



Contenu

Le port série	5
debuglog.....	11
Launchtype.....	17
outils.....	18





Je ne sais pas s'il y a longtemps que vous êtes allés sur notre site favori <http://dev.karotz.com/sdk/>

On y voit un nouveau paragraphe (Launch Type) et il nous rappelle qu'il y a encore des choses que nous n'avons pas étudiées comme la webcam ou le port série.

- [Introduction](#)
- [KarotzVM JAVA](#)
- [Examples](#)
- [Launch type](#)
- [General methods](#)
- [Karotz methods](#)
 - [Parameters and command line](#)
 - [karotz.connectAndStart](#)
 - [karotz.callback](#)
 - [karotz.ping](#)
 - [karotz.tts](#)
 - [karotz.asr](#)
 - [karotz.ears](#)
 - [karotz.led](#)
 - [karotz.webcam](#)
 - [karotz.multimedia](#)
 - [serial](#)
 - [karotz.listeners](#)

Le launch Type :

On peut lire « You can get the origin of the application launch » ou en français dans le texte « vous pouvez savoir ce qui est à l'origine du lancement de votre application. »

Whaou, génial !

Enfin j'imagine !

Si cela a été ajouté c'est forcément qu'il y avait un besoin.

Mais petit problème : comment allons-nous tester ceci en local ? C'est impossible puisqu'un « lancement d'application » est forcément en connecté.

Ceci m'amène tout naturellement à cette question : mais comment faisons-nous pour tester notre appli en ligne ?

Si tout fonctionne pas de soucis, nous l'avons déjà testée en local mais vous je ne sais pas mais moi je n'ai pratiquement jamais eu une appli qui a fonctionné du premier coup une fois connectée alors qu'elle fonctionnait parfaitement en local. Bon je passe sur l'adresse IP à mettre en localhost, le fichier descriptor qui doit reprendre tous les listener que

karotz - tutoriel pour débutant sous Windows -

nous gérons, non je parle du cas où ça devrait fonctionner et que ça ne fonctionne pas. Horreur ? Ou sont les « logs » qui m'aident à mettre au point mon appli en local ?

Si vous avez parcouru le forum de développement consacré à Karotz

<https://groups.google.com/forum/#!forum/karotzdev>

ou même le magnifique WIKI

http://wiki.karotz.com/index.php/Main_Page

Vous avez vu qu'il existait une solution de « Callback PHP avec log » développée par Athanasus mais il est précisé « API REST »

([http://wiki.karotz.com/index.php/API_REST / Exemple de Callback PHP avec log](http://wiki.karotz.com/index.php/API_REST/_Exemple_de_Callback_PHP_avec_log))



karotz - tutoriel pour débutant sous Windows -



Ou celle de KarMen pour le SDK (ce que nous utilisons jusqu'à présent)
<https://groups.google.com/forum/#!topic/karotzdev/9-BS-3nfUCo>

Mais, car il y a un mais, tous deux font appel à un serveur « APACHE » avec page en PHP, un autre monde, le monde des « pros ».
Je ne me vois pas vous expliquer comment installer un tel serveur, cela dépasserait largement le cadre de ces tutoriels à usage des « débutants »

Alors on en a terminé avec les tutoriels, on a atteint notre limite, place aux pros ?

Triste fin vous ne trouvez pas ?

Pourquoi, nous les débutants devrions en rester là ? Vous vous souvenez de ma devise ? Pourquoi faire compliqué lorsqu'on peut faire simple ? Oui c'est bien mais s'il y avait une solution simple ça se saurait ? non ?

Ne vous ai-je pas dit que c'étaient les problèmes les plus ardues qui me motivaient le plus, là il m'en a fallu de la motivation, j'en ai fait des tentatives et puis l'étincelle est venue des essais faits par KarMen avec la carte Arduino. Je pense que nous aurons l'occasion de reparler de cette carte car j'ai bien l'intention de faire des choses avec elle et vous en faire profiter.

Mais quel rapport entre la carte Arduino et la possibilité de débogger en ligne sans passer par un serveur PHP ? Vous ne voyez pas ? Cherchez bien la réponse est d'ailleurs dans la première page de ce tutoriel, dans les choses non encore abordées : « le port série »

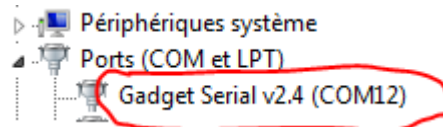
karotz - tutoriel pour débutant sous Windows -

Le port série

Nous n'allons pas appeler professeur Karotz à la rescousse ni en ce qui me concerne vous parler d'un temps que les jeunes de 20 ans ne peuvent pas connaître, l'époque des ports séries, des ports parallèles, des liaisons synchrones, asynchrones, RS232, Centronics tout cela a été remplacé par l'USB, le sans-fil même, Bluetooth, Wifi et autres technologies des temps modernes. Mais l'expérience des anciens a finalement du bon et je suis heureux de pouvoir vous en faire profiter.

