Activité : mise en oeuvre de la console Enttec

Thierry Vaira <tvaira@free.fr>

07/12/2015 (rev. 1)

Table des matières

Mise en oeuvre de la console Enttec

Expression du besoin

La console Enttec est une console professionnelle permettant de s'intégrer dans une installation destinée à contrôler des appareils DMX.

La console Enttec playback

La console Enttec playback est équipée d'une carte réseau **Ethernet** et elle est capable de communiquer en **UDP/IP**.

Elle utilise un protocole propriétaire **WING** comme protocole de couche Application. Celui est décrit précisément dans le fichier wing_api_spec.pdf.

Il s'appuie sur deux types de messages :

– le message de type **WODD** est envoyé par la console à chaque appui sur une touche ou une modification des contrôle de type glissière (*slider*). Il contient l'état de l'ensemble des touches de la console.

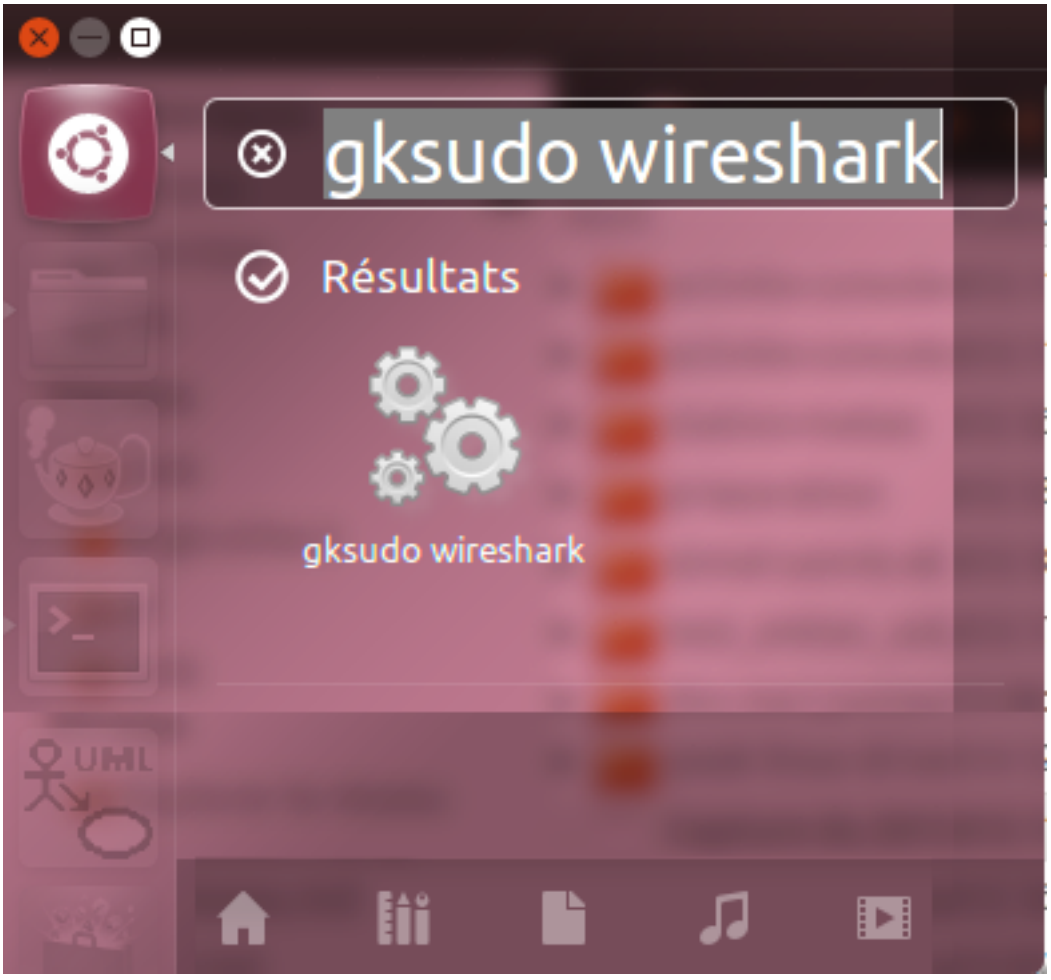
– le message de type **WIDD** est reçu par la console pour commander les 2 afficheurs 7 segments. Il contient la valeur à afficher.

