

Exploiter les canaux

Fiche pratique de script LSL niveau débutant

Ahuri Serenity

Professeur de script à l'Ecole SL

EM@iL

30 juillet 2010

Résumé

Dans cette fiche nous allons voir comment exploiter les canaux pour la communication, permettant notamment la commande personnalisée et dynamique des scripts.

Ce document n'est pas une substitution aux nombreux tutoriels disponibles sur Internet et ne vaut pas un cours, il est cependant indispensable et impératif que vous ayez bien intégré tous les points évoqués ici.

Table des matières

1	Introduction	1
2	Quelques généralités	2
3	Les éléments utilisés	2
4	Implémentation	4
5	Aller plus loin ... moins de LAG !	5
5.1	Délai de réponse	5
5.2	Plusieurs filtres	7
5.3	Réfléchir	7
6	Récapitulatif	9

1 Introduction

Les moyens de communication et d'interaction pour un script sont très nombreux, le plus utilisé (de loin) est celui que nous allons voir tout au long de ce document : les canaux de communication (channels). Lorsque nous écrivons un message dans chat avec ou sans le prefix "/", notre message transite dans un canal particulier. Le receveur du message doit alors être capable de lire ce qu'il se passe dans ce même canal pour le voir. C'est exactement comme pour le talkie-walkie!

Nous commencerons par étudier les différents éléments utilisés pour l'exploitation des canaux puis, après avoir vu une exemple d'implémentation, nous verrons quelques points d'optimisation.

2 Quelques généralités

Lorsque vous parlez dans le chat publique, vous utilisez un canal particulier : PUBLIC_CHANNEL (qui est en fait le canal 0). De même, lorsqu'un script déclare une erreur script il utilise un canal pour vous le communiquer DEBUG_CHANNEL. Cependant lorsque vous discutez, il ne vous est pas nécessaire d'indiquer que vous souhaitez utiliser le canal 0, c'est implicite, ce qui n'est pas le cas des autres canaux.

Les scripts peuvent utiliser tous les canaux allant de -2147483648 à +2147483647 (ça fait beaucoup hein!?) donc aussi bien des canaux positifs que négatifs, contrairement aux avatars qui ne peuvent utiliser "que" la partie positive (il suffit de faire précéder son message de "/" puis du numéro du canal, ex : "/32 Bonjour!"). Par contre, seuls les scripts peuvent lire les messages sur d'autres canaux que le publique (sauf si le viewer est un peu modifié pour cela ..).

Les messages qui transitent sur les canaux ne peuvent excéder 1023 octets. Cette quantité représente un nombre variable de caractères selon votre encodage (ex : chinois, latin, ...) au delà, le message est tronqué.

Il est impossible pour un script de s'écouter lui même, par contre, au sein d'un même objet les différentes primitives s'entendent MAIS c'est inutile et idiot de faire communiquer les primitives de cette manière car il existe les messages liés via `llMessageLinked` et `link_message` qui sont faits pour cela.

Un script ne peut écouter que 65 canaux simultanément.

Lorsqu'on change d'état dans un script, les écoutes en cours sont supprimées (ce qui n'est pas le cas par exemple pour le timer).

La file d'attente pour l'évènement listen (voir Chap. 3) est limitée à une trentaine d'entrées donc on peut perdre des données avec le LAG si il y a trop d'entrées en peu de temps.

Il est bien de privilégier des canaux non communs pour vos scripts (ex : -1428418) et surtout d'éviter le canal publique.

Il faut faire attention, lorsque l'objet est destiné à être transmis, à bien mettre à jour l'écoute sur la bonne personne, pour éviter par exemple que le script écoute l'ancien proprio!

3 Les éléments utilisés

Lorsqu'on désire prononcer un message particulier dans un canal particulier, il faut commencer par se poser la question de la distance à laquelle se trouve le(s) receveur(s) du message. Le choix de la fonction d'émission du message se fait ensuite en fonction de ce critère.

Fonctions d'émission avec leurs distances associées :

Nom de la fonction	Distance associée
<code>llWhisper</code>	10 m.
<code>llSay</code>	20 m.
<code>llShout</code>	100 m.
<code>llRegionSay</code>	Tout le sim.