Moi ce que j'ai vu c'est simplement le mot « série » ! Ça je connais par cœur. Je me suis donc demandé si le port USB, qui utilise le terme serial, qui parle de « baudrate » (la vitesse de transmission série se mesure en Baud) fonctionnait comme une ancienne liaison série.

Et puis vous je ne sais pas mais moi j'en ai bavé avec ma première connexion Karotz, le port USB qui ne voulait pas fonctionner, il avait donc fallu entrer un peu plus dans la technique et savez-vous comment s'appelle le driver USB pour le Karotz ?

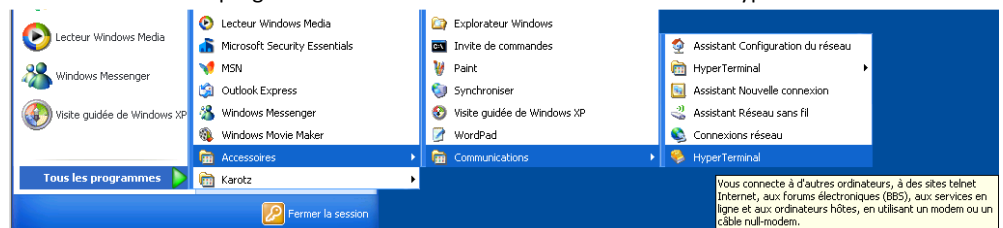


Ou c'est une conspiration ou on a tous les éléments pour réussir ☺
Ça n'est pas professeur Karotz que nous allons appeler à notre secours mais Inspecteur Gadget comment ça vous ne connaissez pas non plus ?)

Ne restait plus qu'à retrouver le logiciel qui me servait à tester les ports séries, les modems « RTC ».

HyperTerminal ! Yesss ma mémoire ne me fait pas défaut. Sauf que, pas de bol HyperTerminal a disparu avec l'arrivée de Windows Vista (et donc seven), seuls ceux qui ont encore XP ont cet utilitaire.

Démarrer => tous les programmes => Accessoires => Communications => HyperTerminal



Et pour ceux qui n'ont pas XP, ils sont nombreux normalement no panic, je fournis ici <http://www.bregeon.net/karotz/HyperTerminal.zip> (HyperTerminal.zip qui comprend la dll et l'exécutable) les 2 fichiers sont à mettre dans le même répertoire (de votre choix, pourquoi pas le karotz qui sert à nos tests ?)

Avant de lancer ce logiciel il va falloir connecter le lapin à notre ordinateur.

karotz - tutoriel pour débutant sous Windows -

En effet HyperTerminal va « écouter » le port série et nous restituer ce qu'il « entend » mais pour cela il faut lui indiquer le port série à écouter et ce port étant « virtuel » il n'existera que lorsque le lapin sera connecté.

Pour connecter le lapin, c'est très simple vous reprenez le câble qui vous a servi à sa première initialisation et vous le branchez de nouveau (sous la queue du lapin côté lapin et sur n'importe quel port USB de votre ordinateur)



Pour savoir si tout va bien, démarrer => Périphériques et imprimantes

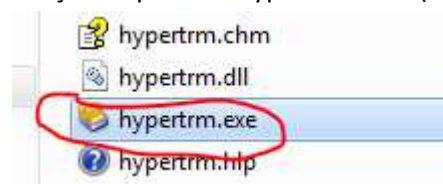


Et vous devriez avoir ceci :



Ce qui nous importe c'est le numéro du com , ici : 12

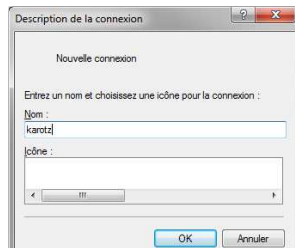
Lançons à présent HyperTerminal (double clic sur Hypertm.exe)



karotz - tutoriel pour débutant sous Windows -



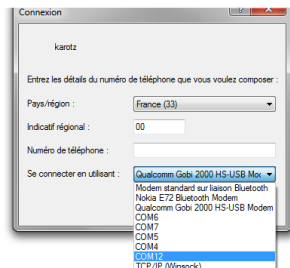
Première fenêtre :