Le transport des messages WODD/WIDD est assuré par le protocole **UDP**. Le modèle DoD est donc le suivant :

Couche	Protocole
Application	WING
Transport	UDP
Réseau	IP
Interface	Ethernet_II

Test

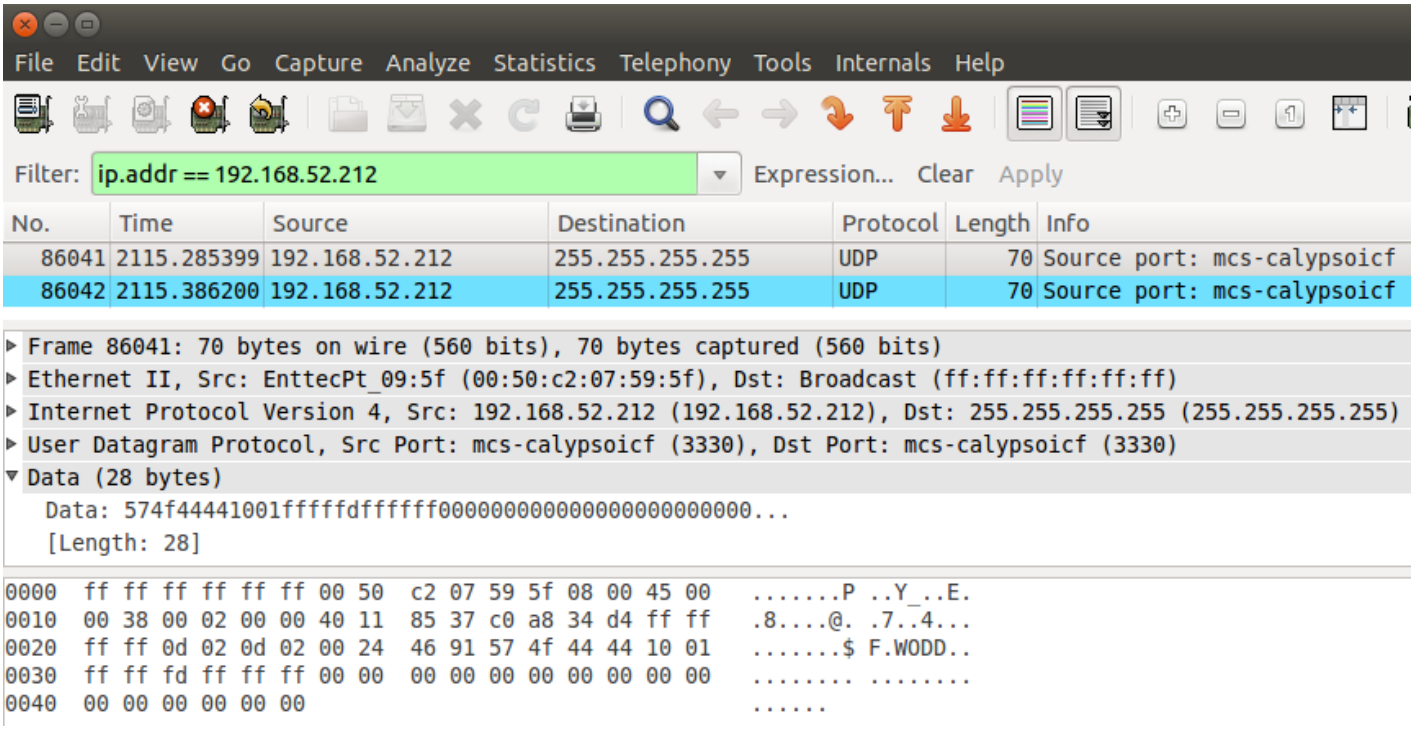
On va réaliser une capture de trames sur le réseau avec **wireshark**.
On démarre wireshark avec les droits *root* avec `gksudo` :



