THE COOPER UNION
ALBERT NERKEN SCHOOL OF ENGINEERING

Robot Control and Communication Interface for the Tele-Robotic Theater

By

Marcin Arkadiusz Balicki

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Mechanical Engineering

May 2004

Advisor
Professor Chih-Shing (Stan) Wei

THE COOPER UNION
ALBERT NERKEN SCHOOL OF ENGINEERING

This thesis was prepared under the direction of the Candidate's Thesis Advisor and has received approval. It was submitted to the Dean of The School of Engineering and the full Faculty, and was approved as partial fulfillment of the requirements for the degree of

Master of Engineering

_____
Dean, School Of Engineering - Date

_____
Professor Chih-Shing (Stan) Wei - Date
Candidate's Thesis Advisor

Acknowledgement

First of all, I would like to thank Professor Stan Wei for his faith and support of James and me taking up such a large project and for his guidance as my thesis advisor. Many thanks to Adrianne Wortzel for her inspirational drive, brilliant ideas, endless energy, and her persistence and foresight in the development of **StudioBlue**. Thanks to Professor Jean LeMee for sharing his vast knowledge of art and engineering, as well the investment of his valuable time and department resources in the development of **StudioBlue**. James Cruickshanks has been an incredible friend, roommate, and a colleague in the quest to create **StudioBlue**. Thank you for late night brainstorming sessions, philosophizing about life over cheeseburgers and beer, and sharing your passion and inspiration to perfect the project and general good hearted friendship. John Rossiello and Megan Neill could not have been more helpful in those last few months by cooking, entertaining and encouraging me to finish on time, muchas gracias! Thanks to George Ortega for putting up with my constant requests for help with electronics and computer issues. I would also like to express my gratitude to: James Cole and Ericson Mar, Carl Weiman for answering my robotics questions; Roger Tooze and the Department of Buildings and Grounds for prompt implementation of construction requests; Chris Simon, Mike Sudano, Mohammed Yusuf and Yoongeu Kim for taking care of the theater while I was away and for building a wonderful turntable; Huy Truong for transforming the dull lab into a bright, well lit professional video production studio; Winston and his Audio-Visual Department for guidance in choosing lighting and Chromakey equipment; and all my friends for understanding and allowing me to be a hermit for the duration of report compilation. Great thanks to my parents for supporting me along the way through six and a half years of engineering school. And last but not least thank you to the Cooper Union!

Abstract

This work describes the creation of a web interfaced robotic multimedia theater (**StudioBlue**) with funding from the National Science Foundation and the Cooper Union. The project's goal is to establish a common ground for collaboration between art and engineering that will foster an interdisciplinary learning environment for the creation of theatrical performances where the primary performers are robots and the audience is present virtually through Internet audio/video stream. **StudioBlue** is a laboratory outfitted with theatrical lights, Chromakey equipment, a turntable stage, and a variety of audio and video production equipment as well as a number of ActivMedia robots. This report also presents a communication platform created to enhance the interactivity and the programmability of **StudioBlue's** robotic actors. The Robot Control and Communication Interface (**RCCI**) provides a clean and efficient text based robot control interface using the **TCP/IP** protocol. **RCCI** was developed in **C++** on top of ActivMedia's robot application programming interface (**ARIA**). Intuitive text commands control major robot functions via *Telnet* operation or software with **TCP/IP** capabilities. **RCCI** serves as the robot side control interface for James Cruickshanks' visual robot control software: Graphical Robotic Activity Scripting Platform (**GRASP**).

Keywords: robot control, theater, ActivMedia

Table of Contents

Nomenclature

ActivMedia	ActivMedia Robotics Inc.
API	Application Program Interface
ARIA	ActivMedia Robotics Interface Application
A/V	Audio/Video
Botspeak	Wrapper for ViaVoice
C	ANSI C Programming Language
C++	ANSI C++ Programming Language
CPSS	Communication Platform Server Software
CRSP	Center for Robotic and Synthetic Performance
GCC	Very high quality, very portable compiler for C, C++ and Objective C.
GRASP	**G**raphical **R**obotic **A**ctivity **S**cripting **P**latform
GUI	Graphical User Interface
HTML	Hyper-Text Markup Language
IDE	Integrated Developing Environment
Java	Platform independent programming language (Sun Microsystems)
Linux	Open-source version of the UNIX operating system.
ME	Mechanical Engineer
MS	Microsoft Corporation
NSF	National Science Foundation
OS	Operation System
PC	Personal Computer
PTZ	Pan Tilt Zoom
RAM	Random Access Memory
RCCI	Remote Control and Communication Interface
Saphira	SRI International's library of gradient-based navigation and localization software including GUI and simulator
TTS	Text-To-Speech
UNIX	Multi-user general-purpose computer disk operating system.
ViaVoice	IBM's TTS and Voice recognition software
WECP	Web Enabled Communication Platform

Table of Figures

1 Introduction

Considering that theater has been evolving for 2,500 years, “robotics” in theatrical environments have been used long before the word “robot” existed. The Greeks invented numerous ingenious stage machineries including the *deus ex machina*, a system for lowering actors; the *ekkyklema*, a wheeled cart for revealing static *tableaux*; and *periaktoi*, an early visual display system that used triangular set elements mounted on pivots and turned by ropes and gears for quick backdrop changes [1].

More recently, an area of robotics called animatronics has extensive applications in the entertainment industry. For example, the film industry uses robots to represent fantastic creatures (i.e. dinosaurs in *Jurassic Park*), while amusement parks like *Disney World* use humanoids to represent important figures in history. These robots tend to be remote controlled puppets or designed for specific task and exclusively created for the character they represent. With the New Media Art movement, there has been an increased interested in more intelligent and versatile robots that engage in dialogs, maneuver intelligently on stage, and interact with the audience. However, such machines to be widely used need to be publicly available, easily reprogrammable and allow for real time control.

The Robotic Theater concept attempts to integrate the field of robotics engineering and theater to establish a common ground for collaboration between art and engineering education via performance art. It was initiated by Adrianne Wortzel and Carl Weiman with assistance of Cooper Union Faculty: Professor Stan Wei, Professor Jean LeMee; and Gateway Engineering Education Coalition [2,3,4]. **StudioBlue** is a pioneering concept for research in theatrical productions involving robots and the Internet as a new art form especially in an institutional setting. It is hoped that **StudioBlue** will become a platform for development of new technologies and a venue for artistic expression.

This paper describes the history and the development of the **StudioBlue** robotic theater, construction of laboratory, implementation of audio/visual and Internet technologies, as well as the development of necessary software for theater's robots. The project is financed by National Science Foundation Grant DUE9980873, and the Cooper Union [2]; it is a collaboration with James Cruickshanks who was heavily involved in the creation of the lab and has created a user friendly, graphical application for directing the robots in performance [6]. His application directly interacts with Robot Control and Communication Interface (**RCCI**) application designed by the author that provides a clean and efficient software interface for simple robot control capability of ActivMedia-made robots [5,7,8] for. It decreases software development time, and encourages cross disciplinary usage of the robots. This interface paradigm can be also applied to other robotic platforms by implementing the required (existing) robot functions and by forcing the programmer to adhere to particular control structure and a transmission standard for communicating with the robot.

This project provides a venue to explore a wide field of interdisciplinary education through the development of remotely controlled robots in public theatrics. Theatrical space was created at the Albert Nerken School of Engineering at the Cooper Union; it is a fully functional multimedia studio equipped with professional lighting and video recording technologies, as well as a video editing, and Internet broadcasting capabilities. **StudioBlue** includes a number of autonomous and semiautonomous robots that can be controlled wirelessly over the Internet with full audio and video feedback. The main function of the space is to allow remote directing of a performance where primary performers are robots, and the audience is present virtually through Internet audio/video stream.

2 History of the project

The following accomplishments and events were crucial stepping stones leading to the creation of **StudioBlue** and **RCCI** software.

2.1 National Science Foundation Grant – "Robotic Renaissance"

"Robotic Renaissance: Bridging Engineering, Art, and Science via Web Robotics" is an initiative funded by the **NSF** to create an environment which adopts mobile robotic technology from engineering and implements it in a fundamentally interdisciplinary education [2]. The initial goal of the project was to purchase off-the-shelf robots for use in teaching and art performances.

Traditionally, versatile programmable robots have been only used in industrial and academic research environments and rarely by non-engineers [5,9]. Great financial requirements and extensive programming knowledge limited their usage to engineering researchers and computer scientists. However, if there existed a public robot venue with an interface such that the robots could be controlled via an intuitive graphical interface requiring no specialized systems experience a team of artist, directors, set designers and engineers could interact in productive cross-disciplinary participation and mutual enrichment of engineering and arts students. As summed up in the **NSF** grant abstract, its goal is to:

> *"provide an open development environment - free of operating system or platform expertise requirements - for students and faculty in all academic disciplines to develop creative applications for mobile robots. By analogy, this is what browsers did for the Internet, revolutionizing both commerce and culture while fueling further technical progress. Outcomes are enrichment of cross-disciplinary curricula in engineering and arts, faculty development, and exciting applications including WWW robotic theatrical performances directed by a professional artist. Results are being disseminated via Web and publications."* [2]

Furthermore, a laboratory that implements this idea establishes a common ground for collaboration between educators and students, and provides an interdisciplinary educational environment for further project-based learning in the area of multi-agent, mobile robotics.

The initial step towards this goal was a creation of an undergraduate interdisciplinary course: *EID111 – Design Illusion and Reality* by Adrianne Wortzel and Carl Weiman that focuses on a robots and art, and is a great complement to the new robotic development environment initiative [3].

2.2 Camouflage Town

Camouflage Town was an interactive tele-robotic installation by Adrianne Wortzel, commissioned by The Whitney Museum of American Art in New York City as part of the *Data Dynamics* exhibit on display from March 22 to June 10, 2001 [10,11,12]. The art piece was centered on a semi-autonomous robot: *Kiru*, which lived in the museum and acted as a physical avatar for Internet users interacting with museum visitors (Figure 1.1). A person was able to navigate the robot around the museum lobby, use text to speech, control camera movements, hear, see, interact with and experience the robot's environment via a web site from anywhere in the world (Figure 1.2). Whitney Museum visitors communicated with online visitors by speaking to the robot and gesturing to its camera through a live web-cast.

Figure 2.1 Kiru Interacting with Whitney Visitors.

Kiru played the role of a "cultural curmudgeon," interacting with visitors and contextualizing the exhibition through its comments on issues of "mapping" physical and virtual space and physical/virtual identity. While *Kiru* was in the autonomous mode, he had five different personalities, all present at once: Wizard, Librarian of Juxtapositions, Philosopher, Preacher and Storyteller. The sixth personality was left to visitors to define through the robot's speech, camera movements and motion. They created the camouflage and made *Kiru* a new persona; two minutes at a time.

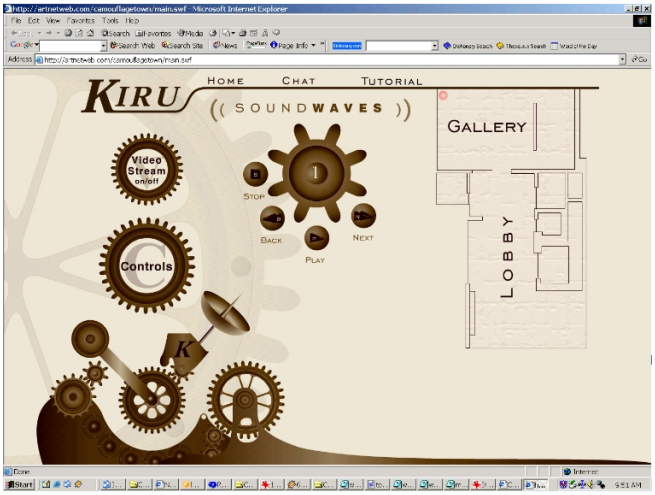


Figure 2.2 Kiru's Web Interface for Camouflage Town.

The robot was the ActivMedia's Pioneer PeopleBot [7], outfitted with an extra camera for navigation and a small LCD display. Localization was done via image processing of the ceiling with a camera mounted on the top of the robot. Groups of multicolored discs mounted on the ceiling allowed the robot to identify its position within the lobby of the museum and display its location on the graphical user interface on *CamouflageTown.tv* website [10,11].

The development of the project was part of James Cole's Master Thesis [11] and the author's senior year engineering project [12], both at the Cooper Union. Notably, this was the first time the Whitney Museum has exhibited **NSF** funded work.

2.3 Rededication of The Foundation Building – Ribbon Cutting.

On April 12th, 2001 **StudioBlue's** Performance PeopleBot was a guest at the "Rededication of the Cooper Union's Foundation Building" ceremony. The robot, called *Bruno* at that time, represented the student body in the ribbon cutting ceremony. *Bruno* was outfitted with scissors, an abstract tuxedo costume made of painted black sheet metal and a matching top hat (Figure 1.3). The robot was controlled wirelessly from near by using software written by Marcin Balicki and James Cruickshanks, which used the keyboard to control the robots movement and gripper actions.

Figure 2.3 Rededication of The Cooper Union Foundation Building.
Photo: Juliana Thomas

2.4 Cooper Union : End-Of-Year Show 2002

In 2002, the Engineering school was invited to participate in the Annual End-of-Year Student Work Exhibition held in the Cooper Union Foundation Building [13]. The two month show exhibits past year's art and architecture student's work, and for the first time, engineering projects.

Figure 2.4 End-Of-Year Show 2002.

