Mechanical Work and Further Software Development for the Former Prof. Perky Humanoid Robot Project.

Final Project Report

for

Intelligent Robotics II (ECE 579)
Portland State University, Spring 2004

Student: Stefan Gebauer

1. Abstract

This term, my work on the project was focused on the resurrection and further mechanical development of the robot. On the other hand I released two new drivers for the PC's serial port which are incorporated into a class that wraps control of the ASC16 board, movement of every single DOF of the robot and of course simple behaviors. Further work was done by the project's group members Normen Giesecke (Movement Editor) and Myron Machado (Gesture Recognition). The interactive game which was suggested by Akashdeep wasn't integrated to existing TTS and SR systems, though. This task will remain. During this term we had the chance to test the robot in an environment with many children. These demonstrations gained our experience towards Human Machine Interface (HMI) and were a good inspiration for new ideas.

2. Mechanical Work on the robot

At the beginning of the term, I started to resurrect the humanoid robot which was standing in the lab for at least 8 month. I tried to rebuild it as it was originally. Later on, my teammates supported me in setting up the robot. After finishing the body, we started to mount the Prof. Perky Head – it is basically a wood made frame which forms a head – on the body. Ultimately, we hooked up the Marc Medonis ASC16 board to the back of the robot. We laid cables, to connect the ASC16 board with each servo. For our first demo for talented high school students I bought a nice shirt and a tie from Goodwill. Finally the robot got a Dr. Frankenstein mask. It was

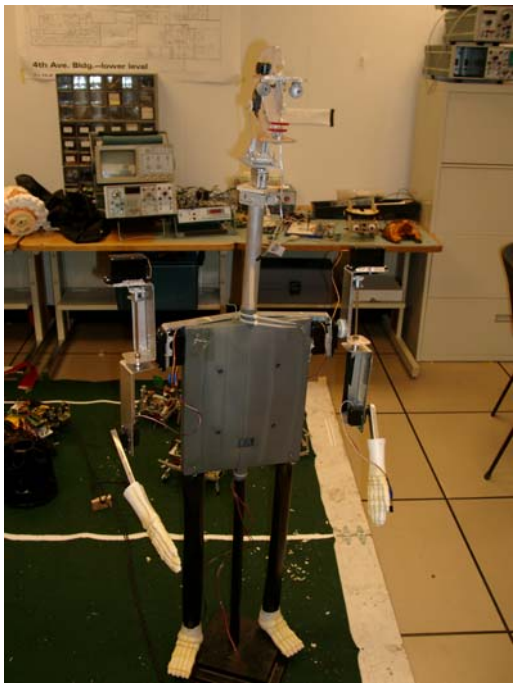

Fig. 1: The frame of “The Frank” robot after ressurection.

the last mask that remained in the lab. Actually, this is the reason why we relabeled the project to “TheFrank” project.

The robot made a quite scary impression to kids. They didn’t like it much even though we putted a nice hat on the mask. Besides, it turned out there are some improvements necessary:

- every time when the robot moved a arm, it dragged the shirt at the shoulders. It made the big servos even stuck. The reason for that can be found in the original design. The shoulder joint is not were it is supposed to be. The robot’s arm overtops the shoulder (Fig. 1 and Fig. 3). Therefore the joint is in the middle of the upper arm which is not natural.
- the whole neck is too high, the proportions of the robot are not correct. Moreover the servo for turning the head to the left or right broke during the demo.
- the whole construction is too shaky. We were afraid of the robot is falling over. When we moved a servo the whole robot was wiggling. One of us had to hold the robot and keep prevent it to fall over.

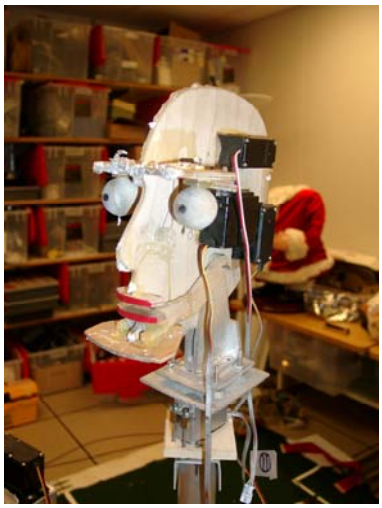

Fig. 2: Frame of Prof. Perky head

We changed those points. The robot got a massive and robust iron made stand. I putted the shoulders to it's right place. The others took care of the head and added a new movement. We call it moving the waist. Actually, the robot moves completely, not only the waist. At the next demonstration (Intel ISEF) the robot was in a much more better shape and we were able to present first movements. Again, the feedback of the people there taught us the following things:

- The mask is bad and scares the people. It needs to be removed.
- We need more interactivity, all the kids the were in front of the robot looked for something that somewhat influences Frank's behavior.
- Still the redesign of the head was not very successful. I is too weak.
- Up to that point we used batteries to support the robot with current.
- The movement of the jaw is too less.
- movement in general is too slow
- there has to be an easier way to program movements

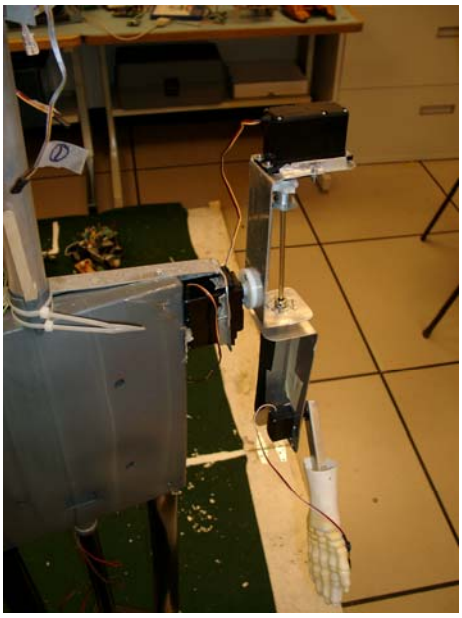

Fig. 3: Robot's shoulder. As you can see it
isn't at it's right position

Again, there were obviously grievances. The first thing I did was hooking up a new small switching power supply to the robot. Then I started to make the legs of the robot moving. I used two of the big servos (Fig. 4) to build a robust design. In the night before the PDXBot demo I replaced the old solution for turning the head by a new. On the other hand I started to program the movement editor. Later I assigned this work to Normen since I was busy with driver programming and mechanical work. The others fixed the problem with the jaw. It opens now much more wider than before. Together with Myron's gesture recognition, that we made in the meanwhile to an interactive tool and Normen's behavior editor, we have changed our presentation. On the PDXBot, children were able to play with the robot. They created new behaviors opened their mouth and surprisingly the robot did the same. The robot works mechanically good. I made the following observations:

- Again, the Dr. Frankenstein mask has to be removed. It scares people especially children. There were some little children that even haven't been approached and kept hiding because of the robot.
- More interactivity is needed. We had great success with the things that we showed but there can be more invented.

- A tracker feature would be really great. I got the code from the student who programs the light tracker for OMSI. I think it can be integrated to our code since it is completely written in Visual C++ 6.0 as our code is.
- I decided to devote myself more on image processing. The face recognition module needs to be finished. The tracker code needs to be adapted to our stuff.
- a game, like Akashdeep's would make the presentation much more interesting, even for people who are usually not interested in that field.

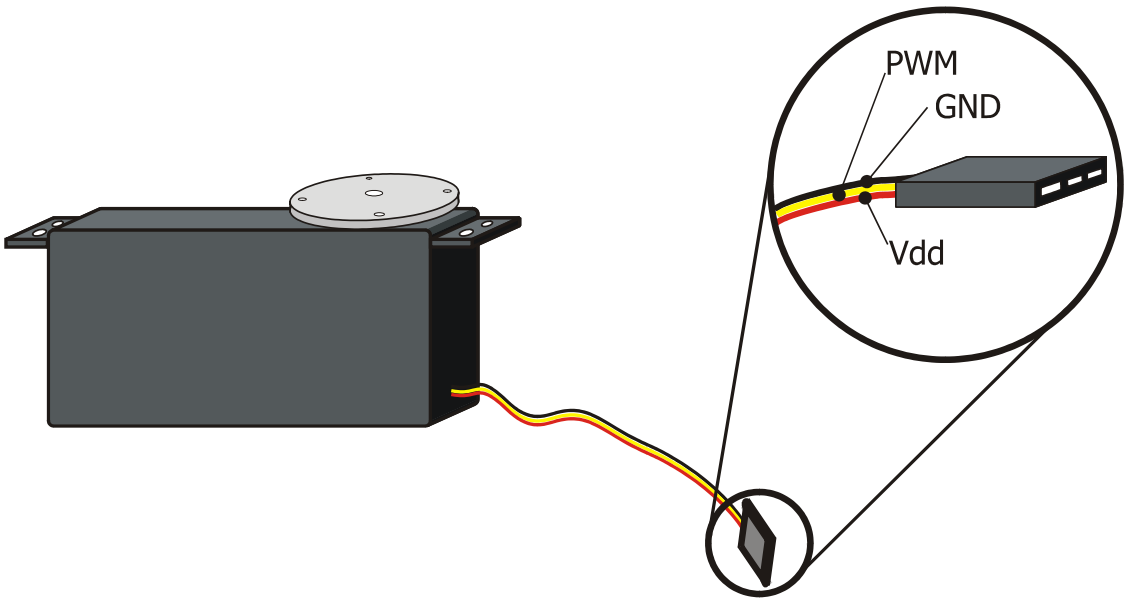

Fig. 4: A servo like we use it. It has three pins.

2.1 General Description of Hardware

The ASC16 board is our servo controller. It drives the servos, creates the PWM position signal for each of the servos and provides a serial connection (RS323) to the PC. We're using two sizes of servos. The bigger for joints which exhibit or have to resist a higher torque, for instance, the shoulder joint, the legs and waist. Actually, in case of the waist DOF the servo has to apply a higher force since the move ability has been accomplished by using a long bolt and two nuts. The robot's body is attached to a L-backed which in turn on the other end is fixed between the nuts. The nuts are still moveable on the thread. A small rod that is hooked to the robot on one side and on the other to the servo turns the robot. The whole construction is mounted at the robot's back. Thus, it has a hunchback. All together the robot has got 13 DOF's. They are listed in Table1. The

problem is, that every servo has a different range and initial value. Although the servos look identical, in fact, they are not. The range is different in every single case. Moreover when we exchanged a servo or when we fixed problems, the ranges changed. There was the need to make life easier for us. We made an ini file which stores the servo ranges. If something changes, we have the chance to change it in the file instead of compiling and changing the whole source code.

Servo #	DOF	Range Min	Initial Value	Range Max
1	eyes: left, right	2600 (left)	1690 (center)	780 (right)
2	mouth: open, close	2000	2000	3200
3	neck vertical: up, down	600	2000	3300
4	neck horizontal: left, right	2500	1000	-700
5	right shoulder: up, down	3700	2900	-400
6	right arm: turn inwards, outwards	3400	1720	45
7	right forearm: up, down	-700	-700	2800
8	left shoulder: up, down	60	650	4000
9	left arm: turn inwards, outwards	500	1300	3800
10	left forearm: up, down	3200	3200	-500
11	waist: left, right	3400	2320	1100
12	right leg: up, down	-300	1500	3200
13	left leg: up, down	3200	1700	-100

Table 1: All DOF's of The Frank robot with ranges.

