Erasmus+

Project funded by: **Erasmus+ / Key Action 2 - Cooperation for innovation and the exchange of good practices, Strategic Partnerships for school education (European Commission, EACEA)**

WORLD OF PHYSICS

TECHNICAL GUIDE

Communication Between Objects and their Parts

1. Introduction

One of the most useful features of the LSL scripting language are the functions and events that allow different objects to communicate between them. You can use this feature to design and implement very complex behaviours!

There are two different cases that you need this:

- Two different objects must coordinate so they need to be informed when something happens related to the other object. For example when the user clicks a switch object, a lamp (other object) needs to turn on.

- Parts of complex object (linked set of basic prims) need to cooperate. For example when you click on the power button of a TV Monitor, the screen (part of the same object) turns on.

In the first case the way to implement it is using messages through chat channels. Opensim apart from the public chat channel which is used when an avatar writes a message in chat, has a very large number of channels that can be used to send messages to. Each channel is associated with an integer value and the range of available channels is -2147483648 through 2147483647. Using scripts you can instruct each separate object or parts of it to wait for messages in one or more of these channels. When a message is sent in a channel that the object is listening on, an event triggers and the object can perform some predetermined actions. So, in the example we mentioned, the lamp object should wait for a message (e.g “turn_on”) in a specific channel (e.g channel 333). The switch object when clicked should send the required message (“turn_on”) to that specific channel (channel 333) thus triggering the lamp to perform the necessary script commands to turn on.

In the second case, the approach is even easier. Parts of a linked set can be set to wait for messages from other parts of the same set (no channel needs to be specified). Additionally parts of a linked set can send messages to another part or parts of the same object. Each part of a linked set has a unique link number id that

can help you identify where to send the message. Keep in mind however that one of the parts of the linked set is considered the root prim and can use a specific set of script commands that allow it to directly change properties of the child prims, so you may not even have to send messages to them! In our example that we mentioned earlier the TV screen should wait for a specific message (e.g. “turn_on”) from other parts of the TV. The button prim just needs to send the message (“turn_on”) to all other parts or specifically to the screen part (we can get the link number that identifies it from the build menu when we select it).

2. Technical Details

In the case that different objects need to communicate through chat channels we use the following:

Function: integer llListen(integer channel, string name, key id, string msg);

‘llListen’ is the function you need in order to set one object to wait for specific messages, from specific channels. Usually you run this function in a state_entry event but you may want to dynamicaly start and stop listening to channels (also check llListenRemove in that case).

Event: listen(integer channel, string name, key id, string message){ ; }

‘listen’ is the event that will be triggered when a message is indeed sent in the channels the object is listening on. Inside the block of the event you may want to check the value of the message and perform the required actions

Function: llSay(integer channel, string msg);

‘llSay’ is the function other objects will have to use to send messages to the specific channel that you have specified so the above mentioned ‘listen’ event will trigger. Notice that llSay has a specific range (20 meters) that it can be transmitted. You may have alternative functions for greater disatances such as llShout or llRegionSay.

In the case that different parts of the same object (linked set of prims) need to communicate we use the following:

Event: link_message(integer sender_num, integer num, string str, key id){ ; }

'link_message' is the event that will be triggered when a message is sent from one of the other parts of the same object.

Function: llMessageLinked(integer link, integer num, string str, key id);

'llMessageLinked' is the function a part of an object can use to send messages to another part or parts of the object, triggering their 'link_message' event.

We have used these two approaches in many cases in WOP and you can find the scripts we used and more instructions in the "Scripts Section" here: http://aigroup.ceid.upatras.gr/wop-oer/scripts.html

3. References/Links

http://aigroup.ceid.upatras.gr/wop-oer/scripts.html
http://wiki.secondlife.com/wiki/Chat_channel
http://wiki.secondlife.com/wiki/LlListen
http://wiki.secondlife.com/wiki/LlSay
http://wiki.secondlife.com/wiki/Listen
http://wiki.secondlife.com/wiki/LlMessageLinked
http://wiki.secondlife.com/wiki/Link_message