Representing the Mechanical Engineering department Marcin Balicki and James Cruickshanks submitted a surveillance themed interactive robotic installation (Advisors: Prof. C. Stan Wei and Adrianne Wortzel.) Visitors were able to control a robot located in the basement of the Foundation Building through an interface located in the lobby. Using wireless Ethernet, and wireless audio/video transmission the interactivity between the user controlling the robot and the robot's environment was truly real time. The project deployed an ActivMedia PeopleBot and several smaller ActivMedia AmigoBots in a free-roaming environment in which each robot is preprogrammed for obstacle avoidance. Adrianne Wortzel summed up the installation in the following way:

> *"Upon entering the first Floor Lobby space, a visitor found themselves at a control station consisting of a computer with two monitors. One of the monitors displayed the interface console which controlled Bruno, the large robot. A visitor can control Bruno's physical locomotion as well as the pan, tilt and zoom features of his camera while he roams the gallery on the lower level. The visitor can also initiate the robot to speak by typing in text which becomes speech for the robot. The second monitor shows the visitor what the robot is seeing and has accompanying speakers so that the visitor can also hear what takes place in the downstairs gallery. This first floor lobby control station also had a hidden camera which made the user visible to visitors downstairs.*
>
> *Bruno is accompanied by three smaller robots, traveling close to the floor and equipped with a camera. These robots are programmed to roam the lower level floor and interact with visitors.*
>
> *Meanwhile, downstairs in the lower gallery a visitor may not necessarily realize they are engaged, through Bruno, in an interactive relationship with the visitor upstairs. Inevitably they will come upon a monitor which reveals that they are indeed, engaged in mutual surveillance and interchangeable communication. Each person may be under the impression that the communication is one-*

way, that they are the observer and not the observed. But that is not the case." [13]

3 Statement of Problem

3.1 StudioBlue

One of the first art projects completed utilizing one of the new intelligent robots at Cooper Union was *COLOR AID* by Daniel Arsham for Adrianne Wortzel's *Design, Illusion and Reality* class in the Fall of 2000 [14]. The theme in the video depicts a young woman's relationship with her partner much more robotic than with a machine. The premise of the video focuses on the fact that a young woman finds her relationship with a robot less robotic than her relationship with her partner. The robot learns to feel emotions through color; while, the woman's boyfriend is unable to notice the clearest signs of human emotion. During the shooting the robot (Pioneer PeopleBot) was confined to a small area in the Rapid Prototyping Laboratory at Cooper Union [15] which is congested with loud machinery, students, and constant street noise along with poor overhead fluorescent lighting. The production was done with assistance from James Cole who wrote custom control code for audio file queuing and limited robot motion. The artist did not directly control the robot, but rather directed the engineer to command it. Despite these hurdles, the team managed to produce an amazing creation.

Figure 3.1 Scene from *COLORAID.*

The principle investigators of the **NSF** grant promised a creation of a venue, software tools for developing robotic performances, and to facilitate interdisciplinary learning. Such controlled environment would have been very convenient for producing work like Arsham's. Implementation of the proposal requires versatile and intelligent robots, a space large enough for them to freely roam among other actors, computers for software development and control, proper stage lighting, audio and video recording equipment and simple scenery changing method.

Lastly, an avant-garde concept was proposed for the theater to have the means to operate it from a remote location via the World Wide Web. This would allow general public to direct plays or become spectators. Implementation of this functionality requires an Internet web server, audio/video encoders and live streaming capabilities; as well as an intuitive robot control interface that is preferably platform independent.

3.2 Robotic Control and Communication Interface (RCCI)

Previous research in the area of robot interface programming has focused on providing a control environment that is specific to a particular task. While such approach is very helpful, it is often very low level forcing the programmer to spend much time to alter the application for another task [16]. An interface that is user friendly and versatile enough for robotic control programmers that can also be used on any platform would be a great addition to **StudioBlue** and the robotics community in general.

3.2.1 James Cole's Master Thesis

James Cole's thesis "A Web-Enabled Communication Platform for the ActivMedia PeopleBot" does a great job of providing an interface for controlling a PeopleBot robot at the Whitney Museum [11]. However, the software running on a *Microsoft Windows 98 OS* is based on outdated **Saphira 6.2** robot operating software, created by SRI International, which is not open source and is discontinued since 2002 [17]. Moreover, the most recent robot purchases are based on the *Linux OS* and are equipped with **ARIA**, a new open source robot operating software created by ActivMedia [18]. *Kiru*,

the only robot running *Windows 98*, was converted to *Linux* as well. The above developments and the specific theater requirements (different than the Whitney) have rendered Cole's software inoperable without major overhaul. More information can be found in Section **5.1.1.**

3.2.2 Graphical Robotic Application Scripting Platform.

GRASP is an acronym for **G**raphical **R**obotic **A**ctivity **S**cripting **P**latform which is an application created concurrently with the development of the robotic theater as the main control interface for **StudioBlue's** robots (**Figure 3.2**) [6]. Written in Java, a platform independent programming language created by Sun Microsystems, it can run on virtually any computer OS, including but not limited to: *Windows*, *Mac OS X*, and *Linux* [19]. **GRASP** allows users to create a script of activities on a synchronous timeline. The parameters of these activities or events can be defined graphically using the appropriate modules: text-to-speech (TTS), pan-tilt-zoom (PTZ) camera, motion, gripper, etc. The script can then be used to send messages request to a robot server via the Internet to perform predefined activities. Event instructions can be transmitted in primarily two separate manners. The user may click through the script, sending events one at a time at his/her discretion or s/he may choose to automatically send them at predefined time intervals. Each event consists of a Text-To-Speech action, a movement, a playing of an audio file, or any other possible action in any combination. These options allow the user to visually direct the robots motions, as well as change its behaviors.

GRASP needs to establish communication with the robot which also requires a type of message interpreting engine that translates general activity request from **GRASP** and materializes them in physical robot activity (i.e. motion, speech). For the most part these predefined activities are implemented in **ARIA**, but need to be set up properly and organized for easy maintenance and live feedback.

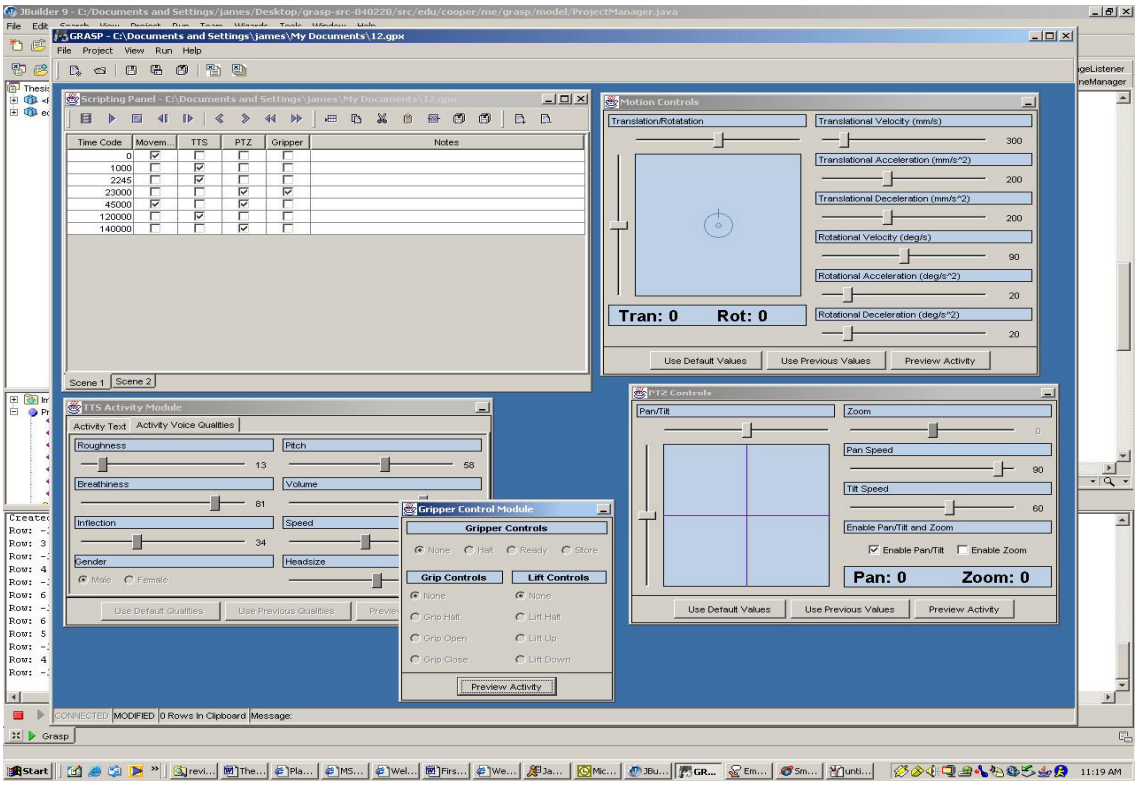


Figure 3.2 Graphical Robotic Activity Scripting Platform (GRASP).

3.2.3 ARIA

The robots are, for the most part, provided by ActivMedia Robotics and come with application programming interface written in **C++** called **ARIA**, along with a higher level navigation package - **Saphira** [18,20,21]. This platform is rather cumbersome and difficult to utilize for first time users, requiring expert knowledge of **C++** as well as familiarity with a large application. It resides on the robot's PC, or in case of the AmigoBot, on a desktop machine. **ARIA** has no graphical interface that is useful in a theatrical environment and developing applications requires time and expertise. Therefore there is a need for a way to quickly and easily allow non-technical person to take control over the robot whether it is prescript or real time via a clean and efficient but high-level interface that hides the complexity involved in robot control.

The application needs to be easily maintainable due to frequent software updates released by ActivMedia. The communications interface (over **TCP/IP** [22]) needs to be modeled in a structured and consistent fashion and be platform independent and easily implemented, so a graphical user interface could be created in any language to connect with the robot (exampled by **GRASP**) but it should also interact with a simple text based **TCP/IP** client like *Telnet*. This approach permits distributed processing to expand the capabilities (intelligence) of the robot and take advantage of more powerful computers.

Lastly, the software should be structured in layers with modules allowing for possibility of implementing other robot software by replacing a layer or implementing functions as modules rather than redesigning the whole application.

4 StudioBlue – Tele-robotic Theater

StudioBlue is a brainchild of Adrianne Wortzel and was made possible by an **NSF** grant and the support of The Cooper Union and its faculty [2]. Even though the Cooper Union is comprised of Art, Architecture and Engineering schools the space that is common to all three is very limited; and the academic and social interaction between the students of these schools is minimal. As mentioned in the introduction the robotic theater attempts to improve this situation. Once available space was found, with the support of Cooper Union's Building and Grounds Department, the new interdisciplinary laboratory was created.

Figure 4.1 StudioBlue: Preparation for the First Shoot.

The Mechanical Engineering Department at the Cooper Union is in charge of developing this theater that includes multiple autonomous and semi-autonomous robots that serve as actors in a virtual reality theatrical space. The goal of the project is to create a "playground" for students of all disciplines, creating an interdisciplinary research environment based on robotics. Marcin Balicki and James

Cruickshanks planned out the appropriate equipment: from robots, stage lighting, to computers and video cameras.

The physical creation of the lab began in January of 2001 and has officially completed in January of 2004. Its name has gone through a few iterations, starting with *Renaissance Theater*, and then naturally evolved into *BlueLab* from its overbearing blue shade, and finally **StudioBlue**. The main goal of the theater is to provide an environment for interdisciplinary learning and collaborating but it is also a research facility for robotics investigations in software and hardware. The current Artists in Residence Adrianne Wortzel, who is also the Art Director, and Huy Truong have successfully used the studio to produce video art (**Figure 4.2**).

Figure 4.2 The Cast of Huy Truong's production: "Shakespeare Robots".

4.1 Related Research Projects

StudioBlue as a tele-robotic theater is an innovative concept in educational sense and can be considered Avant-Garde in the art world. The studio is a self-contained audio and video production space that is connected to the Internet with robots as actors. A few projects involving intelligent robots in theatrical performances have been publicized in the late nineties. The most prominent

robotic performance groups are *Ullanta Performance Robotics* [23] and *OmniCircus* [24]. The recent development at Carnegie Mellon University of the *Center for Robotic and Synthetic Performance* (CRSP) [25] is the closest publicized concept to **StudioBlue**.

4.1.1 CMU - The Center for Robotic and Synthetic Performance

CRSP is a project in development by Frank Garvey, and is a part of Robotics Institute at the Carnegie Mellon University (CMU). According to Garvey, CRSP is "a vehicle for the creation of new technologies, performance languages, and engineering visions which interface science and technology with the primal effort to understand the human condition through the arts"[25]. A planned centerpiece of the CRSP is a

> *"smart" theater, or a space whose structural elements built-in electromechanical systems (robots) with integrated virtual-reality. Garvey proposes that the theatrical laboratory environment will "extend concepts of artificial architectural intelligence and will have numerous built in performance-enhancing sensors and the capability of integrating virtual reality sets, and be capable of exploring other strategies allowing robotic and human actors to interact with and within a three-dimensional, architectural-scale, digitally-imaged and/or roboticized space."* [25]

According to CRSP website, the following projects are or were recently in development at the research center:

> ***Smart Theater Research Project*** *– Development of new technologies for a performance environment integrating state-of-the-art robotic performers with virtual reality sets and performers.*
>
> ***House of the Deafman*** *- A full-length robotic-integrated music-dance-theater piece about the Black Paintings of Francisco Goya, in collaboration with the CMU Theater Dept. and other elements in the CMU community.*

Robotic Performer Research Project *– Development of new mechanical actors including 'Humper', with organic movements and interactivity exploring the social strategies of streetwalking, and 'Laugh Riot', a robotic Jack-in-the-Box whose purpose is the dissemination of false hope.*

Synthetic Performer Research Project *– Development of new strategies for the creation of virtual actors in the digital realm and the programming of 'actual' (robotic) synthetic equivalents through the animation of virtual actors [26].*

4.1.2 *Ullanta* Performance Robotics

Ullanta uses new technologies and concepts in robotics to create new experiences in theater and new challenges for scriptwriters and directors. The group believes that the most impressive features of behavior based robots are their ability to look like "living, intelligent critters that adapt to elements of their environments." Applying them to understand how emotion influences both individual and social behavior.

Figure 4.3 One of *Ullanta*'s street performances.

Their robotic performance pieces take place at scheduled times, demand a standard of robustness, adaptivity, and interaction that few laboratory experiments display and most importantly should appeal to audiences. Furthermore, the casting of robots in dramatic roles allows the creators some justification for the "stereotypical anthropomorphizing by robotics engineers" [23].

