //al declarar la variable dentro del evento solo la
                              //podemos utilizar dentro de este
                suma= numero1+ numero2;
                llSay (0, (string) suma);
        }
}
```

Cambiar el color la posición y la rotación a un objeto:

Cambiar el color:

Podemos cambiar el color de un objeto con la función llSetColor (), esta función tiene dos parámetros, el primero es un vector que indica el color y el segundo un integer que indica a que cara del objeto se le cambia el color, por ejemplo:

llSetColor (<1.0, 0.0, 0.0>,1);

Para cambiar de color todas las caras hay que escribir: llSetColor (<1.0, 0.0, 0.0>, ALL_SIDES);

```
default
{
        touch_start (integer total_number)
        {
                vector rojo=<1.0, 0.0, 0.0>;
                llSetColor (rojo, ALL_SIDES);
                state azul;
        }
}
state azul
{
        touch_start (integer total_number)
        {
                vector azul=<0.0, 0.0, 1.0>;
                llSetColor (azul, ALL_SIDES);
                state default;
        }
}
```

Cambiar la posición:

Para cambiar la posición de un objeto se usa la función llSetPos (), por ejemplo si queremos poner un objeto en la posición x=100, y=120.5, z=50 escribiríamos

llSetPos (<100, 120.5, 50>)

Si queremos por ejemplo mover un metro hacia arriba un objeto y no sabemos en qué posición se encuentra necesitamos la función llGetPos () que nos dice en qué posición se encuentra el objeto en ese momento

```
default
{
        touch_start (integer total_number)
        {
                llSay (0, "Estoy en la posición"+( string)llGetPos ());
                llSetPos (llGetPos ()+<0, 0, 1>);
                llSay (0, " Y ahora en la posición"+( string)llGetPos ());
        }
}
```

Cambiar la rotación:

Con la función llSetRot (rotacion); podemos rotar un objeto, para ello el parametro rotación debe ser del tipo rotation, por ejemplo para aumentar la rotación Y de un objeto 90 grados al cada vez que lo tocamos se haría de la siguiente forma.

```
default
{
        touch_start (integer total_number)
        {
        vector xyz=<0.0, 90 ,0.0>;    //con un vector le indicamos los grados
                                      //que  deseamos girar y sobre que eje
        vector xyz_en_radianes=xyz*DEG_TO_RAD ;//Lo cambiamos a
                                //radianes multiplicándolo por DEG_TO_RAD
        rotation rotacion= llEuler2Rot(xyz_en_radianes); //convertimos el
                                //vector en rotation con la función llEuler2Rot.


        llSetRot(llGetRot()*rotacion);
        }
}
```

If/else:

La forma general de esta sentencia es:

```
if (expresion)
        {
        acciones 1
        }
else
        {
        acciones 2
        {
```

Si **expresión** es verdadera, entonces se ejecuta las **acciones 1**; en caso contrario, se ejecuta las **acciones 2**.

Ejemplo:

```
default
{
        touch_start (integer total_number)
        {
        integer a=2;
        integer b=5;
        integer menor;
        if (a<=b)
        {
        menor=a; //si la "a" es menor o igual que "b" se ejecuta menor=a
        }
        else
        {
        menor=b; ; //si la "a" no es menor o igual que "b" se ejecuta menor=b
        }
        llSay(0 , "El numero menor es: " + menor);
        }
}
```

For:

Sintaxis :

for (inicialización; condición; incremento o decremento)

{

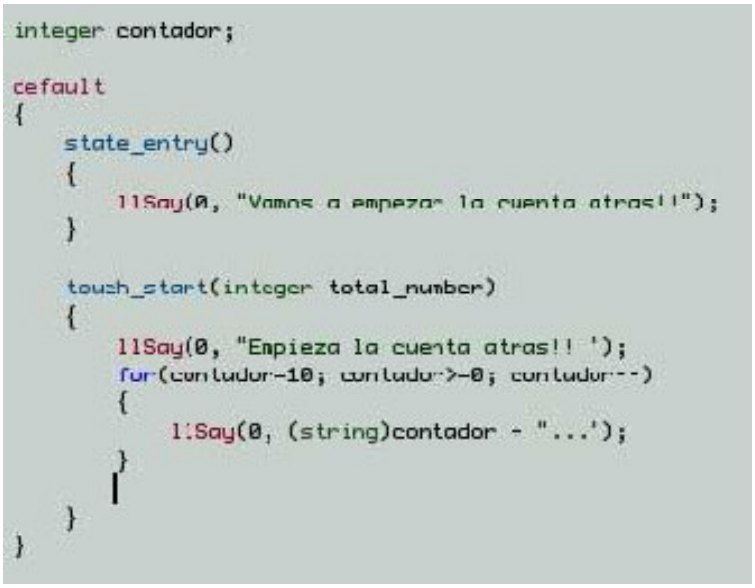
acciones

}

Da lugar a un bucle en el que se ejecutan unas acciones cierto número de veces. Este número de veces se determina entre los paréntesis mediante tres términos.

- INICIALIZACION: asignamos el valor numérico a un “contador”.
- CONDICION: comprobamos que la variable de inicialización cumple una condición. Si la cumple se sale del bucle.
- INCREMENTO o DECREMENTO: cada vez que realiza el bucle incrementará o decrecerá según se lo indiquemos el valor de la variable de inicialización

Un ejemplo:



El primer término del for inicializa la variable contador con inicial que va a tener en nuestro bucle. En este caso 10. En el segundo término se comprueba si se cumple la condición para salirse del bucle si esta se cumple. Y en el tercero en este caso resta uno a contador y comienza el proceso de nuevo. Estos incrementos y decrementos también los podéis escribir de las siguientes formas:

contador - - es lo mismo que contador = contador-1.

contador = contador +3 incrementaría de 3 en 3.

Evento timer:

Con la función *llSetTimerEvent* (x) se ejecutara las acciones del interior del evento timer cada x segundos, por ejemplo:

