Interface Ottomator100

```
#include <string>
#include <vector>
#include "robot.h"
#include "ottoutils.h"

class Ottomator
{
public:
    Ottomator();
    ~Ottomator();

    // Utilities
    std::string getAllStatusesOfDSS1ForBit(int index, bool verbose = false);
    void updateM_statusMatrix();
    void resetM_statusMatrix();
    std::vector<std::string> manageStatusGate();
    int manageNextFor(int sequence, int actionSuccess, int error, int
specialIndex = 0, int specialManagement = 0);

    // Main Sequences
    std::vector<std::string> birthSequence();
    std::vector<std::string> frameworkSequenceTo(bool initiate);
    std::vector<std::string> workSequenceTo(bool fetch, int sampleN);

    // Demontrators
    std::vector<std::string> cycler();
    void displayM_statusMatrix();

    // Case getters and setters
    int** getM_statusMatrix();
    int getBirthSequenceCase();
    void setBirthSequenceCase(int n);
    int getFrameworkSequenceCase();
    void setFrameworkSequenceCase(int n);
    int getWorkSequenceCase();
    void setWorkSequenceCase(int n);
    std::vector<std::string> getM_sampleArray();
    void setM_sampleArray(std::vector<std::string> sampleArray);
    int getNCurrentSample();
    void setNCurrentSample(int n);

    // Integration
    void writeLog(std::string text);

    // Components
    Robot m_robot;

private:
    // State attributs
    int m_statusMatrix[16][6];

    // Cycler attibuts
    std::vector<std::string> m_sampleArray;

};
```

Comment utiliser Ottomator :

Méthode 1 :

```
#include "ottomator.h"

using namespace std;

int main()
{
    // Create objects
    Ottomator theSequencer;

    // Connect to serial port
    string port;
    theSequencer.m_robot.m_busManager.modbusConnectRTU(port); // Eg: "COM9"

    // Populate the sample array
    vector<string> sampleArray;
    theSequencer.setM_sampleArray(sampleArray);
    theSequencer.resetM_statusMatrix();

    // Run cycler
    messages = theSequencer.cycler(); // cycler s'occupe de tout jusqu'à ce que le 1er échantillon de la liste sampleArray est mis dans le château. A ce moment, le message retourné est "Fetch completed".

    // Comptage
    // ...

    messages = theSequencer.cycler(); // en relançant le cycler sans réinitialiser la m_statusMatrix, le cycler s'occupe de tout jusqu'à ce que le 1er échantillon de la liste sampleArray est remis à sa place. A ce moment, le message retourné est "Putback completed". De plus le cycler retirera le 1er élément de la liste sampleArray.

    // Run cycler
    messages = theSequencer.cycler(); // cycler s'occupe de tout jusqu'à ce que le 1er échantillon de la (nouvelle) liste sampleArray est mis dans le château. A ce moment, le message retourné est "Fetch completed".

    // Comptage
    // ...

    // Putback
    messages = theSequencer.cycler();

    // ... etc etc etc

    // ... Et quand le dernier échantillon est reposé :
    messages = theSequencer.cycler(); // en relançant le cycler sans réinitialiser la m_statusMatrix, le cycler détecte que la liste sampleArray est vide. Il s'occupe alors de tout remettre en place pour terminer le batch. Le message retourné est alors "Finish completed"


}
```

Méthode 2 :

Tu peux aussi organiser le cycle comme tu le sens. Pour cela voici les fonctions utiles :

- Toutes les informations sont dans la m_statusMatrix. Les 3 fonctions suivantes permettent de la mettre à jour, la réinitialiser et l'obtenir :

```
void updateM_statusMatrix();
void resetM_statusMatrix();
int** getM_statusMatrix();
```

- Les séquences de base sont les suivantes :

```
std::vector<std::string> birthSequence();
std::vector<std::string> frameworkSequenceTo(bool initiate);
std::vector<std::string> workSequenceTo(bool fetch, int sampleN);
```

    - Pour initialiser la machine :
        - **birthSequence()**
    - Pour préparer le terrain avant les mouvements :
        - **frameworkSequenceTo(1) ;**
    - Pour chercher un échantillon sampleN :
        - **workSequenceTo(1, sampleN) ;**
    - Pour reposer un échantillon sampleN :
        - **workSequenceTo(0, sampleN) ;**
    - Pour remettre tout en place à la fin d'un batch :
        - **frameworkSequenceTo(0) ;**

Chacune de ces fonctions retourne un vecteur de messages. Le premier permet de savoir si l'opération est s'est passé avec succès. Si erreurs, les erreurs apparaissent.

- En bonus, si tu veux surveiller le passeur en dehors d'un processus, il y a une fonction de surveillance de base.

```
std::vector<std::string> manageStatusGate();
```

    - Si tout va bien, elle retourne un vecteur vide. Sinon, les erreurs apparaissent.