Ullanta repertoire ranges from "Play" by G. Stein to a more contemporary “Fifi and Josie, A Tale of Two Lesbiots: A Story of Autonomy, Love, Paranoia, and Agency “ by Roxanne Linnea Rae [23]. The group uses robots provided by ActivMedia Robotics.

4.1.3 *OmniCircus*: Junkyard Cabaret and Robot Ensemble

OmniCircus is an “experimental, surreal-psychedelic musical-cabaret group” founded in 1988 by composer-artist **Frank Garvey** [24]. Its shows integrate 'live' avant-fusion music and performance with life-sized computer-controlled robotics and midi-controlled virtual-reality performers” [24].

4.1.4 The Virtual Theater Project

The project studies the creation of intelligent, automated characters that can act either in well-defined stories or in improvisational environments. Virtual reality initiatives that have a remote aspect but do not utilize robots are of interest because such system’s interfaces and intelligence can be extended to robotic (physical) “actors” [27] .

4.1.5 Synthetic Characters Group

This research project is under the direction of Bruce Blumberg at Massachusetts Institute of Technology. His group aims to create intelligent, interactive characters for storytelling, games, and Web-based worlds. Their passion is “to create creatures whose behavior, form and underlying architecture not only informs our understanding of the natural intelligence displayed by animals and ultimately ourselves, but that also touches a person interacting with them on a profound personal level” [26]. Similarly to Virtual Theater project, the group’s results may be applied to physical avatars (robots.)

4.2 Robots

Central to **StudioBlue** is a group of various intelligent robots that serve as actors. The robots were chosen based on their programmability, available options, support quality and modularity. Most of the

robots are from the ActivMedia family, regarded as most popular and widely used in academic research [5].

4.2.1 *Kiru*

Kiru is the first robot purchased for the theater. Its name was coined by Adrianne Wortzel for *Camouflage Town* exhibit at the Whitney Museum. *Kiru* is a Pioneer PeopleBot made by ActivMedia [8]. It has an onboard PC (running *Linux* 7.0) with a 266 MHz processor, 60 Watt Pioneer speakers and a Sony PTZ camera [28].

Figure 4.4 *Kiru* – Pioneer PeopleBot.

This robot performs quite well, operating eight hours on one charge. The onboard computer uses an old IEEE 802.11 3-Mbps wireless Ethernet equipment made by BreezeNet that is not compatible with

current widely used 802.11b standard; as a result a separate BreezeNet Access Point is required [11,12]. It has 16 front sonar sensors and 10 bumper sensors.

4.2.2 *Woody*

Woody, changes his name with each new production, during the ribbon cutting his pseudo name was *Bruno*. He is a "younger brother" of *Kiru*. This Pioneer Performance PeopleBot is also manufactured by ActivMedia Robotics. As mentioned earlier he was outfitted by the author and James Cruickshanks with an abstract tuxedo made of sheet metal for a formal event.

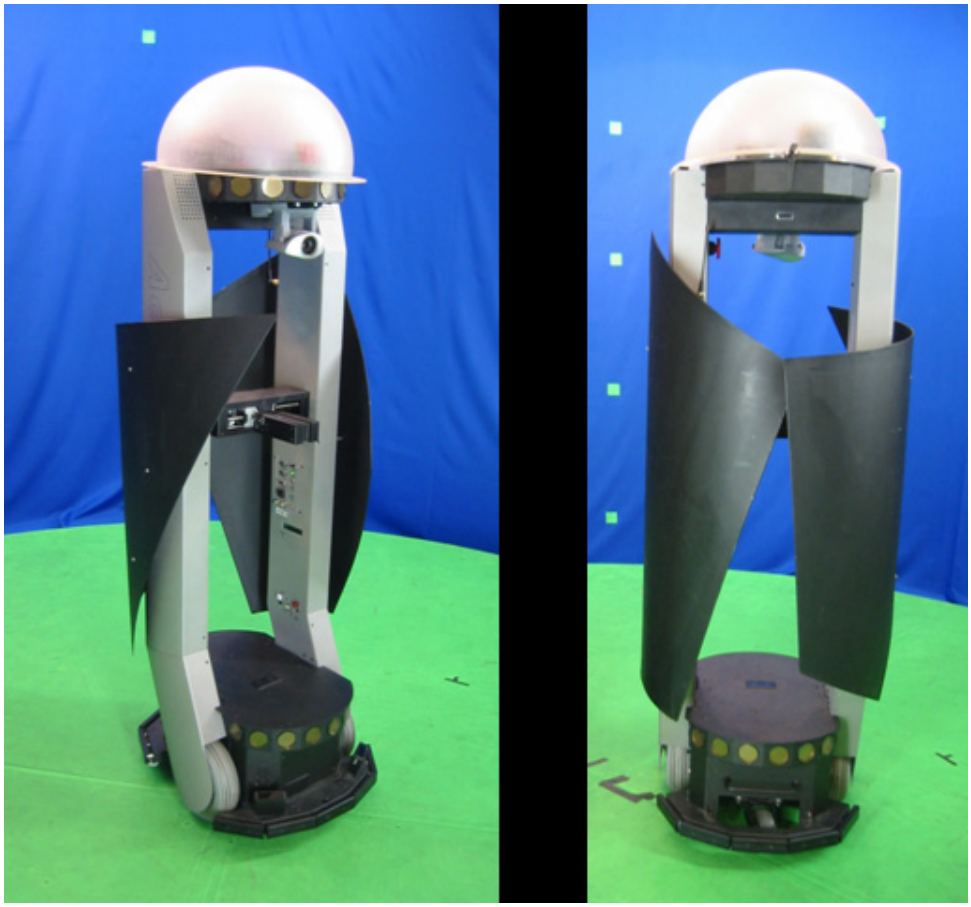


Figure 4.5 *Woody* – Pioneer Performance PeopleBot.

Woody operates only about four hours on one charge due to many peripherals and an onboard computer with a fast, power demanding processor (400 MHz AMD-K6). The robot is also outfitted with a high bandwidth (11 Mbs) WaveLan wireless Ethernet Interface (WiFi 802.11b) PCMCIA card.

It has a total of 24 (8 in the back) sonar sensors, 10 bumper sensors, and 2 Infra Red table sensors. The robot comes with horizontal grippers that can move up and down and are outfitted with break beam sensors [29].

4.2.3 AmigoBots

Another ActivMedia product that is very robust and fun to play with is the AmigoBot [30].

StudioBlue has three of these, one of which is equipped with a wireless video camera. The AmigoBots do not have an onboard PC like the above mentioned robots, but can be controlled via wireless radio (serial communication) link. This allows AmigoBots to run on the same software as the larger robots with addition of a dedicated PC. These run about two hours on a charge requiring four to eight hours to recharge; however, their batteries are not easily replaceable and therefore these robots are difficult to use for long shoots.

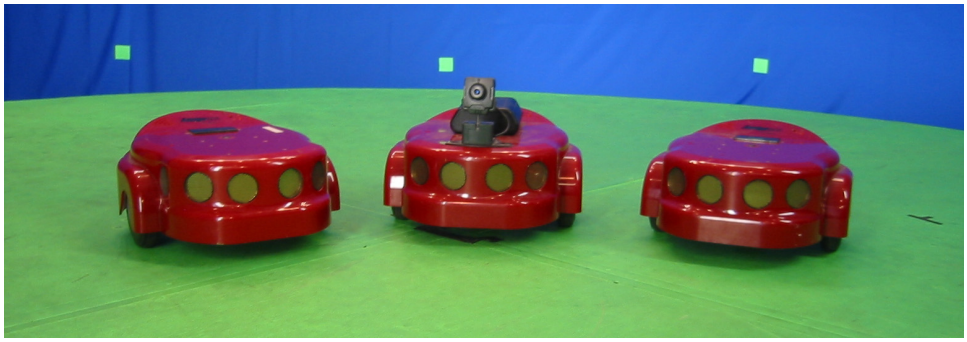


Figure 4.6 AmigoBots.

4.2.4 Magellan Pro

Magellan Pro is manufactured by I-Robot but it has recently been discontinued [9]. It runs *Linux 6.2 OS* on its onboard computer with 350 MHz processor. It operates using Mobility robot operating software but is currently not operating and has been discontinued by I-Robot.

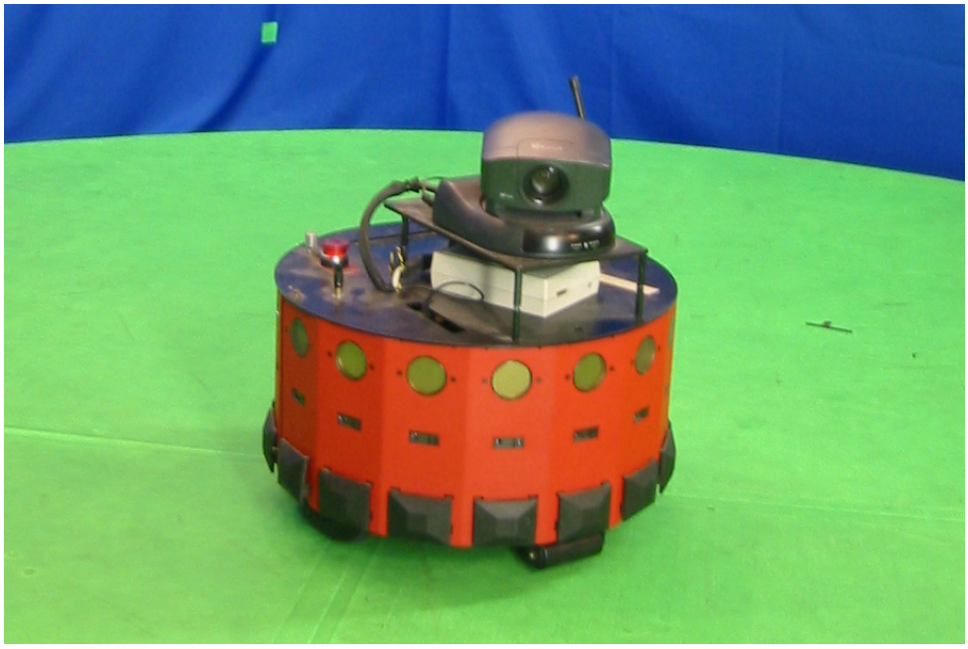


Figure 4.7 Magellan Pro.

4.3 Structures and Electric

Figure 4.8 Entrance to room 232.

4.3.1 Location and Layout

StudioBlue is located on the second floor of the Cooper Union's Albert Nerken School of Engineering at 51 Astor Place NY, NY 10003. The theater is situated in the back of the Cooper Union

Acoustics Laboratory (CUAL) in room 232 [31]. The CUAL was divided into two parts by a full non-structural partition wall, made out of a standard four inch aluminum studs and sheetrock. The wall was insulated with extra heavy insulation for increased soundproofing. A standard fireproof steel door was installed. The construction and electrical work were done by the Building and Grounds Department at Cooper Union and are summarized in the appendix (Request for Services**).**

Figure 4.9 StudioBlue.

The floor plan of the theater is rectangular in shape with two distinct areas: the performance space and the control and development area. This is a traditional three wall theater. The space is about 330 square feet, has three windows looking out onto 9th street. Two of the windows near the stage were boarded up to decrease sunlight and street noise. The lab has an industrial size 12,000 BTU Air Conditioner installed in the control area window. The performance area (stage) is about 13' by 17' while the control area is 10' by 12'.

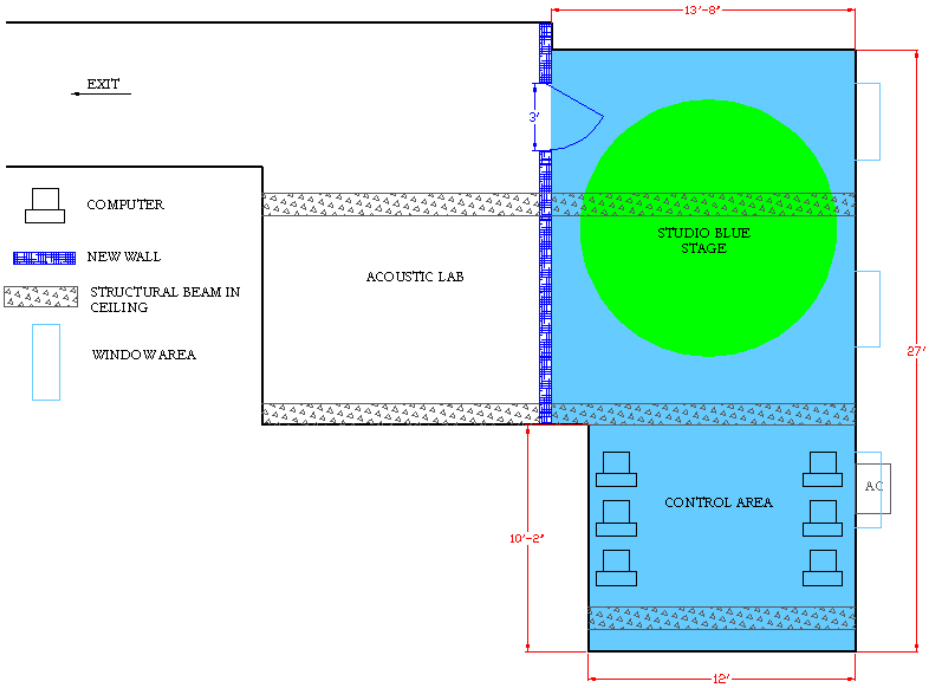


Figure 4.10 Lab Dimensions.

4.3.2 Chromakey Backdrops

The Chromakey, also called Bluescreening, is the process by which a portion of the image that is a specific color of hue is replaced by another image.

Figure 4.11 Chromakey effect.

Generally, the camera signal is fed into the keyer's foreground input as well as the background video. The color level control on the keyer is adjusted to cause all the blue on the foreground input feed to be replaced by the background video. This technique is often used in a news weather forecast, where the weathercaster is in the foreground feed standing in front of a blue screen. The blue part of the first

feed is replaced by the background feed. A particular shade of blue is used because it is not very common in clothing and nature, as well as the fact that blue is the complementary color to flesh tone. Figure 3.10 is a photo capture from a psychoanalysis session scene part of Adrianne Wortzel's work [32]. It displays before and after effect of Chromakey. Two major issues with working with this technology are uneven backdrop lighting and reflected color spilling onto the foreground subject [33].

