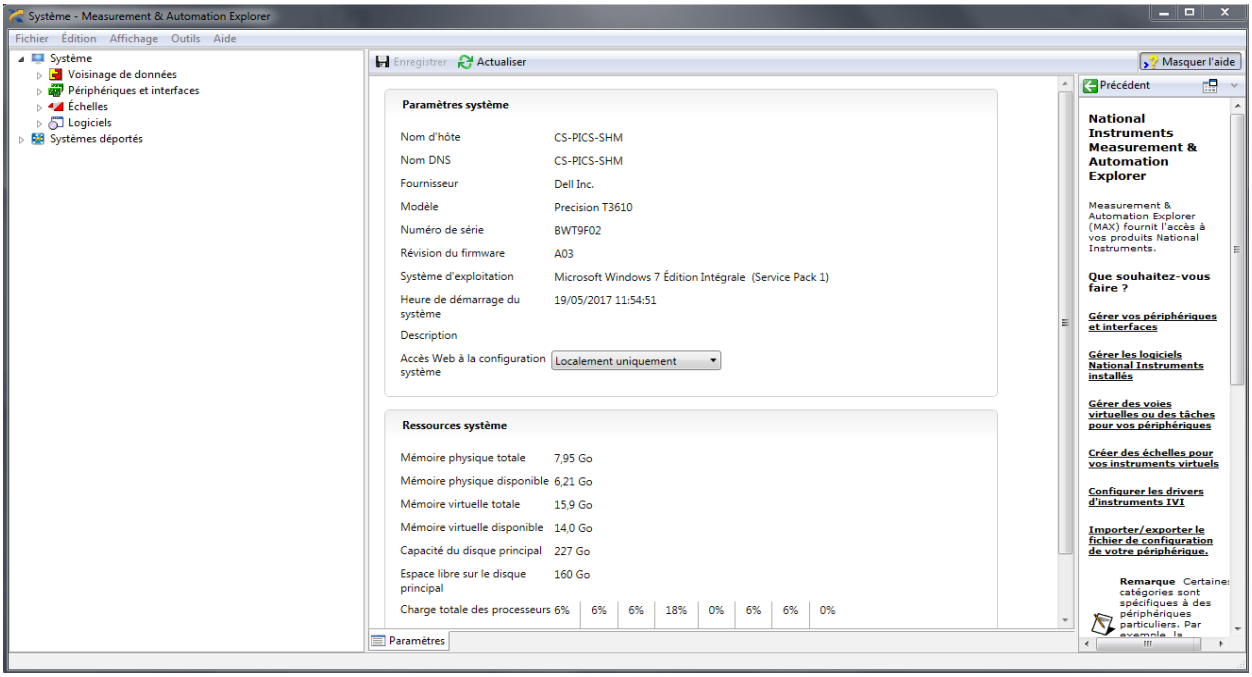
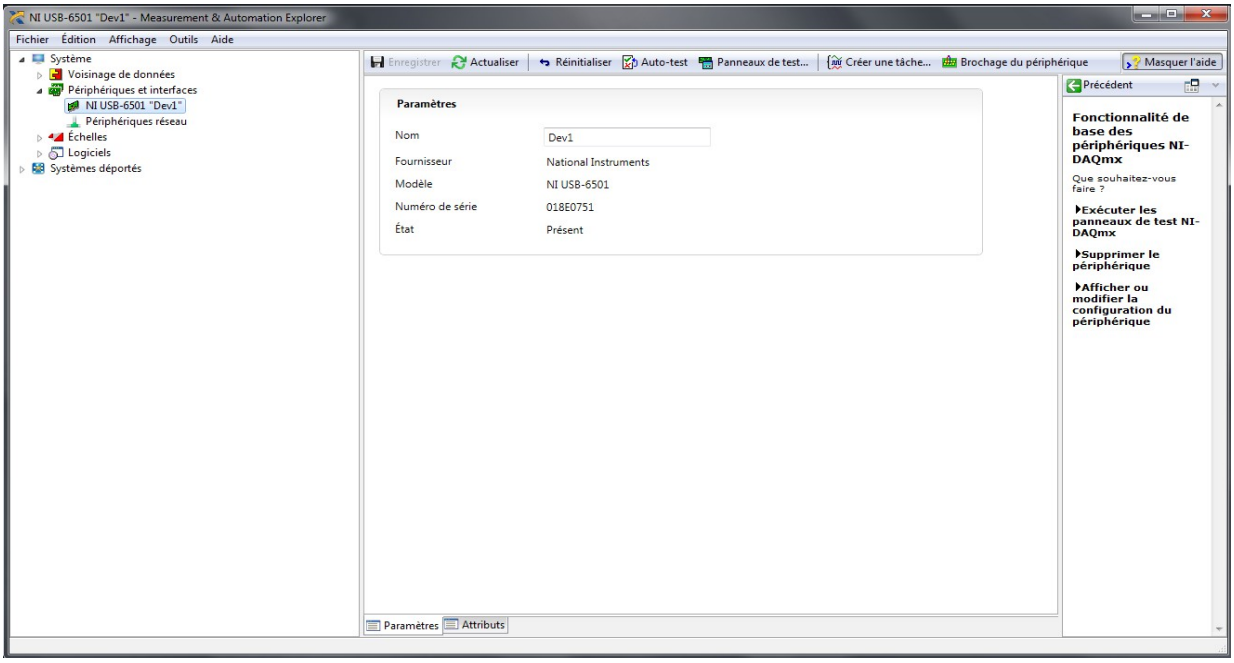
Documentation périphériques National Instrument

