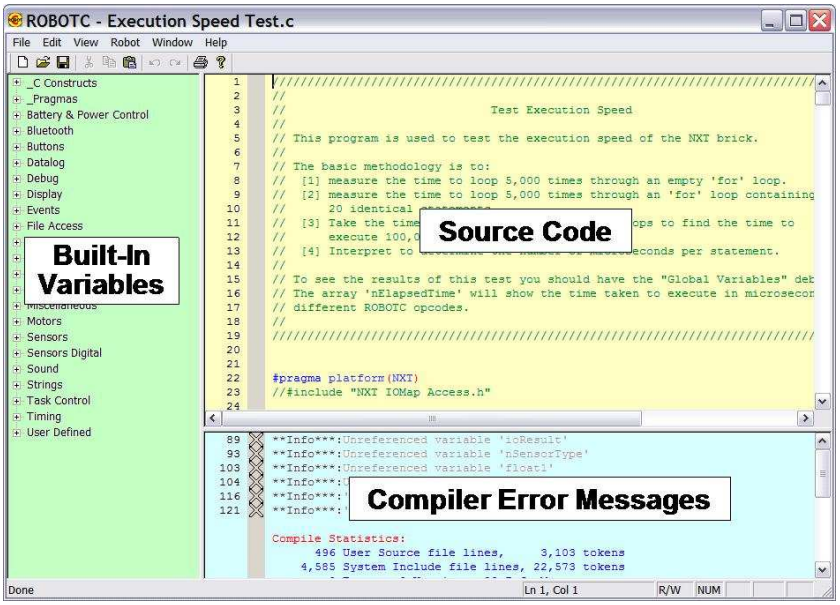
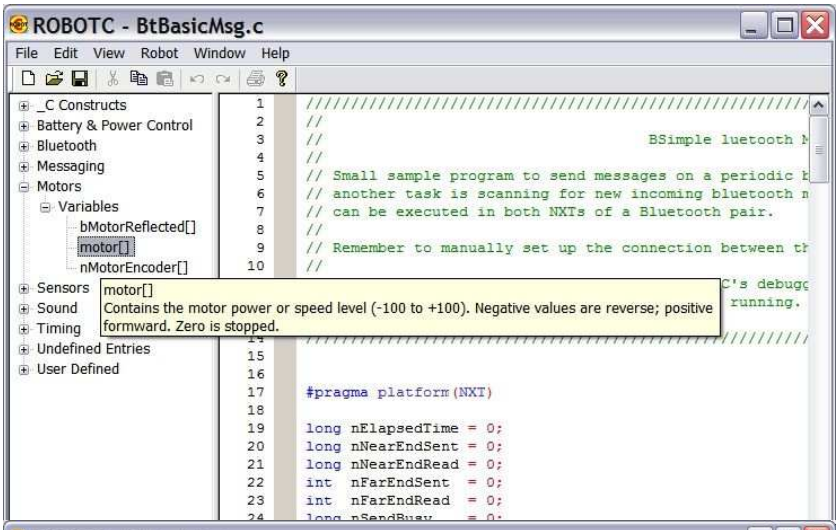
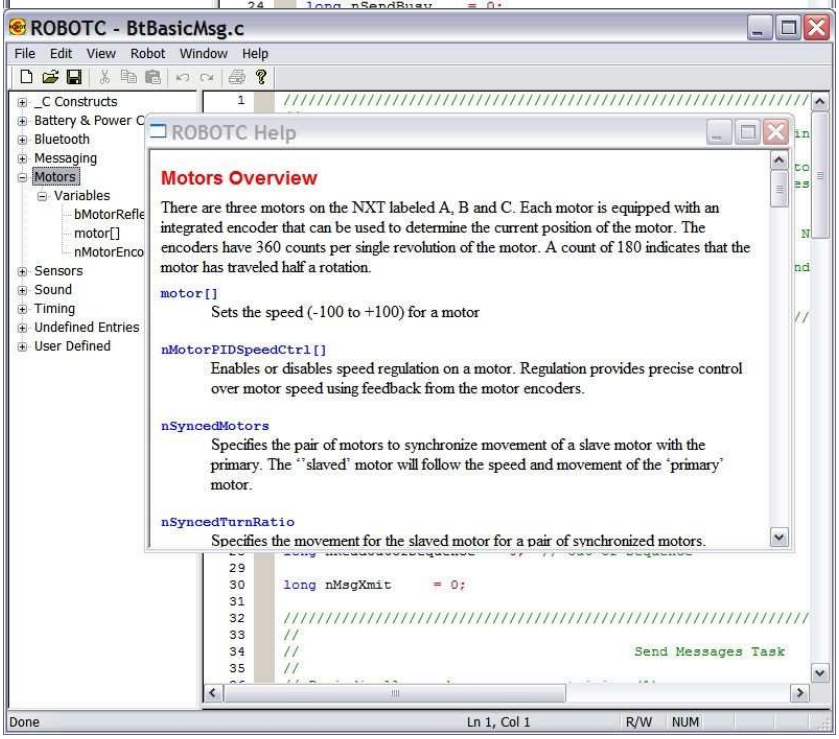
LYCÉE ST-JOSEPH
HASPARREN
SCIENCES ET TECHNIQUES
1, route des missionnaires, 64240 Hasparren/Hazparne
Tél : 05 59 70 29 30 Fax : 05 59 70 29 31
Courriel : accueil@st-joseph-hasparren.fr
Site : http://www.st-joseph-hasparren.fr

PROGRAMMATION DU NXT LEGO AVEC ROBOTC

1. L'éditeur de commande RobotC







Double cliquer sur l'icône du logiciel RobotC.

ROBOTC étend le langage de programmation « C » au contrôle des moteurs et des détecteurs d'un robot par des variables spécifiques.

Par exemple, la variable « motor [ ] » est une variable « tableau » qui contrôle les moteurs du NXT dans notre cas; chaque élément dans le tableau contient le niveau de puissance (vitesse) pour contrôler un moteur simple.

L'interface ressemble à un rédacteur de texte standard, avec le menu habituel et des boutons pour ouvrir et sauver des fichiers, fichiers d'édition, etc. Mais il y a aussi quelques menus spéciaux pour la compilation et le téléchargement de programmes vers le robot et pour l'obtention de l'information depuis le robot.

La fenêtre gauche de l'éditeur ROBOTC fournit une description intégrée de toutes les variables et les fonctions qui sont disponibles dans ROBOTC.

1. Vous pouvez faire glisser des commandes du menu hiérarchique dans la fenêtre de votre programme source.

2. Si vous survolez avec la souris une commande ou une rubrique, un menu contextuel apparaît la décrivant.

3. Un double clic sur une rubrique de menu de niveau supérieur ouvrira un fichier d'aide contenant une vue d'ensemble de ce sujet.

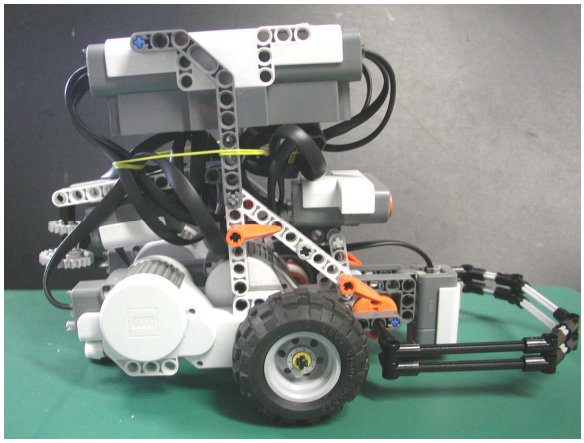
L'image ci-contre montre un déployé de la variable « Motors » avec « moteur [] » comme variable sélectionnée : nous pouvons voir le menu contextuel associé à cette commande. Il ne reste plus qu'à faire glisser cette variable dans la fenêtre code source « source code ».

Un double clic sur un article de menu de niveau supérieur ouvrira un fichier d'aide contenant une vue d'ensemble de ce sujet, ici pour le moteur.

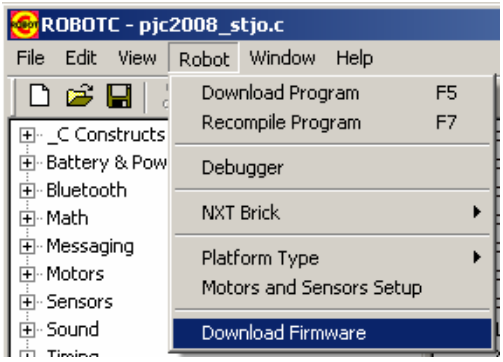
2. Écrire des programmes de base

Dans ce chapitre vous apprendrez à écrire un programme extrêmement simple. Vous allez programmer le robot pour qu'il effectue des déplacements de base, puis vous étudierez l'influence des commandes sur la vitesse de rotation des moteurs et leur sens de rotation.

Le robot que nous emploierons est nommé « robot test » et est issu du challenge 2008. Il est à noter que le moteur de la roue droite est connecté au port A, celui de la roue gauche étant connecté au port C. Le moteur B est utilisé pour lever et baisser la cage permettant de piéger des balles de ping-pong (ou autres).

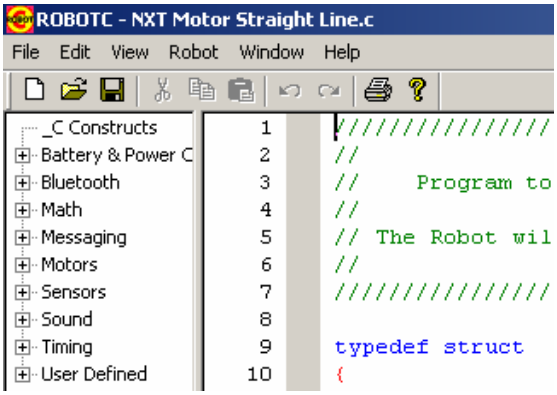


RobotC possède un firmware (microprogramme) plus performant que celui de LEGO. Ce firmware est à charger dans la brique NXT (si ce n'est pas déjà fait) avant toute chose.

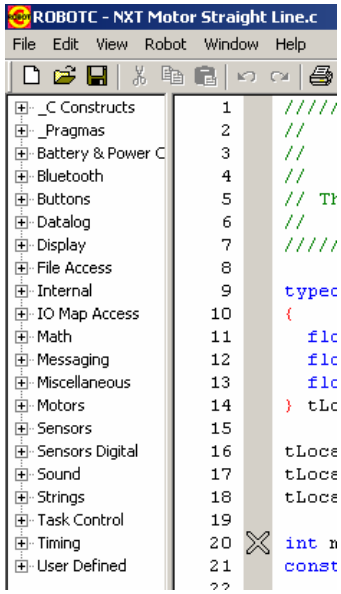


Vous allez écrire un nouveau programme. Donc sélectionnez

- « **File New** » (**CTRL+N**) pour créer une nouvelle fenêtre, vide,
- **Window\menu Level\expert** pour avoir **les menus étendus pour les paramétrages et les déboguages**. En mode basic vous n'aurez que très peu de commande à votre disposition.



Fonctions et variables disponibles en mode Basic



Fonctions et variables disponibles en mode Expert

1. *Inventorier les commandes des moteurs*

Faire avancer le robot pendant 3s

Écrire le programme suivant en sélectionnant les commandes standard dans la fenêtre de gauche et en les faisant glisser dans celle de droite.

```
1 task main()
2  {
3
4    motor[motorC] = 100;
5    wait1Msec(3000);
6
7  }
```

Chaque programme RobotC contient une tâche principale ( ). La tâche principale est habituellement au début du corps principal de votre programme.

```
1 task main()
2  {
3
4    motor[motorC] = 100;
5    wait1Msec(3000);
6
7  }
```

Le programme est délimité par des accolades { } en début et fin.

```
1 task main()
2  {
3
4    motor[motorC] = 100;
5    wait1Msec(3000);
6
7  }
```

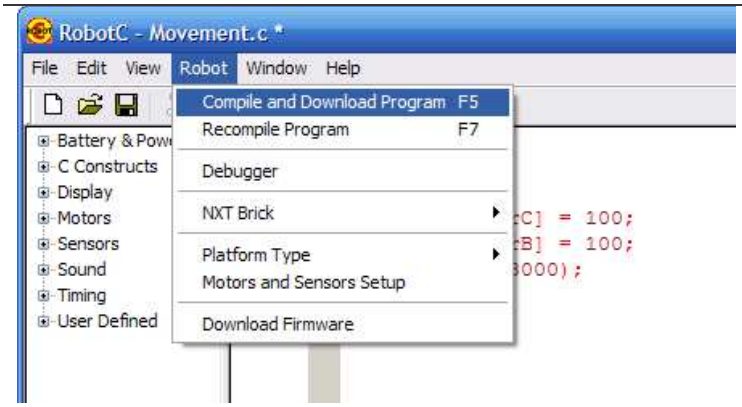
ligne 4 : faire tourner le moteur C avec un niveau de puissance de 100 (c'est le maximum).

```
1  task main()
2  {
3
4     motor[motorC] = 100;
5     motor[motorA] = 100;
6     wait1Msec(3000);
7
8  }
```

Rajouter la commande du moteur A.

Ligne 6 : "attendre …". Elle contient le mot instruction réservé par RobotC "wait1Msec".

Un "M" seconde est une milliseconde ou un millième de seconde. 3000 "Msec" est 3000 millièmes ou trois secondes. Par conséquent, RobotC demande de faire tourner le moteur pendant trois secondes.

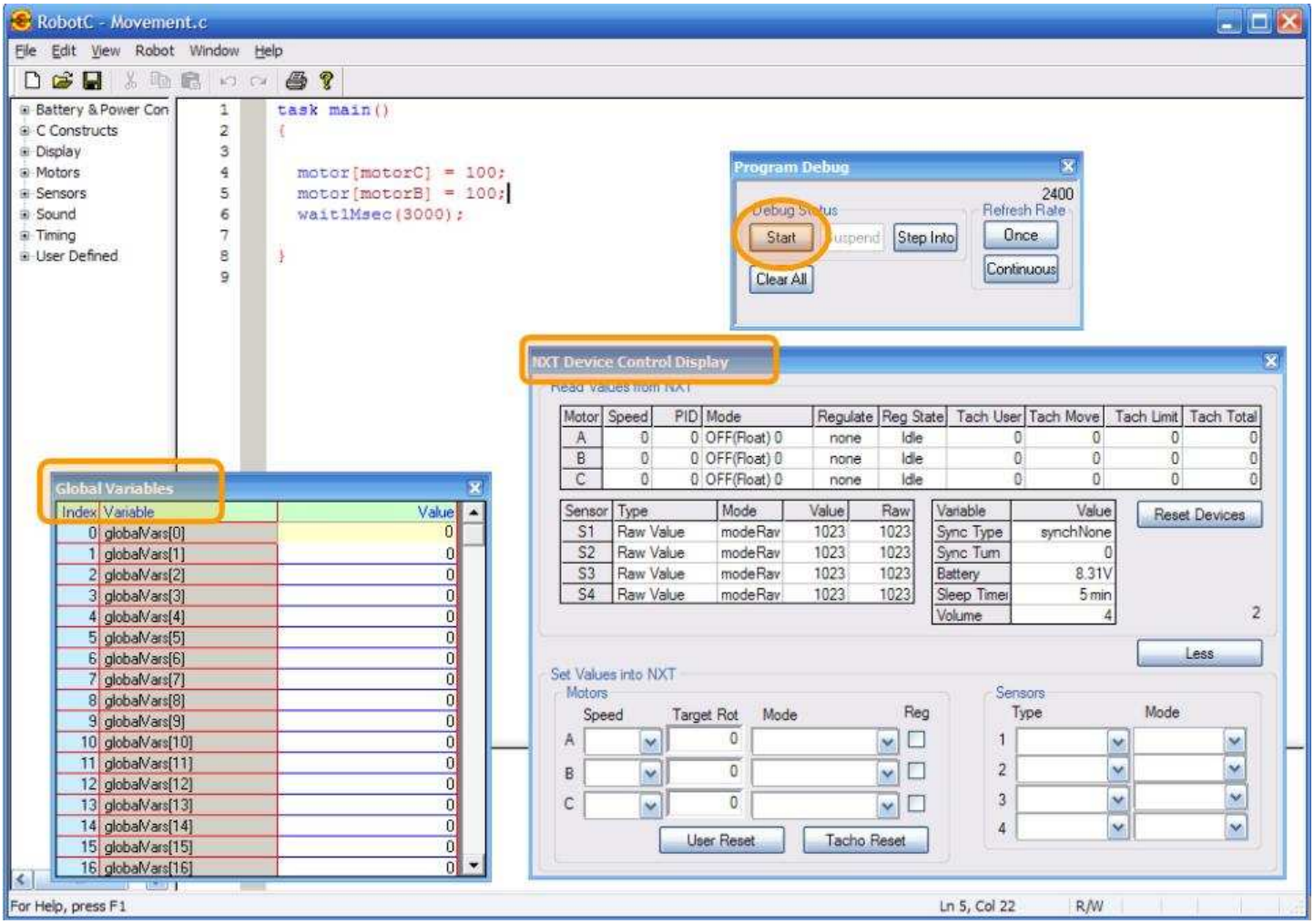


Sauvegarder le programme « **save as** » puis le compiler (transformer les commandes en code machine) et le charger dans la brique NXT.