Figure 4.12 Different Lights and Chromakey screen on a track.

It is imperative that the blue background is lit as evenly as possible. Huy Truong suggested and installed extra lights (see lighting section) and filters to achieve uniform lighting effect.

Assuming that the cinderblock wall painted a matt blue paint was sufficient for future Chromakeying, the room was painted (except the ceiling) to function as a background.

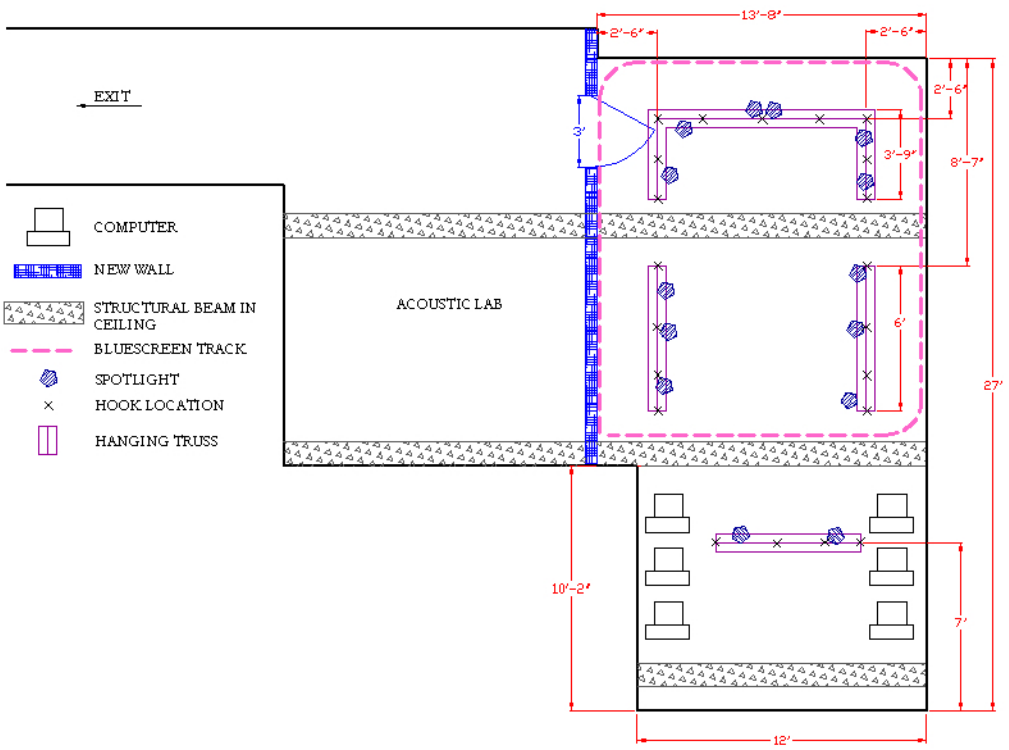


Figure 4.13 Truss location.

Turned out that that was not the case and a smoother and more maintainable surface was desired. With the help of James Cruickshanks and Chris Simon, as a part of Yoongeu Kim and Mohammed Yusuf's senior project [34] a professional blue screen curtain along with a rack system was purchased and installed. The 10' x 24' sheet fabric background (made by WESTCOTT) was mounted on seven 10' rail tracks that were mounted along the perimeter of the performance space (17'4" x 15'3"). The rails required bending to round the corner of the room. A soft bend (36" radius) in the corner decreases the difficulties in evenly lighting corners and removing the corner seem; essential for successful Chromakey effect. The track is made by Flex-I-Tracks and is 13 gauge extended aluminum medium duty cyclorama I-beam type with mill finish used for TV studio curtains.

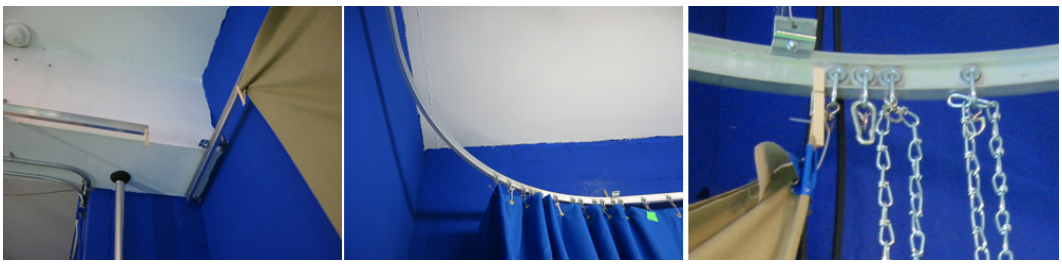


Figure 4.14 Blue curtain track.

The dimensions of the I-beam track were 5/8” x 1-11/16” x 10” with a 1/8” flange. A lock plate was used to join sections assuring proper alignment of track and channels. The Flexi-I-Track was held to the wall using L-braces attached ceiling clamp attached. Single carrier nylon ball bearings were used to attach general S-hooks, which hold the curtains through metal grommets.

The extreme amount of blue color in the room caused the subjects on the stage to acquire a blue tint. To counteract the blue hue spill onto actors he implemented a series of golden/gray curtains was mounted on the periphery of the set that cancel out the blue reflection (**Figure 4.15**).

Figure 4.15 Gray curtain.

The floor was also prepared for Chromakey by installing blue linoleum matching the color of the blue used for the walls. This is not used very often, but can be advantageous in case a top to bottom shot is required.

4.3.3 Electric

Due to an unusually power hungry equipment such as lights, computers, chargers and Air Conditioner, extra power lines/outlets and a circuit breakers were required for safe operation of the lab.

Figure 4.16 Circuit Breaker Box.

The original electrical system within the room 232 was inadequate to power all of the devices and hardware needed for the theatre. Moreover, the room was divided into two separate labs making only one outlet available for theater's use.

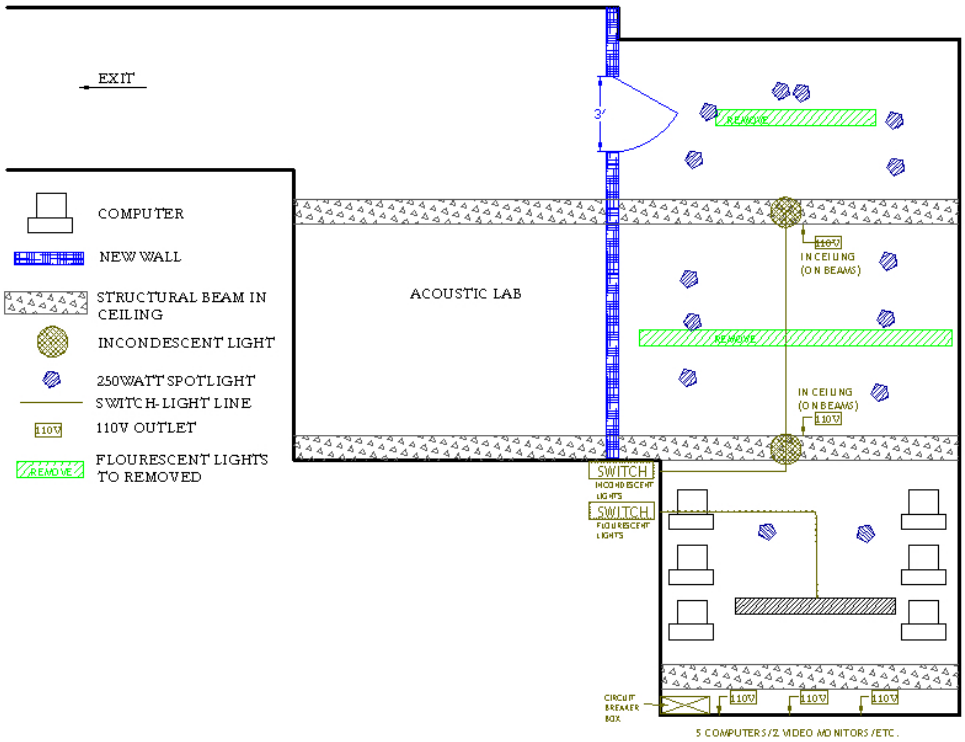


Figure 4.17 Electric layout of the lab.

In addition, the sole available outlet was shared with appliances in the adjacent room (Materials Lab). The following devices required significant power to be provided by the theater:

(5) Computers (CPU and Monitor)
(2) Video Monitors
(12) 250 Watt Stage Lights
(1) Air Conditioner
(5) Robot Power Supplies
Plus other Extraneous Hardware

The power requirements were confirmed by a professional electrician and implemented as required. A circuit breaker box was installed in the room for emergency shutdown access and for wiring simplicity. Ample extensions cords and power strips were purchased for connecting spot lights, fluorescent lights, auxiliary equipment (A/V) and robot battery chargers.

4.3.4 Triangle Truss

To support lights, props, camera, cabling and any other theatrical equipment a truss system was installed in the ceiling. This truss system was required for mounting spot lights, ambient light fluorescent lamps and cameras allowing for easy adjustment and maintenance.

Figure 4.18 Triangle Truss System.

The truss is a standard lighting industry 8-3/4" triangle and came in four 6' sections, two 2' sections and two 90 Degree turns.

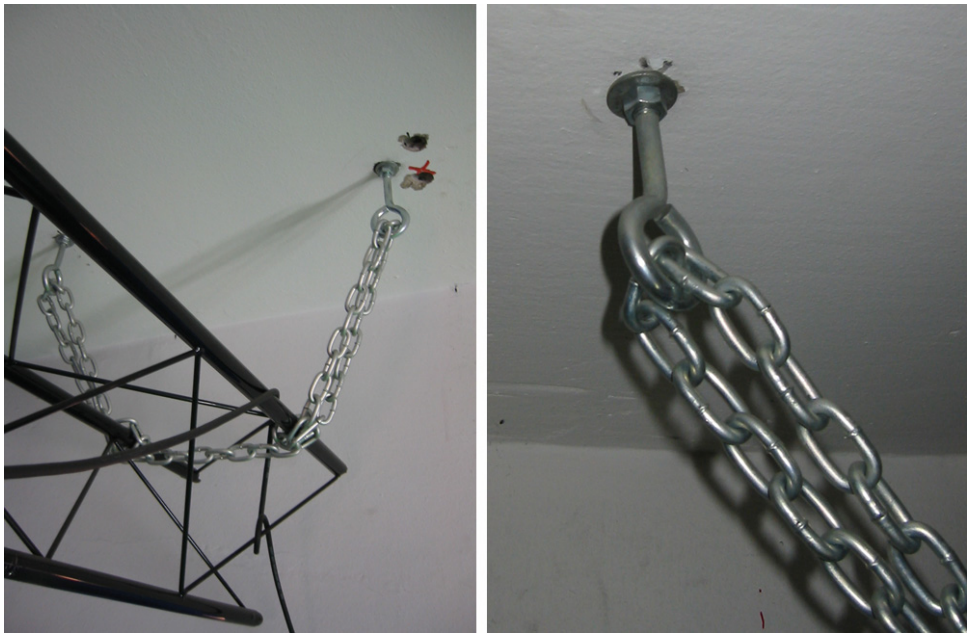


Figure 4.19 Truss mounting anchors and chains.

The triangle truss system was installed with two vertices facing the floor to accommodate more points to affix lighting to. It was determined that 5/8" diameter concrete anchors in the ceiling and ¼" chain

were sufficient to support the truss. The mounted trusses were tested to hold 350 lbs, but are expected to support twice that.

4.3.5 Turntable Stage

Since the space is not very large and camera positioning as well as altering lightings is difficult, a large ("Lazy Susan") turntable stage was designed and built in the summer of 2003 by Yoongeu Kim, Mohammed Yusuf, James Cruickshanks, Huy Truong with the assistance of Professors Stan Wei and Adrianne Wortzel [34].

Figure 4.20 Turntable fabrication.

The circular platform has a diameter of 11' 3 " and is made of a layer of ½ inch plywood supported by a web layer of standard 2x4 wooden studs. The studs are arranged to provide support and redistribute unbalanced stresses through out the circular platform which is capable of supporting up to 1300 pounds.

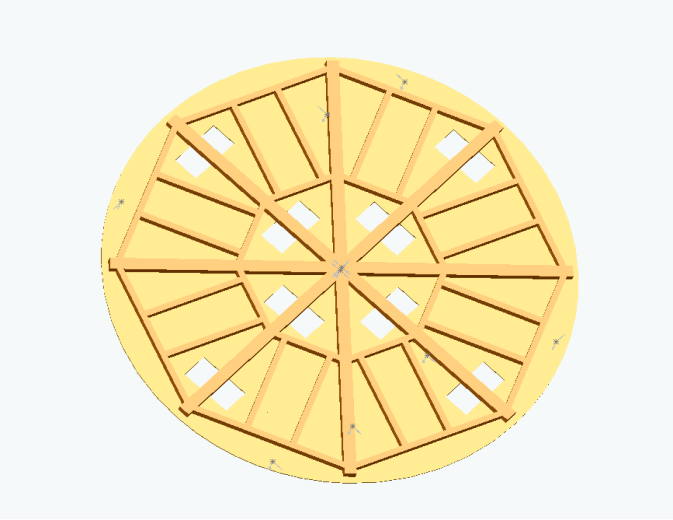


Figure 4.21 Turntable design [34**].**

Four 3” polyurethane caster wheels are mounted tangent to the edge on bottom of the turntable. A “Lazy Susan” bearing is mounted at the center allowing smooth and quiet rotation. The table is made up of four wedge sections enabling transport through the door when disassembled. Its surface was painted Chromakey green. The turntable is easily rotated by one person, and if needed could be motorized in the future.

Figure 4.22 Green Turntable.

4.4 Multimedia Equipment

The support required for smooth theater operation included robot control computers, flexible stage environment, web publishing, and archiving, and span many multimedia areas. The following equipment was found to be required to create a self sustaining production studio.