2. 2 Software Development.

2.2.1 *Driver and Movement Development*

Originally I wanted to concentrate myself on the development of further SR and TTS features. The other group members were supposed to give me their parts and I wanted to integrate these to one program.

Because of several demos which had to give throughout the term we had to change plans. At the beginning the robot was not even moving. We had several parts working like TTS and SR which I did in previous terms. However, nothing was connected to the robot.

I devoted myself to the development of the robot movements, the robot control (See Fig. 5) and driver programming for the serial connection to the robot. It turned out that the driver development is a real challenge. I'll present results and issues at a latter point.

For the first presentation I used a serial driver which I started to program back in December 2003. I continued this work in the winter term since Jeff Morris was supposed to resurrect and connect

the robot to a PC. I decided at this time to support him and used a proprietary DLL for serial communication. The driver was never tested though since Jeff failed to resurrect the robot. In spring term, I gave the C++ class to Akashdeep. Because I was the only one how was involved in the source code and Akasdeep had really difficulties to do something with it, I took over again. For the talented student presentation, I had a simple version running. But there are several disadvantages:

- The driver doesn't work on my computer. I use a Serial to USB adapter to emulate a RS232 port on my computer. An expert who develops those adaptors told me those adaptor can not emulate the physical ports to 100%. So, I programmed the whole class without testing it. However, Normen's and Myron's laptops are equipped with such an old port. Newer laptops are usually not. For the future I would recommend to look for servo boards which come along with a USB connector.
- The first release was in a really early development state. There was no initialization of the servo position included. Neither behaviors nor movements we had to type every command to an edit box in a simple interface.

However our current version of the movement editor and all movements and behaviors are still based on the simple driver. Thus it is not running on computers with just USB connectors. The class is included in the appendix of this document.

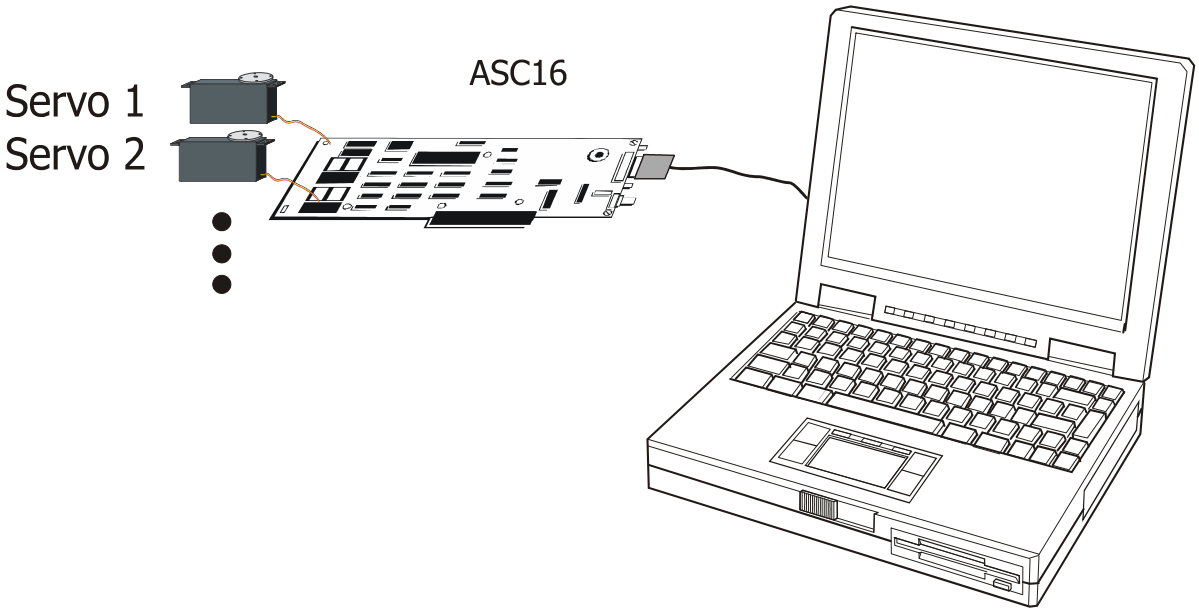

Fig. 5: The scheme of the robot control. The laptop is used to send commands to the ASC16 board which controls every single servo.

For the Intel ISEF conference the rest of the group programmed behaviors. I added a random generator to the code. So, we were able to pick a behavior by random. That was pretty much the innovation for the ISEF conference. After the conference I really made effort to get around the emulation limitation. First the class was divided into two parts, a driver part and a part which incorporates ASC16 communication, movements and certain behaviors. A modular concept is always good to extract and replace modules. In this case the driver is the crucial part. Second, I enhanced the ASC16 communication. Several functions for controlling the servo's speed and it's accelerations were added.

However, the new driver wasn't working on my computer. Even Normen's computer wouldn't work with the new software. It only worked on the desktop computer in the lab. Normen and Myron switched back to the old driver. For the PDXBot presentation, I added to the old driver the acceleration and speed functions. The robot was doing it's movements really fast.

There are three types of how to program a serial driver under windows NT based operating systems. I tried two. Therefore the last one is the one which should give full communication functionality. I programed this in the last two days before the PDXBot demonstration but I wasn't able to finish it. Last week I got it done, though. Unfortunately it has a bug which I haven't been found yet. The source code of the driver is in appendix listed.

2.2.2 *Speech Recognition*

For the reasons I mentioned above, I had to interrupt programming the SR class. However I started and a rough outline of the whole class was designed. Right now, only uncomfortable programming with the FONIX API's is possible which I did last term. The draft of the SR class can be found in appendix.

2.2.3 *Solving of Dependencies*

In all our demonstration there was a big part missing: Speech Synthesis and Speech Recognition. Even though these parts are working properly they couldn't be used for the demonstration. These parts work only on my computer. They are not portable to any other PC's since there are too many dependencies and requirements that have to be fulfilled in order to give access to those tools. Actually, the idea was to integrate all modules to my (main)program since it is the biggest chunk of code. For reasons I mentioned above this plan failed. A possible workaround is to solve all dependencies which are in my source code, put them in a *.ini file where they could perfectly maintained. I started doing that every time I was not working on the robot or driver. Finally there will be a graphical interface for this which is already a part of my program. The software will work on every computer where MS SAPI 5.1 and FONIX embedded 2.1 software packages are installed. The draft of the *.ini file which the program uses is in appendix listed.

3. Conclusions

The biggest accomplishment is the resurrection and the mechanical improvement of the robot. Simple application, like the movement editor and the gesture based mouth and laugh control are working. However, a really big part is not accessible. SR and TTS were not integrated since the serial connection is a serious problem for newer computers which do not have a build in serial connector.

Still, there remain many things to do. The Frank robot needs urgent a better mask. The driver problem has to be solved once and for all. More programming work has to be done for the

FONIX wrapper class, the FONIX Speech API would be another nice feature. The FONIX Speech system is nicer than the MS SAPI 5.1. Perhaps the behavior editor could be improved, too. Firstly, the format we're currently using is not compatible to this what the group around Martin uses. Secondly, a neat feature is to react on certain actions that take place in front of the robot. We got the same features from the gesture module. We could those integrate to the editor. If a gesture (or more) is detected we could react with an behavior. Moreover the gesture can be connected to some logic function which control then the event. By event I mean behaviors.

I decided to devote myself more on image processing. The face recognition module needs to be finished. A tracker feature would be really great. I got the code from Siciliano. He programs the light tracker for OMSI museum. I think it can be integrated to our code since it is completely written in Visual C++ 6.0 as our code is. It just needs to be adapted to our classes. A control needs to be designed for that. I'm thinking of a Fuzzy Control. Chris Brawn did it for hexor. It might be interesting to introduce it to humanoid robots as well.

Ultimately, a game, like Akashdeep's would make presentations in future much more interesting, even for people who are usually not interested in that field.

4. Abbreviations

TTS	Text to Speech Recognition
SR	Speech Recognition
HMI	Human Machine Interface
DOF	Degree of Freedom
PWM	Pulse Width Modulation
DLL	Dynamic Link Library
USB	Universal Serial Port

5. Appendix

The First Driver Class – Header File

```
// Filename: RobotCtrl.h: declarion of the class CRobotCtrl.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_ROBOTCTRL_H__822B2913_5160_4D42_8CD4_9402096C5C67__INCLUDED_)
#define AFX_ROBOTCTRL_H__822B2913_5160_4D42_8CD4_9402096C5C67__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CRobotCtrl
{


public:


	int ComplexBehavior(char *lpszPos);
	int MvNeck(char *lpszPos);
	int MvJaw(char *lpszPos);
	int MvEyes(char *szPos);
	void InitASC16(void);
	short GetCurPosition(short ServoNum);

	/*
		the following part describes certain behaviors of the
		robot.

	*/
	short GetPrevPosition(short usServoNum);  // returns the previous position of a servo

	/*
		The following section consits of methods that give control
		over the pc' serial port
	*/


	int COMTimeOut(unsigned short time_in_ms);	// sets timeout of the interface
							// reference: new
timeout in milliseconds
							// return value: old
timeout
	void DTR(unsigned short DTRstatus);		// set serial DTR pin to 0 or 1 (DTRstatus)
	void TXD(unsigned short TXDstatus);		// set serial TXD pin to 0 or 1 (TXDstatus)
	void RTS(unsigned short RTSstatus);		// set serial RTS pin to 0 or 1 (RTSstatus)
	unsigned short DCD(void);			// gets status of serial DCD pin -> return value
0 or 1
	unsigned short CTS(void);			// gets status of serial CTS pin -> return value
0 or 1
	unsigned short RI(void);			// gets status of serial RI pin -> return value 0 or 1
```

```
	unsigned short DSR(void);			// gets status of serial DSR pin -> return value
0 or 1
	int ReadByte(void);				// reads a byte from serial interface
							// returns -1 if error
occured, otherwise
							// the recieved bit
	void SendByte(short int w);			// sends a byte thru the serial interface
	void CloseCOM(void);				// closes serial interface
	int OpenCOM(char *s);				// opens the serial interface (RS232)
							// returns zero, if
failed
	int LoadDLL(char *name);

	CRobotCtrl();					// constructor

	virtual ~CRobotCtrl();				// destructor



private:
	int SetPosition(short ServoNum, short speed, short position);

	short nInitPos[16];				// array that stores all initial positions of the
servos
							// if the robot is turned on,
the servos will be set
							// to that positions
	short nPrevPos[16];				// array to store all of the previous servo
positions
	short nCurPos[16];				// contains the current positions of the servos


	// variables that are used just for the serial communiction

	HINSTANCE hDLL;					// handle for DLL

	// variables that are used for gesture stuff


};

#endif // !defined(AFX_ROBOTCTRL_H__822B2913_5160_4D42_8CD4_9402096C5C67__INCLUDED_)
```