Vous mettez le nom que vous voulez, ici « karotz »

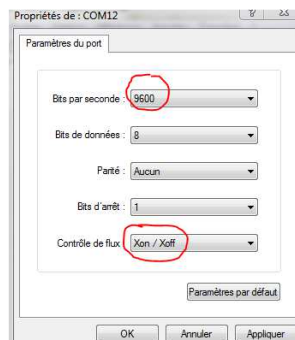
Puis OK (pas d'icône, il doit manquer un fichier mais ça n'a pas d'importance ;)

Ecran suivant vous sélectionnez le port noté un peu plus haut (ici com12)



Puis OK (on ne renseigne rien d'autre sur cet écran)

Ecran suivant

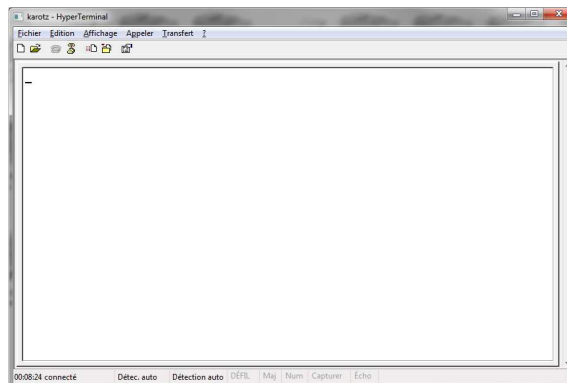


Vitesse : 9600

Contrôle de flux : Xon/Xoff

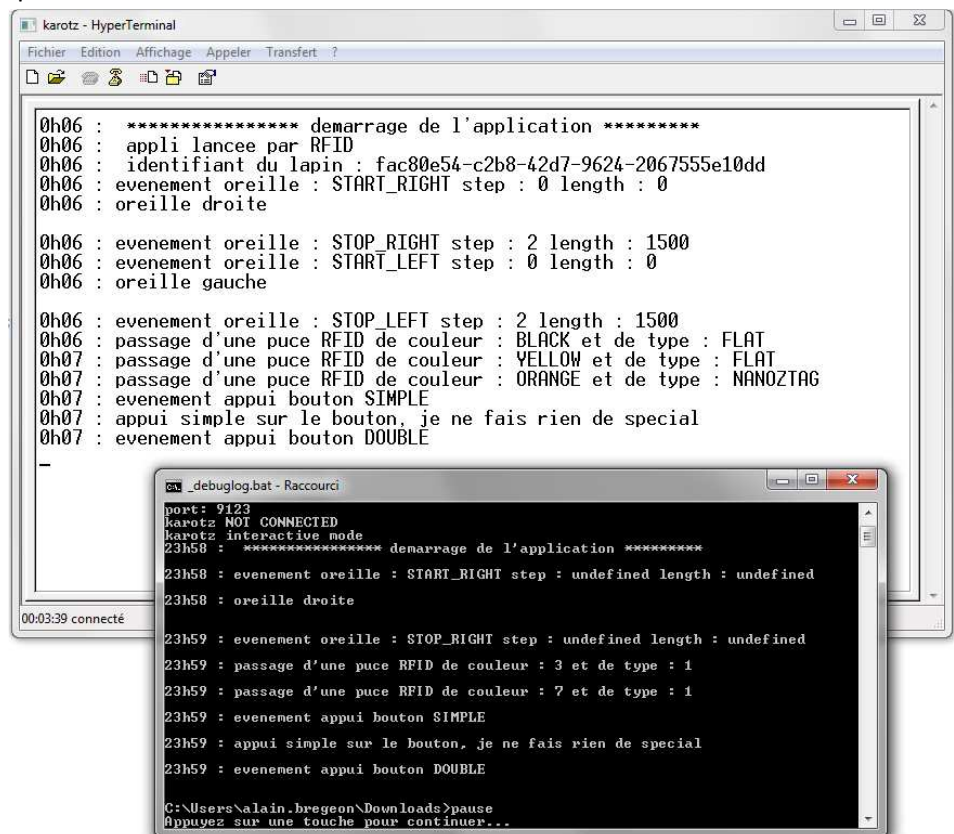
Puis OK

Et voici The fenêtre, la fenêtre de tous nos espoirs, celle qui devrait nous donner plein d'informations, si possible les mêmes que l'on a sur notre fenêtre Dos en local (sauf que le fond n'est pas noir 😊)