1) Récupérer les informations du périphérique

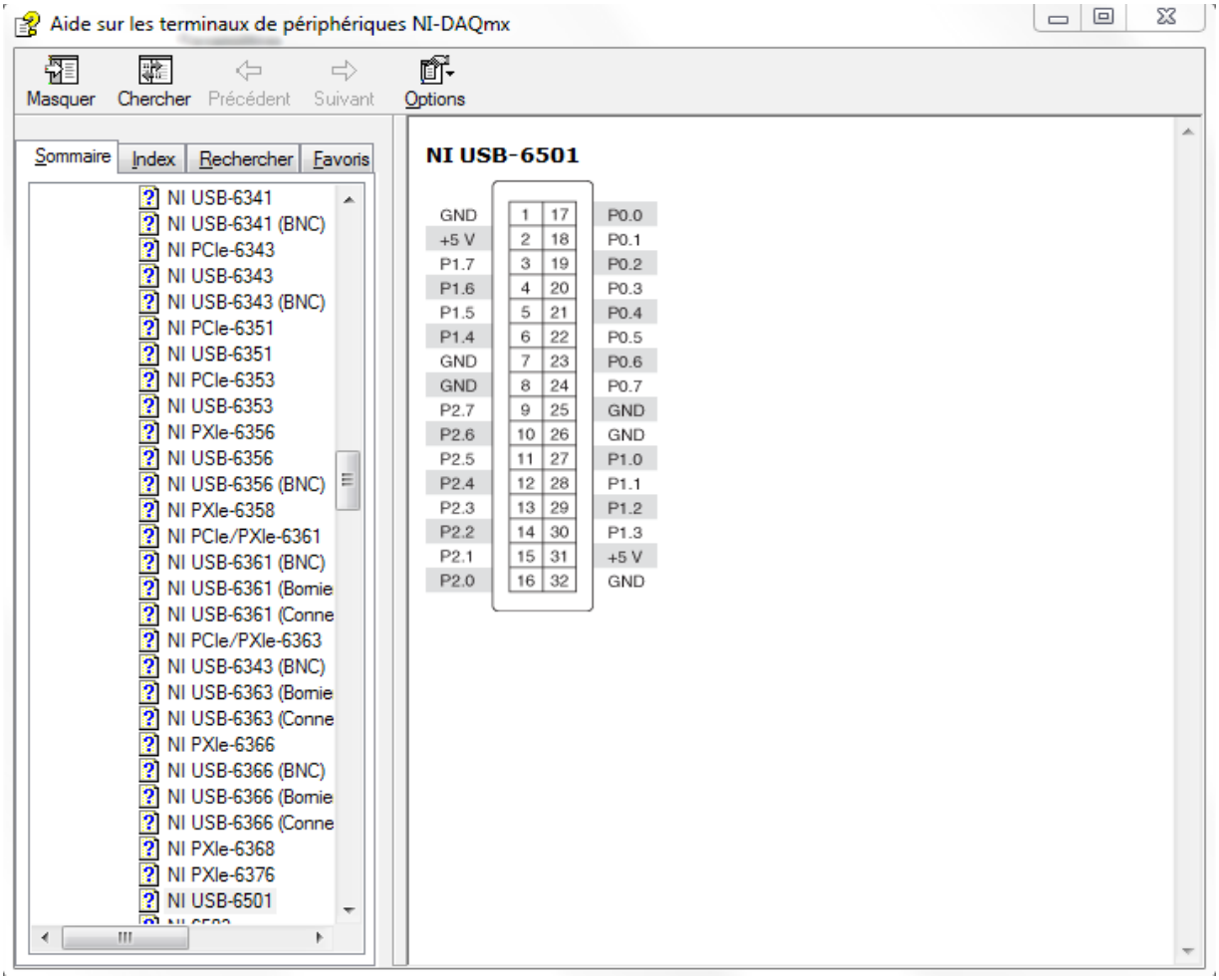
Nous allons utiliser le logiciel NI MAX, pour tester les différentes sorties de nos périphériques. L'interface d'accueil du logiciel se présente comme tel :