Depuis l'objet Ottomator toutes les fonctions des étages inférieurs sont accessibles. Donc tu peux tout piloter si tu veux.
Les étages sont :

- Ottomator
    - Robot
        - BusManager
            - (libmodbus)

Coordonnées des points :

	1	2	3	4	5	6
1	D					
2	1	2	3	4	5	
3	6	7	8	9	10	
4	11	12	13	14	15	
5	16	17	18	18	20	
6				A		C

	6	5	4	3	2	1
1	16	11	6	1		
2	17	12	7	2		
3	18	13	8	3		
4	19	14	9	4		
5	20	15	10	5		
6				A	D	C

```
// State attributs
int m_statusMatrix[16][6];
```

	0	1	2	3	4	5
0	nC	x	y	z	p	v
1	F	CLBS	CLBS	CLBS	CLBS	LAT
2	I	CEND	CEND	CEND	CEND	EMGV
3	B	PEND	PEND	PEND	PEND	P1
4	Fw	HEND	HEND	HEND	HEND	P2
5	W	STP	STP	STP	STP	MOTO
6						OPEN
7		BKRL	BKRL	BKRL	BKRL	CLOS
8		ABER	ABER	ABER	ABER	
9		ALML	ALML	ALML	ALML	
10		ALMH	ALMH	ALMH	ALMH	
11		PSFL	PSFL	PSFL	PSFL	
12		SV	SV	SV	SV	
13		PWR	PWR	PWR	PWR	
14		SFTY	SFTY	SFTY	SFTY	
15		EMGS	EMGS	EMGS	EMGS	

nC = nCurrentSample
F = isFetching
I = isInitiating
B = birthSequenceCase
Fw = FrameworkSequenceCase
W = WorkSequenceCase

X = position of x axis (0..6, 0 = Home)
Y = position of y axis (0..6, 0 = Home)
Z = position of z axis (0..3, 0 = Home, 1 = High, 2 = Table, 3 = Detector)
P = position of plier (0..2, 0 = Home, 1 = Opened, 2 = Closed)
V = position of castel actuator (1 = Opened, 2 = Closed)

(12) Data of device status register 1 (Address = 9005_H) (DSS1)

Bit	Symbol	Name	Function
15	EMGS	EMG status	0: Emergency stop not actuated 1: Emergency stop actuated This bit indicates whether or not the controller is currently in the emergency stop mode due to an emergency stop input, cutoff of the drive source, etc.
14	SFTY	Safety speed enabled status	0: Safety status disabled 1: Safety status enabled Enable/disable the safety speed of the controller using the "safety speed command bit" of device control register 1.
13	PWR	Controller ready status	0: Controller busy 1: Controller ready This bit indicates whether or not the controller can be controlled externally. Normally this bit does not become 0 (busy).
12	SV	Servo ON status	0: Servo OFF 1: Servo ON The servo ON status is indicated. After a servo ON command is issued, this bit will remain 0 until the servo ON delay time set by a parameter elapses. If the servo cannot be turned ON for some reason even after a servo ON command is received, this bit will remain 0. The RC controller does not accept any movement command while this bit is 0.
11	PSFL	Missed work part in push-motion operation	0: Normal 1: Missed work part in push-motion operation This bit turns 1 when the actuator has moved to the end of the push band without contacting the work part (= the actuator has missed the work part) according to a push-motion operation command. Operation commands other than push-motion do not change this bit.
10	ALMH	Major failure status	0: Normal 1: Major failure alarm present This bit will turn 1 if any alarm at the cold start level or operation cancellation level is generated. Alarms at the operation cancellation level can be reset by using an alarm reset command, but resetting alarms at the cold start level requires turning the power supply off and then on again.
9	ALML	Minor failure status	0: Normal 1: Minor failure alarm present This bit will turn 1 when a message level alarm is generated.
8	ABER	Absolute error status	0: Normal 1: Absolute error present This bit will turn 1 if an absolute error occurs in case the absolute specification is set.
7	BKRL	Brake forced-release status	0: Brake actuated 1: Brake released This bit indicates the status of brake operation. Normally the bit remains 1 while the servo is ON. Even when the servo is OFF, changing the "brake forced-release command bit" in device control register 1 to 1 will change this bit to 1.
6	-	Cannot be used	
5	STP	Pause status	0: Normal 1: Pause command active This bit remains 1 while a pause command is input. If the PIO/Modbus Switch Setting (5.4.16 or 6.5.16) is PIO enabled, paused PIO signals are monitored (set the switch to AUTO in case of RC controllers with a mode toggle switch). If Modbus is enabled, the Pause Commands (5.4.6 or 6.5.6) are monitored.
4	HEND	Home return completion status	0: Home return not yet complete 1: Home return complete This bit will become 1 when home return is completed. In case the absolute specification is set, the bit is set to 1 from the startup if absolute reset has been completed. If a movement command is issued while this bit is 0, an alarm will generate.
3	PEND	Position complete status	0: Positioning not yet complete 1: Position complete This bit turns 1 when the actuator has moved close enough the target position and entered the positioning band. It also turns 1 when the servo turns on after the actuator has started, because the controller recognizes that the actuator has completed a positioning to the current position. This bit will also become 1 during the push-motion operation as well as at the completion.
2	CEND	Load cell calibration complete	0: Calibration not yet complete 1: Calibration complete This bit turns 1 when the load cell calibration command (CLBR) has been successfully executed.
1	CLBS	Load cell calibration status	0: Calibration not yet complete 1: Calibration complete Regardless of whether or not a load cell calibration command has been issued, this bit is 1 as long as a calibration has completed in the past.
0	-	Cannot be used	

The corresponding bit gets a 1 if not normal value.