Quand le téléchargement de votre programme est fait quelques fenêtres de mise au point de programme apparaissent.

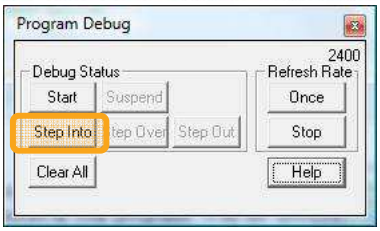
Appuyez le bouton « **Start** » pour commencer à exécuter le programme. Le robot démarre immédiatement.

On peut ajouter des fenêtres de contrôles, pour voir l'état des variables par exemple, par le menu déroulant **Robot\Debug Windows\Global variables** par exemple.



- ***Déboguer (recherche et correction d'erreurs) un programme***

En écrivant les programmes, il est probable que vous fassiez quelques erreurs. Le compilateur répertorie les erreurs et les affiches au bas de la fenêtre : il faut fermer la fenêtre d'avertissement d'erreur et rectifier la syntaxe du code. Ici il faut écrire motor et non pas motot. Lorsqu'il y a plusieurs erreurs, il est possible de double cliquer sur un message pour atteindre la ligne du code en cause. En règle générale, il faut corriger les erreurs une à une (corriger et recompiler pour voir) car une erreur à elle seule, peut en générer d'autres plus bas dans le programme.



Le débogueur permet aussi de contrôler pas à pas « **Step Into** » le déroulement de votre programme lorsque vous êtes en mode expert.

Bouton	Action
Start / **Stop**	Commence ou arrête l'exécution d'un programme.
Suspend / **Resume**	Suspend temporairement ou reprend l'exécution de votre programme.
Once	Simple rafraîchissement du débogueur (une seule fois)
Continuous / **Stop**	Démarre ou arrête le rafraîchissement continu des fenêtres du débogueur.
Step Into	Exécution pas à pas du programme. Si une ligne de programme appelle une fonction à un moment donné alors le pointeur se placera à la première ligne de cette fonction qui s'exécutera si l'on poursuit le pas à pas.
Step Over	Exécution pas à pas du programme. Si une ligne de programme appelle une fonction à un moment donné alors le pointeur se placera à la première exécution fonction qui s'exécutera sera lancé automatiquement.
Step Out	Lance d'entrée l'exécution des fonctions, sans suivre le programme main.

Pour des informations plus détaillées vous pouvez lire la partie « Mise au point du programme ROBOTC » en page 24

Rotation avant/arrière puis arrêt des moteurs.

Nous allons maintenant apprendre à agir sur la vitesse et le sens de rotation (déplacement en avant ou en arrière) du moteur. Le niveau de puissance permet à votre robot d'aller plus rapidement, plus lentement et de s'arrêter en programmant le pourcentage de puissance du moteur employé.

Le niveau de puissance disponible s'étend de -100, (rotation à pleine puissance dans le sens trigonométrique ↶), à 100 (rotation à pleine puissance dans le sens inverse). La pleine puissance pour le NXT, avec une batterie pleinement chargée et une mécanique récente, génère une vitesse de rotation d'environ 1000 impulsions d'encodage par seconde (vitesse de rotation mesurée par le codeur couplé au moteur).

Avec une batterie légèrement déchargée et une mécanique légèrement usée, la vitesse de rotation sera réduite à 750 impulsions par seconde. La vitesse de rotation est dépendante de la puissance transmise ce qui est cohérent.

```
1   task main()
2   {
3     motor[motorC]= 100;
4     motor[motorA]= 100;
5     wait1Msec(1000);
6
7     motor[motorC] = 0;
8     motor[motorA] = 0;
9
10    motor[motorC] = -100;
11    motor[motorA] =  100;
12    wait1Msec(350);
13
14    motor[motorC] = 0;
15    motor[motorA] = 0;
16  }
```

Moteur C : rotation à pleine vitesse avant
Moteur A : rotation à pleine vitesse avant
Pendant 1s (temporisation)

Arrêt moteurs A et C
Moteur C : rot. pleine vitesse arrière
Moteur A : rot.pleine vitesse avant
} Rotation du robot sur lui-même d'environ 90°.
pendant 350 ms

Arrêter les Moteur A et C

Chargez le programme (touche F5) dans le robot et testez-le.

Remarque : Ce programme n'est pas applicable dans la réalité : il est trop aléatoire et pas assez précis. L'angle de rotation effectivement atteint dépend de la nature du sol, de la charge de la batterie, … Il faudra utiliser le codeur couplé au moteur pour atteindre réellement l'angle désiré. Nous verrons cela un peu plus loin.

Définir des variables en tête de programme pour faciliter sa mise au point

```
1   #define MOVE_TIME 1000
2   #define TURN_TIME  350
3
4   task main()
5   {
6     motor[motorA]=motor[motorC]=100;
7     wait1Msec(MOVE_TIME);
8     motor[motorC]=-100;
9     wait1Msec(TURN_TIME);
10    motor[motorA]=motor[motorC]=0;
11  }
```

Vous avez pu constater que le programme précèdent n'est pas très performant : le robot n'effectue pas une rotation de 90 très précise.

Nous allons un peu améliorer notre programme pour rendre plus facile la mise au point du fonctionnement désiré.

Nous allons utiliser deux constantes (**Move_time = temps d'avancée** et **Turn_time = temps de rotation**) à placer en tête du programme. Nous pourrons modifier ces constantes, essai après essai en fonction du comportement du Robot, pour gagner en précision lors de la rotation à 90°.

Nous avons gagné en facilité d'intervention sur le programme mais la précision de la rotation n'est toujours pas assurée automatiquement : c'est le programmeur qui juge du résultat à l'œil.

Le bloc moteur du NXT possède un codeur qui relève la position angulaire du moteur. Nous allons utiliser ce codeur pour arrêter avec précision le robot.

Utiliser les codeurs intégrés du NXT et une boucle While

```
1   //*****************************************************************************************//
3   //                              Configuration                                              //
4   //                                                                                         //
5   //  [I/O PORT]              [Type]                [Description]                            //
6   //  Port A                  moteur                  moteur droit                           //
7   //  Port C                  moteur                  moteur gauche                          //
8   //  diametre roue (diam_roue) : 57 mm                                                      //
9   //  entraxe (entraxe) : 117 mm                                                             //
10  //  calcul liant l'angle de la roue (cod_roue)a l'angle de rotation du robot (ang_rob)      //
11  //  cod_roue = (entraxe / diam_roue) * ang_rob
12  //                                                                                         //
13  //*****************************************************************************************//
14
15  #define diam_roue 57
16  #define entraxe 117
17  #define ang_rob 90
18
22  task main()
23  {
24    nMotorEncoder[motorA] = 0;                    //capteur de rotation du moteur A remis  0
26    nMotorEncoder[motorC] = 0;                    //capteur de rotation du moteur C remis  0
27
30    float cod_roue = entraxe / diam_roue * ang_rob; // calcul de la valeur de codage cod-roue
30                                                    //atteindre pour l'angle de rotation desire
30
30    //les moteurs vont etre alimentes tant que la valeur de codage cod_roue n'est pas atteinte
31    while(nMotorEncoder[motorA] < cod_roue || nMotorEncoder[motorC] > cod_roue)
32   {
33        motor[motorA] = 30;                       //motor A tourne a 30%
34        motor[motorC] = -30;                      //motor B tourne a -30%
35     }
36  }
```

Programme 1 : rot_par_codeur.c

Il faut tout d'abord prendre en compte les paramètres du robot pour le faire tourner sur place avec précision. Il faut connaître le diamètre des roues et l'entraxe entre les deux roues.

Nous allons voir une autre manière d'obtenir ce résultat avec la commande « **`nMotorEncoderTarget[motorA]`** » mais pour cela il nous faut d'abord apprendre à utiliser les nombres réels (virgule flottante) et les commandes de synchronisation des moteurs.

Utiliser les nombres réels

```
1   // utilisation de nombre reels suivant le mode de declaration des variables //
2
3   task main()
4   {
5       float f;        // nb a virgule => nb rel
6       //float x=17;   // nb a virgule => nb rel
7       //float y=9;    // nb a virgule => nb rel
8       int x=17;       // les grandeurs x et y sont entieres
9       int y=9;
10
11          //f = x/y;              //si x et y ont ete declare en float cette ecriture suffit
12          f = (float)x/(float)y; // x et y declare en int il faut rajouter "(float)"
13          nxtDisplayTextLine(1,"Test:%f",f);
14          wait1Msec(10000);       // on attend 10s pour laisser le temps de lire le message
15       eraseDisplay();            // l'affichage est efface, l'ecran est "blanchi"
16  }
17
```

Programme 2 : utilisation_nb_reels.c

Nous avons deux cas de figures :

- Soit les variables en entrée sont définies comme réels (float pour virgule flottante) et dans ce cas il n'est pas besoin de spécifier que l'on veut pour f un résultat réel **`(f = x/y ;)`**
- Soit on les a déclarés entières (int pour integer) et dans ce cas il est nécessaire de spécifier l'état à utiliser dans l'équation **`(f = (float)x/(float)y;)`**

Synchroniser les deux moteurs d'avance : 3 modes de marche

```
1   // utilisation de la commande de synchronisation des moteurs //
2
3   #define vit 50                  //affectation de la vitesse desiree
4
5   task main()
6   {
7      nMotorEncoder[motorA] = 0;
8      nMotorEncoder[motorC] = 0;    //capteur de rotation des moteurs A et C est mis a 0
9      nSyncedMotors = synchAC;     //synchronisation sur le moteur A qui est maitre (C est esclave)
10
11     nSyncedTurnRatio = 100 ;     // 100 indique une marche dans le meme sens des 2 moteurs (avant ou arriere)
12
13         motor[motorA] = vit;     //marche avant a vitesse definie pendant 500 ms
14         wait1Msec(500);
15         motor[motorA] = 0;       //arret pendant 500 ms
16         wait1Msec(500);
17         motor[motorA] = -vit;    //marche arriere a vitesse definie pendant 500 ms
18         wait1Msec(500);
19
20     nSyncedTurnRatio = -100 ;    // -100 indique une marche inversee de la roue C en fonction de A
21         motor[motorA] = vit;     // marche avant a vitesse definie pendant 1s
22         wait1Msec(1000);         // rotation sur lui meme vers la gauche du robot
22         motor[motorA] = 0;       //arret pendant 500 ms
22         wait1Msec(500);
23         motor[motorA] = -vit;    // marche arriere a vitesse definie pendant 1s
26         wait1Msec(1000);         // rotation sur lui meme vers la droite du robot
27
28     nSyncedTurnRatio = 0 ;       // 0 indique un arret de la roue C en fonction de A
29         motor[motorA] = vit;     // marche arriere a vitesse definie pendant 1s
30         wait1Msec(1000);         // rotation autour de la roue C vers la gauche du robot
31  }
```

Programme 3 : moteur_maitre_esclave.c

Utiliser les codeurs intégrés du NXT et la commande nMotorEncoderTarget[motorA]

```
5   //  Port A                  moteur                  moteur droit                                //
6   //  Port C                  moteur                  moteur gauche                               //
7   //  diametre roue (diam_roue) : 57 mm                                          //
8   //  entraxe (entraxe) : 117 mm                                                                  //
9   //  calcul liant l'angle de la roue (cod_roue)a l'angle de rotation du robot (ang_rob)          //
10  //  cod_roue = (entraxe / diam_roue) * ang_rob
11  //                                                                                              //
12  //*********************************************************************************************//
13
14  #define diam_roue 57
15  #define entraxe 117
16  #define ang_rob 90
17  #define vit 20
18
19
20
21  task main()
22  {
23
24    nMotorEncoder[motorA] = 0;      //capteur de rotation du moteur A remis  0
25    nMotorEncoder[motorC] = 0;      //capteur de rotation du moteur C remis  0
26    nSyncedMotors = synchAC;        //synchronisation sur le moteur A qui est maitre (C est esclave)
27    nSyncedTurnRatio = -100 ;       // -100 => marche inversee de la roue C en fonction de A
28
29    float cod_roue = (float)entraxe /(float)diam_roue * (float)ang_rob;
30    nxtDisplayTextLine(1,"val codeur:%d",cod_roue);   //afficher la valeur cod_roue
31    wait1Msec(1000);                                  //pendant 10s
32    eraseDisplay();                                   // effacer l'affichage
33
34    nMotorEncoderTarget[motorA] = cod_roue;     // marche avant a vitesse definie tant que la valeur
35                                                // du codeur cod_roue n'est pas atteinte
36
37    motor[motorA] = vit;        // attention a bien placer la commande motor apres nMotorEncoderTarget
38    while (nMotorRunState[motorA] != runStateIdle) // le moteur s'arretera quand le valeur de consigne
39                                                   // cod_roue sera egale a la valeur encodeur
40
41    nxtDisplayTextLine(1,"val codeur:%d",nMotorEncoder[motorA]);  //afficher la valeur de l'encodeur
42    wait1Msec(1000);                                              //pendant 10s
43    eraseDisplay();                                               // effacer l'affichage
44
45
46  }
```

Programme 4 : rotation_codeur_target sans affichage.c

La commande **`while (nMotorRunState[motorA] != runStateIdle)`** utilise la variable d'état du moteur « **`Idle`** ». Le moteur tourne tant que la valeur de consigne chargée en **`nMotorEncoderTarget[motorA]`** n'est pas atteinte (i.e. tant que la variable d'état du moteur **`nMotorRunState[motorA]`** est différente de Idle). Lorsque c'est le cas la variable passe à « idle » et le moteur s'arrête.

Les 2 modes de commande PWM* des moteurs à l'arrêt

L'alimentation des moteurs se fait sous la forme d'impulsions* plus ou moins larges : cette méthode permet de réguler la vitesse des moteurs en fonction des informations transmises par les encodeurs. Si le moteur tourne moins vite que la valeur de consigne, la valeur moyenne du train d'impulsions sera plus élevée (La tension augmentant, la vitesse augmente également), dans le cas contraire cette valeur moyenne diminuera.

** PWM : Pulse Width Modulation = MLI : Modulation de Largeur d'Impulsion.*

On peut à partir de ce principe créer un couple moteur à vitesse nulle (commande vectorielle), ce qui est impossible en commande classique : quand U=0, I=0 et le couple est donc nul.

La commande **`bFloatDuringInactiveMotorPWM`** permet d'activer ou non cette fonction. Pour plus de détail sur les commandes de moteurs lire **Commande des Moteurs** pages **37**.