Chaque fonction s'utilise exactement de la même façon (canal + message), voyons cela avec la fonction `llSay` :

```
1 llSay( integer canal, string msg );
```

Le paramètre *canal* est le numéro du canal dans lequel le message *msg* doit être prononcé. Si vous voulez interagir avec des avatars avec le chat public il faut mettre 0. Exemple :

```
1 llSay( 0, "Bienvenue visiteur !");
```

Maintenant si votre script doit communiquer avec un autre script, cet autre script doit écouter le bon canal. Disons que deux scripts s'échangent des informations sur le canal -48. Celui qui envoie des informations utilise :

```
1 llSay( -48, "informations ici");
```

Et le script qui reçoit les informations doit écouter sur le même canal donc le canal -48. Pour cela il doit utiliser deux choses inséparables : la fonction `llListen` et l'évènement `listen` dont voici les prototypes :

```
1 integer llListen( integer channel, string name, key id, string msg );
```

Cette fonction agit comme un filtre sur l'ensemble des messages envoyés sur les différents canaux. On doit préciser sur quels éléments on souhaite filtrer et comment le faire :

- *channel* est le canal sur lequel on doit écouter (obligatoire);
- *name* correspond au nom de l'objet ou de l'avatar qui prononce le message, si on met une chaîne de caractères vide (""), tous les noms sont acceptés;
- *id* est la clé de l'objet ou de l'avatar qui prononce le message, si la clé est nulle (NULL_KEY) ou vide (""), toutes les clés sont acceptées;
- *msg* est le message attendu, si il est vide (""), tous les messages sont acceptés.

La fonction renvoie un entier qui est l'identifiant de l'écoute permettant de la supprimer ultérieurement ou bien de la désactiver seulement un moment. Vous devez donner le plus d'éléments possible à filtrer ! Lorsqu'un message passe à travers le filtre, l'évènement `listen` est appelé ...

```
1 listen( integer channel, string name, key id, string message ){ ; }
```

Cet évènement livre plusieurs informations : le canal *channel* dans lequel le message a été envoyé, le nom *name* de l'objet ou de l'avatar source, la clé *id* de cette source et bien entendu le contenu *message*. C'est dans cet évènement que nous allons traiter les informations reçues.

Attention à n'oublier ni la fonction `llListen` ni l'évènement `listen` !

Un `listen` se supprime grâce à son numéro d'identification avec la fonction `llListenRemove` ...

```
1 llListenRemove( integer number );
```

... et s'active/désactive grâce à la fonction `llListenControl`

```
1 llListenControl( integer number, integer active );
```

où *number* est le numéro du `listen` et *active* est l'état d'activation (TRUE pour actif).

Remarque : Les avatars ne peuvent pas recevoir les messages envoyés par la fonction `llRegionSay`.

Remarque : La fonction `llDialog` qui permet d'afficher les fenêtres bleues utilise aussi le couple `llListen/listen` car la réponse, après avoir pressé un des boutons du menu, est envoyée (de la part de l'utilisateur) sur un canal (voir sa page du wiki).

4 Implémentation

Voici un script qui fait en sorte que l'objet ne réponde qu'au propriétaire. L'objet a des réponses adaptées à certains messages (ex : il répond "Bonjour maitre!" à "bonjour"). On fait en sorte également que l'écoute se fasse bien sur le propriétaire même lorsque l'objet est transféré.

```
1 integer iEcoule;
2
3 default
4 {
5     state_entry()
6     {
7         // On souhaite n'ecouter que le owner (aucune autre restriction)
8         iEcoule = llListen(0, "", llGetOwner(), "");
9     }
10    listen( integer channel, string name, key id, string message )
11    {
12        if( message == "bonjour" )
13        {
14            llSay(0, "Bonjour maitre !");
15        }
16        else if( message == "tchao" )
17        {
18            llSay(0, "A bientot !");
19        }
20        else
21        {
22            llWhisper(0, "Je n'ai pas compris le message ...");
23        }
24    }
25    on_rez(integer param)
26    {
27        // On s'assure qu'au rez de l'objet, le listen soit bien reinitialise :
28        llResetScript();
29    }
30    changed(integer mask)
31    {
32        // On s'assure que l'objet sache quand il change de proprietaire pour ne plus ecouter l'
33        // ancien ...
34        // Cette option est optionnelle si l'objet est destine a etre pose sur le sol.
35        if(mask & CHANGED_OWNER)
36        {
37            llResetScript();
38        }
39    }
40 }
```

5 Aller plus loin ... moins de LAG !

Le LAG est un fléau sur Second Life et le système de canaux n'est pas là pour arranger la chose .. voilà pourquoi je veux insister sur l'importance d'utiliser proprement les canaux. Il existe plusieurs choses à considérer lorsqu'on les utilise que nous allons aborder ici.