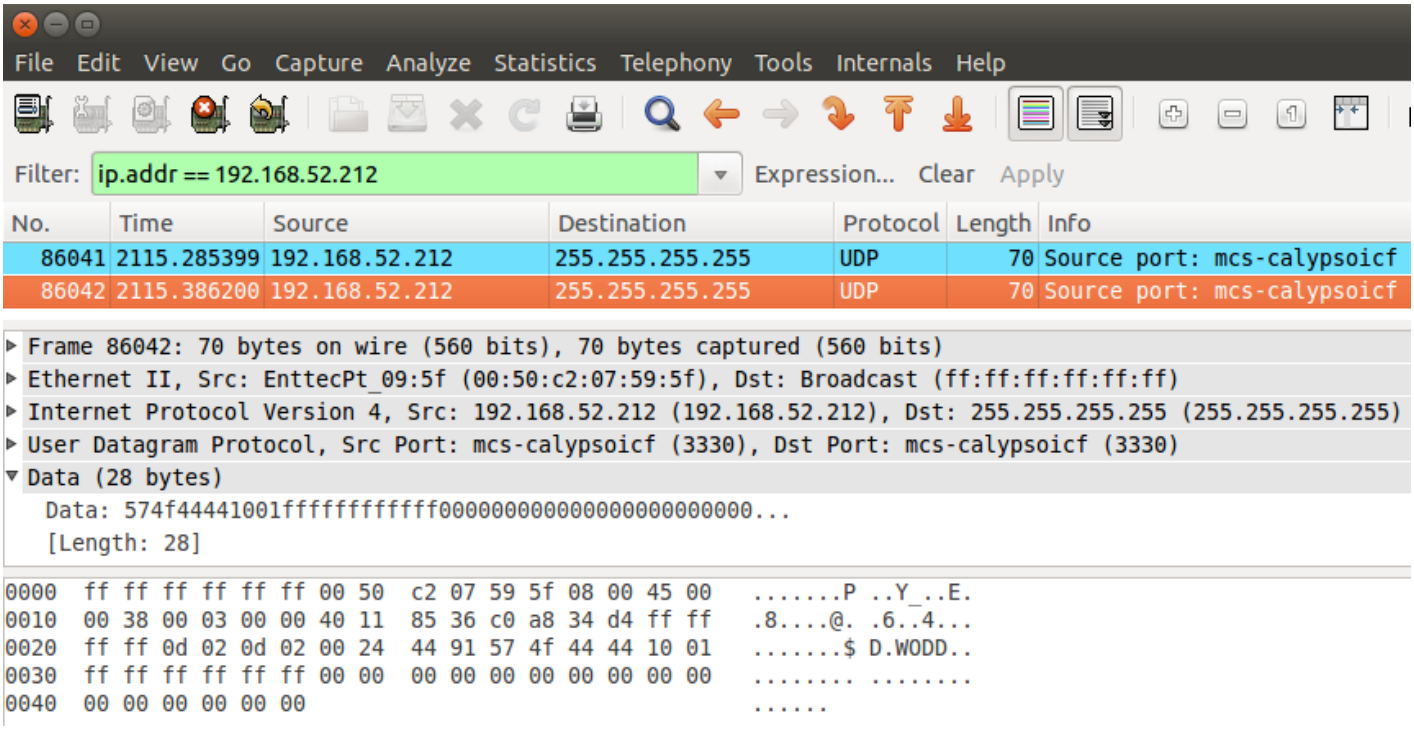
Puis on lance une capture sur l'interface relié au réseau (ici `eth0`).

Aucune trame est reçue. La console Enttec n'envoie donc pas périodiquement l'état des ses boutons et glissières.

On appuie sur le bouton 1. On capture alors 2 trames :



La première trame correspond à l'appui sur le bouton



La deuxième trame correspond à son relachement

La console Enttec émet ses datagrammes en *broadcast* IP (adresse destination **255.255.255.255**) vers le port UDP **3330**. Son port d'écoute pour la réception de datagramme est aussi le **3330**. Son adresse IP est 192.168.52.212.

Important : ici, le choix du *broadcast* IP permet à la console d'envoyer ses trames contenant les états à tous les postes du réseau sans connaître spécifiquement leur adresse. C'est la solution retenue par le fabricant Enttec qui leur a évité de mettre en place un protocole de type requête-réponse.

Les 28 octets de données du protocole WING sont « décodables » à partir de la documentation Enttec wing_api_spec.pdf. On peut remarquer que l'octet à l'adresse 0x32 passe de la valeur 0xFD (appui) à 0xFF (repos).

Les curseurs (*fader* ou *slider*) sont numérotés de 0 à 9, de gauche à droite. Ils renvoient une valeur sur 8 bits suivant la position du curseur. L'affectation des touches est ici.

API Qt

On utilise `QUdpSocket` pour dialoguer avec la console.

- La classe QUdpSocket Qt4 (en)
- La classe QUdpSocket Qt5 (en)

```
Apparu dans les systèmes UNIX, un socket est un élément logiciel qui est aujourd'hui répandu dans la plupart des
    systèmes d'exploitation. Il s'agit d'une interface de communication logicielle avec les services du systè
    me d'exploitation, grâce à laquelle un développeur exploitera facilement et de manière uniforme les
    services d'un protocole réseau. Il s'agit d'un modèle permettant la communication bidirectionnelle inter
    processus IPC (Inter Process Communication) afin de permettre à divers processus de communiquer aussi bien
    sur une même machine qu'à travers un réseau TCP/IP. Les sockets se situent entre la couche Transport et la
    couche Application. En résumé, une socket est un point de communication par lequel un processus peut é
    mettre et recevoir des informations. Ce point de communication devra être relié à une adresse IP et un numé
    ro de port et associé à un mode de communication, le plus souvent : le mode connecté (TCP) ou le mode non
    connecté (UDP).
```

Objectifs

Être capable de dialoguer avec la console Enttec.