Exemple de MLI

```
1   task main()
2   {
3
4   nxtDisplayTextLine(1,"Motor Brake",);
5   bFloatDuringInactiveMotorPWM = false; // Moteur en mode brake (couple  l'arret)
6
7   nxtDisplayTextLine(2,"Target 1000",);
8   nMotorEncoder[motorA] = 0;            // initialisation de la variable donnant la valeur en cours de l'encodeur
9   nMotorEncoderTarget[motorA] = 1000;   // consigne
10  motor[motorA] = 100;
11
12
13  while (nMotorRunState[motorA] != runStateIdle) // le moteur tourne tant que le registre de l'encodeur
14                                                 // n'indique pas que la valeur de consigne est atteinte
15  {
16  nxtDisplayTextLine(3,"Encoder=%d", nMotorEncoder[motorA]);
17  }
18
19  wait1Msec(3000);
20
21  nxtDisplayTextLine(1,"Motor Float",);
22  bFloatDuringInactiveMotorPWM = true; // Moteur en mode float (roue libre  l'arret)
23
24  nxtDisplayTextLine(2,"Target 1000",);
25  nMotorEncoder[motorA] = 0;
26  nMotorEncoderTarget[motorA] = 1000;
27  motor[motorA] = 100;
28
29
30  while (nMotorRunState[motorA] != runStateIdle)
31  {
32  nxtDisplayTextLine(3,"Encoder=%d", nMotorEncoder[motorA]);
33  }
34  }
```

Programme 5 : PWM_modes.c

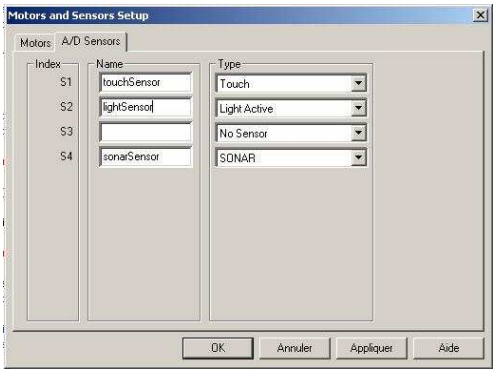
2. Utiliser les capteurs pour guider le robot.

L'affectation des capteurs et des moteurs sur les ports du NXT se fait par le menu **Robot\Motors and Sensors Setup\.**

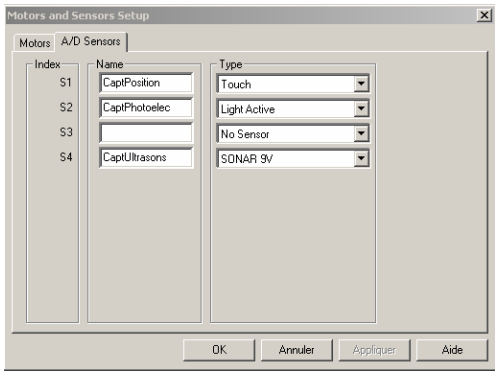
RobotC génère alors du code automatiquement pour les déclarer.

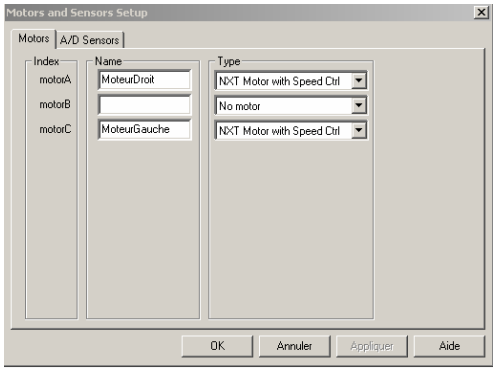
Ces lignes sont repérées par « Auto » en lieu et place du N° de ligne.

Exemples de déclarations



En anglais





En français

```
Auto   const tSensors touchSensor          = (tSensors) S1;   //sensorTouch        //*!!!!*//
Auto   const tSensors lightSensor          = (tSensors) S2;   //sensorLightActive  //*!!!!*//
Auto   const tSensors sonarSensor          = (tSensors) S4;   //sensorSONAR        //*!!!!*//
Auto   //*!!CLICK to edit 'wizard' created sensor & motor configuration.                !!*//
```

Résultat de la déclaration automatique en anglais

```
Auto   const tSensors CaptPosition         = (tSensors) S1;   //sensorTouch        //*!!!!*//
Auto   const tSensors CaptPhotoelec        = (tSensors) S2;   //sensorLightActive  //*!!!!*//
Auto   const tSensors CaptUltrasons        = (tSensors) S4;   //sensorSONAR9V      //*!!!!*//
Auto   const tMotor   MoteurDroit          = (tMotor) motorA; //tmotorNxtEncoderClosedLoop //*!!!!*//
Auto   const tMotor   MoteurGauche         = (tMotor) motorC; //tmotorNxtEncoderClosedLoop //*!!!!*//
Auto   //*!!CLICK to edit 'wizard' created sensor & motor configuration.                !!*//
```

Résultat de la déclaration automatique en français

🗐 Capteur de position mécanique (détecteur de position « touch sensor »)

Ce programme permet au robot d'avancer indéfiniment en fonction des informations qu'il reçoit de son détecteur de position :

- si le détecteur de contact est actionne (à la main puisque sur le robot test ce capteur est à l'arrière) le robot tournera à gauche et continuera à avancer,

La commande "random" est employée pour générer un nombre aléatoire entre le zéro et le nombre (2000 dans notre cas) contenu dans parenthèses. Cela nous permet de créer une temporisation aléatoire de 0 à 2s afin de donner l'impression que le robot est plus ou moins autonome, qu'il n'a pas un fonctionnement immuable.

```
Auto  const tSensors touchSensor          = (tSensors) S1;   //sensorTouch        //*!!!!*//
Auto  //*!!CLICK to edit 'wizard' created sensor & motor configuration.                !!*//
   1
   2  task main()
   3  {
   4     int randTime;                          //la variable "randTime" est declaree
   5     wait1Msec(500);                        //tempo de 500 ms avant d'executer la suite du programme
   6
   7     while(true)                            // boucle infinie avec la condition booleene "true" vraie
   8     {
   9        motor[motorA] = 75;                 //vitesse des moteurs A&C a 75 %
  10        motor[motorC] = 75;
  11
  12        if(SensorValue(touchSensor) == 1)   // si detecteur S1 actionne alors executer prog. suivant
  13        {
  14           motor[motorA] = -75;             //vitesse des moteurs A&C a 75 % en rotation inverse
  15           motor[motorC] = -75;
  16           wait1Msec(750);                  //tempo de 750 ms avant d'executer la suite du programme
  17
  18           motor[motorA] = 75;              //moteur A a 75 % en avant
  19           motor[motorC] = -75;             //moteur A a 75 % en arriere
  20           randTime = random(2000);         //un nombre entier aleatoire entre 0 et 2000 est affecte
  21                                            //a la variable "randTime"
  22           wait1Msec(randTime);             //tempo de valeur aleatoire avant execution suite prog.
  23        }
  24
  25     }
  26  }
```

programme 6 : avance_touch arrière.c

Capteur photosensible (« light sensor »)

```
Auto  const tSensors photoelectrique      = (tSensors) S3;   //sensorLightActive  //*!!!!*//
Auto  const tMotor   moteur_droit         = (tMotor) motorA; //tmotorNxtEncoderOpenLoop //*!!!!*//
Auto  const tMotor   moteur_gauche        = (tMotor) motorC; //tmotorNxtEncoderOpenLoop //*!!!!*//
Auto  //*!!CLICK to edit 'wizard' created sensor & motor configuration.                !!*//
   1
   2  //*********************************************************************************************//
   3  // Le robot suit le cote gauche d'une ligne en fonction des informations donnees par        //
   4  // le capteur de luminosite. Si le niveau de luminosite lu est en deca du seuil, c'est      //
   5  // que le robot est sur une surface noire (ligne). Alors, il tourne vers la gauche jusqu'à  //
   6  // trouver une surface plus claire, à ce moment il retourne sur la droite. En avancant      //
   7  // de gauche a droite le robot realise un mouvement " linéaire " le long de la ligne noire. //
   8  //*********************************************************************************************//
   9  int seuil ;                                        // declarer variable "seuil" en nombre entier
  10
  11  task pingTask
  12    {                                                // definir la tache sons "pingTask"
  13    ClearSounds () ;                                 // eteindre tout son
  14    while (true)
  15      {                                              // tant que la tempo de 1s n'est pas acheve
  16        PlaySound(soundBlip) ;                       // faire le son "Blip" toutes les 1s
  17        wait10Msec (100);                            // tempo 100*10 ms = 1s
  18      }
  19    }
  20  void attente_poussoir_entree()                   // fonction gestion des touches orange (validation)
  21  {                                                // et grise (devalidation)
  22    StartTask (pingTask) ;                         // lancer la tache "pingTask"
  23    while (nNxtButtonPressed != kEnterButton) ;    // tant que le bouton orange n'est pas appuye
  24    while (nNxtButtonPressed != kNoButton) ;       // tant que le bouton orange n'est pas appuye
  25    StopTask (pingTask) ;                          // fin pingTask
  26  }
  27
  28  task main()
  29  {
  30    {
  31    eraseDisplay() ;                                     // effacer tout affichage de l'ecran LCD
  32    nxtDisplayCenteredTextLine(0, "CALIBRAGE") ;         // afficher "CALIBRAGE" centre en ligne 0
  33    nxtDisplayTextLine(1, "- niveau blanc :") ;          // afficher "- niveau blanc" en ligne 1
  34    nxtDisplayCenteredTextLine(7, "APPUYER ENTREE") ;    // afficher "APPUYER ENTREE"
  35    attente_poussoir_entree() ;                          // lancer tache d'attente d'action sur ENTER
  36    int niveau_blanc = SensorValue(photoelectrique) ;    // variable niveau_blanc recoit valeur lue
  37                                                         // par le capteur photoelectrique
  38    nxtDisplayTextLine(1, "- niveau blanc : %d", niveau_blanc) ; // afficher valeur "niveau blanc"
  39
  40    nxtDisplayTextLine(2, "- niveau noir :") ;           // meme procedure pour le niveau noir
  41    attente_poussoir_entree() ;
  42    int niveau_noir = SensorValue(photoelectrique) ;
  43    nxtDisplayTextLine(2, "- niveau noir : %d", niveau_noir) ;
  44
  45    seuil = (niveau_blanc + niveau_noir) / 2 ;           // valeur seuil de declenchement
  46
  47    nxtDisplayCenteredTextLine(4, "Robot OK ?") ;        // Demander si le robot est pret a demarrer
  48    attente_poussoir_entree() ;                          // procedure d'attente de confirmation (son)
  49
  50    nMotorEncoder[motorA]=0; //initialisation de la valeur courante du capteur moteur A
  51    nMotorEncoder[motorC]=0;  //initialisation de la valeur courante du capteur moteur C
  52    nSyncedMotors = synchAC; // Synchroniser mot C par rapport au mot
  53    nSyncedTurnRatio = 100 ;
  54  }
  55    while(true)                                          //boucle infinie
  56
  57    motor[motorA]= 40;
  58     {
  59        if(SensorValue(photoelectrique) < seuil)         // si valeur capteur lue < seuil faire
  60        {
  61          motor[moteur_droit] = 75;                    //moteur A a -75% de puissance
  62          motor[moteur_gauche] = 0;                      //moteur C arrete
  63        }
  64
  65        else                                             // sinon (valeur capteur lue >= seuil) faire
  66        {
  67          motor[moteur_droit] = 0;                       //moteur A arrete
  68          motor[moteur_gauche] = 75;                     //moteur C a -75% de puissance
  69        }
  70     }
  71  }
```

programme 7 : suivi de ligne basique.c

Ce programme plus évolué que les précédents comporte une partie dialogue avec l'utilisateur qui est écrite sous forme de fonction « **attente_poussoir-entree** » car elle est appelée à plusieurs reprises dans le programme principal. Cette fonction appelle elle-même une sous tâche « **pingTask** ».

Détecteur à ultrason (« sonar sensor »)

```
Auto  const tSensors CaptPosition         = (tSensors) S1;   //sensorTouch        //*!!!!*//
Auto  const tSensors CaptPhotoelec        = (tSensors) S2;   //sensorLightActive  //*!!!!*//
Auto  const tSensors CaptUltrasons        = (tSensors) S4;   //sensorSONAR9V      //*!!!!*//
Auto  const tMotor   MoteurDroit          = (tMotor) motorA; //tmotorNxtEncoderClosedLoop //*!!!!*//
Auto  const tMotor   MoteurGauche         = (tMotor) motorC; //tmotorNormal       //*!!!!*//
Auto  //*!!CLICK to edit 'wizard' created sensor & motor configuration.                !!*//
   1
   2  //*******************************************************************************//
   3  //                        Detection d'obstacle par sonar                         //
   4  //  Le robot avance jusqu'a detecter un obstacle. A cet instant, il stoppe       //
   5  //  et affiche la distance de detection.
   6  //*******************************************************************************//
   7
   8  #define detection_dist 20                                  // entrer la distance de detection desiree
   9
  10  task main()
  11  {
  12     do                                      //executer le code entre {} tant que la condition est vraie
  13      {
  14      motor[MoteurDroit] = 75;                //motor A&C en marche avant a 75%
  15      motor[MoteurGauche] = 75;
  16      }
  17     while(SensorValue(CaptUltrasons) > detection_dist);  // tant que la condition est vraie reboucler
  18
  19     motor[MoteurDroit] = motor[MoteurGauche] = 0;      // stopper les moteurs lorsque la condition n'est plus vraie
  20
  21     int dist=0;                                       // la variable entiere dist est mise 0
  22
  23     while(true)                                       // boucle infinie
  24     {
  25        dist = SensorValue(CaptUltrasons);                 //charger lecture sonar dans la variable "dist"
  26        nxtDisplayTextLine(1,"Dist obstacle:%d",dist);   //"Dist. obstacle:" suivi de la valeur "dist"
  27                                                          //est affiche sur la ligne 1 de l'ecran LCD
  28        wait1Msec(1000);                                //tempo 1s
  29        eraseDisplay();                                 //effacer l'ecran entre chaque affichage
  30     }
  31  }
  32
```

programme 8 : detection_obstacle_sonar_fr.c

3. *Écrire les boucles de bases dans un programme*

Boucle While (Tant que … faire)

Ce programme permet à votre robot de se déplacer dans un carré en répétant un mouvement d'avancé et de rotation à 90°. Il répétera ce mouvement 20 fois donc votre robot fait 5 carrés.

Seule l'étude de la boucle nous intéresse dans ce petit programme nous n'avons donc pas réalisé une rotation à 90° très précise. Vous remarquerez que nous avons utilisé une tempo de 750 ms au lieu de réaliser un asservissement de position par codeur.

La commande « **`while`** » permet de tester l'état d'une variable (variable d'un programme, état d'un capteur, événement extérieur, …). Le programme s'exécutera tant que la condition est vraie. Le tableau ci-dessous récapitule les tests disponibles en langage C.

Tests	<	Strictement inférieur	if a < b
	<=	Inférieur ou égal	if a >= b
	>	Strictement supérieur	if a > b
	>=	Supérieur ou égal	if a >= b
	==	Egal	if a ==b (si a est égale à b)
	!=	Différent	if a != b
	&&	ET logique	if ((a=5) && (b=2))
	||	OU logique	if ((a=5) ||(b=2))
	?:	Condition	z=(a>b)?a:b (Si a>b a z=a sinon z=b)

```
1   //*****************************************************************************//
2   //                              Boucle While                                   //
3   //*****************************************************************************//
5
6
7   task main()
8   {
9      int i = 0;                    //variable i de type entier et mise  0
10
11        while(i < 20)              //tant que i<20 executer le programme suivant sinon passer
12        {
13          motor[motorA] = 75;     //moteur A et C en avant a 75%
14          motor[motorC] = 75;
15          wait1Msec(4000);        //tempo de 4s => 4s en marche avant
16
17          motor[motorA] = 75;     //moteur A en avant a 75%
18          motor[motorC] = -75;    //moteur C en arriere a 75%
19          wait1Msec(750);         //tempo 750ms pour realiser une rotation aproximativement de 90 degres
20
21           i++;                   //la variable "i" est incrementee de 1. Le code sera execute 20 fois.
22         }
23  }
```

programme 9 : boucle_while.c

Boucle For (Pour … faire)

Cette boucle est utilisée pour réaliser un nombre d'itération connu. Ici nous voulons que le robot répète 20 fois le même cycle donc cette structure est parfaitement adaptée.

La variable `i` est mise à `0` puis l'action est réalisée, la variable `i` est alors incrémentée de `1`.