4.4.1 Chromakeyer

Software/Hardware based, real-time Chromakey solutions were considered but turned out to be too expensive, requiring another PC. Dedicated hardware Chromakeyers are readily available and work well enough to be used in production environment such as **StudioBlue**. The DataVideo dual-channel Time-Base-Corrector with Chromakey (TBC – 6000) was chosen to serve as the main Chromakeyer, If more control over the visuals is desired a post-production (not real-time), software based solution is provided. The tools are included with most Non-Linear Video Editors. A short manual was created describing procedure for using the TBC-6000 (appendix).

Figure 4.23 TBC-6000 Chromakeyer.

The dual channel, TBC-6000 model provides a Chromakey effect for two camera sources and allows to cleanly switch between sources with no distortion or noise.

4.4.2 Cameras

There are two types of camera's used at the lab, three small close circuit TV type (CCTV) color 1/3" CCD cameras as well as two professional hand-held digital video cameras.

4.4.2.1 Sony SSCDC14

The close circuit cameras are predominantly used for web casting applications and video capture for image processing (i.e. localization). Three High Resolution Sony SSCDC14 color cameras were purchased for SSCDC14's small size and ability to produce high quality images in adverse conditions, i.e. low light. Adjustable zoom lenses were selected to allow for more flexibility in placement and added another visual production tool. The 2.8-12mm Vari-Focal lenses are manufactured by Tameron and have a viewing angle from 89 Degrees (2.8 mm) to 24 Degrees (12mm) with F-Stop of 1.4 and have a DC Auto Iris. They are CS mount type and operate on 24 Volts.

Figure 4.24 Sony CCTV Camera.

While the above cameras are very small and are easily mounted on the truss system, hand held cameras give the lab another dimension and can be used both for live performances and filming, and are another high quality video and audio source. The following cameras are digital with FireWire capability for quick interface to common Mac based video editing stations.

4.4.2.2 AG-DVX100

High-End AG-DVX100 cinema camera by Panasonic was chosen as the primary tool for filming. The DVX100 has the following features:

- 1/3" 3-CCD.
- Outstanding sensitivity: F11 @2000 lux, min illumination: 3 lux (at +18dB).
- Supports 480i/60 (NTSC), Cinema-style 480p/24fps, and 480p/30fps image capture.
- Precision wide-angle Lens with Servo/Manual.
- Auto/Manual Focus f1.6 with 72mm filter size. Advanced optical image stabilization.
- 2-ch. XLR audio inputs with phantom power supply (48V) and manual audio levels.
- IEEE-1394 FireWire™ interface for transfer of digital video/audio to NLE platforms.

Figure 4.25 Panasonic MiniDV Camera.

4.4.2.3 Canon GL1

Supplementary camcorder, Canon GL1 – MiniDV, one of the more popular options, is perfect due to its compactness and versatility. Extra batteries and a protective carrying case were also purchased.

Figure 4.26 Canon GL1 and Carrying Case.

The GL1 has the following characteristics:

- 20x Optical / 100x Digital Zoom.
- Color Viewfinder.
- Digital Video Format and IEEE 1394.
- 3CCD, Lens and Optical Image Stabilization.
- Shooting Modes and Aspect Ratios.
- Programmed Auto Exposure.
- Shooting Enhancements.
- Picture Adjustments.
- Audio and Video Inputs and Outputs.

As mentioned in earlier sections, that the pioneer robot possess wireless camera with RCA video output. Although, the transmission is prone to interference, the feed can be used for a unique, first person (robot) perspective.

4.4.3 Lighting

Lighting is the most important factor in the final quality of video, especially when Chromakey effect is applied. The theater space was originally designed to only work with 12 standard, small theater 3" Fresnel spot lights, that project up to 15 feet. They produce a soft diffused light that can be focused for spot of flood applications and are excellent for use in small theater. They require quartz lamps up to 250 Watts and feature a hinged top for easy re-lamping.

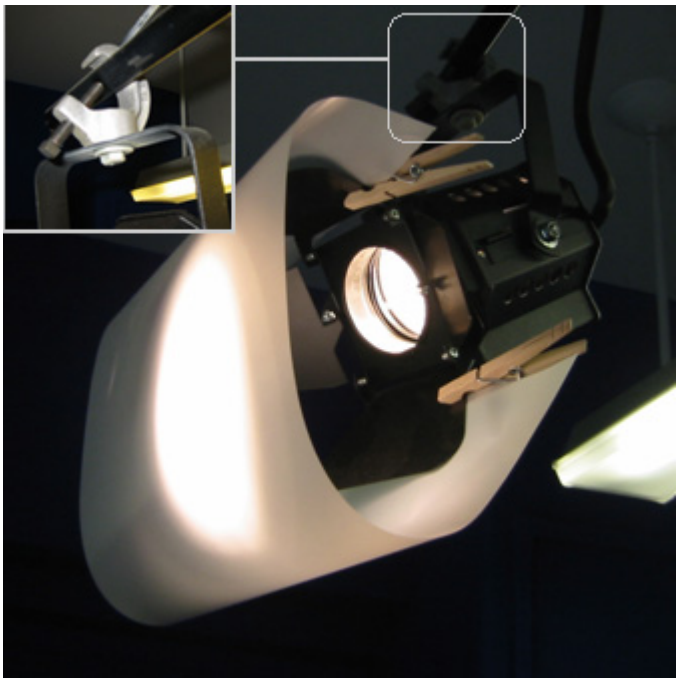


Figure 4.27 3" Fresnel Spot Light with Filter.

Required power cables, barn doors, and frames for color gels were also acquired. To independently control the Fresnels a 24 channel, 2 scene DMX scene setter console manufactured by American DJ was selected along with three 4x500 Watt DMX Dimmer Packs (DP-DMX 20L by American DJ) and three 25' XLR control cables to interconnect them. Small C-style clamps were used for mounting the spot lights. NOTE: Because the lamps become extremely hot its imperative that all safety precautions are followed.

Figure 4.28 Lighting Control Console, Dimmer Pack.

Fresnels are designed for a situation where a small area needs to be highlighted which does not work well for large backgrounds or ambient light such as the Chromakey screen. Artist in residence, Huy Truong solved this problem by applying diffusers (type of flexible, translucent velum) to the spot lights and adding 8 fluorescent shop lights which also use the diffusers. Standard 48" fluorescent, double bulb, lighting fixtures are excellent for evenly lighting the background and even the actors. All the lights are covered by one of these Rosco diffusers:

- Rosco RSCLX diffuser #116 Tough White.
- Rosco RSCLX diffuser #102 LT Tough Frost.

Figure 4.29 Fluorescent lamps with filters.

The velum like sheets are attached to the light fixtures using common clothes pins.

For more accessory mounting options extra stands and mounting equipment was acquired:

- 3 x Avenger A205SCB 40" C Stand.
- 3 x Avenger A205SCB 40" Century Stand.
- 3 x Avenger D200B 2-1/2" Black Grip Head.
- 3 x Avenger D520B 40" Extension Arm.
- 2 x Bogen Deluxe AutoPole.

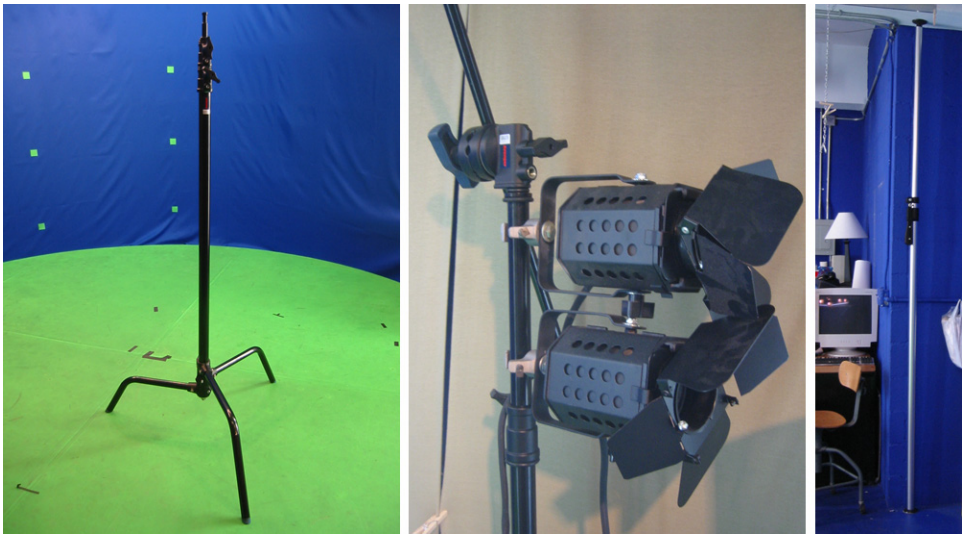


Figure 4.30 Lighting Accessories.

Lighting is an art form; but with the right equipment and with time desired effects can be achieved. For more information on lighting consult Dan Porvin's guide [33] (appendix). The following figure displays the typical arrangement of combination of lights. The working area is defined by the limits of the blue screen.

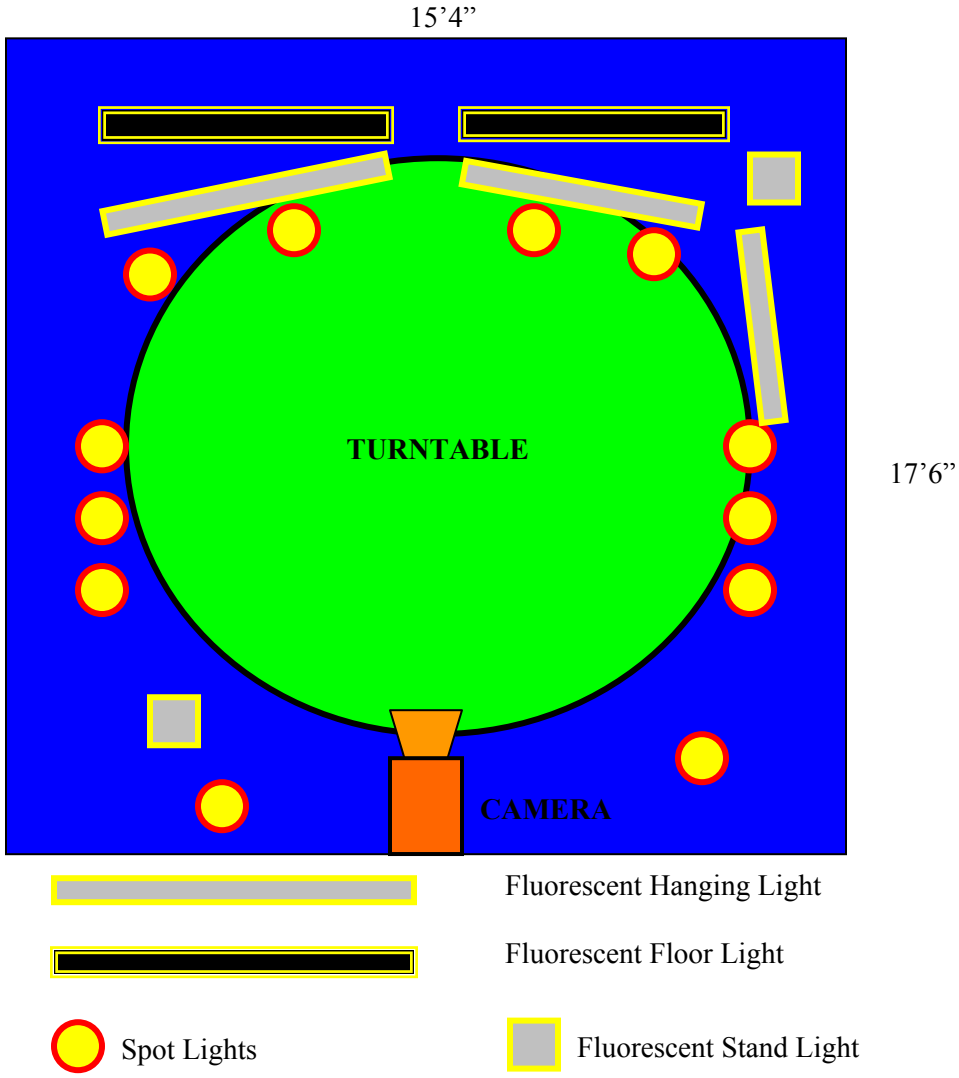


Figure 4.31 Sample layout of lights.

4.4.4 Audio Mixer and Microphones

Controlled audio recordings are essential for success of the broadcast and films produced in **StudioBlue**. Due to poor room acoustics, acoustic interference from adjacent street and weak robot speaker systems, the following high end audio system was put together.

- Two Audio-Technica AT 853a Unipoint Microphones.
- Two 10', 1/4 " jack to RCA cables.
- Samson Mixpad 12 inputs.

The ambidirectional microphones were sufficient for actors; however the robot speech came in muffled so a Shure Wireless Microphone system (used in *Camouflage Town*) was used as a remedy. The wireless transmitter accepts ¼' XLR accessory plugs, in this case a conference microphone.

Figure 4.32 Audio Mixer / Shure Wireless Receiver and Transmitter.

Unacceptable noise from the street was diminished by boarding up two out of three windows with ¼" plywood, which was done by the maintenance department. The window directly opposite the control stations was left untouched, due to the placement of Air Conditioning unit and to allow some natural sun light. During performances the window was covered with fabric for replicable lighting consistency.

4.4.5 Supplementary Video Equipment

High resolution video monitors are used for live preview from video/audio feeds. JVC High Resolution 13” color monitors (TM-H1375SU) can accept and switch between two audio/video feeds.