karotz - tutoriel pour débutant sous Windows -

Allez juste pour le plaisir, pour vous donner du courage regardez ce vers quoi nous allons



```
karotz - HyperTerminal
Fichier Edition Affichage Appeler Transfert ?
0h06 : ***** demarrage de l'application *****
0h06 : appli lancee par RFID
0h06 : identifiant du lapin : fac80e54-c2b8-42d7-9624-2067555e10dd
0h06 : evenement oreille : START_RIGHT step : 0 length : 0
0h06 : oreille droite
0h06 : evenement oreille : STOP_RIGHT step : 2 length : 1500
0h06 : evenement oreille : START_LEFT step : 0 length : 0
0h06 : oreille gauche
0h06 : evenement oreille : STOP_LEFT step : 2 length : 1500
0h06 : passage d'une puce RFID de couleur : BLACK et de type : FLAT
0h07 : passage d'une puce RFID de couleur : YELLOW et de type : FLAT
0h07 : passage d'une puce RFID de couleur : ORANGE et de type : NANOZTAG
0h07 : evenement appui bouton SIMPLE
0h07 : appui simple sur le bouton, je ne fais rien de special
0h07 : evenement appui bouton DOUBLE

_debuglog.bat - Raccourci
port: 9123
karotz NOT CONNECTED
karotz interactive mode
23h58 : ***** demarrage de l'application *****
23h58 : evenement oreille : START_RIGHT step : undefined length : undefined
23h58 : oreille droite
23h59 : evenement oreille : STOP_RIGHT step : undefined length : undefined
23h59 : passage d'une puce RFID de couleur : 3 et de type : 1
23h59 : passage d'une puce RFID de couleur : 7 et de type : 1
23h59 : evenement appui bouton SIMPLE
23h59 : appui simple sur le bouton, je ne fais rien de special
23h59 : evenement appui bouton DOUBLE
C:\Users\alain.bregeon\Downloads>pause
Appuyez sur une touche pour continuer...
```

Enfin nous allons pouvoir visualiser et donc tester le « step » et le « length » des oreilles (bien que la VM sera certainement corrigée lorsque paraîtra ce tutoriel), constater que contrairement à l'appli en local qui renvoie des nombres pour les couleurs, là nous avons la couleur directement, je peux vous dire que ce bug m'avait donné du fil à retordre ;) Notez au passage que nous ferons en sorte d'avoir le même code afin d'avoir les mêmes log que ce soit en local ou en connecté.

D'ailleurs vous voulez essayer, là de suite ? Votre fenêtre HyperTerminal est restée ouverte, le lapin est bien connecté ?

Alors exécutez la merveilleuse application « zanimaux » (version française)

Alors vous êtes avec moi pour la suite ?

Ce ne sera pas très dur en fait, on a déjà fait l'essentiel.

Reprenons notre tout premier programme, « Hello World », que d'émotion d'un coup, il s'en est passé des choses depuis ces premières lignes...

Comme on a acquis un peu d'expérience on le modifie de suite pour gérer les oreilles et les puces RFID et mettre quelques « log » afin de voir ce qu'il

karotz - tutoriel pour débutant sous Windows -

se passe dans notre fenêtre DOS) et on complète le fichier descriptor.xml en conséquence ;)

```
include("util.js");
var karotz_ip="192.168.1.46"; //ici votre adresse IP
//var karotz_ip="localhost"

var buttonListener = function(event) {
  if (event == "DOUBLE") {
    karotz.tts.stop();
    exit();
  }
  return true;
}
var rfidListener = function(data) {
  log("je suis dans le rfid listener");
  log("type rfid " + data.type);
  log("couleur rfid " + data.color);
  return true;
}
var earsListener = function(event,step,length) {
  log("je suis dans l'ears listener");
  log("event = " + event);
  log("step = " + step);
  log("length = " + length);

  return true;
}
var buttonListener = function(event) {
  log("je suis dans le bouton listener");
  log("event = " + event);

  if (event == "DOUBLE") {
    karotz.tts.stop();
    exit();
  }
  return true;
}
var exitFunction = function(event) {
  if((event == "CANCELLED") || (event == "TERMINATED")) {
    log("j'ai fini de parler");
    // exit();
  }
  return true;
}
var onKarotzConnect = function(data) {
  karotz.button.addListener(buttonListener);
  karotz.ears.addListener(earsListener);
  karotz.rfid.addListener(rfidListener);
  log("je vais parler");
  karotz.tts.start("Salut toi ! Je suis ton ami Karotz !", "fr", exitFunction);
}
karotz.connectAndStart(karotz_ip, 9123, onKarotzConnect, {});
```