The First Driver Class - Implementation

```
// RobotCtrl.cpp: Implementation, for declaration look up RobotCtrl.h
//
// Stefan Gebauer, Portland 04/13/2004
//
// Template to control the ASC16 servo board. For documentation view
// RobotCtrl.h!
// It's also going to wrap robot's behavior (Akashdeep Singh Aulakh)
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
```

```
#include "stdafx.h"
#include "RobotCtrl.h"


#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Constructor
//////////////////////////////////////////////////////////////////////

CRobotCtrl::CRobotCtrl()
{
        // load DLL and open COM-Port

   int i;
   if(LoadDLL("port"))
   {   i=OpenCOM("COM4: baud=9600 parity=N data=8 stop=1");
      if(i==0)
      {
                        MessageBox(GetFocus(),"It's not possible to open port
COM1","ERROR",MB_OK|MB_ICONSTOP);
      }
   }

        // initialization of the ASC16 board
        SendByte(121);
        SendByte(255);
        SendByte(245);

}


//////////////////////////////////////////////////////////////////////
// Destructor
//////////////////////////////////////////////////////////////////////

CRobotCtrl::~CRobotCtrl()
{
        CloseCOM();
        // close port
        hDLL = NULL;
}

int CRobotCtrl::LoadDLL(char *name)
{
        char s[256];
   hDLL = LoadLibrary(name);                                                    // Load DLL and
get handle
   if (hDLL == NULL)                                                                    // Success?
   {
                wsprintf(s,"%s nicht gefunden.",name);                          // no!
                MessageBox(GetFocus(),s,"Can't load port.dll",MB_OK|MB_ICONSTOP);
                return 0;
```

```
    }
    return 1;
}

int CRobotCtrl::OpenCOM(char *s)
{
        typedef int (CALLBACK* LP2INT)(char*);
        LP2INT p;
    p = (LP2INT)GetProcAddress(hDLL,"OPENCOM");
    return p(s);
}

void CRobotCtrl::CloseCOM()
{
        typedef int (CALLBACK* LP2INT)();
        LP2INT p;
    p = (LP2INT)GetProcAddress(hDLL,"CLOSECOM");

}

void CRobotCtrl::SendByte(short w)
{
        typedef int (CALLBACK* LP2INT)(WORD);
        LP2INT p;
    p = (LP2INT)GetProcAddress(hDLL,"SENDBYTE");
    p(w);
}

int CRobotCtrl::ReadByte()
{
        typedef int (CALLBACK* LP2INT)();
        char szFuncName[9] = "READBYTE";
        LP2INT p;
        p = (LP2INT)GetProcAddress(hDLL, szFuncName);
    return p();

}

unsigned short CRobotCtrl::DSR()
{
        typedef WORD (CALLBACK* LPFNDLLFUNC)();
        char szFuncName[4] = "DSR";
        LPFNDLLFUNC p;
        p = (LPFNDLLFUNC)GetProcAddress(hDLL, szFuncName);
        return p();


}

unsigned short CRobotCtrl::RI()
{
        typedef WORD (CALLBACK* LPFNDLLFUNC)();
        char szFuncName[3] = "RI";
        LPFNDLLFUNC p;
        p = (LPFNDLLFUNC)GetProcAddress(hDLL, szFuncName);
    return p();
```

```
}

unsigned short CRobotCtrl::CTS()
{
	typedef WORD (CALLBACK* LPFNDLLFUNC)();
	char szFuncName[4] = "CTS";
	LPFNDLLFUNC p;
	p = (LPFNDLLFUNC)GetProcAddress(hDLL,szFuncName);
   return p();
}

unsigned short CRobotCtrl::DCD()
{
	typedef WORD (CALLBACK* LPFNDLLFUNC)();
	char szFuncName[4] = "DCD";
	LPFNDLLFUNC p;
	p = (LPFNDLLFUNC)GetProcAddress(hDLL,szFuncName);
   return p();
}

void CRobotCtrl::RTS(unsigned short RTSstatus)
{
	typedef int (CALLBACK* LP2INT)(WORD);
	char szFuncName[4] = "RTS";
	LP2INT p;
	p = (LP2INT)GetProcAddress(hDLL, szFuncName);
	p(RTSstatus);
}

void CRobotCtrl::TXD(unsigned short TXDstatus)
{
	typedef int (CALLBACK* LP2INT)(WORD);
	char szFuncName[4] = "TXD";
	LP2INT p;
	p = (LP2INT)GetProcAddress(hDLL, szFuncName);
	p(TXDstatus);
}

void CRobotCtrl::DTR(unsigned short DTRstatus)
{
	typedef int (CALLBACK* LP2INT)(WORD);
	char szFuncName[4] = "DTR";
	LP2INT p;
	p = (LP2INT)GetProcAddress(hDLL, szFuncName);
	p(DTRstatus);
}

int CRobotCtrl::COMTimeOut(unsigned short time_in_ms)
{
	typedef int (CALLBACK* LPFNDLLFUNC)(WORD);
	char szFuncName[8] = "TIMEOUT";
	LPFNDLLFUNCp = (LPFNDLLFUNC)GetProcAddress(hDLL, szFuncName);
	return p(time_in_ms);

}

/*
```

```
*/


short CRobotCtrl::GetPrevPosition(short usServoNum)
{
	return nPrevPos[usServoNum];
}

int CRobotCtrl::SetPosition(short ServoNum, short speed, short position)
{
	return 1;
}

short CRobotCtrl::GetCurPosition(short ServoNum)
{
	return nPrevPos[ServoNum];
}

void CRobotCtrl::InitASC16()
{

}

int CRobotCtrl::MvEyes(char *szPos)                    // szPos can be either "left" or "right"
{
	return 1;
}

int CRobotCtrl::MvJaw(char *lpszPos)                   // lpszPos can be either "open" or "close"
{
	return 1;
}

int CRobotCtrl::MvNeck(char *lpszPos)                  // lpszPos can be either "left" or "right"
{
	return 1;
}

int CRobotCtrl::ComplexBehavior(char *lpszPos)     // more complex behavior like laugh, no, yes
{
	return 1;
}
```

The Robot Control Class – Header File

```
/************************************************************************
   filename                              :        SerialCom.h
        -------------------------------------------------------
   begin                                 :        Fri May 7 2004
        last change                              :        5/8/2004
   copyright                             :        (C) 2004 by Stefan Gebauer
   email                                 :        gebi@ece.pdx.edu
        -------------------------------------------------------
        Description:
        ============
```

```
	This class wrapps the communciation with the serial port / ASC 16 board.
	For implementation, please look into the file SerialCom.cpp.

 ********************************************************************************/



#if !defined(AFX_SERIALCOM_H__7637CC97_83F4_47C0_BD7C_804161AEC451__INCLUDED_)
#define AFX_SERIALCOM_H__7637CC97_83F4_47C0_BD7C_804161AEC451__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include <vector>
using namespace std;
//#include "serialport.h"                                   // Serial Communication
#include "SerialCtrl.h"      // Hinzugefügt von der Klassenansicht


const unsigned char DEF_INIT = 150;

class CSerialCom
{
public:
	int ConvertToRange(int Servo, int Position);
	void ReadInServerValues();
	void Move(int servo, int position);
	char ReadByte();
	void SendByte(char Buffer);
	void CloseCOM();
	int OpenCOM(char *port);

/*
Version Change to a more general one 5/6/2004


	CSerialPort SerialCOM;                                  // create a serial communication object
*/


	// Connection scheme of the servos

	enum BoardChannels
                                {
                                        Eyes = 1,
                                        Jaw,
                                        NeckTilt,
                                        NeckLateral,
                                        ShoulderRight,
                                        ArmRightTurn,
                                        ForeArmRight,
                                        ShoulderLeft,
                                        ArmLeftTurn,
                                        ForeArmLeft,
```

```
                                        Body,
                                        Waist,
                                        RightLeg,
                                        LeftLeg

                                };
/*      enum
        {
                                        Eyes = 1,
                                        Jaw,
                                        NeckTilt,
                                        NeckLateral,
                                        ShoulderRight,
                                        ArmRightTurn,
                                        ForeArmRight,
                                        ShoulderLeft,
                                        ArmLeftTurn,
                                        ForeArmLeft,
                                        Body,
        }
*/
        int     ComplexBehavior(char *szBehavior);
                                        // makes a behavior just from a given string as parameter
        int     ComUndoSeveralMovements(char HowMany);
                                        // undoes the given amount of movements
        int     GetAnalogueInput(int InputNum);
                                // read from the anolog input of the ASC16 board
        int  ComTurnUpperArmRight(int Pos, char Speed=DEF_INIT, char Accel=DEF_INIT);
        int  ComTurnUpperArmLeft(int Pos, char Speed=DEF_INIT, char Accel=DEF_INIT);
        int  ComMvForearmRight(int Pos, char Speed=DEF_INIT, char Accel=DEF_INIT);
        int  ComMvForearmLeft(int Pos, char Speed=DEF_INIT, char Accel=DEF_INIT);
        int  ComMvUpperArmRight(int Pos, char Speed=DEF_INIT, char Accel=DEF_INIT);
        int  ComMvUpperArmLeft(int Pos, char Speed=DEF_INIT, char Accel=DEF_INIT);
        int  ComMvNeckLateral(int Pos, char Speed=DEF_INIT, char Accel=DEF_INIT);
        int  ComMvNeck(int Pos, char Speed=DEF_INIT, char Accel=DEF_INIT);
        int  ComMvEyes(int Pos, char Speed=DEF_INIT, char Accel=DEF_INIT);
        int  ComMvJaw(int Pos, char Speed=DEF_INIT, char Accel=DEF_INIT);
        int  ComMvBody(int Pos, char Speed = DEF_INIT, char Accel=DEF_INIT);
        int  ComLoadAcceleration(BoardChannels Servo);
        int  ComLoadSpeed(BoardChannels Servo);
        int  ComLoadPosition(BoardChannels Servo);
        int  ComGetAcceleration(BoardChannels Servo) const;
        int  ComSetAcceleration(BoardChannels Servo, char Accel=DEF_INIT);
        int  ComSetSpeed(BoardChannels Servo, char Speed=DEF_INIT);
        int  ComGetSpeed(BoardChannels Servo);
        int  ComGetPosition(BoardChannels Servo);
        int  ComMoveRelative(BoardChannels Servo, int RelPos, char Speed=DEF_INIT, char Accel=DEF_INIT);
        int  ComMoveAbsolute(BoardChannels Servo, int Position, char Speed=DEF_INIT, char
Accel=DEF_INIT);
        int  ComGetErrorMsg(char *lpszError);
        int  ComGoToInitPos();

        //      the following functions are old function which were used in the first function
        //  they have the same name here just for the sake of compatibility

/*      char ReadByte();                                        // reads a byte from serial interface
```

```
                                                                    // returns -1 if error
occured, otherwise
                                                                    // the recieved bit
        void SendByte(char Buffer);                         // sends a byte thru the serial interface
        void CloseCOM();                                    // closes serial interface
        int OpenCOM(char *port);                            // opens the serial interface (RS232)
                                                                    // returns zero, if
failed
*/

        void ComLoadServoPositionFromFlash();
        void ComSaveServoPositionsToBoard();
        CSerialCom();
        virtual ~CSerialCom();

private:
        void ReadInLineByLine(char line[200], int input_count);
        CSerialCtrl CommObj;
        char *lpszErrorMsg;
        int nErrorMsgLength;
        char HistryCount;



        // max_min range of all servos
        // for initialization look to constructor

        int nEyesMin;
        int nEyesMax;
        int nJawMin;
        int nJawMax;
        int nNeckTiltMin;
        int nNeckTiltMax;
        int nNeckLateralMin;
        int nNeckLateralMax;
        int nShoulderRightMin;
        int nShoulderRightMax;
        int nArmRightTurnMin;
        int nArmRightTurnMax;
        int nForeArmRightMin;
        int nForeArmRightMax;
        int nShoulderLeftMin;
        int nShoulderLeftMax;
        int nArmLeftTurnMin;
        int nArmLeftTurnMax;
        int nForeArmLeftMin;
        int nForeArmLeftMax;
        int nBodyRight;
        int nBodyLeft;


        vector <int>    InitPos;                    // initial positions for servos
        vector <char>   InitSpeed;                  // initial speed of the servos
        vector <char>   InitAccel;                  // initial acceleration of the servos
        vector <char>   CurPos;                     // stores current servo positions
        vector <char>   CurSpeed;                   // current servo speed
        vector <char>   CurAccel;                   // current Acceleration
```

```
        vector <int>    PrevPos;                        // stores previous servo positions
        vector <char>   PrevSpeed;                      // previous Speed settings
        vector <char>   PrevAccel;                      // previous Acceleration settings

        vector <int>    HistPos;                // stack, represents a kinda history of moves
        vector <char>   HistServo;                      // stack, history of which servos were in use
        vector <char>   HistSpeed;                      // same with speed
        vector <char>   HistAccel;                      // and with acceleration

        BoardChannels Servo;


protected:


        bool IsAccelerationSet;
        bool IsSpeedSet;
        bool IsServoEnabled;
        bool bComInitialized;

        void ComSetErrorMsg(char *lpszError, int length);
        void ComServosOff();
        void ComServosOn();
        void StopCom();
        void StartCom(char *COMport, int ModuleID);
        int  IntASC16(int ModuleID);

};

#endif // !defined(AFX_SERIALCOM_H__7637CC97_83F4_47C0_BD7C_804161AEC451__INCLUDED_)
```