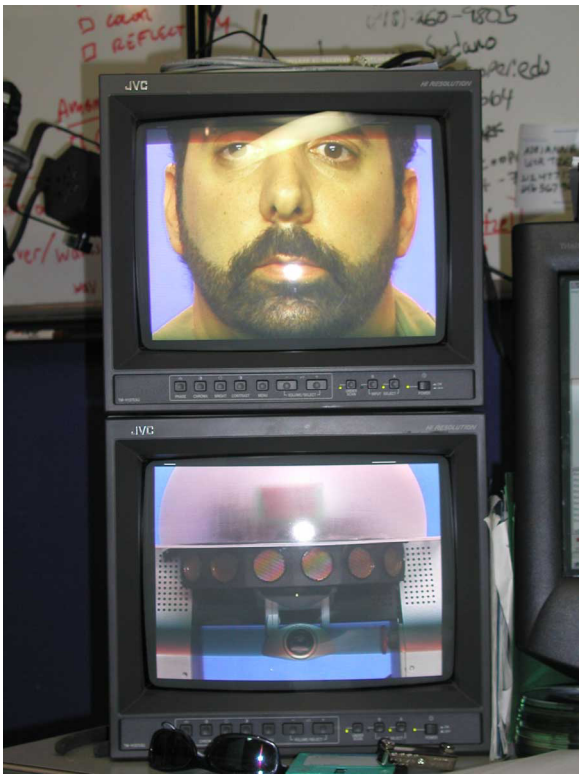


Figure 4.33 High Resolution JVC 13” color video monitors.

Vertical Interval Video Switcher (12/2) made by Comprehensive (CVG-1202RS232) can switch between 12 different sources and send them to two different outputs. It can be controlled via serial communication (RS232). James Cruickshanks has written a Java graphical application to automate the switcher and give it ability of remote control over the Internet.

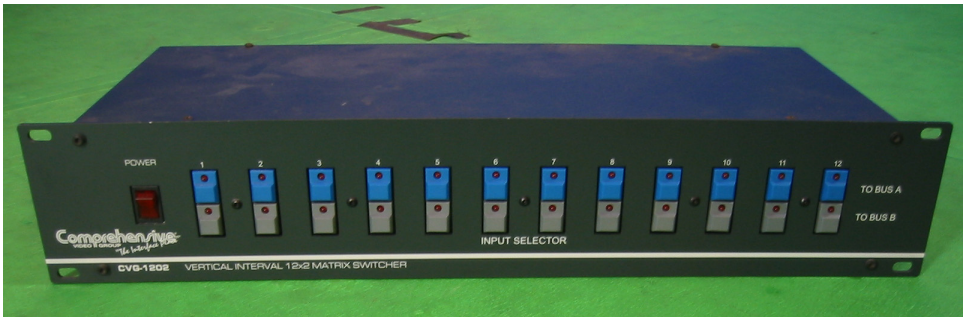


Figure 4.34 Vertical Interval Video Switcher .

- Bescor TX-25 New Tek Heavy Duty Fluid Head Tripod (30mm leg)
- Fuji DVM 60 minute MiniDV (digital) tapes

A number of BNC to RCA 50 foot cables were purchased to connect to the CCD cameras. An assortment of BNC to RCA, and other RCA connectors were also acquired.

- 3',6',12' Composite RCA Audio/Video Calves.
- 25' standard RCA male to male cable.
- RCA to BNC adapters.

4.4.6 Computers

Currently **StudioBlue** houses five workstations and a laptop. They are wired together via a Netgear DS309 100/10 Mbps 10 port Ethernet Switch and/or IEEE 802.11b (WaveLan) wireless access point which also provided the necessary Internet and local network connectivity for the robots.

Figure 4.35 Computers.

The three Dell Dimension Series PCs have ample RAM (1 Gigabyte), latest video cards (128 Mbs), 1.8 GHz processors and run Microsoft Windows 2000 operating system. Another older Dell Dimension machine (1 GHz) runs *Linux* Redhat 7.2.

An IBM ThinkPad laptop is used for remote development and control. Last addition was a Mac G3 functions as a video editing station. Most of these machines are typically used for software development and testing; however during performances they may take on another role.

4.4.6.1 Video Encoder

One of the Dell stations has two ViewCast Ospray 200 PCI capture cards for video and audio capture exhibiting 30 fps performance for uncompressed video. The machine can encode the Audio/Video stream and send it out to a streaming server or create digital archive.

4.4.6.2 Video Tracking

One of the more powerful Desktops can be used for robot localization if image based navigation system is used; otherwise, it can be setup to control AmigoBots or used for general software development.

4.4.6.3 Robot Control / Web Server

The third high end Machine is reserved for robot control applications such as **GRASP** and/or a Web or Media Server. Web server function typically does not require much processing power unless high traffic is expected.

4.4.6.4 AmigoBot Control / Linux Interface

An older DELL 1 GHz machine is running *Linux* operating system (*Redhat 7.2*) and is used for robot maintenance and operating the required robot-side applications. This is favorable over *Microsoft Windows OS* due to issues arising with exporting *X Windows* [35] to non *UNIX* environment. This machine also has two 2-port PCI Serial Accelerator I/O cards made by StartTech which are used to interface with AmigoBots.

4.4.6.5 Video Editing

The video editing station is used for video and audio post-production. It has FireWire port to directly interface with the MiniDV cameras and RCA A/V inputs to connect with a VCR. It is composed of

MAC G3, Apple made computer, a FireWire I/O, M100 I/O card, a Super VHS VCR, and a High Resolution Video Monitor. *FinalCut Pro* is the primary video editing software package used for non-linear editing.

4.4.6.6 Wireless Control

A ThinkPad A20p IBM laptop is used for software development, presentations, and remote control. It has an 802.11b wireless card which connects to the wireless Ethernet network in **StudioBlue**.

4.5 A/V Connections / Web Interface

Tele-operation was one of the original aims in creation of **StudioBlue** and drove the design of control software platform. Once the robots are wirelessly connected to an Ethernet network they can be controlled from any computer in the world via Internet. **RCCI** coupled with **GRASP** provide the means to control the robot functions, however audiovisual interaction is done via other systems.

As for Audio and Video, the limiting factor is the encoding (compression) type and bandwidth issues when Internet if the transmission medium. This is a difficult task in web based A/V robotics especially with the limitations of the Internet, current audio/video streaming capabilities and limits in Cooper Union's resources. *Camouflage Town* used Real Media technology which encoded video and audio at the performance location sent it to a streaming server located in California. From there it was distributed to the users [12]. A significant problem arose due to the interactive nature of the exhibit and the limit of the technology resulting in a 10-30 second delay between round-trip audio/visual exchange, which was irritating and confusing to the user. *Real Player* streaming has acceptable bandwidth requirements but uses a large buffer responsible for the delay. With the advent of *FlashMX* and *FlashMX Server* (Macromedia) video and audio streaming, real time interaction is plausible with a typical delay of five seconds or less. The audio quality is excellent, while the video quality is related to amount of motion (difference from frame to frame) as well as available bandwidth and encoding choice. The other advantage of *FlashMX* is that it is platform independent

and provides nice interface creation capabilities (including animation), and is widely used on the Internet. For more information on this topic consult James' Cruickshanks thesis [6].

4.6 General Usage

The lab could be used purely as a general robotics software development facility but is much more functional in producing interdisciplinary works of art. There are two types of approaches to using the lab: video production or live performance.

4.6.1 Video Production

Video production is the preferred method for beginners to **StudioBlue**. The process involves working on a small part of the script at a time, which gives a great deal of control to the director and the actors. Chromakey and special effects are much easier to achieve in post production editing and can be redone if not satisfactory.

4.6.2 Live Performance

Theatrical or live performance is uninterrupted (except few planned intermissions) and done sequentially as one event much like classical theater. The performers go through the script without repetition while audience watches in person or over the web in real time. This leaves no room for errors and requires great preparation and discipline. Since this method is a real time production, it demands on the fly alteration of backdrops, mobile audio microphones, and special effects. Furthermore, it is much more difficult to produce due to robots inability to repeat its motion exactly and putting more pressure on the actors to adapt to the situation. The performance can be archived and then altered as a video production.

5 Robot Control and Communication Interface

StudioBlue has highly functioning robots capable of moving forward, rotating, avoiding obstacles, manipulating objects with a gripper and even speaking. These functions are typically utilized by "hard coding" them in control software in **C++** programming language [36] using complicated robot's application programming interface (API) and invoking them via key strokes or automatically upon preprogrammed event trigger such as sensor data, or timed alarm. Such application typically runs on robot's onboard computer and is interacted with via terminal emulation application. Through this approach direct human input is required to control the preprogrammed functions and there is no immediate ability for other software to do so. To create desired control code extensive knowledge of robot API and its programming language (**C++**) is required and is often very time consuming; both are discouraging hurdles in discovering the joys of working with robotics.

Hence, a need for a simple interface for the robots that would fit in the multidisciplinary environment of **StudioBlue** was discovered. High level, platform independent communication interface would be optimal for controlling most of the robot functions and feedback systems. High level means that the most of the mechanics of performing a task are taken care by a subsystem. For example, a high level "go to" position instruction consist of the robot rotating towards a goal, accelerating to a preset maximum velocity and then decelerating to precisely arrest at the desired position accounting for distance and heading. Furthermore, lower level control system controls each wheel's velocity, translates sensor feedback into usable data, etc. Extensive mapping of robot functions and feedback services was required for the user to be able to fully control the robot behavior.

Such communication and control interface should also be effortlessly utilized by inexperienced users as well as software developers with minimal programming, hence platform and client application independence are design priorities. It was decided that most computers have *Telnet*

terminal emulation application which allows a user to connect to another computer (i.e. web server) and enter text commands that can be executed as if s/he was entering them directly on the server console. Using this application to control the robot using intuitive text commands would be useful immediately. Moreover, software written in any modern language probably has **TCP/IP** capabilities which can create *Telnet* like connections and send text and receive text from the host which would mean that exactly the same command syntax can be sent by a *Telnet* user or an application. Furthermore, **RCCI** supports multiple concurrent client connections creating new possibilities for distributed and collaborative sensing and control.

Typically, robot software is created in layers where each layer builds on top of another to accomplish increasingly complicated tasks; however, a layer does not need to know how other layers implement their functionality. The lowest software layer, the "mobile robot server", is a mobile robot platform that provides a set of robotic services in a standard format. Above layers are clients of this server and are not confined to particularities of any one robot, as long as they adhere to common the server protocol. This client/server paradigm fits perfectly in Object Orientated programming methods [37] and is directly used in ActivMedia's robots. It was also applied in design of **RCCI** model**. RCCI's** architecture is very modular allowing robot function components to be altered, added, removed and combined without disturbing the structure of the model. In this way one could port **RCCI** model to other non ActivMedia robots as long as they implement the command structure defined in the following sections.

In order to develop this software package, great effort was invested in becoming familiar with *Linux*, **C++**, ActivMedia's API, robots and general programming conventions. Although the code is extensively commented knowledge of the above is essential to understand and manipulate **RCCI**.

5.1 Existing Projects

5.1.1 A Web-Enabled Communication Platform for the ActivMedia PeopleBot

For his Master's thesis at Cooper Union, James Cole created a Web-Enabled Communication Platform (WECP) for a single ActivMedia PeopleBot implemented on *Kiru* for *Data Dynamics* exhibit and later on to be used in Cooper Union's Robotic Theatre [11]. His work integrated the **Saphira 6.2** programming environment, the **Botspeak** *Via Voice* interface, the PTZ Camera drivers, the ACTS Color Tracking software, and *Windows'* multimedia functionality through a **TCP/IP** communication channel. Shortly after the completion of the work ActivMedia created **ARIA** and SRI Research Company rewrote its **Saphira** package using **ARIA**. In addition, more ActivMedia robots were acquired; all included *Linux* Operating System. *Kiru* was also converted to use *Linux* (from *Microsoft Windows 98*) and the new **ARIA** programming interface rendering Cole's code obsolete without major redesign.

Cole's Web Enabled Communication Platform (WECP) was created to "fit the needs of the *Camouflage Town* exhibit's program's director, Adrienne Wortzel " [11] and was not designed for the robotic theater, especially since the desired functionality was not known. WECP's structure included many behaviors and camera movements that were hard-coded requiring recompiling of the application to alter any part. Moreover, the software model was tailored to only one robot, which was solved in **RCCI** by designing a flexible modular architecture allowing fast integration (even plug and play in some cases) into other ActivMedia ARIA based robots.

Coles **TCP/IP** Command Interface syntax was simple and concise, but rather inflexible and a slightly vague. The fact that his system does not allow for mixing numeric and text commands, a workaround was implemented via two types of command types. Since at the time of development only one function required text arguments, the talk command was one type of command:

t_ [TEXT] _ \0

while all other commands fell under the standard command type:

[CHAR][CHAR][CHAR] _ [NUM] _ [NUM] _ [NUM] _ [NUM] _ [NUM] _ \0

For example, "mvi 5 \0" would represent "move at a velocity of five inches a second." The standard command type had a maximum of five numeric characters and three layers of command structure (3 chars). A more flexible messaging structure was desirable: one that would allow for any number of mixed types: numeric and alphanumeric arguments. It is also important to keep the same messaging pattern for response messages, so same content independent parsing and message creation algorithm was applicable. Unlike WECP, the message listener/handler model used in **RCCI** clearly separates the parsing from command interpretation improving modularity and maintenance.

It is noteworthy to credit Cole's genius in resolving robot localization problem via his Constellation Vision Navigation System. A camera mounted on the robot facing the ceiling would "looked" for three colored discs mounted on the ceiling. Image processing engine would determine the colors and the orientation of this discs (arranged in a triangle) and compare them with a database of the constellations' real world coordinates and would use equations similar to those used in trilateration to calculate its position and orientation.

5.1.2 The Player/Stage Project

The Player/Stage Project is a set of Open Source development tools for robot and sensor applications [38]. The player seems to be most similar research project to **RCCI**. It is used with complementing robot simulator called Stage.

5.1.2.1 Stage

> *"Stage is a scaleable multiple robot simulator; it simulates a population of mobile robots moving in and sensing a two-dimensional bitmapped environment, controlled through Player. Stage provides virtual Player robots which interact with simulated rather than physical devices. Various sensor models are provided,*

*including sonar, scanning laser rangefinder, pan-tilt-zoom camera with color blob detection and odometry." [*38*]*

5.1.2.2 Player

Player is a device server that provides a powerful, flexible interface to a variety of robotic elements. Since Player uses a TCP socket-based client/server model, robot control programs can be written in any programming language and can execute on any computer with network connectivity to the robot. In addition, Player supports multiple concurrent client connections enabling distributed and collaborative sensing and control [16].