Et voici le résultat



```
connected
karotz interactive mode
je vais parler
karotz should say [fr]: Salut toi ! Je suis ton ami Karotz !
j'ai fini de parler
je suis dans l'ears listener
event = START_RIGHT
step = undefined
length = undefined
je suis dans l'ears listener
event = START_LEFT
step = undefined
length = undefined
je suis dans l'ears listener
event = STOP_RIGHT
step = undefined
length = undefined
je suis dans l'ears listener
event = STOP_LEFT
step = undefined
length = undefined
je suis dans le rfid listener
type rfid 1
couleur rfid 2
```

karotz - tutoriel pour débutant sous Windows -

On y voit entre autre que les informations « step » et « length » de l'oreille ne sont pas définis (c'est la raison pour laquelle je n'en ai jamais parlé dans mes tutoriels et que le type de puce RFID retourné est un chiffre ainsi que la couleur (mon tutoriel sur les puces RFID fonctionne sur ce principe et il fonctionne bien)

Par contre s'il vous prenait l'envie d'utiliser mon programme fakir « qui vous « lit » l'information de la clé en « embarqué » et bien il ne fonctionnerait pas. En effet en « embarqué » le type de puce retourné est un « string », « FLAT » par exemple alors qu'en local on avait 1 et la couleur également « RED » là où on avait 1 en local.

Notre « debuglog » c'est comme cela que je l'ai appelé va donc grandement nous aider.

Allez assez parlé, passons à l'action.

Que dit notre site favori à propos de la gestion du port « série »

<http://dev.karotz.com/sdk/#serial>

Add the access to the descriptor : serial

```
karotz.serial.open(port,baudrate)
```

open a serial port

port: /dev/ttyUSB0

baudrate: 50, 75, 110, 134, 150, 200, 300, 600,1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400 or 460800

```
karotz.serial.write(data)
```

write serial data

data: ascii data to write

On pense à ajouter :

```
<access>serial</access>  
Au fichier descriptor.xml
```

Initialisation du port :

```
karotz.serial.open("/dev/ttyUSB0",9600);
```

Ecriture (envoi vers la sortie USB) :

```
karotz.serial.write("je sais ecrire !");
```

karotz - tutoriel pour débutant sous Windows - debuglog



Pour le fichier descriptor je vous laisse faire mais il doit ressembler à ceci :

```
<accesses>
  <access>button</access>
  <access>rfid</access>
  <access>tts</access>
  <access>ears</access>
  <access>serial</access>
</accesses>
```

Pour l'initialisation on ajoute cette ligne au début de notre programme :

```
var onKarotzConnect = function(data) {
  karotz.serial.open("/dev/ttyUSB0",9600);
  karotz.button.addListener(buttonListener);
  karotz.ears.addListener(earsListener);
  karotz.rfid.addListener(rfidListener);
  log("je vais parler") ;
  karotz.tts.start("Salut toi ! Je suis ton ami Karotz !", "fr",
  exitFunction);
}
karotz.connectAndStart(karotz_ip, 9123, onKarotzConnect, {});
```

Alors comment dire... ?

Pourquoi cela m'a-t-il pris un peu de temps pour faire fonctionner tout cela?

Simplement parce que tant que ça ne fonctionne pas intégralement il n'y a aucun moyen de savoir ce qui cloche car il faut le préciser (et ça n'est pas sur notre page favorite) il y a des règles à connaître.

La première c'est que le port Série n'est géré qu'en embarqué ! Impossible donc de jouer avec en local, ça ne facilite pas les choses avouez-le.

La seconde c'est que le port qui est documenté sur la page du SDK correspond à la grosse prise USB, pas au mini USB que nous voulons utiliser.

Alors merci WIKI qui donne le nom de la mini USB et au forum des développeurs sur lequel j'ai retrouvé une réponse de Julien indiquant que le port série ne fonctionnait qu'en embarqué.

Tout ça pour dire quoi ? Que je vais être malheureux car je ne vais pas pouvoir jouer à mon jeu favori qui consiste à vous laisser essayer, passer par les mêmes erreurs que moi afin de mieux apprendre, de vous imprégner de la bonne solution.

Initialisons notre port USB :

le « normal »:

```
karotz.serial.open("/dev/ttyUSB0",9600);
```

Ou

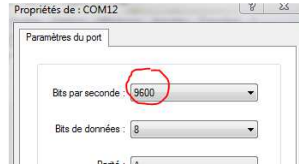
le « mini USB »: (celui que nous utiliserons dans ce tutoriel)

```
karotz.serial.open("/dev/ttyGS0",9600);
```

9600 correspond à la vitesse et ici on doit mettre la même valeur que dans

le paramétrage de HyperTerminal, souvenez-vous :

karotz - tutoriel pour débutant sous Windows -



Hyperterminal va jusqu'à 921600 et Karotz lui s'arrêtant à la valeur précédente (460800) donc pourquoi pas 460800.