The Robot Control Class - Implementation

```
/********************************************************************************
   filename                             :       SerialCom.cpp
        ------------------------------------------------------
   begin                                :       Fri May 7 2004
        last change                             :       5/8/2004
   copyright                            :       (C) 2004 by Stefan Gebauer
   email                                :       gebi@ece.pdx.edu
        ------------------------------------------------------
        Description:
        ============

        This class wrapps the communciation with the serial port / ASC 16 board.
        The file also describes behaviors. This is achieved by a combination of
        servo movements.


        History:
        --------

        Dezember 2003:
        Begin to develop a simple Comunication Class

        January 2004:
        The first Version was working with a dll driver
```

```
        March 2004:
        Start to develop an own driver; extension of the ASC16 commands

        May 7th 2004:
        New Version completed; Driver is working, ASC16 control as well,
        everything was presented on Intel ISEF Contest
        - more functions were added, old remained to be compatible to older
          versions.

        June 7th 2004:
        New Driver was developed, much simpler, than the one before. It turned
        out the prvious driver wasn't working with every computer.
        - Changes were made to stay compatible to previous versions

        What has to be done for the future:

        - Put servo ranges to the ini-file for the whole program
        - Add Comunication settings to configuration dialog
        - Random Behavior Generator
        - figure out a way how to synchronize speech with movement of lips
        - Thread Programming: Some movements over time
        - check system with error msgs!
        - need a function that:
                * checks Ranges for every Servo
                * normalizes movement to a Range between 0 and 1000
        - History to MoveRelative
        - Concept of Global Speed

 ***************************************************************************/


#include "stdafx.h"
#include "SerialCom.h"
#include "SerialCtrl.h"                                         // RS232 communication


#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif


// Global Variables --------

//Structure for Servo.ini
struct servo_values
{
        int servo_num;
        int servo_pos_low;
        int servo_pos_high;
        int servo_init;
};
servo_values servo_val[16];
```

```
//Saves the number of variables
int behav_cnt=0;



struct movement
{
	int servo_array[16];		//int array with the moved servo number
	int servo_pos_array[16];   //int array with desired servo position info
	int speed;					//speed of movement
	int accel;					//accelaration of movement
	int delay;					//delay in ms after movement
	int servo_num;				//saves the number of servos that are in use for a particular
movement
};


struct behavior
{
	char name[30];
	movement mov[20];		//array of max 20 movements in on behavior
	int mov_cnt;
};
//behavoirs[30];

behavior behaviors[30];

// Global Variables --- END -------



/*
============================================================================

	Constructor

============================================================================
*/

CSerialCom::CSerialCom():bComInitialized(false)
{
	// set serial port status:


	StartCom("COM2", 255);				// open port, first is com-port
	IsServoEnabled = false;				// no servo active
	IsAccelerationSet = false;		// no general value for acceleration


	lpszErrorMsg = NULL;				// no Error Msg

	HistryCount = 0;					// history of movements counter set to zero;

	// set servo ranges (you may change these if you change the mechanically design)

	nEyesMin			= 700;		// right
	nEyesMax			= 2500;		// left
```

```
nJawMin                         = 1250;         // close
nJawMax                         = 1700;         // open

nNeckTiltMin        = 600;              // down
nNeckTiltMax        = 3200;             // up

nNeckLateralMin             = -200;             // left
nNeckLateralMax             = 1400;             // right

nShoulderRightMin   = 400;              // up
nShoulderRightMax   = 3500;             // down

nArmRightTurnMin    = 45;               // most outer position
nArmRightTurnMax    = 3400;             // most inner position

nForeArmRightMin    = -700;             // down
nForeArmRightMax    = 2800;             // up

nShoulderLeftMin    = 50;               // down
nShoulderLeftMax    = 3500;             // up

nArmLeftTurnMin             = 500;              // most inner position
nArmLeftTurnMax             = 3700;             // most outer position

nForeArmLeftMin             = 1;                // up
nForeArmLeftMax             = 3200;             // down

nBodyLeft                   = 0;                // left
nBodyRight                  = 2000;             // right side



        // set initial positions of all servos (Range 0..4000)
//      InitPos.push_back(1000);
//      InitPos.push_back(300);
//      InitPos.push_back(300);
        InitPos.at(Eyes - 1)                =   1000;                   // middle
        InitPos[Jaw - 1]            =   50;                     // closed
        InitPos[NeckTilt - 1]       =   1000;                   // middle
        InitPos[NeckLateral - 1]    =   1000;                   // middle
        InitPos[ShoulderRight - 1]  =   100;                    // straight down
        InitPos[ArmRightTurn - 1]   =   1000;                   // middle = no turn
        InitPos[ForeArmRight - 1]   =   300;                    // a little bit bended
        InitPos[ShoulderLeft - 1]   =   100;                    // like the right side
        InitPos[ArmLeftTurn - 1]    =   1000;                   // like the right side
        InitPos[ForeArmLeft - 1]    =   300;                    // like the right side
        InitPos[Body - 1]           =   1000;                   // in the middle

        // set initial speeds of all servos (Range 0..255)

        InitSpeed[Eyes - 1]                 =   (char)255;
        InitSpeed[Jaw - 1]                  =   (char)255;
        InitSpeed[NeckTilt - 1]             =   (char)255;
        InitSpeed[NeckLateral - 1]          =   (char)255;
        InitSpeed[ShoulderRight - 1]    =   (char)255;
```

```
	InitSpeed[ArmRightTurn - 1]		=	(char)255;
	InitSpeed[ForeArmRight - 1]		=	(char)255;
	InitSpeed[ShoulderLeft - 1]		=	(char)255;
	InitSpeed[ArmLeftTurn - 1]		=	(char)255;
	InitSpeed[ForeArmLeft - 1]		=	(char)255;
	InitSpeed[Body - 1 ]			=	(char)255;


	// set initial accelerations of all servos (Range 0..255)


	InitAccel[Eyes - 1]				=	(char)255;
	InitAccel[Jaw - 1]				=	(char)255;
	InitAccel[NeckTilt - 1]			=	(char)255;
	InitAccel[NeckLateral - 1]		=	(char)255;
	InitAccel[ShoulderRight - 1]	=	(char)255;
	InitAccel[ArmRightTurn - 1]		=	(char)255;
	InitAccel[ForeArmRight - 1]		=	(char)255;
	InitAccel[ShoulderLeft - 1]		=	(char)255;
	InitAccel[ArmLeftTurn - 1]		=	(char)255;
	InitAccel[ForeArmLeft - 1]		=	(char)255;
	InitAccel[Body - 1]				=	(char)255;

	IsAccelerationSet = true;				// general value for
acceleration set

	//ComGoToInitPos();					// finally
go to the initialized positions

	ReadInServerValues();
	int temp=0;
	int i=0;

	for(i=0;i<13;i++)
		Move(i+1,servo_val[i].servo_init);

}



/*
============================================================================

	Destructor

============================================================================
*/

CSerialCom::~CSerialCom()
{
		if (lpszErrorMsg != NULL)

		{
			delete [] lpszErrorMsg;
			lpszErrorMsg = NULL;
```

```
                }

                if (bComInitialized) CloseCOM();


}

/*
================================================================================

        Communication with serial Driver

================================================================================
*/



//----------------------------------------
// Open COM port for serial communication
//----------------------------------------


int CSerialCom::OpenCOM(char *port)                     // opens the serial interface (RS232)
{                                                                                       // returns zero, if
failed

                                                                                        // open port
  if (!CommObj.OpenPort(port))
  {
          return 0;
  }
  bComInitialized = true;
  return 1;

}


//---------------
// Close COM port
//---------------


void CSerialCom::CloseCOM()                     // closes serial interface
{
  if (!bComInitialized)
  {
                AfxMessageBox("Communication is not initialized!", MB_OK | MB_ICONSTOP);
  }
  else
  {
                if (CommObj.ClosePort())
            bComInitialized = false;
        }
}
```

```
//--------------------------------------
// Read a byte fron the opened COM port
//--------------------------------------

char CSerialCom::ReadByte()                         // reads a byte from serial interface
{                                                                                       // returns -1 if error occured,
otherwise
                                                                                        // the recieved bit
        char *szRecieved = NULL;
        const unsigned int buf = 1;
        unsigned long length = 1;

        if (CommObj.ReadSerialPort(szRecieved, buf, length))
                return *szRecieved;
        else return (char)(-1);


}


//-----------------------------------
// Send a byte to the opened COM port
//-----------------------------------

void CSerialCom::SendByte(char Buffer)              // sends a byte thru the serial interface
{
        const unsigned int buf = 1;
        unsigned long length = 1;

        CommObj.WriteToSerialPort(&Buffer,buf,length);
}


/*
================================================================================

        Initialize ASC16 Board

================================================================================
*/

int CSerialCom::IntASC16(int ModuleID)
{
        char send = 0;
/*

  Version Change to a more general one 5/6/2004

        DWORD dwErrors;
        SerialCOM.ClearError(dwErrors);
        SerialCOM.TerminateOutstandingWrites();

        //terminate
        SerialCOM.TransmitChar(send);
        SerialCOM.TransmitChar(send);
        SerialCOM.TransmitChar(send);
```

```
	send = 121;

	//enable ASC-16 module
	SerialCOM.TransmitChar(send);
	SerialCOM.TransmitChar((char)ModuleID);

*/
	//terminate
	for (int i=0;i<2;i++) SendByte(send);

	send = 121;

	//enable ASC-16 module
	SendByte(send);
	send = (char)ModuleID;
	SendByte(send);

	return 1;
}


/*
============================================================================

	Start and stop Serial communication with the ASC16 Board

============================================================================
*/


void CSerialCom::StartCom(char *COMport, int ModuleID)
{
	/*

  Version Change to a more general one 5/6/2004

	if (bComInitialized) StopCom();

	SerialCOM.Open(COMport, 9600, CSerialPort::NoParity, 8, CSerialPort::OneStopBit,
CSerialPort::NoFlowControl);

	DWORD dwErrors;
	SerialCOM.ClearError(dwErrors);
	SerialCOM.TerminateOutstandingWrites();

	IntASC16(ModuleID);                                    // initalize the ASC16 board

	ComServosOn();                                         // enable Servos



	bComInitialized=true;

  */

	if (bComInitialized) StopCom();
	OpenCOM(COMport);
```

```
	IntASC16(ModuleID);                                    // initalize ASC16 board
	ComServosOn();                                         // enable Servos

	bComInitialized = true;                          // port is open

}

void CSerialCom::StopCom()
{

  ComServosOff();
  /*
Version Change to a more general one 5/6/2004
  SerialCOM.Close();
  */

  CloseCOM();
  bComInitialized=false;

}



/*
================================================================================

	Hook up Servos

================================================================================
*/

//-----------------
// engergise servos
//-----------------

void CSerialCom::ComServosOn()
{

	const unsigned char byte = 245;
/*
	SerialCOM.TransmitChar(byte);
*/
	SendByte(byte);
	IsServoEnabled = true;

}



//---------------
// disable Servos
//---------------

void CSerialCom::ComServosOff()
{
	const unsigned char byte = 246;
```

```
	/*
Version Change to a more general one 5/6/2004
	SerialCOM.TransmitChar(byte);
	*/
	SendByte(byte);
	IsServoEnabled = false;

}



/*
================================================================================

	functions to set up Servos, Speed, Acceleration, Position

================================================================================
*/


//---------------------
// do an absolute move
//---------------------

int CSerialCom::ComMoveAbsolute(BoardChannels Servo, int Position, char Speed, char Accel)
{

	char UpperByte;
	char LowerByte;


	// check range


	switch(Servo)
	{

		case Eyes:
					if ( (Position < nEyesMin) || (Position > nEyesMax))
					{
						// the servo range has been exceeded.
						// set error msg

						ComSetErrorMsg("The position that you are trying to reach is
out of range! (Eyes)", 65);

						return 0;
					}
					break;

		case Jaw:
					if ( (Position < nJawMin) || (Position > nJawMax))
					{
						// the servo range has been exceeded.
						// set error msg
```

```
                                        ComSetErrorMsg("The position that you are trying to reach is
out of range! (Jaw)", 64);

                                        return 0;
                                }
                                break;

        case NeckTilt:

                                if ( (Position < nNeckTiltMin) || (Position > nNeckTiltMax))
                                {
                                        // the servo range has been exceeded.
                                        // set error msg

                                        ComSetErrorMsg("The position that you are trying to reach is
out of range! (NeckTilt)", 69);

                                        return 0;
                                }

                                break;
        case NeckLateral:

                                if ( (Position < nNeckLateralMin) || (Position > nNeckLateralMax))
                                {
                                        // the servo range has been exceeded.
                                        // set error msg

                                        ComSetErrorMsg("The position that you are trying to reach is
out of range! (NeckLateral)", 72);

                                        return 0;
                                }

                                break;
        case ShoulderRight:

                                if ( (Position < nShoulderRightMin) || (Position > nShoulderRightMax))
                                {
                                        // the servo range has been exceeded.
                                        // set error msg

                                        ComSetErrorMsg("The position that you are trying to reach is
out of range! (Shoulder Right)", 75);

                                        return 0;
                                }

                                break;
        case ArmRightTurn:

                                if ( (Position < nArmRightTurnMin) || (Position > nArmRightTurnMax))
                                {
                                        // the servo range has been exceeded.
                                        // set error msg
```

```
                ComSetErrorMsg("The position that you are trying to reach is
out of range! (Arm Right Turn)", 75);

                return 0;
            }

            break;
        case ForeArmRight:

            if ( (Position < nForeArmRightMin) || (Position > nForeArmRightMax))
            {
                // the servo range has been exceeded.
                // set error msg

                ComSetErrorMsg("The position that you are trying to reach is
out of range! (Fore Arm Right)", 75);

                return 0;
            }

            break;
        case ShoulderLeft:

            if ( (Position < nShoulderLeftMin) || (Position > nShoulderLeftMax))
            {
                // the servo range has been exceeded.
                // set error msg

                ComSetErrorMsg("The position that you are trying to reach is
out of range! (Shoulder Left)", 74);

                return 0;
            }

            break;
        case ArmLeftTurn:

            if ( (Position < nArmLeftTurnMin) || (Position > nArmLeftTurnMax))
            {
                // the servo range has been exceeded.
                // set error msg

                ComSetErrorMsg("The position that you are trying to reach is
out of range! (Arm Left Turn)", 74);

                return 0;
            }
            break;
        case ForeArmLeft:

            if ( (Position < nForeArmLeftMin) || (Position > nForeArmLeftMax))
            {
                // the servo range has been exceeded.
                // set error msg

                ComSetErrorMsg("The position that you are trying to reach is
out of range! (Fore Arm Left)", 74);
```

```
                                        return 0;
                                    }
                                    break;
                case Body:

                                    if ( (Position < nBodyLeft) || (Position > nBodyRight))
                                    {
                                        // the servo range has been exceeded.
                                        // set error msg

                                        ComSetErrorMsg("The position that you are trying to reach is
out of range! (Body)", 65);

                                        return 0;
                                    }
                                    break;
                default:
                                    break;
        };


        // separate integer value into lower and upper bytes
        LowerByte = Position & 255;
        UpperByte = Position >> 8;

        // set speed;

        // What is the rule for setting a new speed?
        // Highest priority has a speed value that is being passed to this function.
        // If a value is passed, the new speed value is then activated.
        // Future implementations might use the flag IsSpeedSet to force a global
        // speed, equal for all servos.

        PrevSpeed[Servo-1] = CurSpeed[Servo-1];
        PrevAccel[Servo-1] = CurAccel[Servo-1];
        if (!IsSpeedSet || (Speed != 150))
        {
                // use the given speed, otherwise the
                // current speed
                ComSetSpeed(Servo, Speed);
                // save value to the current speed vector
                CurSpeed[Servo-1] = Speed;                                                    //
a speed value was passed to the

                        // function, therefore IsSpeedSet will be

                        // reseted and the local value for speed

                        // will be used instead.

        }

        // set acceleration

        if (!IsAccelerationSet || (Accel != 150))
        {
```

```
                // use the referenced acceleration
                ComSetAcceleration(Servo, Accel);                                   // same as
speed
                CurAccel[Servo-1] = Accel;

        }



        if (!IsServoEnabled) ComServosOn();

/*
        Version Change to a more general one 5/6/2004

        SerialCOM.TransmitChar(Servo);
        SerialCOM.TransmitChar(UpperByte);
        SerialCOM.TransmitChar(LowerByte);
*/

        SendByte(Servo);
        SendByte(UpperByte);
        SendByte(LowerByte);

        ComServosOff();

        //History Stuff;

        if (HistPos.front() == 0)
        {
                HistPos.at(0) = Position;
        }

        HistPos.push_back(Position);
        HistryCount++;
        // increase history counter

        PrevPos[Servo] = CurPos[Servo];                                             // previous
position
        CurPos[Servo] = Position;

        return Position;

}