L'interface nous donne des informations sur le système. Ce qui nous intéresse est le menu de gauche. Sélectionner le sous-menu "Périphériques et interfaces" pour avoir un visuel sur tous les périphériques National Instrument connectés à l'ordinateur.

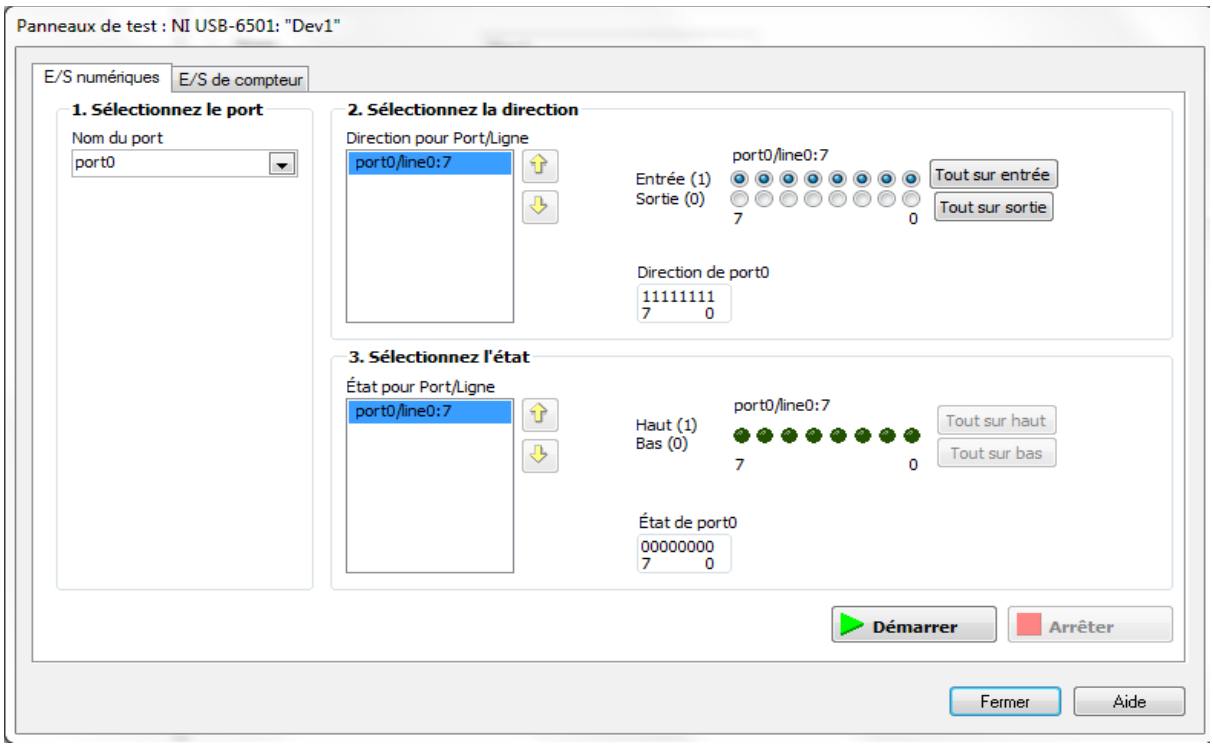


Sur notre exemple il n'y a qu'un seul périphérique de connecté, une carte d'acquisition "USB-6501". En cliquant dessus, nous avons accès à différentes informations sur