Lorque `i = 20` l'exécution du programme s'arrête.

```
1   //*****************************************************************************//
3   //                              Boucle For                                     //
4   //*****************************************************************************//
5
6
7   task main()
8   {
9      int i;                     //la variable "i" est declaree de type entier
10
11     for(i = 0; i< 20; i++)     // boucle FOR, i est mis  0, la condition a tester est i<20 et i
12                                // est incremente a chaque rotation.
13     {
14          motor[motorA] = 75;    //moteur A et C en avant a 75%
14          motor[motorC] = 75;
14          wait1Msec(4000);       //tempo de 4s => 4s en marche avant
14
14          motor[motorA] = 75;    //moteur A en avant a 75%
14          motor[motorC] = -75;   //moteur C en arriere a 75%
14          wait1Msec(750);        //tempo 750ms pour realiser une rotation aproximativement de 90 degres
21     }
22  }
```

programme 10 : boucle_for.c

🗐 Boucle Do … While (faire … tant que)

Cette boucle while a de particulier que le test est placé à la fin : le programme sera donc exécuté au moins une fois.

```
1   //*****************************************************************************//
3   //                              Boucle Do While                                //
4   //*****************************************************************************//
5
6   task main()
7   {
8      int i = 0;                //la variable "i" est declaree de type entier et mise a 0
9
10     do                        // faire (executer) le code entre {}
11     {
12        motor[motorA] = 75;    //moteur A et C en avant a 75%
12        motor[motorC] = 75;
12        wait1Msec(4000);       //tempo de 4s => 4s en marche avant
12
12        motor[motorA] = 75;    //moteur A en avant a 75%
12        motor[motorC] = -75;   //moteur C en arriere a 75%
12        wait1Msec(750);        //tempo 750ms pour realiser une rotation aproximativement de 90 degres
19
20        i++;                   //la variable "i" est incrementee de 1
21     }
22     while(i < 20);            //verification de la condition apres chaque iteration
23
24  }
```

programme 11 : boucle_do_while.c

🗐 Switch (variable) …..case ….. default (faire ça ou ceci ou cela ou par défaut)

L'instruction « **switch** » permet de faire plusieurs tests de valeurs sur le contenu d'une même variable. Ce **branchement conditionnel** simplifie beaucoup le test de plusieurs valeurs d'une variable, car cette opération aurait été compliquée (mais possible) avec des **if imbriqués**. Sa syntaxe est la suivante :

```
switch (Variable) {

case Valeur1 :
        Liste d'instructions;
        break;

case Valeur2 :
        Liste d'instructions;
        break;

case Valeurs... :
        Liste d'instructions;
        break;

default:

        Liste d'instructions;

}
```

Les parenthèses qui suivent le mot clé switch indiquent une expression **dont la valeur est testée successivement par chacun des case.** Lorsque l'expression testée **est égale** à une des valeurs suivant un « **case** », **la liste d'instructions qui suit celui-ci est exécutée**. Le mot clé « **break** » indique la sortie de la structure conditionnelle. Le mot clé « **default** » précède la liste d'instructions qui sera exécutée si l'expression n'est jamais égale à une des valeurs.

N'oubliez pas d'insérer des instructions « **break** » entre chaque test, ce genre d'oubli est difficile à détecter car aucune erreur n'est signalée...

En effet, lorsque l'on omet le « **break** », l'exécution continue dans les blocs suivants !

Cet état de fait peut d'ailleurs être utilisé judicieusement afin de faire exécuter les mêmes instructions pour différentes valeurs consécutives, on peut ainsi mettre plusieurs « **case** » avant le bloc :

```
switch(variable)
{
case 1:
case 2:
{ instructions exécutées pour variable = 1 ou pour variable = 2 }
break;

case 3:
{ instructions exécutées pour variable = 3 uniquement }
break;
default:
{ instructions exécutées pour toute autre valeur de variable }
}
```

```
Auto  const tSensors CaptPosition         = (tSensors) S1;   //sensorTouch        //*!!!!*//
Auto  //*!!CLICK to edit 'wizard' created sensor & motor configuration.                !!*//
   1
   2  //*****************************************************************************************//
   3  // Le robot realise une des 3 actions en fonction de l'etat de la variable de SWITCH (v)    //
   4  //*****************************************************************************************//
   5  int i ; int j ; int k ; int v ; // declaration des variables entieres
   6  {
   7  i=0;          // initialisation de la variable i a incrementer et tester
   8    while (i<4) // boucler tant que i<4 (permet de tester les 4 cas possibles
   9    {
  10      v=0;        // initialisation de la variable v de SWITCH a tester
  11      k=0;        // initialisation de la variable k a incremnter et tester
  12        while (k<3) // boucler 3 fois pour pouvoir affecter 0 ou 1 ou 2 ou 3 a la variable v
  13        {
  14          v = v + SensorValue(CaptPosition) ;   // v = 0 + etat du capteur
  15          wait10Msec (100);           // attendre 1s (tempo 1s)
  16          k++;                        // incrementer k de +1 idem k==k+1
  17        }
  18
  19        switch (v)                    // tester la variable v
  20        {
  21          case 1:                     // v = 1
  22            j=0;
  23            while (j<4)               // repeter 4 fois la sonnerie
  24            {
  25              PlaySound(soundBlip) ; // faire sonner pendant 1s (tempo 1s)
  26              wait10Msec (100);      // tempo 1s
  27              j++;                   // incrementer j de +1 idem j==j+1
  28            }
  29          break;                     // fin du traitement du cas 1
  30
  31          case 2 :
  32            j=0;
  33            while (j<4)
  34            {
  35              PlaySound(soundBeepBeep) ;
  36              wait10Msec (100);
  37              j++;
  38            }
  39          break;                     // fin du traitement du cas 2
  40
  41          case 3 :
  42            j=0;
  43            while (j<4)
  44              {
  45              PlaySound(soundUpwardTones) ;
  46              wait10Msec (100);
  47              j++;
  48              }
  49          break;                 // fin du traitement du cas 3
  50
  51          default:               // traitement fait par defaut (v=0 ou v>3)
  52            PlaySound(soundDownwardTones) ;
  53        }
  54   i++;
  55    }
  56  }
  57
```

Ce programme est à tester en mode pas à pas pour bien voir les différentes boucles « **i, j, k** » et les choix en fonction de la valeur de la variable « **v** ».

Commandes RobotC appliquées au NXT

1. Contrôle des Boutons de la brique NXT par programmation

Il y a quatre boutons sur le boîtier NXT. Les boutons GAUCHE, ENTREZ et DROIT sont sur la rangée supérieure et le bouton de SORTIE est sur la deuxième rangée.

Normalement, quand un programme est en exécution, les boutons GAUCHE et DROIT sont employés pour basculer entre des affichages standards divers. Et le bouton de SORTIE est employé pour arrêter l'exécution du programme utilisateur. Ces caractéristiques peuvent être utilisées pour que les actions des boutons puissent être contrôlées dans une application courante

nNxtButtonTask

La valeur normale de cette variable est -1 ' pour indiquer que le traitement de bouton doit être traité par le microprogramme NXT et pas dans une application utilisateur. Mettre cette variable à une autre valeur permet le contrôle d'applications programmées des boutons NXT.

Quand la valeur correspond à une tâche valide, alors cette tâche est commencée (ou reprise) chaque fois qu'un bouton est appuyé. Si c'est une tâche invalide, alors aucune action n'est prise autre que la permission du contrôle d'application de tous les boutons.

nNxtButtonPressed

Le microprogramme ne peut seulement reconnaître un appui de bouton à la fois. Cette variable contient l'index (0.. 3) du bouton actuellement appuyé ou -1 pour indiquer qu'aucun bouton n'a été appuyé.

nNxtExitClicks

Cette variable contient le nombre d'appui sur le bouton de sortie consécutif qui doivent être reçus avant qu'un programme utilisateur ne soit interrompu. La valeur de défaut est un clic. Si un programme utilisateur veut employer le bouton de sortie, il peut alors entrer dans ce champ un nombre supérieur. Le microprogramme contrôle toujours le bouton de sortie pour l'arrêt de programme, même quand l'application gère le contrôle des boutons; cela empêche un programme incorrect (ou malveillant) de fermer le NXT.

2. Datalog (journal des variables)

Le NXT a une capacité d'enregistrement chronologique des données qui permet de sauver les variables et les valeurs de détecteur dans un journal (« log ») pendant l'exécution de programme. C'est utile pour des données collectées pendant des mesures. C'est aussi utile pour la mise au point de programme car vous pouvez analyser la valeur d'une variable particulière après l'exécution d'un programme a été dirigé. Il y a la place pour le stockage d'environ 10000 points de données.

Les points de données sont stockés dans la RAM. Ils peuvent être par la suite copiés dans un fichier flash de NXT. Ils ne sont pas immédiatement stockés dans un fichier de flash pendant l'exécution de programme parce que l'écriture d'un fichier dans la mémoire flash peut prendre 3 à 6 millisecondes qui pourraient déranger l'exécution de l'application utilisateur.

Il y a des utilités dans le RobotC IDE pour télécharger le datalog et l'afficher dans une fenêtre de PC. Ceux-ci peuvent être trouvés dans « debug display » et dans « NXT -> Files Management ».

`AddToDatalog (data)`

Ajoute un nouveau point de données au « datalog » le contenant de la valeur « data ».

`CreateDatalog (taille)`

Cela créera un « datalog » de la taille indiquée.

`SaveNxtDatalog ()`

Cette fonction sauvera le datalog courant dans un fichier de données au standart NXT. Le fichier de données a le nom « DATAnnnn. RD »" où « nnnn » est un nombre numérique à quatre chiffre. Le microprogramme RobotC parcourra tous les fichiers existants sur le NXT à la recherche du fichier de données de plus grand nombre existant. Il incrémentera alors nombre de un pour créer un nouveau fichier datalog.

`nMaxDataFiles`

Défini le nombre maximal de fichiers datalog (DATAnnnn. RTD) qui peuvent être stockés sur le NXT. En sauvant un nouveau fichier, des fichiers plus vieux seront supprimés pour tenir dans cette limite. Cette variable n'est pas volatile et n'est pas affectée par les séquences de « marche/arrêt ».

`nMaxDataFileSize`

Défini la taille totale des fichiers datalog (DATAnnnn. RTD) qui peuvent être stocké sur le NXT. En sauvant un nouveau fichier, des fichiers plus vieux seront supprimés pour tenir dans cette limite. Cette variable n'est pas volatile et n'est pas affectée par les séquences de « marche/arrêt ».

`bHideDataFiles`

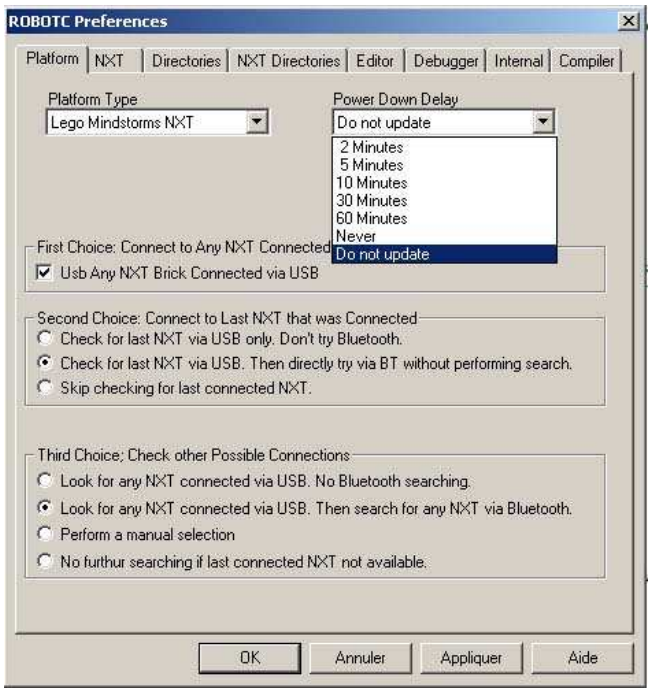
Variable booléenne pour indiquer si les fichiers de données doivent être montrés ou cachés sur le NXT GUI. Par défaut les fichiers de données - i.e fichiers avec l'extension de fichier « rdt » - ne sont pas montrés sur le NXT GUI. Déclarer cette valeur « vraie » (« true ») permettra leur affichage. Cette variable n'est pas volatile et n'est pas affectée par les séquences de « marche/arrêt ».

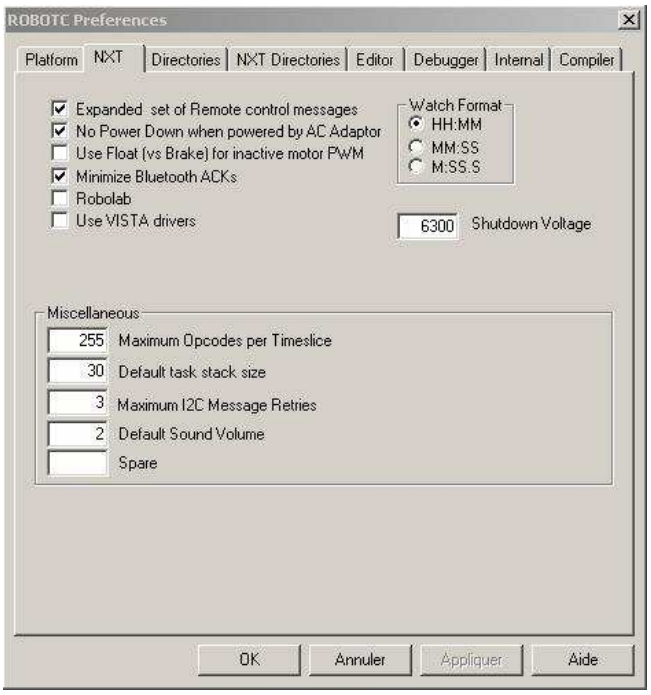
`UploadDatalog (nStart, nSize)`

C'est une fonction interne de RobotC qui envoie un message du PC au NXT pour charger une partie du datalog.

3. Contrôle du niveau de charge de la batterie et de la puissance

Le microprogramme NXT scrute le temps d'inactivité du NXT et éteint la brique si le temps limite est dépassé. La valeur de temps mort d'inactivité peut être configurée via le NXT GUI. Cela peut se faire aussi depuis RobotC en ouvrant **view\preferences**.





A chaque connexion à un NXT RobotC charge la valeur de « temps mort » définie par l'utilisateur (ou celle par défaut).

Un niveau de tension suffisant doit être maintenu pour qu'un circuit électronique fonctionne correctement ; il en va de même pour le NXT. Aussi à chaque connexion RobotC vérifie la tension de la batterie ou pile ; si la tension est trop basse la connexion est interrompue.

`alive()`

Cette fonction remet à 0 le timer de « temps mort ».

Une application en cours qui sollicite périodiquement la fonction «**`alive()`**» empêchera l'extinction du NXT.

`powerOff ()`

Cet appel de fonction éteindra immédiatement le NXT

`bNoPowerDownOnACAdaptor`

Un « drapeau » (« flag ») Booléen qui indique que le NXT ne doit pas être éteint automatiquement lorsqu'il est alimenté par un adaptateur AC.

`bNxtRechargable`

Un drapeau Booléen qui indique si le NXT est alimenté ou non par batterie LEGO rechargeable. Un commutateur caché à l'intérieur du compartiment de la batterie NXT est actionné en présence d'une batterie LEGO. Cette variable lit la valeur de ce commutateur.

`nAvgBatteryLevel`

La moyenne de plusieurs prélèvements du niveau de tension de la batterie. Une valeur de 7500 indique une tension de 7.5 volts.

`nImmediateBatteryLevel`

Le dernier prélèvement du niveau de tension de la batterie.

`nPowerDownDelayMinutes`

Le nombre de minutes d'inactivité qui déclenchera l'extinction automatique du NXT.