Sauf que...je ne suis pas sûr du contrôle de flux, j'ai mis Xon / Xoff mais il se peut qu'il n'y ait aucun contrôle auquel cas la liaison étant dite « asynchrone » il est important que le récepteur puisse recevoir TOUT ce qu'on lui envoie et plus ça va vite et plus il y a de chance d'en louper, je n'ai pas expérimenté car en l'espèce la vitesse n'est pas importante donc restons « sage » et laissons 9600.

Envoyons nos « logs » vers Hyperterminal donc à chaque fois que nous avons une ligne log (quelque chose) on ajoutera une ligne
`karotz.serial.write(le même quelque chose)`



Voici donc notre premier programme prêt pour le debuglog

```
include("util.js");
//var karotz_ip="192.168.1.46"; //ici votre adresse IP
var karotz_ip="localhost"

var buttonListener = function(event) {
  if (event == "DOUBLE") {
    karotz.tts.stop();
    exit();
  }
  return true;
}

var rfidListener = function(data) {
  log("je suis dans le rfid listener") ;
  log("type rfid " + data.type) ;
  log("couleur rfid " + data.color) ;
  karotz.serial.write ("je suis dans le rfid listener") ;
  karotz.serial.write ("type rfid " + data.type) ;
  karotz.serial.write ("couleur rfid " + data.color) ;
  return true;
}

var earsListener = function(event,step,length) {
  log("je suis dans l'ears listener") ;
  log("event = " + event) ;
  log("step = " + step) ;
  log("length = " + length) ;
  karotz.serial.write ("je suis dans l'ears listener") ;
  karotz.serial.write ("event = " + event) ;
  karotz.serial.write ("step = " + step) ;
  karotz.serial.write ("length = " + length) ;

  return true;
}

var buttonListener = function(event) {
  log("je suis dans le bouton listener") ;
  log("event = " + event) ;
  karotz.serial.write ("je suis dans le bouton listener") ;
  karotz.serial.write ("event = " + event) ;

  if (event == "DOUBLE") {
    karotz.tts.stop();
    exit();
  }
  return true;
}

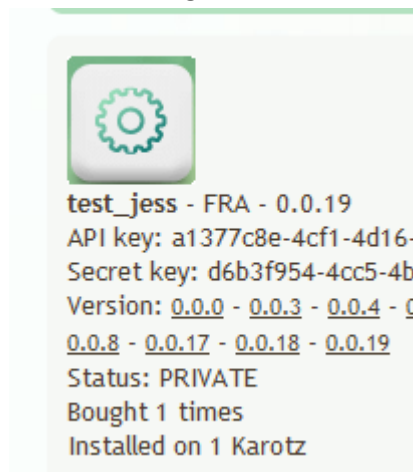
var exitFunction = function(event) {
  if((event == "CANCELLED") || (event == "TERMINATED")) {
    log("j'ai fini de parler") ;
    karotz.serial.write ("j'ai fini de parler") ;
    // exit();
  }
  return true;
}

var onKarotzConnect = function(data) {
  karotz.serial.open("/dev/ttyUSB0",9600);
  karotz.button.addListener(buttonListener);
  karotz.ears.addListener(earsListener);
  karotz.rfid.addListener(rfidListener);
  log("je vais parler") ;
  karotz.tts.start("Salut toi ! Je suis ton ami Karotz !", "fr", exitFunction);
}

karotz.connectAndStart(karotz_ip, 9123, onKarotzConnect, {});
```

karotz - tutoriel pour débutant sous Windows -

On le met en ligne



Et si vous ne faites pas la même erreur que moi, de ne pas mettre à jour le descriptor.xml vous obtenez ceci



Et surtout

Vous voyez ceci dans la fenêtre hyperTerminal



Bon il reste encore un peu de mise en forme mais ça n'est pas mal, non ?