5.1 Délai de réponse

Lorsqu'on s'adresse à une personne, il est préférable de laisser un délai de réponse, puis si au bout de ce temps il n'y a toujours pas de réponse, on annule l'écoute en cours. Pour utiliser le temps, on prendra la fonction `llSetTimerEvent` et l'évènement `timer` :

```
1 integer iEcoule;
2 key    kAgent;
3
4 default
5 {
6     touch_start(integer p)
7     {
8         kAgent = llDetectedKey(0);
9         llWhisper( 0, "Bonjour "+llDetectedName(0)+" ! Ca va ?");
10
11         state enCours;
12     }
13 }
14
15 state enCours
16 {
17     state_entry()
18     {
19         iEcoule = llListen(0, "", kAgent, "");
20         llSetTimerEvent(45.0);
21     }
22     on_rez(integer param)
23     {
24         // On s'assure qu'au rez de l'objet, le listen soit bien reinitialise :
25         llResetScript();
26     }
27     listen( integer channel, string name, key id, string message )
28     {
29         if( message == "oui" )
30         {
31             llSay(0, "Super t'as la banane !!!");
32         }
33         else if( message == "non" )
34         {
35             llSay(0, "T'as pas la peche ?");
36         }
37         else
38         {
39             llWhisper(0, "Je n'ai pas compris le message ...");
40         }
41         state default;
42     }
43     timer()
44     {
```

```

45     llInstantMessage( kAgent, "Vous ne m'avez pas donne de reponse assez rapidement, pour eviter
46         le lag je me reinitialise ...");
47     state default;
48 }
49 state_exit()
50 {
51     llSetTimerEvent(0.0);
52 }
53 }

```

Remarque : J'ai fais exprès ici de ne pas employer llListenRemove car les changements d'état annulent les écoutes en cours, ce script permet juste de se rendre compte que l'emploi d'un timer est chose aisée. Si on souhaite rester dans un seul état, il faut ajouter les fonctions llListenRemove et llSetTimerEvent(0.0) dans l'évènement timer :

```

1  integer iEcoule;
2  integer iAttente = TRUE;
3
4  finEcoule()
5  {
6      iAttente = TRUE;
7      llSetTimerEvent(0.0);
8      llListenRemove(iEcoule);
9  }
10
11  default
12  {
13      touch_start(integer p)
14      {
15          if( iAttente )
16          {
17              iAttente = FALSE;
18
19              llWhisper( 0, "Bonjour "+llDetectedName(0)+" ! Ca va ?");
20
21              iEcoule = llListen(0, "", llDetectedKey(0), "");
22              llSetTimerEvent(45.0);
23          }
24      }
25      listen( integer channel, string name, key id, string message )
26      {
27          if( message == "oui" )
28          {
29              llSay(0, "Super t'as la banane !!!");
30          }
31          else if( message == "non" )
32          {
33              llSay(0, "T'as pas la peche ?");
34          }
35          else
36          {
37              llWhisper(0, "Je n'ai pas compris le message ...");
38          }
39          finEcoule();
40      }

```

```

41     timer()
42     {
43         finEcoule();
44         llInstantMessage( kAgent, "Vous ne m'avez pas donne de reponse assez rapidement, pour eviter
                                le lag je me reinitialise ...");
45     }
46 }
47

```

NB : Les deux codes se comportent de la même façon.

5.2 Plusieurs filtres

Dans le cas des systèmes à plusieurs objets communiquant pas mal d'informations par canaux, il est préférable au niveau du LAG de prendre plusieurs filtres ou bien d'utiliser un unique canal avec des tests dans l'évènement listen. Voyons cela ...

Remarque : Bien que l'on puisse faire pas mal d'écoutes simultanées avec pour chaque écoute son propre filtre, il est impossible de superposer les filtres pour un même canal, le filtre sera toujours de la nature du dernier appel de llListen.

Cela implique une chose évidente : impossible d'écouter deux sources quelconques en même temps sur un même canal en les identifiant formellement. Soit on écoute une source particulière, soit toutes les sources ou un groupe de sources en les appelant de la même façon, mais généralement on évite car ce n'est pas sécurisé.

Dans le cas où vous devez gérer différents objets et que les clés sont connues, il faut privilégier l'utilisation de plusieurs filtres c'est à dire on écoute plusieurs canaux mais avec une clé valide comme filtre sur chaque filtre. C'est d'autant plus optimisé si vous n'avez à transmettre qu'une unique commande dans le chat, ça rajoute un élément de filtre. *De manière générale*, plus vous utilisez de filtres et plus vous optimisez.