`nShutdownVoltage`

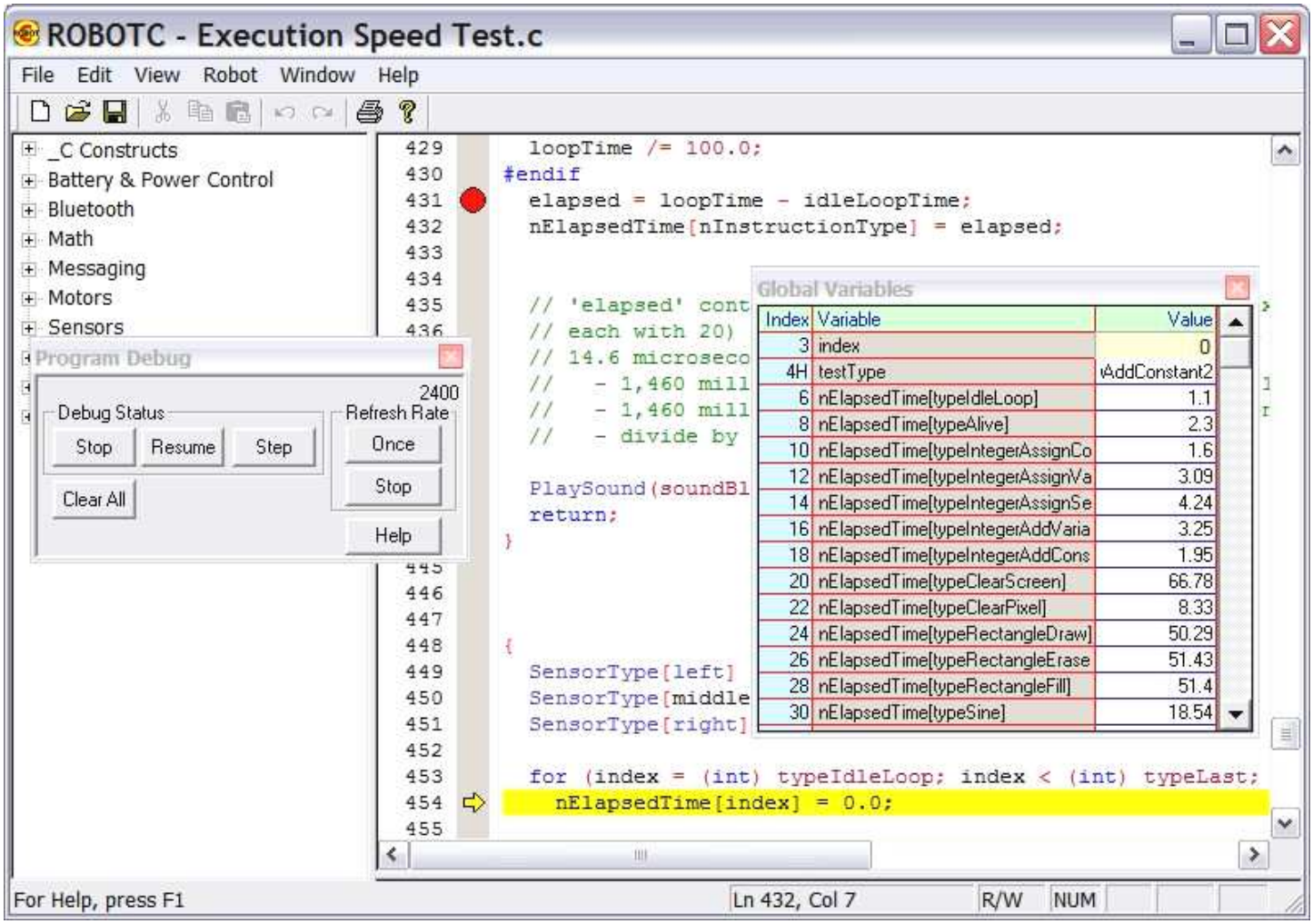
La valeur de tension basse qui déclenchera l'arrêt automatique du NXT. Une valeur de 6300 indique un arrêt à 6.3V.

4. Mise au point du programme ROBOTC

Le « debugger » ou « débogueur » en français (plutôt du FranGlais) de ROBOTC permet l'accès interactif en temps réel au robot lors de l'exécution de votre programme court. Cela peut potentiellement considérablement réduire le temps de recherche et de correction des erreurs dans vos programmes. Avec le programme de mise au point (« debugger ») vous pouvez :

- Commencer et arrêter l'exécution du programme depuis le PC,
- choisir le mode "pas à pas" (« Single step ») pour analyser votre programme ligne par ligne afin d'en contrôler son séquencement et vérifier les valeurs de variables,
- Définir « des points de contrôle » (« breakpoints ») dans votre code qui suspendent l'exécution de programme lorsqu'il les rencontrent,
- Lire et écrire les valeurs de toutes les variables définies dans votre programme
- Lire et écrire les valeurs des moteurs et des capteurs.

L'image suivante montre les fenêtres ROBOTC ouvertes pendant une session de mise au point interactive.



Note : Pour réduire au minimum la taille de l'image, les deux fenêtres de programme de mise au point dans la ont été placées sur le sommet de la fenêtre principale ROBOTC. Ce n'est pas une obligation. Elles peuvent être placées n'importe où sur le bureau de PC.

Quatre moyens de « déboguage » sont actifs dans cette image. Chacun d'entre ceux-ci est mis en évidence dans l'image suivante.

ROBOTC - Execution Speed Test.c

File Edit View Robot Window Help

User Defined Breakpoint

Variable Display

Debugger Controls

Current Execution Point

Program Debug

Debug Status: Stop, Resume, Step, Clear All

Refresh Rate: 2400, Once, Stop, Help

Global Variables

Index	Variable	Value
3	index	0
4H	testType	vAddConstant2
6	nElapsedTime[typeIdleLoop]	1.1
8	nElapsedTime[typeAlive]	2.3
10	nElapsedTime[typeIntegerAssignCo	1.6
12	nElapsedTime[typeIntegerAssignVa	3.09
14	nElapsedTime[typeIntegerAssignSe	4.24
16	nElapsedTime[typeIntegerAddVaria	3.25
18	nElapsedTime[typeIntegerAddCons	1.95
20	nElapsedTime[typeClearScreen]	66.78
22	nElapsedTime[typeClearPixel]	8.33
24	nElapsedTime[typeRectangleDraw]	50.29
26	nElapsedTime[typeRectangleErase	51.43
28	nElapsedTime[typeRectangleFill]	51.4
30	nElapsedTime[typeSine]	18.54

For Help, press F1 — Ln 432, Col 7 — R/W — NUM

	Description
Debugger Controls **Contrôles du déboguage**	C'est la fenêtre principale du débogueur permettant le contrôle de l'exécution du programme à tester.
Current Execution Point **Point d'exécution en cours**	La flèche jaune pointe la prochaine ligne de code à être exécutée dans le programme. La couleur jaune indique que l'exécution de programme est actuellement suspendue. La couleur verte est employée quand le programme est en séquencement automatique.
User Defined Breakpoint **Point de contrôle défini par le programmeur**	Le point rouge indique un « point de pause » (« Breakpoint ») défini par le programmeur. L'exécution de programme sera suspendue si cette ligne de code est atteinte pendant l'exécution de programme. Les points de contrôle sont facilement positionnables par un « clique-gauche » avec la souris dans la colonne grise à gauche du code à contrôler.
Variable Display **Affichage des variables**	C'est un affichage de toutes les variables définies dans votre programme. Les valeurs sont périodiquement rafraîchies. Si vous le désirez vous pouvez choisir une variable et récrire sa valeur.

Contrôles du déboguage

Les boutons sur cette fenêtre permettent :

- d'arrêter le l'exécution du programme (« stop »),
- de suspendre et de reprendre (« suspend » puis « resume ») l'exécution du programme,
- de contrôler le taux de rafraîchissement de l'information provenant du robot,

Les noms de boutons sont changeants car liés à l'état de l'exécution de programme. Par exemple, avant l'exécution du programme c'est « Start » qui est affiché puis « Stop » apparaît lorsque le programme se déroule.

Chacun des boutons est décrit dans la table suivante.

Bouton	Action réalisée
Start / Stop	Démarrer ou arrêter l'exécution du programme
Suspend / Resume	Suspendre (arrêt momentané) et de reprendre l'exécution du programme,
Step	Commande manuelle pour exécuter la ligne immédiatement suivante d'un programme qui a été suspendu. Note: Le lignes de codes « compliquées », particulièrement celles qui comportent des cassures dans le déroulement linéaire du programme comme "for" ou "while",ne seront exécutées que partiellement par la commande step.
Once	Déclencher un « seul » rafraîchissement des fenêtres du débogueur. Le rafraîchissement continu des fenêtres du débogueur est stoppé.
Continuous / Stop	Lancer ou arrêter le rafraîchissement continu des fenêtres du débogueur.

Point de contrôle défini par le programmeur

Les « Break points » sont des points de pause inséré dans votre code source, ayant pour fonction de stopper l'exécution du programme pour permettre une intervention manuelle par exemple.

```
430  #endif
431    elapsed = loopTime - idleLoopTime;
432    nElapsedTime[nInstructionType] = elapsed;
433
```

Les points de contrôle n'ont pas d'impact sur la vitesse d'exécution du programme. Cette fonction est intégrée dans le microprogramme ROBOTC et n'exige pas d'insertion d'instructions supplémentaires dans votre code compilé.

Vous pouvez définir un nombre pratiquement illimité de points de contrôle dans votre programme. Il n'y a aucune restriction du nombre de points de contrôle dans une fonction simple ou une tâche (« task »).

Changement (forcé manuellement) du déroulement du programme

On peut vouloir changer le flux normal d'exécution de programme. En cliquant avec le bouton droit de la souris (voir ci-dessous) un menu contextuel apparaît. Une des commandes est « Set Next Instruction » qui permet d'aller du point d'exécution actuel du programme à la ligne choisie.

```
430  #endif
431    elapsed = loopTime - idleLoopTime;
432                                  actionType] = elapsed;
433
434
435                                  s the number of 10 msec 'ticks' to execute 100,000
436                                  atements. Thus if 'elapsed' is 146, a single statemen
437                                  :
438                                  econs total 'adjusted' time in loop (146 ticks x 10 m
439                                  econds is same as 1,460,000 microseconds
440    //    - divide by 100,000 to get 14.6 microseconds
441
442    PlaySound(soundBlip);
```

Remove Breakpoint
Remove all breakpoints
Toggle bookmark
Set Next Instruction
Go to Disassembly
Go to Listing

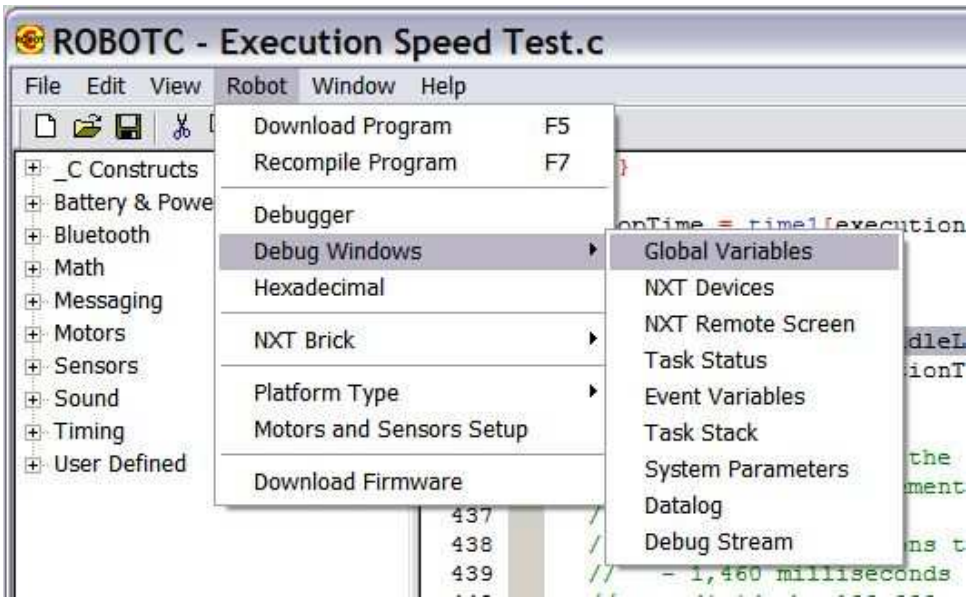
Les fenêtres du débogueur

Il y a plusieurs fenêtres qui peuvent être employées en mode déboguage. Ces fenêtres donnent accès au programme utilisateur et aux variables incorporées. Certaines des fenêtres sont rarement employées, mais peuvent être parfois utiles.

This picture shows the Debugger Windows available for the NXT platform in ROBOTC. Other robot platforms may have a different list of windows customized to the robot controller hardware.

L'image page suivante montre les fenêtres du débogueur disponibles pour la plate-forme NXT dans ROBOTC.

Le nombre de fenêtres disponibles dépend du niveau de menu choisi (« Basic » ou « Expert »). En mode « Basic » les fonctions « avancées » ne sont pas visibles.



Global Variables

Index	Variable	Value
3	index	0
4H	testType	AddConstant2
6	nElapsedTime[typeIdleLoop]	1.1
8	nElapsedTime[typeAlive]	2.3
10	nElapsedTime[typeIntegerAssignCo	1.6
12	nElapsedTime[typeIntegerAssignVa	3.09
14	nElapsedTime[typeIntegerAssignSe	4.24
16	nElapsedTime[typeIntegerAddVaria	3.25
18	nElapsedTime[typeIntegerAddCons	1.95
20	nElapsedTime[typeClearScreen]	66.78
22	nElapsedTime[typeClearPixel]	8.33
24	nElapsedTime[typeRectangleDraw]	50.29
26	nElapsedTime[typeRectangleErase	51.43
28	nElapsedTime[typeRectangleFill]	51.4
30	nElapsedTime[typeSine]	18.54

Cette fenêtre contient un tableau de toutes les variables déclarées dans votre programme, avec leurs valeurs actuelles.

Si besoin, vous pouvez choisir et changer la valeur de n'importe quelle variable.

System Parameters

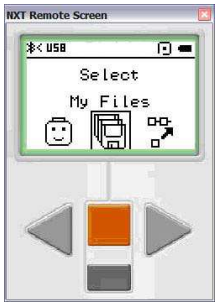
Index	Variable	Value
0	nClockMinutes	910
1	nExceptionReports	0
2	version	7.23
3	nSysTime	39:10:36.102
4	nPgmTime	14.028 sec
5	avgBackgroundTime	13%
6	avgInterpreterTime	86%
7	nShutdownVoltage	6.30 V
8	bNoPowerDownOnACAdaptor	true
9	bNxtRechargable	true
10	nPowerDownDelayMinutes	10
11	nAvgBatteryLevel	8.21 V
12	nImmediateBatteryLevel	8.20 V
13	bRobolab	false
14	nOpcodesPerTimeslice	255
15	nDebugTaskMode	0
16	bFloatDuringInactiveMotorPW	false
17	nVirtualMotorChanges	0

ROBOTC a plusieurs variables incorporées qui caractérisent les performances de votre robot. Les « Paramètres Système" (« System Parameters ») fournissent l'accès à ces variables.

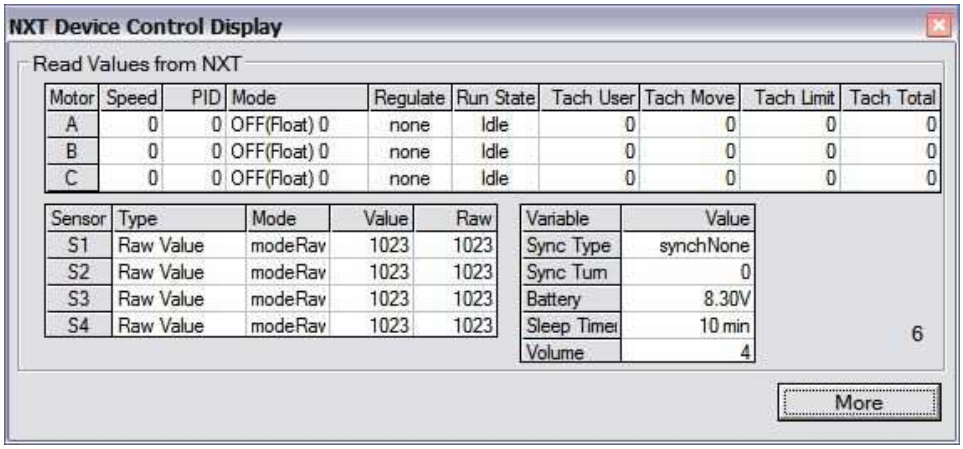
Bien sûr, toutes ces variables sont directement accessibles dans votre code de programme.

L'utilisateur avancé peut y trouver quelques champs particulièrement intéressants. Par exemple :

- 'AvgBackgroundTime' est le temps utilisé par le gestionnaire de code. Le temps restant est disponible pour l'exécution de programme d'utilisateur. Dans cet exemple le « CPU » (« Control Process Unit », « processeur ») consomme 13 % du temps pour la gestion du code. Cette donnée doit être prise en compte lorsqu'on développe de gros programme. Il se peut que le CPU n'est pas le temps de « passer » tout le programme dans le temps imparti (les 87% restant).
- 'BNoPowerDownOnACAdaptor' est une variable propre pour empêcher l'extinction automatique du NXT si sa batterie LEGO est reliée au secteur (par un transformateur).