Similarly to **RCCI,** Player attempts to provide a general high level programming interface with a belief that users will develop their own control system and/or user interfaces. Furthermore, Player's **TCP** socket abstraction can interface with "virtually any programming language making it much more "minimal" that other robot interfaces "[16].

The distinct difference between **RCCI** and the Player is that **RCCI** uses intuitive text messaging, and is tailored towards interaction with non-programmers. Although the initial goal of **RCCI** software was to implement control functions for many different robotic platforms, as done by the creators of Player, only the ActivMedia suite of robots was covered. Moreover, **RCCI** communication syntax is much simpler and its straightforward architecture more manageable. The following figure displays Player's message header.

Message header fields and types

Byte 0 Byte 31

STX	Type	device	index	t_sec	t_usec	ts_sec	ts_usec	reserved	size
short	**Short**	**short**	**short**	**int**	**int**	**int**	**int**	**int**	**int**

Figure 5.1 Player Message syntax [16**]**.

Player's communication interface is much lower low level than **RCCI**, although it is more compact (and cryptic) and therefore likely uses up less bandwidth. The first "short" - STX is a special symbol that signals the start of a message. The type designates the type of the message to follow whether it is

a command, request, data, acknowledgement, etc. Following is the time structure, and the last element in the header specifies the length of the data that is attached to this message.

5.1.3 MissionLab v6.0

The main goal for MissionLab project, done at the Mobile Robot Laboratory at Georgia Institute of Technology, is to "control the motion of a robot or groups of robots in highly dynamic, unpredictable, and possibly hostile environments." [39]. MissionLab v6.0 is a multi-agent robotics mission specification and control software. It takes high-level military-style plans and executes them with teams of real or simulated robotic vehicles. MissionLab is quite cumbersome for a novice as it is composed of the following modules [40]:

Mlab - a console-like program from which a user monitors the progress of experimental runs of the robot executables.

The Configuration Editor (**CfgEdit**) - a graphical tool for building robot behaviors. It generates source code which, when compiled, can directly control a simulated or real robot.

Configuration Description Language (CDL) – CDL is generated by CfgEdit and is translated by a code generator into CNL (Configuration Network Language) code.

CNL - A compiler compiles CNL code generated by the CDL code generator, and produces C++ code which is then compiled with the GNU C Compiler (gcc), the compiled program (or robot executable) may now directly control a robot.

CBRServer – a CaseBased Reasoning Server generates a mission plan based on specs provided by the user by retrieving and assembling components of previously stored successful mission plans.

HServer - a Hardware Server that directly controls all the robot hardware, either via **TCP/IP** or a serial link, and provides a standard interface for all the robots and sensors. The CfgEdit

generated code uses this standard interface to control the real robots. HServer also provides direct control, configuration, and status of the robots and sensors.

MissionLab, although extensive and modular, is very large and complex and using this type of software for interdisciplinary development at **StudioBlue** would be impractical.

The component relevant to the RCCI project is the HServer. It implements only one of the ActivMedia Robots and requires a graphical interface. Its keyboard interface is very efficient as it uses single characters to activate a command (i.e., 'p' will turn on/off sonar sensors). Very few functions are implemented and do not accept arguments. [40]

5.2 ARIA and Saphira

ActivMedia Robotics Interface for Application (**ARIA**) is written in the **C++** language in an object orientated paradigm. It functions as a robot control applications-programming interface for ActivMedia Robotics' line of intelligent mobile robots [18]. **ARIA** is versatile and flexible client-side software for easy, efficient management of the robot server, as well as access to accessory robot sensors and effectors. It can be run multi-threaded using its own wrapper around *Linux pThreads* and *WIN32* threads. It can be used in many different ways, from simple command-control of the robot server for direct-drive navigation, to development of higher-level intelligent behaviors such as obstacle detection and avoidance (intelligent navigation).

ARIA sits on top of the mobile server that is found embedded on the robot's microcontroller. The server manages low-level tasks of robot control and operation, including motion, heading and odometry, as well as acquiring sensor (sonar, Infra Red) information and driving accessory components like the PTZ camera.

Since **ARIA** is released under the *GNU Public License*, any distributed work which uses **ARIA** must be distributed with the entire source code of that work (open source) [41]. **ARIA** is continually updated and version 1.3.2 was used in development of **RCCI**.

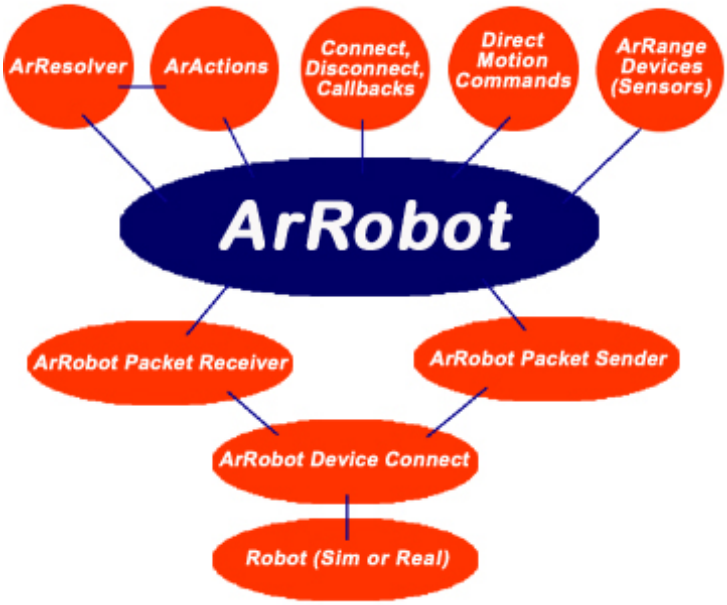


Figure 5.2 ArRobot - The Main Control Class for ARIA [42].

A good example of implementing **ARIA** for high-level intelligent tasks is **Saphira** Robot Control System [20]. It was developed by Kurt Konolige at the *SRI International's Artificial Intelligence Center*. **Saphira** has a real-time path planner based on the gradient method which is typically used with Markov Localization to keep the robot registered with a map as it moves through its environment. Gradient Path Planning is a process for determining optimal paths for the robot taking into account both local obstacles, sensed by sonar and/or laser range-finder devices; and global map information such as the location of walls and other structural obstacles [17].

Since **Saphira** is implemented using **ARIA** API, it is updated regularly to correspond with the latest **ARIA** version. **Saphira** version 8.2 was used in **RCCI**. It is not open source.

5.3 Speech Synthesis Software

Being very particular to humans, speech seems to be an impressive element of theatrical performance involving robots, as it has been demonstrated by K-Aft Robot Radio where all the performers are computer generated voices [43]. It is imperative that this aspect of the theater is technologically

advanced as possible. Text-To-Speech (TTS) apparatus should control voice characteristics (gender, age), voice style, allow for specifying emphasis, tone, intonation and pauses to give the user most flexibility to create desired voices and embed emotions in the dialogues.

ViaVoice Outloud (TTS) [44], developed by IBM, was part of a package originally implemented by ActivMedia not only to synthesize speech but also to perform voice recognition. Its TTS capability was successfully utilized by James Cole in *Camouflage Town* [11]. Since IBM has discontinued *Linux* versions, ActivMedia has adopted *Festival*, which does speech synthesis, and *Sphinx* which does speech recognition. Both are implemented in the **ArSpeech** class. They are not as easy to use as, and produce lower quality speech than **Botspeak** (a wrapper for *ViaVoice)* and therefore will not be implemented in this version of **RCCI**. **Botspeak** and *ViaVoice* for *Linux* were included with **Saphira 6.2** prior to the release of **ARIA**.

Controlling synthesized text via **Botspeak** is done simply by inserting annotations in the input text; the input text string is an argument in a **Botspeak** TTS function. Pitch, head size, roughness, breathiness, tone, pitch fluctuation, speed, emphasis can all be controlled by inserting a string of characters in the text to be converted to speech. For example, *`vrN* changes the roughness for the text that follows with N in range 0 (smooth) to 100 (rough).

5.4 Robot Functions

It has been determined that the following ActivMedia robot functions should be implemented for the use in the **StudioBlue** theater. They represent the most used functions and robot state information.

5.4.1 Motion

- Move forward/reverse mm.
- Rotate in degrees.
- Move to position x, y in mm.
- Set transversal velocity.
- Set rotational velocity.
- Set transversal acceleration.

- Set transversal deceleration.
- Set rotational acceleration.
- Set maximum transversal velocity.
- Set maximum rotational velocity.
- Set current position (x, y, heading).

5.4.2 Robot Status

- Location coordinates, x, y, heading angle.
- Forward/Reverse velocity.
- Wheel velocities.
- Heading.
- Battery voltage.
- Sonar sensor data.
- Table sensor data.
- Bumper sensor data.
- Robot name.
- Robot type
- Sonar on/off.
- Motors on/off.

5.4.3 Gripper

- Gripper system halt.
- Gripper system store.
- Gripper system ready.
- Grip stop.
- Grip close.
- Grip open.
- Lift stop.
- Lift up.
- Lift down.

5.4.4 Camera

- Pan.
- Tilt.

- Zoom.
- Set tilt slew.
- Set pan slew.
- Backlighting on/off.

5.4.5 Audio

- Speech synthesis.
- Digital sound file playback.
- AmigoSound playback.
- AmigoSound on/off.

5.4.6 ARIA Actions

- Bumper avoid on/off.
- Front avoid using sonar on/off.
- Rear avoid using sonar on/off.
- Table avoid using IR on/off.

5.4.7 Saphira

- Gradient Path Planning.

5.5 Operating System

Linux was chosen as it is a stable, secure, modular, and controllable software operating system with good documentation and large Internet community. One of its more attractive aspects is the integrated *X-Windows* system (*X11*) which allows for remote graphical control. Essentially, the screen shot of the robot computer can be easily be displayed on any *X* enabled computer in the world. *Linux* (*UNIX*) is rather transparent compared to *Microsoft Windows* system, allowing the administrators to fully control the processes, permissions, boot sequences and logs. It is open source making it very customizable. It is know to be very secure and widely supported on the World Wide Web. Most of the coding was done on *Mandrake 9.0 Linux* distribution with a robot simulator that is bundled with **ARIA**, while the robots operated on *RedHat 7.0*.

A useful tool in *Linux* is a shell script which is a text file with a set of instructions that the operating system will execute as if the user typed them on the command prompt. Shell scripts issue commands, start processes, control jobs, redirect input and output, etc. [45,35]

In case of **RCCI** the following bash script was used to analyze user input, adjust volume, shutdown any existing instances of **RCCI**, shutdown Text-To-Speech engine, restart the engine, and run **RCCI** with the user input port.

```
#!/bin/bash
echo "RUNNING RCCI: "
if [ $# -ne 1 ]; then
         echo 1>&2 Usage: $0 PORT
         exit 127
    fi
#echo "Adjusting volume and muting microphone "
aumix -w 50
aumix -v 50
aumix -m 0
while ((1>0))
do
echo "Killing ViaVoice engine, Botspeak and RCCI"
killall -9 rcci
killall -9 botspeak.srvr
killall -9 engine
sleep 2
#restart the ViaVoice engine
/usr/lib/ViaVoice/bin/engine
sleep 3
echo "Attempting to RCCI Server on port: " $1
./RCCI $1
sleep 10
echo "TRYING AGAIN"
done
```

The script is activated from the command line by typing in the name of the script file and the *PORT* argument:

./start 5555

If **RCCI** quits for any reason, this script will start it up again. The above retry function was specifically created to solve a problem of delayed clean up of sockets. Sockets, particularly, on *UNIX* based systems behave different than on *Windows* platform, in that, when they are not used anymore, the operating system automatically cleans them up for repeated use. However this takes time and immediate reuse of a socket port is not possible and unpredictable; clean up takes on average a few

minutes. The script automates the testing process and keeps trying until the socket is available. Choosing another port number is another option.

5.6 C++ Development

C++ is probably the most widely used programming language in the world, as it is backwards compatible with ANSI C, another popular high level programming tool. Although newer versions are available, GNU's C++ compiler (*GCC*) version 2.95.3 was used as it is considered very stable and is compatible with **Saphira** and **Botspeak** libraries which were compiled with that version [46]. The new *Linux* distributions come with *GCC* version 3 and above and therefore require installation of version 2.95.3. The process is clearly document on the web [46].

Following common coding guidelines, the **C++** interface for implementers is located in ".h" (header) file which is sometimes called the prototype definition. While the module's (class) definitions or implementation of functions is placed in ".cpp" files. The header files contain sufficient comments presenting what the class does and how to utilize it [35].

5.7 Development Environments.

A combination of text editors and integrated development environments (IDE) was used in the software development process. *VI Improved* (*VIM*) was used to make small code alterations when a graphical interface was not possible, i.e. in the case of *Telnet* sessions [47]. Otherwise, *KDevelop* or *KEdit* text editors tailored for **C++** editing were used extensively to work on multiple files simultaneously. All these are part of the *KDE* Desktop environment and come bundled in most *Linux* distributions [48].

For commenting and cleaning up the code *BorlandX* for windows was used for its friendly interface and excellent code formatting capabilities, as well as clear color coding and publishing.

5.8 Code Documentation.

Extensive commenting is almost as essential as the quality of the code itself. It improves understanding and maintenance of the software. All the classes in **RCCI** follow *Doxygen* type comment formatting [49]. They are extracted by *Doxygen* from the header files (*filename.h*) and used to automatically organize and create a clear and easily navigable documentation in HTML format which is included in the appendix.

5.9 Compilation

After the *GCC* (g++) is setup to be the default compiler, required ActivMedia libraries are installed, and the environmental variables are set, compilation is possible. It is imperative that **ARIA**, **Saphira** and **Botspeak** are installed in their default directories and that their working paths are added to the *PATH* variable as instructed in the appendix.

The typical approach to compiling is to create a compiled object file (filename.o) for each class (filename.cpp) and then to link all the compiled objects and required libraries to form an executable. When a software project includes many libraries and class files, automating the process is necessary. *Makefiles* are much like shell scripts but are adapted for compiling applications [35]. Great advantage in using *makefiles* is that only the files that have been altered (i.e. a class) since last compilation will be compiled and re-linked, saving time. *Makefiles* also simplify compilation options and the locations of include files and corresponding libraries. See the appendix for *makefile* used to compile **RCCI**.