int CSerialCom::ComMoveRelative(BoardChannels Servo, int RelPos, char Speed, char Accel)
{
        char UpperByte;
        char LowerByte;

        // RelPos += ComGetPosition(Servo);                 // absolute Pos

          //separate integer value into lower and upper bytes
        LowerByte = RelPos & 255;
        UpperByte = RelPos >> 8;

/*
```

```
Version Change to a more general one 5/6/2004
   SerialCOM.TransmitChar(Servo);
        SerialCOM.TransmitChar(UpperByte);
        SerialCOM.TransmitChar(LowerByte);
*/
        SendByte((char)(Servo + 41));
        SendByte(UpperByte);
        SendByte(LowerByte);
        return RelPos;
}



//-------------------------------------------------------------
// returns the position of the given servo in the range 0-4000
//-------------------------------------------------------------

int CSerialCom::ComGetPosition(BoardChannels Servo)
{
        char buffer[2] = {0, 0};
        const char byte = 116;
        int currentPosition = -1;
        int success = 0;
        int tries = 0;

        while (success == 0)
        {
                /*
Version Change to a more general one 5/6/2004
                //request position
                SerialCOM.TransmitChar(byte);
                SerialCOM.TransmitChar(Servo);
                */

                //request position
                SendByte(byte);
                SendByte(Servo);

                /*
Version Change to a more general one 5/6/2004
                //Wait for data to come back to the serial port
                DWORD dwRead = SerialCOM.Read(buffer, 2);
                */

//              DWORD dwRead = ReadByte(buffer,2);
                char buf = ReadByte();
                currentPosition = atoi(buffer);

                if ((currentPosition > -1) || (tries > 2)) success = 1;
                tries++;

        }

        // DATA-Structure = currentPosition;
        return(currentPosition);
}
```

```
//---------------------------------------------------------
// returns the speed of the given Servo in the range 0-255
//---------------------------------------------------------

int CSerialCom::ComGetSpeed(BoardChannels Servo)
{
	const char byte = 117;
	char buffer[2] = {0, 0};
	int currentSpeed = -1;
	int success = 0;
	int tries = 0;


  while (success == 0)
  {
    /*
Version Change to a more general one 5/6/2004
	//request speed
   SerialCOM.TransmitChar(byte);
   SerialCOM.TransmitChar(Servo);
	*/

	SendByte(byte);
	SendByte(Servo);
   //Wait for data to come back to the serial port
/*
	DWORD dwRead = SerialCOM.Read(buffer, 1);
*/
   //DWORD dwRead = ReadByte(buffer,1);
	char buf = ReadByte();
	currentSpeed = atoi(buffer);

   if ((currentSpeed > -1) || (tries > 2)) success = 1;
   tries++;
  }
  return(currentSpeed);
}



//-------------------------------------
// set servo speed in the range 0-255
//-------------------------------------

int CSerialCom::ComSetSpeed(BoardChannels Servo, char Speed)
{

	/*
Version Change to a more general one 5/6/2004
	SerialCOM.TransmitChar((char)(61 + Servo));
	SerialCOM.TransmitChar(Speed);
	*/
	SendByte((char)(61 + Servo));
	SendByte(Speed);
```

```
	//HistSpeed

	//vector<char>::iterator iSpeedHist = HistSpeed.begin();

	if (!IsSpeedSet)
		CurSpeed[Servo-1]=Speed;

	if (HistSpeed.front() == 0)
	{
		HistSpeed.at(0) = Speed;
	}
	HistSpeed.push_back(Speed);

	IsSpeedSet = true;
	return Speed;
}



//-----------------------------------------
// set servo acceleration in the range 0-255
//-----------------------------------------

int CSerialCom::ComSetAcceleration(BoardChannels Servo, char Accel)
{

		/*
Version Change to a more general one 5/6/2004

	SerialCOM.TransmitChar((char)(81 + Servo));
	SerialCOM.TransmitChar((char)Accel);
*/
	SendByte((char)(81 + Servo));
	SendByte((char) Accel);
	if (HistAccel.front() == 0)
	{
		HistAccel.at(0) = Accel;
	}

	if (!IsAccelerationSet)
	CurAccel[Servo -1] = Accel;

	IsAccelerationSet = true;
	return Accel;                                                                    //
successful
}



//-------------------------------------------------
// returns current settings for Servo Acceleration
//-------------------------------------------------

int CSerialCom::ComGetAcceleration(BoardChannels Servo) const
{
```

```
	return 1;                                                    // change
to acceleration
}



//-----------------------------------------------------
// Loads the previous servo position and returns to it
//-----------------------------------------------------

int CSerialCom::ComLoadPosition(BoardChannels Servo)
{

	return 1;                                                    //
successful
}


//---------------------------------------
// Loads the previous servo speed setting
//---------------------------------------

int CSerialCom::ComLoadSpeed(BoardChannels Servo)
{

	return 1;                                                    // change
to Speed
}



//-----------------------------------------------
// Loads the previous servo Acceleration setting
//-----------------------------------------------

int CSerialCom::ComLoadAcceleration(BoardChannels Servo)
{

	return 1;                                                    // change
to acceleration
}



//-------------------------------------------------------------------------
// Loads the initial settings and returns all servos to its initial positions
//-------------------------------------------------------------------------

int CSerialCom::ComGoToInitPos()
{

	// set the speed for all servos
	ComSetSpeed(Eyes , InitSpeed[Eyes - 1]);
	ComSetSpeed(Jaw, InitSpeed[Jaw-1]);
	ComSetSpeed(NeckTilt, InitSpeed[NeckTilt-1]);
	ComSetSpeed(NeckLateral, InitSpeed[NeckLateral-1]);
	ComSetSpeed(ShoulderRight, InitSpeed[ShoulderRight-1]);
```

```
	ComSetSpeed(ArmRightTurn, InitSpeed[ArmRightTurn-1]);
	ComSetSpeed(ForeArmRight, InitSpeed[ForeArmRight-1]);
	ComSetSpeed(ShoulderLeft, InitSpeed[ShoulderLeft-1]);
	ComSetSpeed(ArmLeftTurn, InitSpeed[ArmLeftTurn-1]);
	ComSetSpeed(ForeArmLeft, InitSpeed[ForeArmLeft-1]);
	ComSetSpeed(Body, InitSpeed[Body-1]);

	// set initial acceleration of all servos
	ComSetAcceleration(Eyes , InitAccel[Eyes - 1]);
	ComSetAcceleration(Jaw, InitAccel[Jaw-1]);
	ComSetAcceleration(NeckTilt, InitAccel[NeckTilt-1]);
	ComSetAcceleration(NeckLateral, InitAccel[NeckLateral-1]);
	ComSetAcceleration(ShoulderRight, InitAccel[ShoulderRight-1]);
	ComSetAcceleration(ArmRightTurn, InitAccel[ArmRightTurn-1]);
	ComSetAcceleration(ForeArmRight, InitAccel[ForeArmRight-1]);
	ComSetAcceleration(ShoulderLeft, InitAccel[ShoulderLeft-1]);
	ComSetAcceleration(ArmLeftTurn, InitAccel[ArmLeftTurn-1]);
	ComSetAcceleration(ForeArmLeft, InitAccel[ForeArmLeft-1]);
	ComSetAcceleration(Body, InitAccel[Body-1]);

	// move all servos to its initial positions
	ComMoveAbsolute(Eyes, InitPos[Eyes - 1], InitSpeed[Eyes - 1]);
	ComMoveAbsolute(Jaw, InitPos[Jaw-1], InitSpeed[Jaw - 1]);
	ComMoveAbsolute(NeckTilt, InitPos[NeckTilt-1], InitSpeed[NeckTilt - 1]);
	ComMoveAbsolute(NeckLateral, InitPos[NeckLateral-1], InitSpeed[NeckLateral - 1]);
	ComMoveAbsolute(ShoulderRight, InitPos[ShoulderRight-1], InitSpeed[ShoulderRight - 1]);
	ComMoveAbsolute(ArmRightTurn, InitPos[ArmRightTurn-1], InitSpeed[ArmRightTurn - 1]);
	ComMoveAbsolute(ForeArmRight, InitPos[ForeArmRight-1], InitSpeed[ForeArmRight - 1]);
	ComMoveAbsolute(ShoulderLeft, InitPos[ShoulderLeft-1], InitSpeed[ShoulderLeft - 1]);
	ComMoveAbsolute(ArmLeftTurn, InitPos[ArmLeftTurn-1], InitSpeed[ArmLeftTurn - 1]);
	ComMoveAbsolute(ForeArmLeft, InitPos[ForeArmLeft-1], InitSpeed[ForeArmLeft - 1]);
	ComMoveAbsolute(Body, InitPos[Body-1], InitSpeed[Body - 1]);

	return 1;
}



/*
================================================================================

	functions to move head servos

================================================================================
*/


//--------------------------------------------------------------------------
// Opens/closes the Jaw, angle is given in Pos (Range 0 - 4000), Speed of
// movement is given in parameter Speed. Return value is the new position.
//--------------------------------------------------------------------------

int CSerialCom::ComMvJaw(int Pos, char Speed, char Accel)
{
	if (ComMoveAbsolute(Jaw, Pos, Speed, Accel))
	return Pos;
```

```
	else return 0;
}



//---------------------------------------------------------------------------
// Moves eyes to the left/right, angle is given in Pos (Range 0 - 4000),
// Speed of movement is given in parameter Speed. Return value is the new position.
//---------------------------------------------------------------------------

int CSerialCom::ComMvEyes(int Pos, char Speed, char Accel)
{
	if (ComMoveAbsolute(Eyes, Pos, Speed, Accel))
	return Pos;
	else return 0;
}



//---------------------------------------------------------------------------
// Moves neck up/down. Angle is given in Pos (Range 0 - 4000),
// Speed of movement is given in parameter Speed. Return value is the new position.
//---------------------------------------------------------------------------

int CSerialCom::ComMvNeck(int Pos, char Speed, char Accel)
{
	if (ComMoveAbsolute(NeckTilt, Pos, Speed, Accel))
	return Pos;
	else return 0;
}



//---------------------------------------------------------------------------
// Moves neck to the left/right. Angle is given in Pos (Range 0 - 4000),
// Speed of movement is given in parameter Speed. Return value is the new position.
//---------------------------------------------------------------------------

int CSerialCom::ComMvNeckLateral(int Pos, char Speed, char Accel)
{
	if (ComMoveAbsolute(NeckLateral, Pos, Speed, Accel))
	return Pos;
	else return 0;
}



/*
============================================================================

	functions to move arms

============================================================================
*/
```

```
//---------------------------------------------------------------------------
// Moves robot's left upper arm up/down. Angle is given in Position (Range 0 - 4000),
// Speed of movement is given in parameter Speed. Return value is the new position.
//---------------------------------------------------------------------------


int CSerialCom::ComMvUpperArmLeft(int Pos, char Speed, char Accel)
{
	if (ComMoveAbsolute(ShoulderLeft, Pos, Speed, Accel))
	return Pos;
	else return 0;
}



//---------------------------------------------------------------------------
// Moves robot's right upper arm up/down. Angle is given in Position (Range 0 - 4000),
// Speed of movement is given in parameter Speed. Return value is the new position.
//---------------------------------------------------------------------------

int CSerialCom::ComMvUpperArmRight(int Pos, char Speed, char Accel)
{
	if (ComMoveAbsolute(ShoulderRight, Pos, Speed, Accel))
	return Pos;
	else return 0;
}



//---------------------------------------------------------------------------
// Moves robot's left fore arm up/down. Angle is given in Position (Range 0 - 4000),
// Speed of movement is given in parameter Speed. Return value is the new position.
//---------------------------------------------------------------------------

int CSerialCom::ComMvForearmLeft(int Pos, char Speed, char Accel)
{
	if (ComMoveAbsolute(ForeArmLeft, Pos, Speed, Accel))
	return Pos;
	else return 0;
}



//---------------------------------------------------------------------------
// Moves robot's right fore arm up/down. Angle is given in Position (Range 0 - 4000),
// Speed of movement is given in parameter Speed. Return value is the new position.
//---------------------------------------------------------------------------

int CSerialCom::ComMvForearmRight(int Pos, char Speed, char Accel)
{
	if (ComMoveAbsolute(ForeArmRight, Pos, Speed, Accel))
	return Pos;
	else return 0;
}
```

```
//-------------------------------------------------------------------------------------------------
// Turns robot's left arm towards the body or perpendicular to the body. Basically, this
// movement rotates the forearm to a particular angle. The Angle is given in Position
// (Range 0 - 4000), Speed of movement is given in parameter Speed. Return value is the new position.
//-------------------------------------------------------------------------------------------------


int CSerialCom::ComTurnUpperArmLeft(int Pos, char Speed, char Accel)
{
	if (ComMoveAbsolute(ArmLeftTurn, Pos, Speed, Accel))
	return Pos;
	else return 0;
}




//-------------------------------------------------------------------------------------------------
// Turns robot's right arm towards the body or perpendicular to the body. Basically, this
// movement rotates the forearm to a particular angle. The Angle is given in Position
// (Range 0 - 4000), Speed of movement is given in parameter Speed. Return value is the new position.
//-------------------------------------------------------------------------------------------------


int CSerialCom::ComTurnUpperArmRight(int Pos, char Speed, char Accel)
{
	if (ComMoveAbsolute(ArmRightTurn, Pos, Speed, Accel))
	return Pos;
	else return 0;
}


//-----------------------------------------------------------------------------
// Turns the robot's body to the left/right. The return value is the new position.
//-----------------------------------------------------------------------------

int CSerialCom::ComMvBody(int Pos, char Speed, char Accel)
{
	if (ComMoveAbsolute(Body, Pos, Speed, Accel))
	return Pos;
	else return 0;
}




//-----------------------------------------------------------------------------
// ComplexBehavior is a function that combines smaller function which control
// only a single servo (DOF) together in order to create a meaningful Gesture.
//-----------------------------------------------------------------------------


int CSerialCom::ComplexBehavior(char *szBehavior)
{

	return 1;
```

```
}



//---------------------------------------------------------------------------
// save servo positions to flash (look to the manual for detailed description)
// ensure that the servos are off before calling this function
//---------------------------------------------------------------------------

void CSerialCom::ComSaveServoPositionsToBoard()
{
	const char byte = (char)241;
	// Version Change to a more general one 5/6/2004
	// SerialCOM.TransmitChar(byte);
	SendByte(byte);
}



//---------------------------------------------------------------------------
// load servo positions from flash (look to the manual for detailed description)
// ensure that the servos are off before calling this function
//---------------------------------------------------------------------------

void CSerialCom::ComLoadServoPositionFromFlash()
{
	const char byte = (char)242;
	// Version Change to a more general one 5/6/2004
	// SerialCOM.TransmitChar(byte);
	SendByte(byte);
}

//---------------------------------------------------------------------
// returns the analogue value of the given input in the range 0-255
//---------------------------------------------------------------------

int CSerialCom::GetAnalogueInput(int InputNum)
{

	char buffer[1];
	const int AnIn = 141;
	int array[1];

	//request input
	/*
	SerialCOM.TransmitChar((char)(AnIn + InputNum));
	*/
	SendByte((char)(AnIn + InputNum));
	//Wait for data to come back to the serial port
	buffer[0]=0;
	/*
	DWORD dwRead = SerialCOM.Read(buffer, 1);
   */
	//DWORD dwRead = ReadByte(buffer, 1);
	char buf = ReadByte();
	array[1] = atoi(buffer);
	return(array[1]);
```

```
}


//--------------------------------------------------------------------
// This function sets a error message that might occur while using
// memberfunctions of the class. The lenght of the error message is
// stored in nErrorMsgLength, the actual Message in lpszErrorMsg.
// In order to get the Error, please refer to ComGetErrorMsg.
//--------------------------------------------------------------------

void CSerialCom::ComSetErrorMsg(char *lpszError, int length)
{
	nErrorMsgLength = length;

	if (lpszErrorMsg != NULL)
	{
		delete[] lpszErrorMsg;
		lpszErrorMsg = NULL;
	}

	for (int i=0;i<length;i++);
	{
		lpszErrorMsg = new char;
		lpszErrorMsg = lpszError;
		lpszError++;
	}

}



//--------------------------------------------------------------------
// This function retrieves the error messages that might occur while
// using the class. The lenght of the error message is the return
// value.
//--------------------------------------------------------------------

int CSerialCom::ComGetErrorMsg(char *lpszError)
{
	int length;

	lpszError = lpszErrorMsg;

	length = nErrorMsgLength;
	return length;
}



//--------------------------------------------------------------------
// Function is used to undo up to 30 single movements. It can be used
// to undo more than one gesture.
//--------------------------------------------------------------------

int CSerialCom::ComUndoSeveralMovements(char HowMany)
{
```

```
	if ((HowMany > 30) || (HowMany > HistryCount))
	{
		// violation
		ComSetErrorMsg("The amount of movements to undo is out of range", 49);
		return 0;
	}
	else
	{

		for (char i=0; i<HowMany; i++)
		{
			// undo the last moves


			// decrease the history
			HistryCount--;
		}
		return 1;
	}

}


void CSerialCom::Move(int servo, int position)
{
	//variables
	int nUpperByte;
	int nLowerByte;

	// this function takes the servonumber and the desired position and executes it.
	SendByte(servo);			// we call the SendByte function

	nLowerByte = position & 255;
	nUpperByte = position >> 8;

	SendByte(nUpperByte);		// position component 1
	SendByte(nLowerByte);		// position component 2
}

void CSerialCom::ReadInLineByLine(char line[], int input_count)
{
  int i=0,j=0;
  char temp[6];

	servo_val[input_count].servo_num=input_count+1;				//saves servonumber, to
make avaiable in the hole program

	while(line[i]!=':')
		i++;
	i++;

	while(line[i]==' ')
		i++;

	while(line[i]!=' ')
	{
		temp[j]=line[i];
```

```
		j++;i++;
	}
	temp[j]='\0';j=0;
	servo_val[input_count].servo_pos_low= atoi(temp);

	while(line[i]==' ')
	i++;

	while(line[i]!=' ')
	{
		temp[j]=line[i];
		j++;i++;
	}
	temp[j]='\0';j=0;
	servo_val[input_count].servo_init= atoi(temp);
		while(line[i]==' ')
		i++;

	while(line[i]!=' ' && line[i]!='\0')
	{
		temp[j]=line[i];
		j++;i++;
	}
	temp[j]='\0';j=0;
	servo_val[input_count].servo_pos_high= atoi(temp);
	i=0;
}

void CSerialCom::ReadInServerValues()
{
	FILE *servo_file;
	char line_buffer[200];
	int i=0,j=0;
	int input_cnt=0;


	if((servo_file=fopen("servo.ini","r"))==NULL)
	{
		AfxMessageBox("Can't open 'servo.ini' !",MB_OK);
		exit(0);
	}

	while(!feof(servo_file))
	{

		fgets(line_buffer,199,servo_file);
		strlwr(line_buffer);

		if((strncmp(line_buffer,"eyes",4))==NULL)
			ReadInLineByLine(line_buffer,0);

		if((strncmp(line_buffer,"mouth",5))==NULL)
			ReadInLineByLine(line_buffer,1);

		if((strncmp(line_buffer,"neck_vertical",13))==NULL)
			ReadInLineByLine(line_buffer,2);
```

```
		if((strncmp(line_buffer,"neck_horizontal",15))==NULL)
			ReadInLineByLine(line_buffer,3);

		if((strncmp(line_buffer,"right_shoulder",14))==NULL)
			ReadInLineByLine(line_buffer,4);

		if((strncmp(line_buffer,"right_arm",9))==NULL)
			ReadInLineByLine(line_buffer,5);

		if((strncmp(line_buffer,"right_elbow",11))==NULL)
			ReadInLineByLine(line_buffer,6);

		if((strncmp(line_buffer,"left_shoulder",13))==NULL)
			ReadInLineByLine(line_buffer,7);

		if((strncmp(line_buffer,"left_arm",8))==NULL)
			ReadInLineByLine(line_buffer,8);

		if((strncmp(line_buffer,"left_elbow",10))==NULL)
			ReadInLineByLine(line_buffer,9);

		if((strncmp(line_buffer,"waist",5))==NULL)
			ReadInLineByLine(line_buffer,10);

		if((strncmp(line_buffer,"right_leg",9))==NULL)
			ReadInLineByLine(line_buffer,11);

		if((strncmp(line_buffer,"left_leg",8))==NULL)
			ReadInLineByLine(line_buffer,12);

		i++;
	}

	fclose(servo_file);


}

int CSerialCom::ConvertToRange(int Servo, int Position)
{
  // converts a position which is given between 0 and 1000 to the
  // physical servo range which was read in before

  //Mouth
	int slider_value=0;
	int max=1000;
	int servo_range=0;
	int servo_min=0,servo_max=0;
	int converted_value=0;


	servo_min = servo_val[Servo - 1].servo_pos_low;				//these values have to read-
in at the beginning at the program
	servo_max = servo_val[Servo - 1].servo_pos_high;		//these values have to read-in at the
beginning at the program
	servo_range = servo_max - servo_min;
```

```
	converted_value = (servo_range*Position/max) + servo_min;


	return converted_value;

}
```