	Messages	Normal value	Bit weight	Fatal error
0	SV	1	1	
1	STP	0	2	
2	EMGS	0	4	
3	EMGV	1	8	
4	P1	0	16	
5	P2	0	32	
6	LAT	1	64	
7	ALML	0	128	
8	ALMH	0	256	1
9	ABER	0	512	1
10	MOTO	0	1024	1
11	XCAS	PSFL at completion	2048	1
12				
14				
15				
16				

-1 = Stuck in birth sequence
-2 = Stuck in initiation sequence
-3 = Stuck in fetch sequence
-4 = Stuck in putback sequence
-5 = Stuck in finish sequence
-6 = Sequence case out of range
-7 = Empty catch
-8 = Birth completed
-9 = Initiation completed
-10 = Fetch completed
-11 = Putback completed
-12 = Finish completed
-13 = Need input

CYCLOPE2_COLLISION_PINCE = P1 ou P2 basculent (EMGS et EMGV aussi car le passeur est câblé de telle façon que l'erreur sur l'un des capteurs de la pince met immédiatement les axes en Emergency)

CYCLOPE2_COLLISION_CHATEAU = l'axe x dans sa course vers la position château retourne PSFL en mode push. Cela signifie qu'un obstacle infranchissable a été rencontré lors de la course et que l'axe x n'arrive pas à atteindre la position programmée.

CYCLOPE2_DANGER_TIME_OUT = l'une des fonctions de positionnement appelé retourne time out (13s dépassées pour les mouvements vérins ou 40s dépassées pour les mouvements l'axes)

CYCLOPE2_ERREUR_CODEUR_ABSOLU = ABER a basculé sur l'une des axes (information d'une erreur de positionnement absolu, le câble entre carte d'axe et axe a été débranché, l'axe en question a fait un choc important qui a déréglé le codeur absolu… un redémarrage dans de bonne condition suffisent à remettre tout d'aplomb)

CYCLOPE2_ERREUR_MOTEUR = MOTO a basculé (information d'un défaut moteur remontée par le modicon contrôlant le vérin)

CYCLOPE2_ALARME_GRAVE = ALMH a basculé sur l'une des axes (alarme grave remontée par la carte d'axes suite à des problèmes divers. Les ne peuvent pas être acquittés un simple alarmReset, il faut redémarrer voir utiliser Robocylinder pour diagnostiquer. Les ALMH sont documentés dans la doc IAI.

CYCLOPE2_ALARME_ACQUITTABLE = ALML a basculé sur l'une des axes (alarme mineurs remontée par la carte d'axes suite à des problèmes divers (buté de faible gravité). Celles-ci peuvent être acquittées par un simple alarmReset. Les ALMH sont documentés dans la doc IAI.

CYCLOPE2_SERVO_OFF = un SV reste à 0. L'une des axes informe que son servomoteur est éteint. (Les servomoteurs passent automatiquement à OFF lors d'un Emergency, le but de ce mot d'erreur et de discriminer les Emergency des cas où le servomoteur n'est pas allumé alors même qu'il n'y a pas d'Emergency (l'étape servoON n'a pas été effectuée, ou sans succès))

CYCLOPE2_STOP_BIT = un STP bascule : L'une des axes informe que son Stop bit a été activé (elle est donc passée en mode pause). Ottomator règle ce cas en forçant tous les Stop bit à l'état marche.

CYCLOPE2_PORTE_CHATEAU_OUVERTE = LAT bascule. Le capteur de la porte latérale du château indique qu'elle n'est pas correctement fermée. (LAT entrain comme toutes les informations de capteurs de sécurité le basculement de EMGV et EMGS (Emergency). Ceci permet de stopper tout mouvement en cours).

CYCLOPE2_FENETRE_OUVERTE = Quand l'une des fenêtres n'est pas fermée, le câble du passeur fait en sorte que toutes le axes ainsi que le vérin passent en Emergency (arrêt immédiat du mouvement en cours et non reprise tant que l'ordre n'a pas été donnée même après réarmement). Pour discriminer le cas d'une fenêtre ouverte de tous les autres cas d'erreur nécessitant de mettre le passeur en Emergency, ce cas est traité en dernier car c'est le seul impliquant seulement le basculement de EMGV et EMGS. (Scenario : Lorsqu'une fenêtre s'ouvre, le passeur passe en Emergency et Ottomator détectant EMGS attend patiemment qu'EMGS disparaisse. Le flag EMGS disparait si les fenêtres sont refermées et que l'opérateur à réarmé le passeur).

Logique de « cycler » :