la carte. Ce qui nous intéresse dans un premier temps est le bouton "Brochage du périphérique" situé en haut dans la barre.

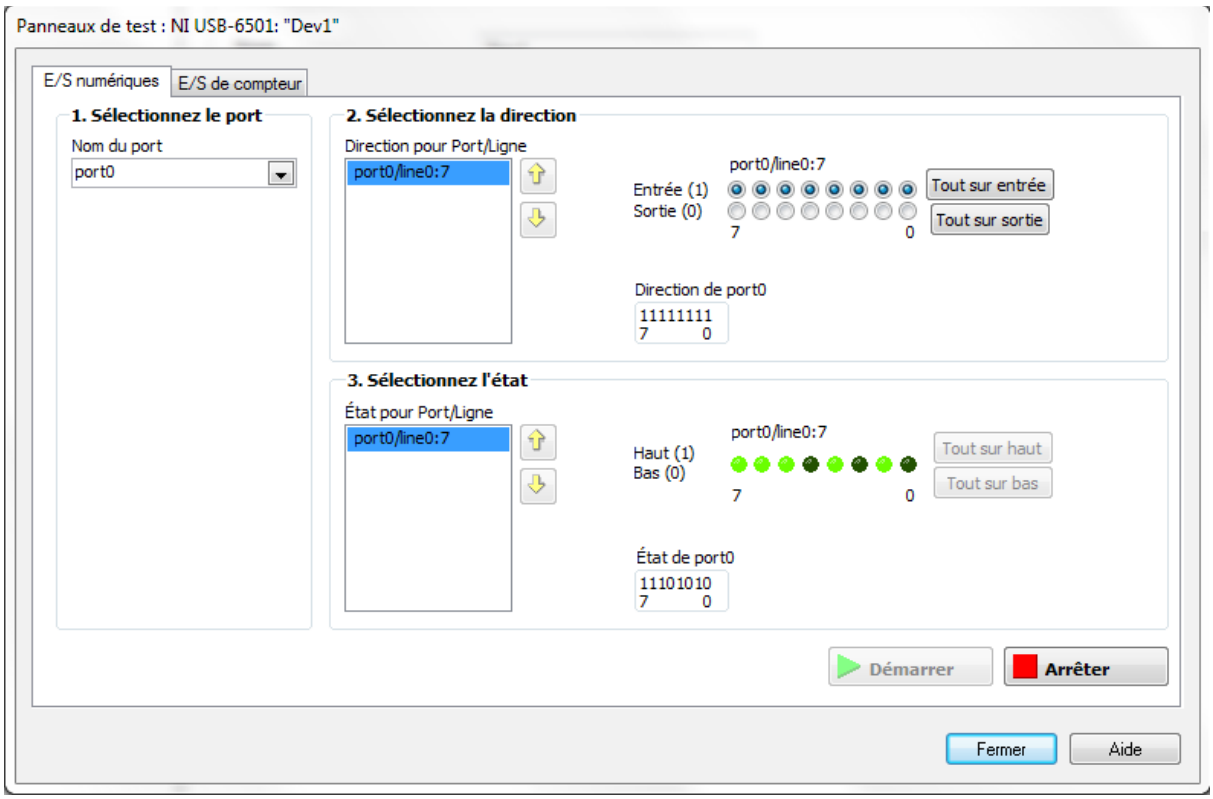


Une nouvelle fenêtre apparaît nous indiquant les différents branchements disponible sur la carte ainsi que le port associé. P0 correspond au port 0, P1 au port 1, etc. Ici chaque port possède 8 lignes, car les ports renvoient un message de 8bit. Chaque ligne correspond à 1bit et à une commande.