The Fonix Wrapper Class – Header File

```
/*
==========================================================================
Stefan Gebauer, Portland 04/13/2004

This class wrapps the Fonix ASR library. The goal is to get easier access to
the Fonix API recognition function and clean up the source code in TabSpeech.cpp

Version 0.1 (04/13/2004)
==========================================================================

*/

#include "./Fonix/FonixCore.h"                                  // Fonix Support

#if !defined(AFX_FNXASR_H__886819FD_73B7_4AB8_BCF1_405DC40DF1D2__INCLUDED_)
#define AFX_FNXASR_H__886819FD_73B7_4AB8_BCF1_405DC40DF1D2__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000



class CFnxASR
{


// member variables (private)

// variables that describe location of Fonix SDK 2.1
/*
	int nInstLen;
	int nRelSharedLen;
	int nSharedPathLen;
	int nGerLen;
	int nUSEngLen;
	int nUKEngLen;
	int nGNNETLen;
	int nDNNETLen;
	int nASRLen;
	int nRulesLen;
	int nGrammarLen;
	int nLangLen;
	int nCurLen;
*/

	char *szInstallDir;                                             // basic
SDK installation path
```

```
	char *szSharedRel;                                  // relative
shared path
	char *szSharedPath;                                 // shared
path = current project path
	char *szGerman;                                     // relative
language directory for German
	char *szUSEng;                                      // relative
language directory for US English
	char *szUKEng;                                      // relative
language directory for UK English
	char *szGEN_NNET;                                   // general
neural net file, language dependent
	char *szDIG_NNET;                                   // neural
net file for numbers, language dependent
	char *szASRFile;                                    // Speech
Recognition Dictionary, language depandent
	char *szRulesFile;                                  // Rules for
Speech Recognition, language dependent
	char *szGrammarFile;                                //
Grammar file for Speech Recognition, language dependent
	char *szLang;                                       // contains
all language settings
	char *szCurLang;                                    // contains
the language which is in use.

	// German UKEngl or USEngl

	bool bWordProb;                                     // controls
the printout of the WordScore
	int nWordScore;
	int nGarbageScore;
	int nWords;                                                 //
number of words for szSpotWords
	int nDigitNodeID;                                   // node ID
for numbers
	int nGenNodeID;                                     // node ID
for word spotter

	FnxCore pCore;                                      // Fonix
Core Pointer
	char *szError;                                      // stores
Error messages in that string
	char *szWord;                                       //
recognized word
	char *szDigitGrammar;                               // holds grammar
data for numbers;
	char *szGrammar;                                    // grammar
for general recognizer;
	char *szSpotWord;                                   // holds a
set of words


public:
	int GetStringLength(char *StringToCount);
	CFnxASR();                                                  //
standard constructor
```

```
	virtual ~CFnxASR();                                                    //
destructor

	// public member functions
	// Path operations
	char FnxASRGetSDKpath() const;                                  // get the SDK installation
path
	bool FnxASRSetSDKpath(char SDK_PATH);                           // set the SDK inatallation
path
	char FnxASRGetSharedPath() const;                               // get project path
	bool FnxASRSetSharedPath(char szPath);                  // set project path
	char FnxASRGetGermanPath() const;                               // get relative path to
German language files
	bool FnxASRSetGermanPath(char szPath);                  // set relative path to German
language files
	char FnxASRGetUSEngPath() const;                                // get relative path to US
English language files
	bool FnxASRSetUSEngPath(char szPath);                   // set relative path to US English
language files
	char FnxASRGetUKEngPath() const;                                // get relative path to UK
English language files
	bool FnxASRSetUKEngPath(char szPath);                   // set relative path to UK English
language files
	char FnxASRGetGNNetFile() const;                                // get the file name of the
general neural net file
	bool FnxASRSetGNNetFile(char szFileName);               // set the file name of the general
neural net file
	char FnxASRGetDNNetFile() const;                                // get the file name of the
digit neural net file
	bool FnxASRSetDNNetFile(char szFileName);               // set the file name of the digit neural
net file
	char FnxASRGetDicFile() const;                                  // get the file name of the
dictionary file
	bool FnxASRSetDicFile(char szFileName);                 // set the file name of the dictionary
file
	char FnxASRGetRulesFile() const;                        // get the file name of the rules file
	bool FnxASRSetRulesFile(char szFileName);               // set the file name of the rules file
	char FnxASRGetGrammarFile() const;                              // get name of current
grammar file
	bool FnxASRSetGrammarFile(char szFileName);             // set file name of grammar file

	// Options
	bool FnxASRRestoreDefaultSettings();                    // set all pathes to its original values
	void FnxASRSetLanguage(char szLanguage);                // set language for speech recognition
	char FnxASRGetLanguage() const;                                         // get current SR
language
	void FnxASRSetSRprobability(bool);

	// Speech Recognition functions

	bool FnxASRSpotWord(char *szWord);                              // retrieves a spotted word



private:
	void RemoveThisChFromString(int place, char *szString);
	void AddChToString(int place, char CharToAdd, char *StrToExtend);
```

```
	void ProcessString(char *StrToProcess, bool bSlash);
};

#endif // !defined(AFX_FNXASR_H__886819FD_73B7_4AB8_BCF1_405DC40DF1D2__INCLUDED_)
```