Cette fenêtre ne fonctionne pas parfaitement dans la version Robot C 1.10. Les touches ne sont pas visibles sur l'écran du PC mais sont actives ; il faut cliquer avec la souris là où elles devraient se trouver.



C'est la fenêtre « devices » (moteurs et capteurs) du débogueur qui permet l'accès aux valeurs en cours des moteurs et des détecteurs.

Le bouton "More" de bouton développe la fenêtre pour fournir les commandes permettant d'écrire la configuration initiale des capteurs et des moteurs.

Il n'y a aucune restriction sur le nombre des fenêtres de débogueur qui peuvent être simultanément ouvertes. Cependant il faut savoir que chaque fenêtre oblige le PC à communiquer avec le contrôleur de robot pour rafraîchir ses données. En multipliant le nombre de fenêtres on sollicite d'autant le CPU.

Compilation : les modes « Debug » (par défaut) et «Release » (avancée)

ROBOTC has a conditional compile and optimization capability similar to that found in many of the professional large-system IDEs. Usually these systems have at least two default settings with the capabilities shown in the following table.

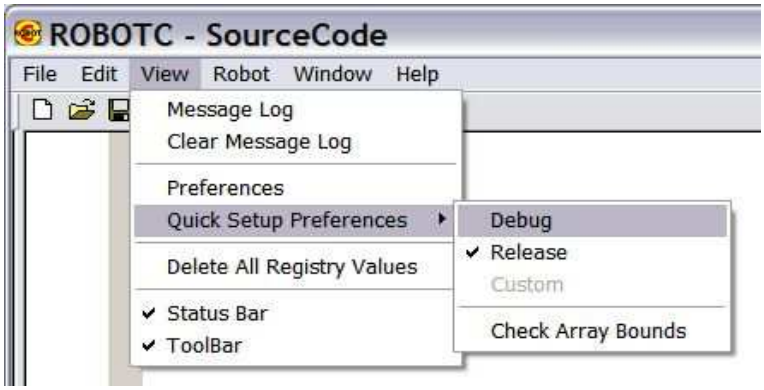
ROBOTC un mode de compilation et d'optimisation similaire aux gros systèmes professionnel. Habituellement ces systèmes possèdent deux configurations par défaut comme détaillé dans le tableau suivant :

Mode	Effet

Debug	• Met hors de service la plupart des fonctions d'optimisation de code de compilateur
	• Permet Automatiquement au compilateur de produire le code pour détecter et créer des exceptions pour l'accès aux tableaux pendant l'exécution de programme.
	• Pour les utilisateurs expérimentés :
	  ○ Valide la macro « ASSERT » qui produira seulement le code pendant une compilation en déboguage. La macro valide que le paramètre que vous avez passé est une valeur «vraie » (c'est-à-dire différent de zéro).
	  ○ Définit la macro variable de préprocesseur "_DEBUG" que vous pouvez employer dans votre propre compilation conditionnelle.

Release	• Permet toutes les techniques d'optimisation de code de compilateur. Cela inclut un « code re-ordering » : le compilateur peut « réordonner » les instructions afin de réduire la quantité de code et/ou le faire exécuter plus rapidement. Le code « Réordonné » peut être difficile à mettre au point parce que le débogueur n'est pas souvent capable de dresser la carte d'instructions du code source qui a produit l'instruction de bas niveau! Le compilateur ROBOTC réduit typiquement la taille de code produit de 10 à 30 % grâce aux techniques d'optimisation de compilateur. L'exécution du programme a tendance à être accélérée de 5 à 15 %.

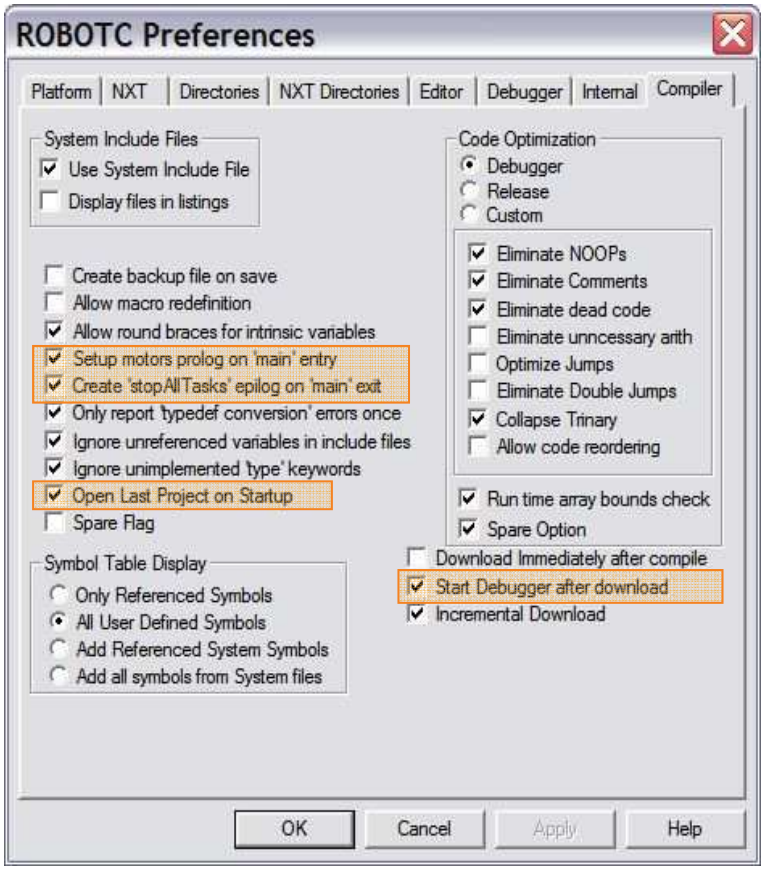
La configuration par défaut ROBOTC est la configuration « déboguage ».



Vous pouvez facilement changer la configuration en cliquant sur « Quick Setup Preferences ».

La sélection de « Debug » permettra automatiquement la génération de code qui vérifiera que les index de tableau sont dans une plage valide.

Vous pouvez aussi changer la configuration en cliquant sur l'index « Compiler » « RobotC Preferences". Cette possibilité n'est offerte que dans le niveau d'utilisateur « Expert ».



Les utilisateurs avertis peuvent être intéressés par l'exploration des techniques d'optimisation des divers compilateurs. En « Expert » on peut voir un index dans « ROBOTC Préférences » qui permet le contrôle individuel des méthodes d'optimisation de compilation.

L'image à gauche montre cet index. Voir « code optimization » à droite de cette fenêtre.

Cette copie d'écran illustre aussi plusieurs options utiles ROBOTC comme :

- « Ouverture automatique » du dernier projet quand ROBOTC est lancé,
- Produire automatiquement le code « prologue » et « epilog » qui, au commencement et la fin de l'exécution du programme, gère la configuration des moteurs et des capteurs,
- Lancement automatique du débogueur après chaque téléchargement de programme,
- Etc.

Les valeurs de défaut sont les options les plus classiques. La plupart des utilisateurs n'auront pas besoin de modifier ces valeurs.

Techniques de déboguage

Mettre au point un programme - découvrir les erreurs et les corriger - peut être long sans débogueur en temps réel. Sans débogueur vous pouvez recourir aux techniques suivantes :

S'il n'y a aucune façon de déterminer si votre programme exécute le bon séquencement alors :

- ajoutez du code pour émettre des sons différents en fin des différentes parties votre programme. Si les sons sont émis c'est que le programme « est passé par là ».
- ajoutez du code pour afficher des messages soit sur l'écran du PC soit sur l'écran LCD du NXT. De même que pour le son vous pourrez contrôler le déroulement de votre programme en fonction des messages affichés.

Ces deux techniques sont disponibles dans ROBOTC. Cependant, un débogueur en temps réel permet la mise au point du programme sans modifier le code source.

RobotC possède des fonctions de déboguage, en temps réel, très puissantes.

`debugPrint (format, parm1, parm2`

`debugPrintLine (format, parm1, parm2)`

Ce sont deux fonctions puissantes que vous pouvez employer dans votre programme pour afficher un message lors de l'exécution de votre programme sur une fenêtre du PC. La première fonction formate une chaîne de caractères en définissant le format et les paramètres. Les caractères formatés sont ajoutés au flux du fichier de débogage (« debug stream file »). La deuxième fonction est la même que la première sauf qu'elle crée un nouveau texte dans le flux du fichier de débogage.

Le flux de débogage est mise en oeuvre comme une grande mémoire tampon (« buffer ») de caractères dans le microprogramme NXT. Chaque fois que la fonction « debugPrint » est appelée les caractères appropriés sont ajoutés à la fin de la mémoire.

Si le débogueur de RobotC est en mode interrogation (« polling ») sur le PC et que la fenêtre du flux de déboguage est ouverte, RobotC interrogera continuellement le microprogramme NXT pour récupérer n'importe quels caractères existants en début de buffer et les ajouter à la fin du texte de la fenêtre du flux de débogage.

Au final n'importe lequel des caractères que vous « imprimez » (« print ») ou « écrivez » (« write ») au flux de débogage apparaît dans une fenêtre sur l'écran de PC.

Si une application est en exécution et écrit du texte dans le flux de débogage sans que le débogueur soit actif alors la somme des données dépassera la capacité mémoire du buffer et les premières données seront écrasées.

Une fois le programme mis au point il est possible d'ignorer les commandes « debugPrint » intégrées dans le code. Pour cela il faut sélectionner le mode « Release » dans le compilateur : toutes les commandes de mise au point seront ignorées par le compilateur.

5. Afficheur à cristaux liquides (LCD)

Le NXT est équipé d'un afficheur 100 x 64 pixels. Le coin supérieur gauche est le point (0, 0) et le coin inférieur droit est le point (99, 63). RobotC permet l'affichage de textes et de dessins sur le LCD.

nViewStateNXT

Quatre écrans différents prédéterminés peuvent être affichés. Ils peuvent être alternativement activés via les boutons gauche et droit. Les affichages possibles sont :

- l'affichage classique du NXT,
- un affichage des capteurs,
- un affichage des moteurs,
- un affichage graphique des derniers points du datalog et des données utilisateur.

L'affichage utilisateur est aussi activé chaque fois qu'une des fonctions suivantes est appelée.

eraseDisplay ()

Efface complètement le LCD.

nxtDisplayString (nLineNumber, sFormatString, parm1, parm2)

Formate une chaîne de caractères selon le format indiqué dans **sFormatString** en utilisant les paramètres parm1 et parm2. Affiche le résultat sur la ligne de texte **nLineNumber**. Le reste de la ligne n'est pas changé; ainsi, si le résultat fait 8 caractères alors les seuls huit premiers caractères de la ligne de texte sont mis à jour.
Notez que parm1 et parm2 sont des paramètres facultatifs; la valeur zéro sera substituée s'ils ne sont pas présents.

There are eight text lines numbers 0 to 7. 0 is the top line and 7 is the bottom line of the display.

Il y a huit lignes de texte numérotées de 0 à 7. La ligne supérieure a pour numéro le 0 et la ligne inférieure a le 7.

nxtDisplayClearTextLine (nLineNumber)

Nettoie (efface, remplit de blancs) la ligne de texte **nLineNumber**.

nxtDisplayTextLine (nLineNumber, sFormatString, parm1, parm2)

Même action **nxtDisplayString** à ceci près que la ligne de texte complète est remplacée. La ligne est terminée par des caractères blancs à la fin.

Exemple :

```
NxtDisplayTextLine (3, "Sensor is %03d", SensorValue [S1);
```

Aboutiraient à l'affichage « Sensor is 038 » sur la ligne de texte 3. « %03d » dit de prendre le paramètre entier et de le formater en une valeur décimale à 3 caractères.

nxtDisplayCenteredTextLine (nLineNumber, sFormatString, parm1, parm2)

Comme **nxtDisplayTextLine** sauf que le texte est centré sur la ligne au lieu de gauche-justifié.

nxtDisplayStringAt (xPos, yPos, sFormatString, parm1, parm2)

Formate une chaîne de caractères et l'affiche en commençant du point **(xPos, yPos)** sur l'écran.

nxtDisplayBigStringAt (xPos, yPos, sFormatString, parm1, parm2)

Semblable **à nxtDisplayStringAt** mais en grande police de 16 pixels.

nxtScrollText (sFormatString, parm1, parm2)

Affiche l'image LCD sur une ligne. Formate alors une chaîne de caractères et l'affiche sur la ligne inférieure de l'écran LCD.

nxtClearPixel (xPos, yPos)
nxtSetPixel (xPos, yPos)

Nettoie le pixel ou le met au point **(xPos, yPos)**

nxtDrawCircle (Left, Top, Diameter)

Dessine un cercle au point **(Gauche, Haut)** avec **le Diamètre** indiqué.

nxtDrawLine (xPos, yPos, xPosTo, yPosTo)

Dessine une ligne entre les points **(xPos, yPos)** et **(xPosTp, yPosTo)**.

nxtDrawRect(Left, Top, Right, Bottom)
nxtEraseRect(Left, Top, Right, Bottom)
nxtFillRect(Left, Top, Right, Bottom)

Agit sur un rectangle défini par la position **(Gauche, Haut, Droite, Bas)** : le dessine, le remplit ou bien l'efface selon la fonction appelée.

nxtDrawEllipse(Left, Top, Right, Bottom)
nxtEraseEllipse (Left, Top, Right, Bottom)
nxtFillEllipse(Left, Top, Right, Bottom)

De même que pour le rectangle mais pour l'ellipse cette fois.

nxtDisplayIconFile (xPos, yPos, sFileName)

Affiche un fichier d'icône au point choisi **(xPos, yPos)**.

Note : cette commande n'est pas actuellement mise en oeuvre. Elle sera disponible dans la prochaine version de RobotC.

Formats d'affichage des variables

Les spécifications du format commencent toujours par un signe « pour cent, (**%**) » et sont lus de gauche à droite. Le format se compose de champs facultatifs et requis, suivant la formule suivante :

flags] [width] [.precision] type indicateurs] [largeur] **[**.precision**]** type

Des champs facultatifs sont affichés avec '[nom de champ]. Les crochets ne font pas partie de l'écriture mais indiquent simplement que le champs est facultatif. Il y a plus de 20 « types » et leurs spécifications détaillées peuvent être trouvées en « C » ou « C++ »

Format des variables de nombres entiers

Le tableau ci-dessous présente un échantillon de plusieurs formats pour un nombre entier qui a pour type de type de champ « d ».

%d	Les sous-programmes de format calculeront la taille de l'élément formaté.
%6d	L'élément sera composé de 6 caractères. Aligné à droite et complété avec des blancs du côté gauche. Les 6 caractères incluent le signe moins « - » pour des nombres négatifs. Le signe plus « + » ne sera pas affiché pour les nombres positifs.
%06d	Même format que ci-dessus mais complété avec des zéros du côté gauche.
%+6d	Même format que « %6d » à ceci près qu'un caractère de signe « + » ou « - » sera toujours inclus.

Format de variables réelles à virgule flottante

De type de champ « f ». Le nombre est affiché dans le format [–]dddd**.**dddd, où dddd correspond à un ou plusieurs chiffres décimaux.

%f	Les sous-programmes de format calculeront la taille de l'élément formaté. Le nombre de chiffres avant la virgule dépend de la grandeur du nombre, et le nombre de chiffres après que la virgule décimale dépend de la précision demandée.
%6f	L'élément sera composé de 6 caractères. Aligné à droite et complété avec des blancs du côté gauche.
%6.3f	L'élément sera composé de 6 caractères. Il aura trois chiffres significatifs après la virgule.

Format des variables de chaîne de caractères

%10c	Composé de 10 caractères. Complété avec des blancs du côté droit.
%-10c	Composé de 10 caractères. Complété avec des blancs du côté gauche

Chaque champ est un caractère unique ou un nombre signifiant une option de format particulier. La spécification du format la plus simple contient seulement le signe « pour cent » et un type le caractère (par exemple, %s). Si un signe « pour cent » est suivi d'un caractère qui ne correspond a aucun format de champ, le caractère est copié dans **stdout**. Par exemple, on peut écrire « **%%** » pour afficher le caractère « **%** ».

6. Fonctions mathématiques

sin(fRadians)
cos(fRadians)

Retourne le sinus ou le cosinus de la donnée à traiter. La donnée est un nombre réel (« float ») exprimé en radians.

asin(Sine)

Retourne l'arcsinus.

acos(Cos)

Retourne l'arccosinus.

atan(Tangent)

Retourne l'arc-tangente.