Séquence 0 : décodage de trames

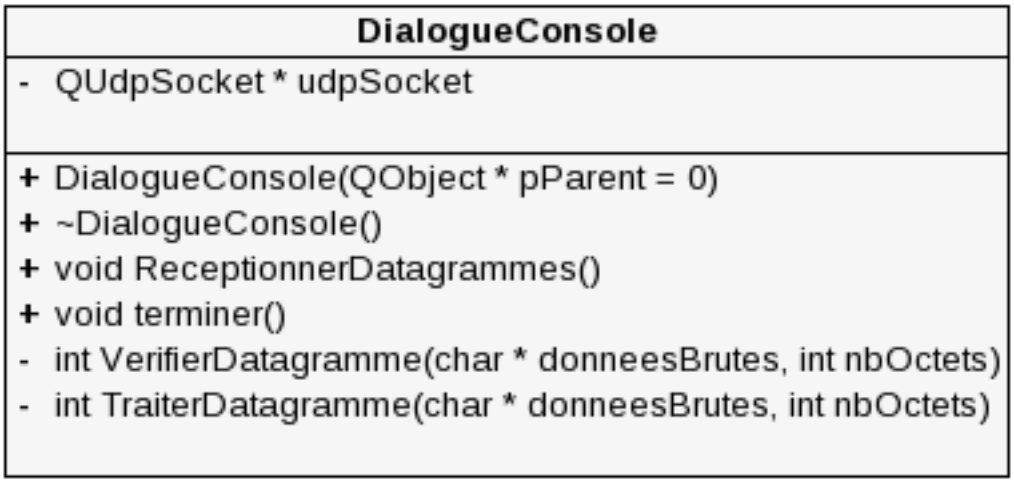
1. En utilisant wireshark et la documentation Enttec, retrouver dans les données échangées où se situent les boutons Page Up et Page Down ainsi que le slider n°0 (le premier à gauche).

Séquence 1 : réceptionner des datagrammes UDP

On va créer une application simple qui permette de recevoir des datagrammes en provenance de la console.

```
$ ./dialogue-udp
<DialogueConsole.cpp:DialogueConsole:23> socket UDP sur le port 3330
<192.168.52.212:3330> datagramme de 28 octet(s) reçu(s)
WODD 0x10 0x01 0xFF 0xFF 0xFD 0xFF 0xFF 0xFF 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00

<192.168.52.212:3330> datagramme de 28 octet(s) reçu(s)
WODD 0x10 0x01 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00
```

Pour cela, on va créer une classe `DialogueConsole` :




```
#define PORT_ENTTEC      3330 //Enttec Port

class DialogueConsole : public QObject
{
   Q_OBJECT

   public:
      DialogueConsole(QObject *pParent = 0);
      ~DialogueConsole();

      void demarrer();

   public slots:
      void ReceptionnerDatagrammes();

   signals :
      void terminer();

   private:
      QUdpSocket     *udpSocket;
      bool           statut;

      int  VerifierDatagramme(char *donneesBrutes, int nbOctets);
      int  TraiterDatagramme(char *donneesBrutes, int nbOctets);
};
```

Le constructeur de cette classe assure la création de la *socket* (`udpSocket`) et son attachement sur le port **3330** des interfaces locales de la machine. Le destructeur fermera tout simplement la *socket*.

```
DialogueConsole::DialogueConsole(QObject *pParent):QObject(pParent)
{
   udpSocket = new QUdpSocket(this);

   // Attachement locale de la socket UDP :
   statut = udpSocket->bind(QHostAddress((QString)"0.0.0.0"), PORT_ENTTEC);

   if(!statut)
   {
      QMessageBox::critical(NULL, "Serveur UDP", "Erreur bind sur le port 3330");
   }
}

DialogueConsole::~DialogueConsole()
{
   udpSocket->close();
}
```


La méthode `demarrer()` (appelée à partir de la fonction `main()`) réalise la connexion du *signal* `readyRead()` au *slot* `ReceptionnerDatagrammes()` (si l'attachement `bind()` a évidemment réussi).

```
void DialogueConsole::demarrer()
{
    if(statut)
    {
       connect(udpSocket, SIGNAL(readyRead()), this, SLOT(ReceptionnerDatagrammes()));
       connect(this, SIGNAL(terminer()), qApp, SLOT(quit()));
    }
}
```