The latest driver:

```
/********************************************************************************
   filename                                :       SerialCtrl.h
	-----------------------------------------------------
   begin                                   :       Fri June 6 2004
	last change                             :       6/6/2004
   copyright                               :       (C) 2004 by Stefan Gebauer
   email                                   :       gebi@ece.pdx.edu
	-----------------------------------------------------
	Description:
	============

	This class wrapps the communciation with the serial port. It can be used
	to read and write to the PC's serial port

 ********************************************************************************/


// SerialCtrl.h: Schnittstelle für die Klasse CSerialCtrl.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_SERIALCTRL_H__47702D98_0EBC_4EE7_A874_1770AB15B7A5__INCLUDED_)
#define AFX_SERIALCTRL_H__47702D98_0EBC_4EE7_A874_1770AB15B7A5__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CSerialCtrl
{
public:
	bool GetPortStatus();
	CSerialCtrl();
	virtual ~CSerialCtrl();
	bool WriteToSerialPort(const char *szData,                                      // write data to the serial port
	                       const unsigned int& nSizeBuffer,
	                       unsigned long& length);
	bool ReadSerialPort(char *szOutputData,                                         // read from serial port
	                    const unsigned int &nSizeBuffer,
	                    unsigned long& length);
	bool OpenPort(const char *szPortName = "COM1");                 // open a serial port
	bool ClosePort();                                                               // close a
port that was previously opend
```

```
private:
	HANDLE PortHandle;                                                    //
handle to an instance of the port
	bool bPortStatus;                                                     // is the
port opened or not?
	DCB DCB_Settings;                                                     //
structure to configure serial communication
};

#endif // !defined(AFX_SERIALCTRL_H__47702D98_0EBC_4EE7_A874_1770AB15B7A5__INCLUDED_)
```