```
default
{
        state_entry ()
        {
                llSetTimerEvent (0.5);
        }
        timer ()
        {
                float x= llFrand(1);    //Con llFrand(1) almacenamos en "x", "y"
                float y= llFrand(1);    // y "z" un numero aleatorio de tipo float
                float z= llFrand(1);    // entre 0.0 y 1.0
                llSetColor (<x, y, z>, ALL_SIDES);
        }
}
```

Con este script conseguimos que cada 0.5 segundos el objeto cambie a un color aleatorio.

Comunicación entre scripts:

Los objetos en Vleaf se comunican entre sí en casi exactamente de la misma manera que los avatares. Todo lo dicho en Vleaf utiliza un canal específico, los avatares hablar normalmente en el canal cero Si un objeto escucha en el canal cero, escuchara lo que los usuarios cerca de él digan, del mismo modo, si un objeto habla por el canal cero los avatares cerca de él le escucharan.

Con la función llListen() le hincamos al objeto a través de los cuatro parámetros que posee, lo que queremos que escuche.

- Con el primer parámetro le indicamos el canal por el cual va a escuchar.
- Con el segundo le indicamos el nombre del objeto al cual va a escuchar, si lo dejamos en blanco escuchara a cualquier objeto.
- Con el tercer le indicamos la key del avatar al cual va a escuchar, si lo dejamos en blanco escuchara a cualquier avatar.
- Con el cuarto parámetro le decimos que solo esuche un determinado mensaje, si lo dejamos en blanco escuchara cualquier mensaje.

Por ejemplo:

Con llListen(0, “objeto1”,””,””); le indicamos que escuche por el canal 0 y solo lo que digan los objetos llamados objeto1.

Con llListen(30, “”,””,””); le indicamos que escuche todo lo que se diga por el canal 30.

Una vez que el objeto está escuchando siempre que escuche algo se ejecutara las acciones que hayan dentro del evento listen.

```
listen( integer channel, string name, key id, string message )

{

    acciones ;

}
```

En las variables channel, name, id y message se almacenara el canal, el nombre del objeto, la id del avatar y el mensaje que se ha escuchado respectivamente.

Interruptor:

```
default
 {
        touch_start (integer i)
        {
                llSay(1, "Encender");
                state apagado;
        }
}

state apagado
{
        touch_start (integer i)
        {
                llSay(1, "Apagar");
                state default;
        }
}
```

Bombilla:

```
default
 {
        state_entry ()
        {
                llListen(1, "","","");
        }
        listen (integer channel, string name, key id, string message )
        {
                if (message=="Encender")
                {
                        llSetPrimitiveParams([ PRIM_FULLBRIGHT,
                        ALL_SIDES, TRUE ]); //Esta función sirve para dar
                                           //"Brillo máximo".
                }
                else
                {
                        llSetPrimitiveParams([ PRIM_FULLBRIGHT,
                        ALL_SIDES, FALSE ]); //Sirve para quitar "Brillo
                                    //máximo".
                }
        }
}
```

Si introducimos el script bombilla en un objeto y el script interruptor en otro, al tocar el objeto con el script interruptor el objeto con el script bombilla se apaga y se enciende sucesivamente.

Menús:

En Vleaf podemos crear menús con la función

llDialog(key avatar, string mensaje, [list botones], integer canal);

Esta función muestra un cuadro de diálogo de color azul en la esquina superior derecha de la pantalla con un mensaje y botones de opción, así como un botón de ignorar.

```
integer canal=1000;
default
 {
        state_entry ()
        {
                llListen(canal, "", "", "");
        }
        touch_start (integer i)
        {
                llDialog(llDetectedKey(0), "Elije una opcion", [" Si", "No"],
                        canal);
        }

        listen (integer channel, string name, key id, string message )
        {
                llSay(0, " Has elegido " + message);
        }
}
```

Sensores:

```
default
 {
        touch_start (integer i)
        {
                llSensor("", "", AGENT, 10, PI);
        }

        sensor (integer num)
        {
                string  msg="Detectados " + (string) num + " avatares : "+
                                                        llDetectedName(0);
                integer i;
                for(i=1;i<num;i++)
                {
                        msg=msg+ ", " + llDetectedName(i);
                }
                llSay(0, msg);
        }

        no_sensor()
        {
                llSay(0, " No detecto nningun avatar ");
        }
}
```