La méthode `ReceptionnerDatagrammes()` assure donc la réception des datagrammes UDP. Pour cela, elle réalise un boucle d'attente sur la présence de données dans la socket. Son pseudo code à compléter est le suivant :

```
void DialogueConsole::ReceptionnerDatagrammes()
{
   int nbOctets = 0;
   int etat;

   // datagramme en attente d'être lu ?
   while (udpSocket->hasPendingDatagrams())
   {
      QByteArray donneesDatagramme;
      QHostAddress emetteurAdresse;
      quint16 emetteurPort;

      // Fixe la taille du tableau au nombre d'octets reçus en attente
      donneesDatagramme.resize(udpSocket->pendingDatagramSize());

      // Lit le datagramme en attente
      //nbOctets = udpSocket->readDatagram(datagram.data(), datagram.size());
      nbOctets = udpSocket->readDatagram(donneesDatagramme.data(), donneesDatagramme.size(),
                                         &emetteurAdresse, &emetteurPort);

      // Vérifie la validité du datagramme
      etat = VerifierDatagramme(donneesDatagramme.data(), nbOctets);
      if(etat == 1)
      {
        #ifdef DEBUG_DialogueConsole
        QString qs_emetteurAdresse = emetteurAdresse.toString();
        cout << "<" << qs_emetteurAdresse.toStdString() << ":" << emetteurPort
             << "> datagramme de " << nbOctets << " octet(s) reçu(s)" << endl;
        #endif

        TraiterDatagramme(donneesDatagramme.data(), donneesDatagramme.size());
      }
      else
      {
        #ifdef DEBUG_DialogueConsole
        QString qs_emetteurAdresse = emetteurAdresse.toString();
        cout << "<" << qs_emetteurAdresse.toStdString() << ":" << emetteurPort
             << "> datagramme de " << nbOctets << " octet(s) reçu(s) : INVALIDE !" << endl;
        #endif
        emit terminer(); /* sinon on quitte ! (à modifier par la suite) */
      }
   }
}
```

La méthode `VerifierDatagramme()` s'assure de la validité du datagramme reçu. Pour cela, on vérifiera que le type du message est bien 'WODD' et que sa taille correspond à celle fournie par le protocole WING. Cette méthode retourne 1 si le datagramme est valide sinon 0.

La méthode `TraiterDatagramme()` réalise pour l'instant seulement l'affichage simple des données reçues.

```
int DialogueConsole::TraiterDatagramme(char *donneesBrutes, int nbOctets)
{
```



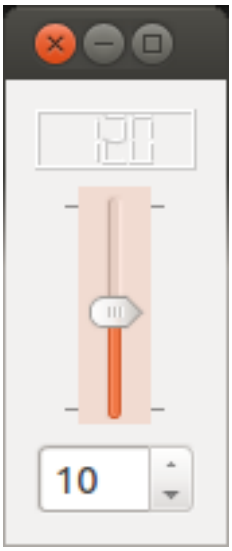
```
  bool commande = false;
  bool enregistrement = false;
  int etat;
  int valeur;

  #ifdef DEBUG_DialogueConsole
  if(nbOctets == LG_MESSAGE_WOOD)
  {
      for(int i=0;i<INDEX_FIRMWARE;i++)
        printf("%c", donneesBrutes[i]); //WODD
      printf(" ");
      for(int i=INDEX_FIRMWARE;i<nbOctets;i++)
        printf("0x%02X ", (unsigned char)donneesBrutes[i]); // le reste des données
      printf("\n\n");
  }
  #endif

  return 0;
}
```

Code source fourni : activite-dialogue-console.zip

1. Tester l'application fournie.
2. Compléter la méthode `TraiterDatagramme()` pour qu'elle assure la détection des touches PAGE_UP ou PAGE_DOWN, et l'affichage du slide n°0 (premier en partant de la gauche).
3. Écrire la méthode `EnvoyerDatagramme()` qui permettra d'envoyer un message de type 'WIDD' vers la console.
4. Modifier l'application pour assurer l'incrémentation et la décrémentation de l'afficheur de la console lorsqu'on appuie sur une touche PAGE_UP ou PAGE_DOWN.

Séquence 2 : application GUI

On vous fournit le code Qt d'un *slider* personnalisée.



Code source fourni : activite-myslider.zip

1. En utilisant la classe `DialogueConsole`, créer un application qui dialogue avec la console Enttec pour synchroniser le *widget myslider* avec le *slider* n°0. Le numéro de canal DMX sera lui aussi synchronisé avec les afficheurs de la console.

Retour au sommaire