Avec ces informations on peut utiliser le bouton "Panneaux de test..." situé en haut dans la barre.



En cliquant dessus, une nouvelle fenêtre apparait. C'est dans cette fenêtre que nous allons pouvoir tester les différentes sorties du périphérique. On doit tout d'abord sélectionner le port à observer grâce au menu déroulant à gauche. Ensuite, on clique sur démarrer pour commencer l'observation du port précédemment sélectionné.



Les LEDs s'allument ou restent éteintes selon le message envoyé par le port. Si on manipule les commandes connectées à la carte National Instrument, le message sera différent, ce qui permet de savoir quelle commande correspond à la ligne du port.

2) Intégration dans un projet C++

Pour intégrer un périphérique National Instrument dans un projet C++ il faut tout d'abord inclure la bibliothèque "NIDAQmx". Une fois cette bibliothèque intégrée dans votre projet, on initialise la récupération du message comme ceci :

TaskHandle taskHandle = 0;
DAQmxCreateTask("", &taskHandle);
DAQmxErrChk(DAQmxCreateDIChan(taskHandle, "Dev1/port0/line0:7", "", DAQmx_Val_ChanForAllLines));

La variable taskHandle est de type *TaskHandle* qui est lui-même un type spécial de la bibliothèque NIDAQmx.
Dans l'exemple ci-dessus la chaine de caractère *"Dev1/port0/line0:7"* signifie que l'on veut récupérer les lignes de 0 à 7 du port 0 de la carte qui porte le nom Dev1.
Si on veut seulement récupérer une seule ligne du port 0 alors on écrit :

"Dev1/port0/line5"

A noté que l'on peut récupérer plusieurs ports avec la même instruction. Pour ce faire on sépare dans la chaine de caractère les différents ports par une virgule.

"Dev1/port0/line0:7,Dev1/port1/line0:7,Dev1/port2/line0:7"

Dans notre exemple la carte National Instrument à 3 ports, avec cette instruction on récupère toutes les lignes de tous les ports.

Pour démarrer la récupération d'information on utilise les commandes suivantes :

```
uInt8 dataTemp[24];
int32 read, bytesPerSamp;
DAQmxErrChk(DAQmxStartTask(taskHandle));
DAQmxErrChk(DAQmxReadDigitalLines(taskHandle, 1, 10.0,
DAQmx_Val_GroupByChannel, dataTemp, 24, &read, &bytesPerSamp, NULL));
```

La variable dataTemp est un tableau d'un type spécial de la librairie "NIDAQmx", qui correspond à un entier unsigned sur 8 bits. Cette variable va stocker les informations que l'on va recevoir de la carte d'acquisition.

Les variables read et bytesPerSamp sont d'un type spécial de la librairie "NIDAQmx", qui correspond à un entier sur 32bit.

Pour fermer les communications avec le port et ainsi le libérer on utilise les instructions suivantes.

```
DAQmxStopTask(taskHandle);
DAQmxClearTask(taskHandle);
```

Pour récupérer les erreurs, il faut exécuter les instructions précédentes dans l'instruction *DAQmxErrChk()* comme ceci :

```
DAQmxErrChk(DAQmxCreateTask("", &taskHandle));
```

Il faut également prévoir une partie *Error* dans notre code.

```
Error:
if (DAQmxFailed(error)){
        DAQmxGetExtendedErrorInfo(errBuff, 2048);
}
if (taskHandle != 0){
        DAQmxStopTask(taskHandle);
        DAQmxClearTask(taskHandle);
}
if (DAQmxFailed(error)){
        cout << "DAQmx Error : " << errBuff << endl;
}
```