Vous notez les valeurs renvoyées par les oreilles (step et length, enfin !) et constatez que si vous persistez à tester un nombre pour la couleur du RFID ça ne va pas bien fonctionner ;)

La première chose que l'on a envie d'ajouter, enfin ce fut mon cas, c'est juste après l'initialisation c'est

```
var onKarotzConnect = function(data) {  
    karotz.serial.open("/dev/ttyGS0",9600);  
    karotz.serial.write ("**** demarrage de l'application ****");  
    karotz.button.addListener(buttonListener);  
};
```

karotz - tutoriel pour débutant sous Windows -

Et surtout lui demander d'aller à la ligne (cela s'écrit "\n")
Aller à la ligne veut dire « reste à la même position en abscisse (longueur)
mais descend d'une ligne ce qui va vous donner un escalier, c'est rigolo
mais pas très lisible

On ajoute donc un « retour chariot » cela s'écrit "\r"

On trouvera donc \n\r à la fin de nos log

```
karotz.serial.write ( "**** démarrage de l'application ****" "\n\r" );
```

Je vous laisse essayer si vous le souhaitez.

Moi pendant ce temps je continue car l'idée est d'éviter d'avoir à doubler
les lignes de log, une pour la version VM et une pour la version
embarquée.

Ce que j'ai fait : j'ai décidé d'appeler tous mes logs (VM ou embarqués)
Debuglog(mon texte)

Et je vais écrire une routine debuglog qui testera si je suis en VM (auquel
cas le texte sera affiché via l'instruction log classique ou en embarqué
auquel cas le sera sera affiché via le karotz.serial.write

J'en profite pour gérer le retour à la ligne, pour heuro-dater mes lignes,
bref avoir une présentation un peu sexy ;)

Question que je vous pose à vous, oui vous qui me lisez : comment savoir
si je suis en VM ou en embarqué ?

Alors, alors je vous écoute...

Moi ce que je me suis dit c'est quoi de mieux que la variable karotz_ip qui
contient « localhost » en connecté et autre chose en local, merveilleux,
nous allons effectivement utiliser cette variable et le tour sera joué.

Voici ma routine

```
var debuglog = function(string) {  
    var d = new Date();  
    var hh = "" + d.getHours();  
    var mm = d.getMinutes();  
    if (mm < 10) { mm = "0" + mm; }  
    hh = hh + 'h' + mm + " : ";  
    if (karotz_ip == 'localhost') {  
        karotz.serial.write(hh + string + '\n\r');  
    }  
    else log(hh + string );  
    return true  
}
```

karotz - tutoriel pour débutant sous Windows -



```
include("util.js");
//var karotz_ip="192.168.1.46"; //ici votre adresse IP
var karotz_ip="localhost"

var debuglog = function(string) {
  var d = new Date();
  var hh = "" + d.getHours();
  var mm = d.getMinutes();
  if (mm < 10) { mm = "0" + mm; }
  hh = hh + 'h' + mm + ':' + '';
  if (karotz_ip == 'localhost') {
    karotz.serial.write(hh + string + '\n\r');
  }
  else log(hh + string);
  return true;
}

var buttonListener = function(event) {
  if (event == "DOUBLE") {
    karotz.tts.stop();
    exit();
  }
  return true;
}
var rfidListener = function(data) {
  debuglog("je suis dans le rfid listener");
  debuglog("type rfid " + data.type);
  debuglog("couleur rfid " + data.color);
  return true;
}
var earsListener = function(event,step,length) {
  debuglog("je suis dans l'ears listener");
  debuglog("event = " + event);
  debuglog("step = " + step);
  debuglog("length = " + length);

  return true;
}
var buttonListener = function(event) {
  debuglog("je suis dans le bouton listener");
  debuglog("event = " + event);

  if (event == "DOUBLE") {
    karotz.tts.stop();
    exit();
  }
  return true;
}
var exitFunction = function(event) {
  if((event == "CANCELLED") || (event == "TERMINATED")) {
    debuglog("j'ai fini de parler");
    // exit();
  }
  return true;
}
var onKarotzConnect = function(data) {
  karotz.serial.open("/dev/ttyGS0", 9600);
  debuglog("***** demarrage de l'application *****");
  karotz.button.addListener(buttonListener);
  karotz.ears.addListener(earsListener);
  karotz.rfid.addListener(rfidListener);
  debuglog("je vais parler");
  karotz.tts.start("Salut toi ! Je suis ton ami Karotz!", "fr", exitFunction);
}
karotz.connectAndStart(karotz ip. 9123. onKarotzConnect. {});
```