Lorsque vous gérez des objets sans connaître leur clé ou que leur nombre est élevé, il faut privilégier l'utilisation des filtres moins restrictifs : nom des objets. Selon la situation, on peut considérer plusieurs filtres et donc plusieurs canaux (plusieurs groupes d'objets) ou bien qu'un seul canal (impossibilité de toucher au nom des objets).

Moins il y a de filtres et plus on multiplie les traitements dans les listen, cela peut altérer de manière significative les performances du simulateur si tout le monde fait ainsi. Cependant parfois on a pas trop le choix et on est obligé de ne rien filtrer sur le canal, cette situation doit être la moins fréquente possible!

5.3 Réfléchir

Il faut faire remonter aux filtres le maximum d'informations! Plus il y a d'informations dans les filtres et plus votre script sera optimisé! Contrairement à ce qu'on pourrait penser, faire une seule écoute sans filtre est plus "laggy" que de faire plusieurs écoutes avec un filtre sélectif, il faut donc réfléchir!

Voilà un exemple de ce qu'on ne doit surtout pas voir dans un script :

```

1 default
2 {
3     state_entry()
4     {
5         llListen(0, "", NULL_KEY, "");
6     }

```

```

7 listen( integer channel, string name, key id, string message )
8 {
9     if( id == llGetOwner() )
10    {
11        llSay( 0, "Tu es bien mon maitre donc je t'ecoute.");
12    }
13 }
14 }

```

Il faudrait faire remonter llGetOwner() sur le filtre :

```

1 default
2 {
3     state_entry()
4     {
5         llListen(0, "", llGetOwner(), "");
6     }
7     listen( integer channel, string name, key id, string message )
8     {
9         llSay( 0, "Tu es bien mon maitre donc je t'ecoute.");
10    }
11 }

```

Par contre on peut être amené à mettre en place une structure semblable pour écouter les objets du proprio plutôt que le proprio lui-même :

```

1 default
2 {
3     state_entry()
4     {
5         llListen(0, "", NULL_KEY, "");
6     }
7     listen( integer channel, string name, key id, string message )
8     {
9         if( llGetOwnerKey(id) == llGetOwner() )
10        {
11            llSay( 0, "Tu appartiens bien a mon maitre donc je t'ecoute.");
12        }
13    }
14 }

```

Dans ce cas, on a pas trop le choix ... on ne peut rien faire remonter au filtre.

De même, ce code est à bannir :

```

1 default
2 {
3     state_entry()
4     {
5         llListen(0, "", llGetOwner(), "");
6     }
7     listen( integer channel, string name, key id, string message )
8     {
9         if( message == "bonjour" )
10        {
11            llSay( 0, "Bonjour maitre !");
12        }
13    }
14 }

```

```
13 }
14 }
```

pour ce code :

```
1 default
2 {
3     state_entry()
4     {
5         llListen(0, "", llGetOwner(), "bonjour");
6     }
7     listen( integer channel, string name, key id, string message )
8     {
9         llSay( 0, "Bonjour maitre !");
10    }
11 }
```

Par contre, si on veut reconnaître un message quelconque, on a pas le choix, il faut enlever le filtre :

```
1 default
2 {
3     state_entry()
4     {
5         llListen(0, "", llGetOwner(), "");
6     }
7     listen( integer channel, string name, key id, string message )
8     {
9         if( message == "bonjour" )
10        {
11            llSay( 0, "Bonjour maitre !");
12        }
13        else if( message == "bye" )
14        {
15            llSay( 0, "Tchao maitre !");
16        }
17        else
18        {
19            // ...
20        }
21    }
22 }
```

Donc réfléchissez bien au problème à savoir quel type de filtre mettre, restrictif ou pas, etc. N'hésitez pas à demander aux plus expérimentés ce genre d'information car il en va de la santé de nos simulateurs et de votre sérieux.

6 Récapitulatif

- Bien choisir sa fonction émettrice : llWhisper, llSay, llShout, llRegionSay, llDialog.
- Utiliser des canaux cohérents et non communs.
- Paramétrer soigneusement llListen avec le plus d'informations possibles pour un filtre efficace. Utiliser l'identité du listen retournée pour optimiser le script.
- L'évènement listen pour traiter les informations recueillies à travers les filtres.
- llListenRemove : supprimer un llListen (à utiliser le plus souvent possible!)
- llListenControl : activer ou désactiver un llListen.

- Optimiser son script!!