PI

Variable flottante constante contenant la valeur du nombre PI (π).

sinDegrees(degrees)
cosDegrees(degrees)

Retourne le sinus ou le cosinus de la donnée à traiter. La donnée est un nombre réel (« float ») exprimé en degrées.

radiansToDegrees(radians)
degreesToRadians(degrees)

Convertit les degrés en radians et inversement. Les valeurs converties en angles de 0 à 359° pour les degrés et 0 à 2π pour les radians.

abs(input)
exp(input)
sgn(input)
sqrt(input)

Applique la fonction mathématique spécifiée sur la variable d'entrée

srand(seed)

Définit le nombre de départ pour le générateur de nombre aléatoire.

random(range)

Retourne un valeurs aléatoire entière comprise dans l'intervalle [0..32767]

7. Fonctions d'accès aux fichiers

Le NXT contient un système capable de gérer les fichiers en 64 bits. Les noms de fichier peuvent contenir jusqu'à 15 caractères suivis d'un point et d'une extension à 3 caractères.

Le système de fichiers NXT supporte un maximum de 16 fichiers ouverts ou en recherche. Ceci inclut :

- les programmes d'utilisateur,
- les transferts de fichier sur l'USB ou la liaison de BT,
- les fichiers sons et dessins (c'est-à-dire jouer un son, dessiner un fichier d'icône par exemple).
-

NOTEZ : les routines d'entrée-sortie de fichier NXT exigent qu'un un fichier soit complètement chargé (ou écrit, ce qui paraît évident). Si vous créez un fichier de 200 octets et qu'ensuite seulement 100 octets sont écrits dans le fichier alors vous obtiendrez une erreur et le fichier sera supprimé !

Il y a une définition « enum » qui définit les codes d'erreur possibles des fonctions de fichier.

`OpenRead (hFileHandle, nIoResult, nFileSize, sFileName);`

Ouvre **`sFileName`** pour la lecture. **`HFileHandle`** permet la lecture du fichier. **`NFileSize`** est la taille fichier. **`NIoResult`** est le « non-zéro » quand une erreur se produit.

`SFileName` doit être un nom de fichier NXT valide. Le fichier doit déjà exister sur le NXT pour que cette fonction réussisse.

`OpenWrite (hFileHandle, nIoResult, nFileSize, sFileName);`

Ouvre **`sFileName`** pour l'écriture avec la taille indiquée **`de nFileSize. HFileHandle`** permet l'écriture dans le fichier. **`NIoResult`** est le « non-zéro » quand une erreur se produit.

`SFileName` doit être un nom de fichier NXT valide. Le fichier doit déjà exister sur le NXT pour que cette fonction réussisse.

`Close(hFileHandle, nIoResult);`

Ferme le fichier indiqué ou la poignée de recherche **`(hFileHandle)`**. La fermeture doit être la dernière opération d'entrée/sortie de fichier après que les opérations de lecture/écriture sont terminées. **`NIoResult`** est le « non-zéro » quand une erreur se produit.

`FindFirstFile (hFileHandle, nIoResult, sSearch, sFileName, nFilesize);`

Cette fonction est employée pour la recherche (itérative) dans la liste de fichiers sur le NXT. **`nIoResult`** est le non-zéro si aucun fichier n'existe dans le système de fichiers NXT.. '**`sSearch`** définit des paramètres de recherche (conditionnelle ou pas). Par exemple :

« *. * » recherche des fichiers de toute extension

« *.rso » recherche tous les fichiers ayant l'extension « rso » qui correspond au fichiers son

`SFileName` est le nom du premier fichier trouvé et **`nFilesize`** est la taille du fichier.

`HFileHandle` permet de tenir trace de la recherche. Vous devez utiliser « **`Close`** » quand la recherche est terminée pour fermer le pointeur (« handle ») et permettre une nouvelle recherche.

`FindNextFile (hFileHandle, nIoResult, sFileName, nFilesize);`

Cette fonction continue une recherche amorcée par la fonction **`de "FindFirstFile"`**. Les paramètres ont la même signification que dans **`FindFirstFile`**.

`CloseAllHandles (nIoResult);`

Fonction interne qui ne doit pas être employée dans l'application utilisateur. Ferme tous les pointeurs ouverts par programme. **`NIoResult`** est le non-zéro si une erreur se produit. Cette fonction est employée par le microprogramme NXT à la fin d'exécution de programme pour s'assurer que tous les fichiers ouverts par une application sont fermés.

`Delete(sFileName, nIoResult);`

Supprime le nom de fichier indiqué du NXT. **`NIoResult`** est le non-zéro si une erreur se produit.

Rename(sFileName, nIoResult, sOriginalFileName);

Renommer un fichier. **NIoResult** est le non-zéro si une erreur se produit.

ReadByte (hFileHandle, nIoResult, nParm);
ReadShort (hFileHandle, nIoResult, nParm);
ReadFloat (hFileHandle, nIoResult, fParm);
ReadLong (hFileHandle, nIoResult, nParm);

Lit une variable numérique dans le format approprié : octet d'entiers (« integer ») ou caractère (à 1 octet, 8bits), mot entier « court » (2 octets, 16 bits), mot entier long (4 octets) ou à virgule (« float ») (4 octets) du fichier ouvert représenté par ' **hFileHandle**

WriteByte (hFileHandle, nIoResult, nParm);
WriteShort (hFileHandle, nIoResult, nParm);
WriteLong (hFileHandle, nIoResult, nParm);
WriteFloat (hFileHandle, nIoResult, fParm);

Écrit une variable numérique dans le format approprié : octet d'entiers (« integer ») ou caractère (à 1 octet, 8bits), mot entier « court » (2 octets, 16 bits), mot entier long (4 octets) ou à virgule (« float ») (4 octets) du fichier ouvert représenté par ' **hFileHandle**

WriteString (hFileHandle, nIoResult, sParm);

Écrit une chaîne dans le fichier indiqué. **NIoResult** est le non-zéro si une erreur se produit. S**Parm** est la chaîne à écrire.
Un caractère de fin « zero-termination » est inséré à la fin de la chaîne. Voir aussi "**WriteText**"

WriteText (hFileHandle, nIoResult, sParm);

Écrit une chaîne dans le fichier indiqué. **NIoResult** est le non-zéro si une erreur se produit. S**Parm** est la chaîne à écrire.
Employez cette routine pour écrire un fichier texte. Un caractère de fin « zero-termination » est inséré à la fin de la chaîne. Voir aussi "**WriteString**"

nAvailFlash

Indique la quantité de mémoire flash qui est actuellement inutilisé et donc disponible pour le stockage de fichier. Les unités sont 1/10 de 1Ko (c'est-à-dire 100 octets).

8. Commande des moteurs

Il y a trois moteurs sur le NXT repérés A, B et C. Chaque moteur est équipé d'un encodeur intégré qui permet de déterminer la position angulaire du moteur. Les encodeurs délivrent 360 impulsions pour une révolution simple du moteur : pour 180 impulsions le moteur a donc effectué la moitié d'une rotation.

Motor []

Permet d'affecter une valeur de -100 à 100 à un moteur (A, B ou C). Le signe – indique une rotation dans le sens trigonométrique, dans le sens inverse lorsqu'il n'y a pas de signe. La puissance demandée au moteur peut varier de 0 à 100% de la pleine puissance. En pratique l'effet constaté est une variation de vitesse de 0 à 100%.

Cette commande est dite en « boucle ouverte », c'est-à-dire que l'on envoie un ordre au moteur sans avoir de retour par l'encodeur. Les moteurs n'étant pas contrôlés la vitesse variera en fonction de la charge de la batterie et d'autres variations de vitesse pourront apparaître entre les moteurs qui ne peuvent être mécaniquement absolument identiques. Le robot risque fort de « tirer à droite ou à gauche », il serait étonnant qu'il suivre une trajectoire parfaitement rectiligne.

NMotorPIDSpeedCtrl []

Modes de régulation de vitesse d'un moteur au nombre de trois : (non, speed, dual sync motor) pas de régulation, régulation de vitesse, régulation des moteurs synchronisés.

La régulation permet de contrôler précisément la vitesse de rotation d'un moteur en utilisant les informations transmises par l'encodeur couplé au moteur. C'est une régulation dite en boucle fermée, c'est-à-dire que le calculateur ajuste à intervalles réguliers, toutes les 25 ms par défaut, la commande de vitesse du moteur en fonction des informations transmises par le codeur.

Exemple: `nMotorPIDSpeedCtrl[motorC] = nMotorPIDSpeedCtrl[motorA] = mtrSpeedReg ;`

`nPidUpdateInterval = 20 ;` **mtrSpeedReg** indique que la régulation est **active** pour les moteurs A et B. Le contrôle de la vitesse se fera toutes les 20 ms (nPidUpdateInterval = 20).

`nMotorPIDSpeedCtrl[motGate] = mtrNoReg ;` **mtrNoReg** indique que la régulation est **inactive**

NSyncedMotors

Indique la paire de moteurs à synchroniser (maître/esclave). Le moteur déclaré en second est esclave du premier : il suivra la vitesse et le mouvement du premier. Ceci permet une course « rectiligne » du robot si le châssis est correct, puisque les moteurs tournent exactement à la même vitesse.

Exemple: `nSyncedMotors = synchAC;` Le moteur C est esclave du moteur A. Il suffit alors d'écrire la commande concernant le seul moteur A, le moteur C suivra sans déclaration.

`NSyncedMotors = synchNone;` aucune synchronisation du moteur

NSyncedTurnRatio

Indique le mouvement pour le moteur esclave lors de la synchronisation d'une paire de moteurs.

Exemple:

`nSyncedTurnRatio = 100 ;`	marche avant des 2 moteurs
`nSyncedTurnRatio = 0 ;`	arrêt de la roue C, A étant en mouvement (rotation autour de la roue C)
`nSyncedTurnRatio = -100 ;`	marche inversée de la roue C en fonction de A (rotation sur place)

NMotorEncoder []

Lecture de la position en cours de l'encodeur du moteur.

C'est une variable 16 bits. La valeur maximale avant débordement de capacité est 32767 qui correspond à environ 95 rotations de moteur. Pour les longues courses, il faudra périodiquement réinitialiser (reset) la valeur dans l'application en cours pour éviter les débordements.

NMotorEncoderTarget []

Permet d'indiquer la consigne de positionnement : autrement dit la position que le moteur doit atteindre pour répondre à la commande. Ce code permet de faire un asservissement de position précis en boucle fermée : le moteur tournera tant que la position réelle ne sera pas égale à la position désirée (consigne).

Exemple: `nMotorEncoderTarget[motorA] = cod_roue;` marche avant à vitesse définie tant que la valeur de consigne de l'encodeur « cod_roue » n'est pas atteinte.

NMotorRunState []

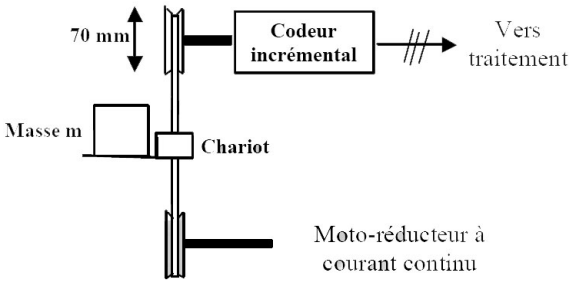
Registre ou tableau qui contient la valeur en cours de l'encodeur *(idle, ramping up or down, steady state, holding encoder position while stopped)* (inactif, accélération ou décélération, état stable, valeur de l'encodeur lorsque le moteur est arrêté. Cette variable fournit une méthode commode de vérification pour savoir si un moteur a atteint une valeur de consigne : l'état de la variable « état moteur `NMotorRunState`» passe à « idle » quand la position est atteinte et que le moteur est arrêté.

Variables complémentaires employées pour les contrôles avancés du moteur

BFloatDuringInactiveMotorPWM

Indique si l'on travaille en mode « flot » (flottant), le moteur est alimenté en tension flottante et est donc en roue libre lorsque le moteur est à l'arrêt (arrêt par inertie) ou en mode « brake » (freinage), le moteur est alimenté en « court-circuit » à l'arrêt c'est-à-dire qu'il a un couple à vitesse nulle (permet de supporter une charge en course intermédiaire sans frein mécanique spécifique (voir figure ci-dessous) car l'axe moteur n'est pas libre mais soumis à un couple, on ne peut pas faire tourner l'axe du moteur à la main).

PWM : Pulse Width Modulation = MLI : Modulation de Largeur d'Impulsion.



Maintient d'une masse par couple à vitesse nulle : moteur alimenté en MLI

Exemple: `BFloatDuringInactiveMotorPWM = faux;` mode frein

`BFloatDuringInactiveMotorPWM = vrai;` mode flottant

NPidUpdateInterval []

Indique la cadence de rafraîchissement de l'information reçue des codeurs, par défaut cette valeur est de 25 ms. Le microcontrôleur traitera ces informations pour réajuster sa commande en fonction du retour des informations transmises par l'encodeur.

BMotorReflected []

Drapeau (flag en anglais : variable ou bit qui renseigne sur un état) booléen pour indiquer si le moteur fonctionne en sens normal ou inversé par rapport à la commande.

NMaxRegulatedSpeed

Indique le nombre d'impulsions d'encodage par seconde en fonction de la vitesse.

A vitesse maximale, le nombre d'impulsions par défaut est de 1000 par seconde. Cette valeur a été choisie car elle correspond à celle utilisée par le microprogramme de la norme NXT-G.

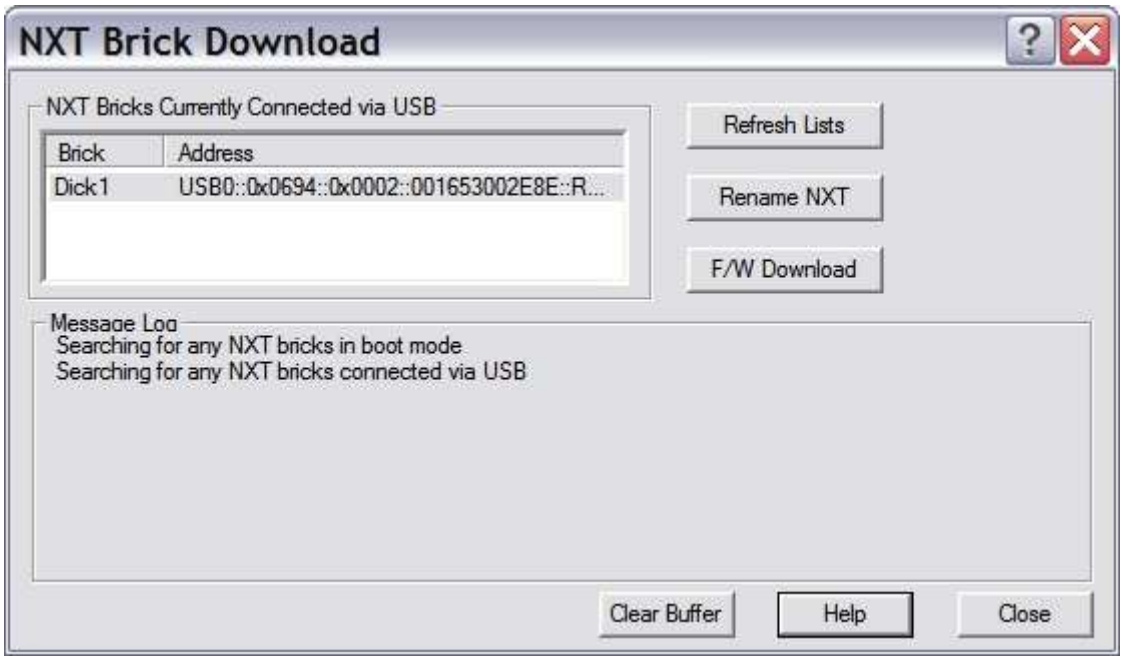
Avec des batteries partiellement déchargées, seules environ 750 impulsions peuvent être réalisées. Cela signifie que si vous voulez utiliser des vitesses cohérentes en fonction du niveau de batterie vous ne devez pas spécifier des vitesses au-dessus de 75 % ou alors vous devrez réduire le niveau de vitesse maximal.