Et le résultat qui a une bonne tête

```
test - HyperTerminal
Fichier Edition Affichage Appeler Transfert ?
19h32 : event = DOUBLE
19h32 : ***** demarrage de l'application *****
19h32 : je vais parler
19h32 : j'ai fini de parler
19h32 : je suis dans l'ears listener
19h32 : event = START_RIGHT
19h32 : step = 0
19h32 : length = 0
19h32 : je suis dans l'ears listener
19h32 : event = STOP_RIGHT
19h32 : step = 3
19h32 : length = 1800
19h32 : je suis dans le rfid listener
19h32 : type rfid FLAT
19h32 : couleur rfid BLACK
19h32 : je suis dans le rfid listener
19h32 : type rfid FLAT
19h32 : couleur rfid GREEN
19h32 : je suis dans le rfid listener
19h32 : type rfid FLAT
19h32 : couleur rfid GREEN
19h35 : je suis dans le bouton listener
19h35 : event = DOUBLE
-
```




Launchtype

Et si on en profitait pour tester les variables renvoyées par LaunchType
Voyons ce que dit notre page préférée

Launch type

You can get the origin of the application launch.

launchType.name

value of launchType.name : { "ASR"; "SCHEDULER", "RFID" }

If "ASR" :

get the asr semantic :

grammar : twitter {\$.cmd='none'}

launchType.semantic.cmd == 'none'

If "RFID" :

get the rfid tag info :

launchType.id

launchType.app

launchType.type

launchType.pict

launchType.color

launchType.data

On trouve également dans le WIKI

params[instanceName].uuid) renvoie l'identifiant de votre lapin

On va donc pouvoir afficher ces informations après l'initialisation du port série, on peut d'ailleurs en profiter pour également afficher les choix faits par l'utilisateur sur nos paramètres de screen.xml . Ces informations n'étant envoyées qu'en connecté on les écrira comme cela

```
var onKarotzConnect = function(data) {
  karotz.serial.open("/dev/ttyGS0", 9600);
  debuglog("***** démarrage de l'application *****");
  if (karotz_ip == "localhost") {
    debuglog(" appli lancée par " + launchType.name);
    debuglog(" identifiant du lapin : " + params[instanceName].uuid);
  }
  karotz.button.addListener(buttonListener);
```

Essayons de suite

Et ça fonctionne ;)

```
21h06 : ***** démarrage de l'application *****
21h06 : appli lancée par RFID
21h06 : identifiant du lapin : dbe381f7-2b5c-4e53-9546-67919ed
21h06 : je vais parler
21h06 : j'ai fini de parler
21h06 : je suis dans l'ears listener
```

Vous avez noté que j'ai écrit démarrage et non démarrage, lancée et non lancée

Là encore Karotz n'aime pas les accents, ça plante l'application. Pensez-y si quelque chose ne fonctionne pas.

Vous l'avez compris ce debuglog n'a pas la prétention de résoudre tous les problèmes mais il vous aidera, j'en suis sûr à une mise au point plus facile.



outils

Je profite de ce tutoriel pour vous donner quelques outils que j'utilise et qui sont bien pratiques.

Vous connaissez NotePad, WordPad mais connaissez-vous notepad++ ?
<http://www.framasoftware.net/article2579.html>

Entre autre :

- Numérote les lignes (pratique pour retrouver la ligne en erreur énoncée par Java)
- Rapproche les accolades (ouverte / fermée, parenthèse ouverte / fermée) permet de voir si on n'a rien oublié et vous permet de choisir
- votre codification (UTF8, ANSI etc)
- Multifenêtre (permet d'avoir main.js, descriptor.xml, screen.xml)
- La coloration syntaxique

```
39 var buttonListener =
40 debugLog("je suis dan
41 debugLog("event = " +
42
43 if (event == "DOU
44 karotz.tts.st
45 exit();
46
47 return true;
48
49 var exitFunction = fu
50 if((event == "CANCELLED") || (event == "TERMINATED")) {
51 debugLog("j'ai fini de parler");
52 // exit();
53 }
54 return true;
55
56 var onKarotzConnect = function(data) {
57 karotz.serial.open("/dev/ttyGS0", 9600);
58 for (var j = 0; j < 1000; j++) {
59 for (var k = 0; k < 1000; k++) {
60 };
61 };
62
63 debugLog("***** demarrage de l'application *****");
64 if (karotz_ip == 'localhost') {
65 mlog(' appli lancée par ' + launchType.name);
66 mlog(' identifiant du lapin : ' + params[instanceName].uuid);
```

Pour les sons j'utilise Audacity <http://audacity.sourceforge.net/?lang=fr>
Editez rapidement vos fichiers audio. Supprimez les silences, ajoutez un écho ou un effet spécial, enlevez les parasites, mixez, etc. Une fois modifié, votre fichier est sauvegardé aux formats OGG, WAV ou MP3. L'éditeur intégré vous permet de copier, coller et assembler des extraits sonores, mais également d'ajuster la vitesse et la hauteur d'un enregistrement.