The latest driver implementation:

```
/********************************************************************************
   filename                    :    SerialCtrl.cpp
	-------------------------------------------------------
   begin                       :    Fri June 6 2004
	last change                 :    6/6/2004
   copyright                   :    (C) 2004 by Stefan Gebauer
   email                       :    gebi@ece.pdx.edu
	-------------------------------------------------------
	Description:
	============

	This class wrapps the communciation with the serial port. It can be used
	to read and write to the PC's serial port

 ********************************************************************************/



// SerialCtrl.cpp: Implementierung der Klasse CSerialCtrl.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "stdafx.h"
#include "SerialCtrl.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

/*
================================================================================

	Constructor / Destructor

================================================================================
*/

CSerialCtrl::CSerialCtrl()
{
```

```
 // default parameter.

  DCB_Settings.ByteSize = 8;            // Byte of the Data.
  DCB_Settings.StopBits = ONESTOPBIT;   // Use one bit for stopbit.
  DCB_Settings.Parity = NOPARITY;       // No parity bit
  DCB_Settings.BaudRate = CBR_9600;     // Buadrate 9600 bit/sec

}

CSerialCtrl::~CSerialCtrl()
{
        if (bPortStatus) ClosePort();
}


/*
================================================================================

        Open COM port and configure for serial communication

================================================================================
*/


bool CSerialCtrl::OpenPort(const char *szPortName)
{

  if (bPortStatus == false)                     // if port is opened already, do not open port again.
  {
    PortHandle = NULL;
        PortHandle = CreateFile(szPortName, // Specify port device: default "COM1"
    GENERIC_READ | GENERIC_WRITE,       // Specify mode that open device.
    0,                          // the devide isn't shared.
    NULL,                           // no security.
    OPEN_EXISTING,                  // Specify which action to take on file.
    0,                          // default.
    NULL);                          // default.

        if (PortHandle == NULL)
        {
          AfxMessageBox("Port couldn't be opened!", MB_OK | MB_ICONSTOP);
      return false;
        }
    // Get current configuration of serial communication port.
    if (GetCommState(PortHandle, &DCB_Settings) == 0)
    {
      AfxMessageBox("Couldn't read port configuration!", MB_OK | MB_ICONSTOP);
      return false;
    }

        // Set current configuration of serial communication port.
        // This is a windows API function, declared in winbase.h
        // DCB = Device Control Block is used to configure the communication
    if (SetCommState(PortHandle, &DCB_Settings) == 0)
    {
       AfxMessageBox("Was not able to setup port", MB_OK | MB_ICONSTOP);
       return false;
```

```
    }

    // instance an object of COMMTIMEOUTS.
    COMMTIMEOUTS COMTimeOut;
    // Specify time-out between charactor for receiving.
    COMTimeOut.ReadIntervalTimeout = 3;
    // Specify value that is multiplied
    // by the requested number of bytes to be read.
    COMTimeOut.ReadTotalTimeoutMultiplier = 3;
    // Specify value is added to the product of the
    // ReadTotalTimeoutMultiplier member
    COMTimeOut.ReadTotalTimeoutConstant = 2;
    // Specify value that is multiplied
    // by the requested number of bytes to be sent.
    COMTimeOut.WriteTotalTimeoutMultiplier = 3;
    // Specify value is added to the product of the
    // WriteTotalTimeoutMultiplier member
    COMTimeOut.WriteTotalTimeoutConstant = 2;
    // set the time-out parameter into device control.
    SetCommTimeouts(PortHandle,&COMTimeOut);
    // Updata port's status.
    bPortStatus = true;                                                    // port was
successfully hooked up!!
            return bPortStatus;
  }
  return false;


}



/*
============================================================================

            close COM port

============================================================================
*/

bool CSerialCtrl::ClosePort()
{
   if (bPortStatus)
   {
            if(CloseHandle(PortHandle) == 0)
            {
                        AfxMessageBox("Port can not be closed!", MB_OK | MB_ICONSTOP);
                        return false;
            }
            PortHandle = NULL;
            bPortStatus = false;
            return true;                                                    // got closed
without complications

   }
 return false;
}
```

```
/*
================================================================================

	Read from an opened Serial Port

================================================================================
*/

bool CSerialCtrl::ReadSerialPort(char *szOutputData, const unsigned int &nSizeBuffer, unsigned long &length)
{

	if (bPortStatus)
	{
		if (ReadFile(PortHandle,    // handle of file to read
			szOutputData,           // handle of file to read
			nSizeBuffer,            // number of bytes to read
			&length,                // pointer to number of bytes read
			NULL) == 0)             // pointer to structure for data
		 {
			AfxMessageBox("A problem occured while reading from serial port!", MB_OK |
MB_ICONSTOP);
			return false;
		 }
		 if (length > 0)
		 {
			szOutputData[length] = NULL; // Assign end flag of message.
			return true;
		 }
		 return true;
	}
	else
	{
		AfxMessageBox("Can't Read from Serial Port.\nOpen at first a Port!", MB_OK |
MB_ICONSTOP);
		return false;
	}

}



/*
================================================================================

	Write to an opened Serial Port

================================================================================
*/

bool CSerialCtrl::WriteToSerialPort(const char *szData, const unsigned int &nSizeBuffer, unsigned long &length)
{
  DWORD dwBytesWritten = 0;
   if (bPortStatus)
  {
	if (length > 0)
```

```
	{
		if (WriteFile(PortHandle,  // handle to file to write to
		  szData,                                  // pointer to data to write to file
		  nSizeBuffer,            // number of bytes to write
		  &dwBytesWritten,NULL) == 0)      // pointer to number of bytes written
		{
		  AfxMessageBox("A problem occured while writing to serial port!", MB_OK | MB_ICONSTOP);
		  return false;
		}
		return true;
	}

	AfxMessageBox("You need to give Data!",MB_OK | MB_ICONSTOP);
	return false;
 }
 else
 {
	AfxMessageBox("No port was opened! \nYou need to open a port before sending data to it!", MB_OK |
MB_ICONSTOP);
	return false;
 }
}



/*
============================================================================

	Write to an opened Serial Port

============================================================================
*/

bool CSerialCtrl::GetPortStatus()
{
	return bPortStatus;
}
```

The INI file:

```
[Standard Fonix ASR]
SDK-DIR=D:\Programming\Speech\Fonix\fre
shared path=fd01\english\asr\
language=US English
US-path=fd01\english\asr\
US-GenNNet=usgp11N3108
US-DigNNet=usdg11N3066
US-Dictionary=USEnglish.dcc
US-Rules=USEnglish.rules
US-suffix=USEnglish.suffix
US-CustomDict=
US-Grammar=
German-path=fd01\german\asr
German-GenNNet=degp11FN3115
German-DigNNet=
German-Dictionary=German.dcc
```

German-Rules=German.rules
German-suffix=German.suffix
German-CustomDict=
German-Grammar=
UK-path=fd01\ukenglish\asr
UK-GenNNet=brgp11N3119
UK-DigNNet=
UK-Dictionary=UKEnglish.dcc
UK-Rules=UKEnglish.rules
UK-suffix=UKEnglish.suffix
UK-CustomDict=
UK-Grammar=
optimize=off

[Custom Fonix ASR]

[Standard Fonix TTS]

[Custom Fonix TTS]

[Standard MS TTS]

[Custom MS TTS]

[Standard MS SR]

[Custom MS SR]