Exemple: `NMaxRegulatedSpeed = 750 ;` niveau de vitesse de vitesse réglé à 750

9. Téléchargement de brique de NXT

Cette fenêtre est utilisée pour charger le microprogramme (« firmware ») de RobotC dans la brique NXT. Vous serez amené lors :

- d'une mise à jour du firmware (actuellement nous chargeons le firmware `NXT0723.rfw`).
- d'un changement de logiciel de programmation (essentiellement passage du logiciel LEGO vers RobotC).



La fenêtre contient la liste des NXTs qui sont sous tension et connectées au PC par l'intermédiaire de l'USB.

A l'ouverture de la fenêtre, tous les périphériques USB connectés au PC scannés pour détecter s'ils sont des NXT apparaissent dans le fenêtre. Il y a un bouton pour régénérer la liste à tout moment.

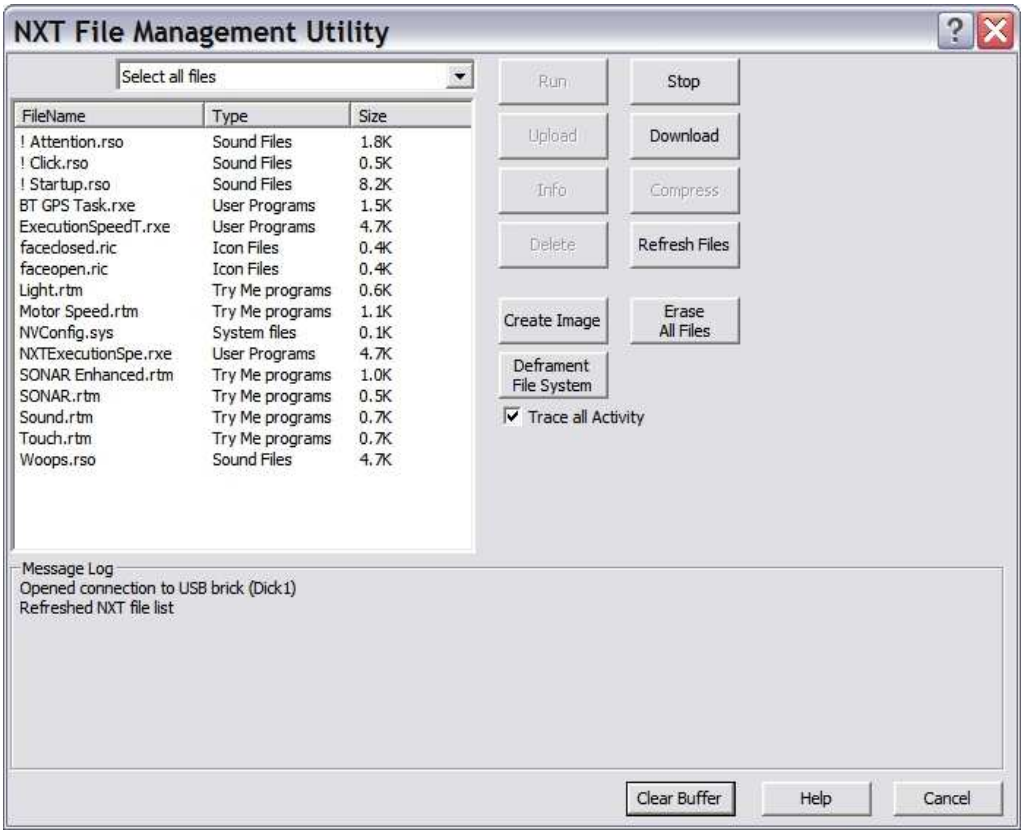
Boutons

Bouton	Action réalisée
Refresh List	Ce bouton régénère la liste de NXTs.
Rename NXT	Ce bouton ouvrira une fenêtre permettant de renommer la brique NXT sélectionnée.
F/W Download	Ce bouton lance le téléchargement du firmware vers le NXT sélectionné. Il ouvrira une seconde fenêtre permettant de sélectionner le firmware à télécharger. Le téléchargement du firmware peut prendre 30 à 60 secondes.

10. Gestion des fichiers

Cette fenêtre sert à la gestion des fichiers - télécharger vers le PC, télécharger du PC, effacer, etc. - sur le NXT.

Le fonctionnement de cette fenêtre est sur le modèle des explorateurs des différents systèmes d'exploitation (affichage trié par nom, type, taille, …). Les boutons de droite deviennent actifs lorsque l'action est possible.

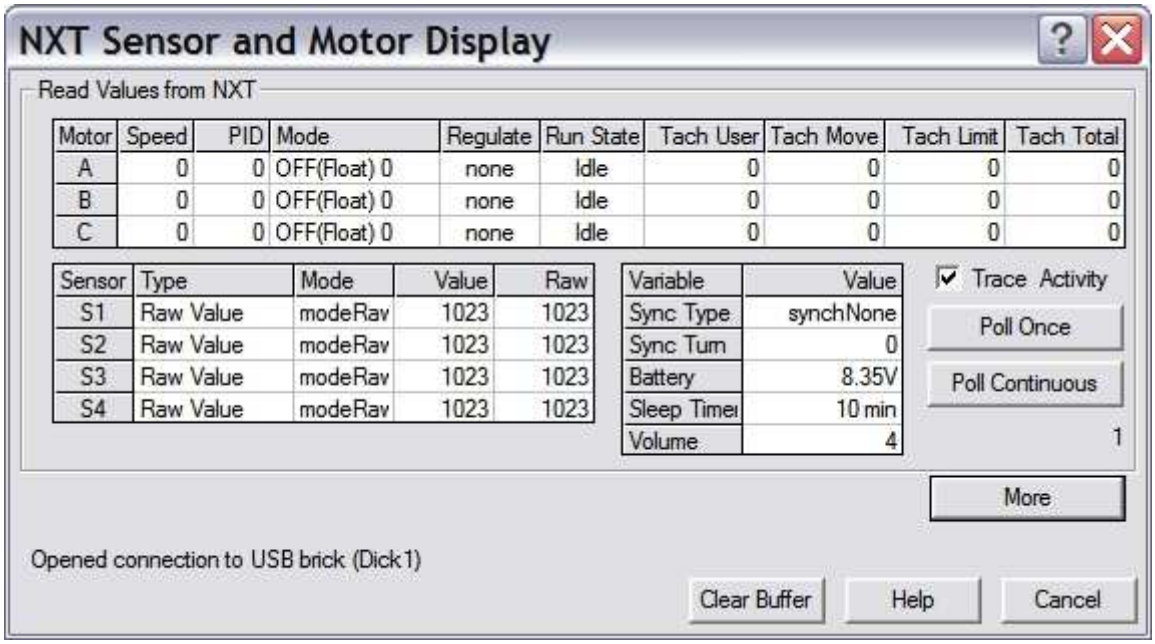


Boutons

Bouton	Action réalisée
Run / Play	Utilisé pour lancer un programme ou pour jouer un fichier audio. Appliqué au fichier sélectionné.
Stop	Utilisé pour arrêter l'exécution d'un programme ou la lecture d'un fichier audio.
Upload	Copie les fichiers sélectionnés du NXT vers le PC.
Download	Copie les fichiers du PC vers le NXT. Il faudra choisir le(s) fichier(s) du PC à transférer.
Info	Fournit une information détaillée sur les fichiers sélectionnés - i.e. la taille exacte et le type. L'information est affichée dans la fenêtre « Message Log ».
Compress	Utilisé pour compresser les fichiers audios choisis sur le NXT. Les fichiers compressés prennent environ la moitié de la taille des fichiers audios non compressés.
Refresh Files	Régénère la liste des fichiers sur le NXT.
Create Image	Crée un fichier image du contenu de la mémoire Flash de NXT. Ce fichiers image peut être alors téléchargés dans un autre NXT qui aura exactement les mêmes fichiers que le NXT source.
Erase All Files	Efface tous les fichiers du NXT.
Defragment File System	Les opération de chargement/effacement de fichiers dans le NXT créent des intervalles dans la mémoire de mémoire Flash trop petits pour être utilisables pour un nouveau fichier. La commande de défragmentation « Defragment » éliminera les intervalles entre les fichiers afin d'optimiser l'utilisation de la mémoire Flash.

11. *Balayage de la brique NXT*

Cette fenêtre permet l'accès aux configurations en cours des moteurs et des capteurs.



Moteurs

The top table in the display contains the current settings for the motors. This includes both the current speed setting and various encoder (or "Tachometer") settings.

Le tableau supérieur de la fenêtre (« motor »)contient les configurations des moteurs. Ceci inclut la configuration de la vitesse en cours des moteurs et des encodeurs (ou « tachymètre ").

Capteurs

Le tableau suivant (« sensor ») contient les configurations en cours pour les quatre capteurs.

Boutons

Bouton	Action réalisée
Poll Once	Recherche et affiche une seule fois les données des moteurs et des capteurs.
Poll Continuously	Recherche et affiche continuement.
More / Less	Agrandit (« more") ou diminue (« less ») la fenêtre pour afficher des champs supplémentaires qui permettent d'écrire des valeurs pour les moteurs et les capteurs. Cette opération d'écriture de valeurs pour les moteurs/ capteurs est réservés aux utilisateurs avertis.

12. Capteurs

Le NXT est équipé de quatre ports de capteur (S1, S2, S3 et S4). Il y a une série de fonctions et variables utilisées pour configurer ces ports et accéder à leurs valeurs.

Les configurations des capteurs peuvent être compliquées. RobotC possède un assistant intégré qui peut être utilisé pour configurer les capteurs de NXT. L'assistant contient un certain nombre de fenêtres de PC qui permettent de configurer les champs suivants pour le capteur :

- **le nom de la variable que vous voulez assigner au capteur**. Il y a un certain nombre de variables « internes » qui contiennent des informations sur les capteurs. Ces variable peuvent être les repères des port de capteur (i.e. S1, S2, S3 et S4) ou même le numéro de positionnement (i.e. 0 à 3). Cependant, il est vaut mieux assigner aux capteurs un nom signicatif (par exemple « lightSensor ou photodétecteur », « leftBump ou contact_gauche », « rightBump ou contact_gauche », « sonar ou ultrason », etc.) afin d'identifier immédiatement le capteur dans le programme. On a tenté de proposer des noms en français pour désigner les cpateurs, mais la langue anglaise offre l'avantage d'esprimer les choses de manière très contractée, ce qui évite d'avoir des variables avec des noms à rallonge.
- **Le type de capteur qui est utilisé.** Le kit NXT propose plusieurs capteurs développés par LEGO - contact, bruit, lumière, ultrason - et il y a beaucoup d'autres capteurs en développement. Les firmware doivent connaître le type de capteur afin de pouvoir correctement le configurer - par exemple certains capteurs requiert une alimentation par la batterie 9V - et savoir adapter les valeurs qu'ils transmets.
- **Le mode du capteur.** Le mode le plus courant est le « percentage ou pourcentage » qui normalisera la valeur transmise par le capteur dans l'intervalle [0 100]. L'autre mode courant est le « raw ou brut » qui renvoie simplement la valeur brute (non normalisée) du capteur. D'autres modes incluent « Celsius » et « Fahrenheit » qui sont employés pour spécifier l'échelle pour des capteurs de température.

`SensorType[]`

Cette variable en lecture/écriture est utilisée pour configurer le type d'un capteur (contact, lumière, ultrason, etc.)

`SensorMode[]`

Cette variable en lecture/écriture est utilisée pour configurer le mode d'un capteur.

`SensorValue[]`

Cette variable contient la valeur normalisée en cours du capteur.

`SensorRaw[]`

Cette variable contient la valeur brute (non normalisée) en cours du capteur.

Le firmware du NXT contient des pilotes de périphérique intégrés pour les capteurs fournis par LEGO. Cependant, il existe d'autres capteurs développés par d'autres entreprises qui sont compatibles avec le NXT. Dans le meilleur des cas ils peuvent être utilisés dans une application d'une manière complétement transparente pour l'utilisateur ; le développeur du capteur fournit les pilotes et RobotC les identifie automatiquement. Au démarrage il augmente automatiquement la liste de types de capteurs disponibles. Dans sa verison 1.10 RobotC ne comprend pas ce mode de détection, cela est prévu pour les mises à jour à venir. En attendant il faut déclarer les capteurs manuellement (i.e. quelques lignes de code). Le concept de base est le suivant :

- Le pilotage de périphériques pour le capteur « non LEGO » sera une tâche séparée (« thread ») de RobotC qui s'exécute parallèlement au programme de l'utilisateur. Normalement le thread initialisera le capteur et assurera l'acquisition continue des valeurs du capteur.

- Pour faciliter l'implementation des pilotes les variables xx et xx sont maintenant des variables en lecture/écriture (contre lu seulement). Ceci permet à la tâche de gestion des pilodes de mettre à jour ces champs avec les résultats des acquisistions de valeurs.
- L'application utilisateur peut accéder simplement aux variables xx et xx des tout comme pour un capteur LEGO dont les pilotes sont integrés dans le firmware. Cettes quasi transparence du fonctionnement fait que l'utlisation d'un capteur tiers (non LEGO) ou LEGO sera quasiment identique pour l'utilisateur.
- Deux nouvelles variables internes ont été ajoutées au firmware pour supporter les capteurs tiers :

  - **`SensorSubType[]`**

    Cette variable contient le **`SubType («`** sous-type ») pour le capteur indiqué. Le type principal peut être assigné en tant que « custom » qui indique que c'est un capteur tiers avec son propre pilote de périphériques. Le type « custom » indiquera au firmware qu'il doit ignorer les pilotes intégrés pour ce capteur et que les zones xx et xx seront mises à jour par la tâche de pilotage de capteur tiers.

  - **`nPSControl[]`**

    Cette variable est employée par le capteur sans fil d'interface de contrôleur de Playstation développé par MindSensors. Elle est employée pour retenir les valeurs des divers joysticks et boutons sur le contrôleur PS.

13. Contrôle du son

RobotC fournit une suite complète de fonctions pour contrôler le haut-parleur du NXT.

Le firmware de RobotC peut mettre jusqu'à 10 fichiers audio en file d'attente de lecture. Ceci permet au programme utilisateur de lancer la lecture d'un fichier audio et de continuer son exécution sans devoir attendre la fin de la lecture. Ceci a pour avantage de libérer le programme qui peut ainsi continuer à traiter « en temps réel » les informations reçues des capteurs par exemple.

RobotC joue les fichiers audios compressé et non compressé. L'environnement de développement de RobotC fournit une commande pour comprimer les fichiers audios. Il est situé dans le menu « Robot\NXT Brick\File ».

`ClearSounds();`

Efface toutes les commandes sons existantes et mises en mémoire tampon

`PlayImmediateTone(frequency, durationIn10MsecTicks);`

Jouer immédiatement une tonalité à la fréquence et la durée indiquée quelque soit l'état de la file d'attente (demande prioritaire).

`PlaySound(sound);`

Jouer un des sons prédéfinis par système (« buzz » ou bruit genre vibreur, « beep » ou bip-bip, « click » cliquetis,…).

`PlaySoundFile(sFileName);`

Joue un fichier audio du système de fichiers de NXT. Ce fichier doit exister sur le système de fichiers de NXT.

Quand un programme utilisateur est téléchargé par RobotC, un contrôle est fait pour s'assurer que tous les fichiers audios référencés par la fonction de « PlaySoundFile » existent sur le NXT ; sinon RobotC essayera de télécharger ces fichiers depuis le PC.

`PlayTone(frequency, durationIn10MsecTicks);`

Joue une tonalité constante à la fréquence et à la durée spécifiées.

`bPlaySounds`

Indicateur booléen qui Indique si de nouvelles demandes de sons doivent être acceptées ou rejetées. Un programmeur peut utiliser cet indicateur pour mettre une lecture de fichier audio en mode « muet » ou pas.

`bSoundActive`

Indicateur booléen qui a l'état « vrai » indique que le système est en train de jouer un son.

`bSoundQueueAvailable`

Indicateur booléen qui indique s'il y a de l'espace disponible dans la file d'attente audio pour un autre fichier.

Cet indicateur est utile à la fin d'un programme, si on veut s'assurer que tous les sons sont terminés avant de quitter le programme. Il suffit d'ajouter le contrôle suivant :

```
while (!bSoundQueueAvailable)
{}
```

`kMaxVolumeLevel`

Constante qui met le volume à son niveau maximum.

`nVolume`

Réglage du volume sain de 0 à 4 (le plus fort).