5.10 SOFTWARE DESIGN

Modularity was a large factor in the design of **RCCI**. Updates of **ARIA**, additional robot functionality, and general software progress were reasons to make **RCCI** as modular as possible. **Figure 5.3** displays the hierarchy of components participating in robot – client communication and command. Client is shielded from the intricacies of components controlled through **RCCI**. The interaction between the two top layers is via **TCP** with **RCCI** taking the role of the server.

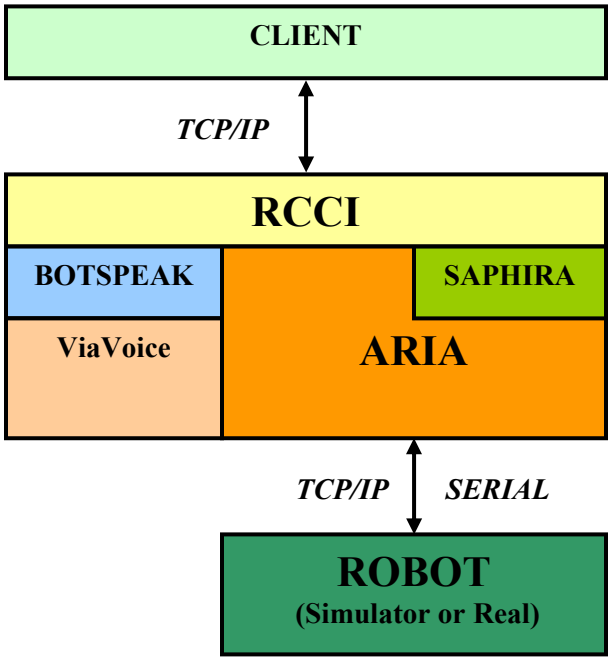


Figure 5.3 Component diagram.

RCCI translates command from formatted string into structured class data and then distributes (via functions) it to other modules. **RCCI** directly controls **Botspeak** which uses the *ViaVoice* engine to synthesize text. **RCCI** similarly commands **Saphira** which implements its functionality via **ARIA**. The rest of the commands are interpreted by **ARIA** which has direct control over the robot and its accessories. This communication is done via **TCP/IP** if the simulator is used or RS232 (serial) if the real robot is connected to the machine running **ARIA**. **Botspeak**, **ARIA** and **SAPHIRA** are all integrated into **RCCI**. **RCCI** also applies some of the **TCP/IP** communication and threading modules provided in **ARIA**.

Threading was implemented using **ARIA**'s **ArThread** class, although its usage was kept minimal. Using threads increases the complexity of the code and often leads to slower performance due to resource demanding data lock and unlock mechanism. Moreover, debugging threaded code is very difficult and often not intuitive.

The software entrance point is through the main function which accepts command line argument specifying the port number used by the **TCP** server. The **main** function initializes **ARIA** with dedicated signal handling thread. It then creates and instance of **SbManagerT** class, which as the name suggests is the primary task manager of the software. The main class keeps running until **ARIA's getRunning()** function returns false.

SbManagerT manages communication infrastructure, and is the hub for control of all available robot function modules, see **Figure 5.4**. **SbManagerT** creates an instance of **SbServerML** and passes the port specified by the user (from *main()*), and then opens up a socket for clients to connect to. **SbServerML** also parses incoming client communication data, checks for connection and format errors and transforms the raw data into **SbMessage** format. The manger also creates and instance of **SbMsgHandlerSingleton** that is responsible for inter-module communications. It serves as a hub for passing information between different classes and indirectly (via **SbServerML**) the user. The class implements the Singleton method which is a computer science design pattern to ensure a class has only one instance and provide a global point of access [50]. **SbMsgListener** is a pure abstract class, a common interface for objects that are interested in "listening" to the messages within **RCCI**. This includes messages from **TCP** clients. Classes that implement **SbMsgListener** can be registered with **SbMsgHandlerSingleton**. This design paradigm allows the handler to interact with different classes without worrying what the class actually does. Furthermore, each listener specifies what types of messages it is interested in receiving which tells the handler where to distribute the messages as they arrive. A module that implements **SbMsgListener** may respond to more than one type of message.

Once all instances of listener modules (i.e. **SbMotion**, **SbSounds**) are created and registered with **SbMsgHandlerSingleton, SbManagerT** thread becomes active and runs until **ARIA** is shutdown. Since this is the primary control function (**ARIA** runs in its own thread) of **RCCI** it runs rather frequently with a 50 millisecond sleep once every loop. First, the robot is checked for

established connection, followed by a scan of the vital sensors, such as a stall or triggered bumpers (*SbRobotML->checkOnce()*). Upon disconnection from the robot server, **SbRobotML** will send one disconnect message, but will continuously report stall or bumper via messages to the user (or print them out on command line if the client is not connected) until they become inactive. Following its 50 millisecond "nap" **TCP** server is checked for new connections, connection errors, and new messages (*tcpServer->cycleOnce()*). If a new message was received, its type is checked for **SHUTDOWN** keyword, in which case **RCCI** disconnects from robot, disconnects all TCP clients and exits the program. Otherwise, the message is handled by **SbMsgHandlerSingleton** and distributed to registered listeners (***SbMsgHandlerSingleton****->fireMsg(msg)*). All registered listeners hold a reference (pointer) to an instance of **SbMsgHandlerSingleton** and therefore could use the handler's functions to fire a response message. The process is repeated until it is stopped by the user or the operating system. **SHUTDOWN** command message can only be sent by a client and at the moment is not part of the inter-module communication system.

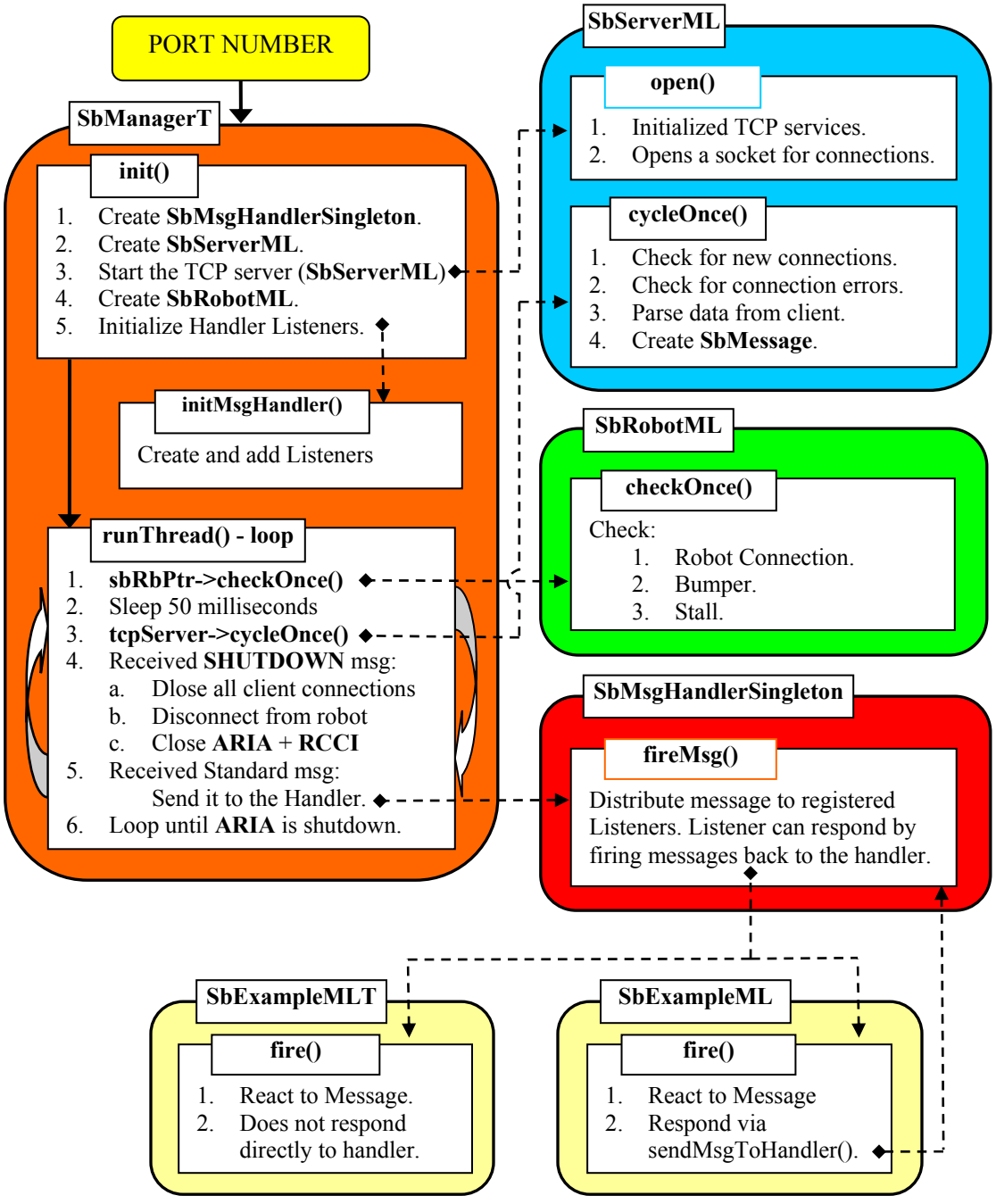


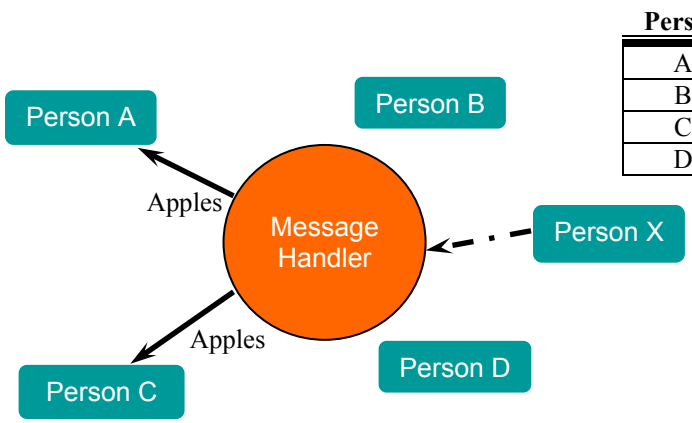
Figure 5.4 Main Logic Flow Chart.

5.10.1 Messaging system

Messaging system implementation described above is at heart of this software design. **SbMsgHandlerSingleton** manages references to listeners registered with it. Each of these listeners has message types associated with it that get "forwarded" by the handler.

One can think of this system as a mass mailing machine (

Figure 5.5). Starting with a pool of people interested in different subjects (listeners: A, B, C, D). If person A was interested in apples and pears s/he would register with the mass mailing (handler) service to be put on the apple and pear list while B was interested in pears only, C in apples, pears and oranges and D is not interested in fruits at all. Once information is available on apples, the handler will mass mail A and C, and if the information is on pears A, B, C will be notified, while C will be the only one to get the mailing if oranges are "in the news." D will never receive a mailing.



Person	Apples	Pears	Oranges
A	X	X	
B		X	
C	X	X	X
D			

Figure 5.5 Message System Analogy.

The message is required to have type associated with it along with the message information as in **Figure 5.6**. Furthermore, the message has to be flexible, and allow for various data types (string, integer, double) and be able to be easily translated in and out of text format to be sent over **TCP/IP**.

Figure 5.6 Message Prototype.

5.10.2 SbMessage Syntax

The idea is that each datagram should have a type specifying the category of the message allowing for quick sorting (addressing) and the rest of the message would be the data. The solution was to store all the elements of the message (type, data) in string format and convert the strings to necessary format when needed. This would allow the message to be very similar in the software and over **TCP**. Initially, *XML* (Extensible Markup Language) was chosen as the text formatting structure for the message to be sent over **TCP** for its popular use and flexible text format [52]. However, it was found that parsing *XML* is not trivial and the overhead in the text message structure (requires bandwidth) outweighs the benefits of its use. **SbMessage** takes care of storing this information and is easily transformed in and out of string format (serialized and de-serialized).

Figure 5.7 displays the syntax of message sent over **TCP** protocol. The suggested (default) delimiter is the pipe symbol: '|'.

Figure 5.7 Message Syntax.

A message begins with the message type, following by the rest of the components separated by one delimiter and ending with two delimiters and a new line character: '\n'. This character signals that the data in the **TCP** buffer should be read and forwarded to the message parser. The message parser looks for the double delimiter to create a message. A sample camera control message looks like this:

"CAM|zoom|56||"

It is imperative that the creator of the message does not include the delimiter as one of the arguments which could happen when sending string for TTS for example. Such mistake will cause the message to be formatted differently then expected.

As for internal structure of the **SbMessage** class, the message is stored as vector of strings. Each element, except for message type (always returned as a string), can be accessed as an integer, double or a string. This structure proves very flexible in implementation. Once the message is parsed

and created, it is up to the class that uses the message to know how many and what type of arguments it should expect. If the stored argument (as a string) is accessed and is not a number or it doesn't exist the **SbMessage** will return 0.

5.11 Components (Classes)

All classes created for this project have the prefix **Sb** for **StudioBlue** and in some case have a suffix suggesting their derivation. **ML** suffix stands for Message Listener and indicates that the particular class derives for **SbMsgListener**. The **T** suffix represents a class that has threading capability derived from **ArASyncTask** which in turn is based on **ArThread**. All class names begin with a capital letter, enums either begin with a capital letter or are all in caps, member function names start with a lower case. Each word in name is capitalized except for the first word in a function or variable: suchAsThis.

The following figures display inheritance diagrams for components of **RCCI**.

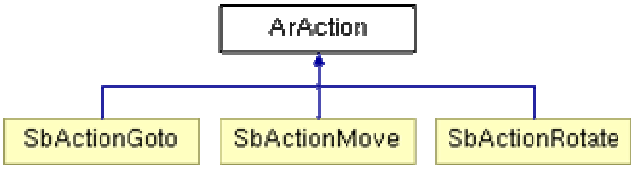


Figure 5.8 ArAction Inheritance Diagram.

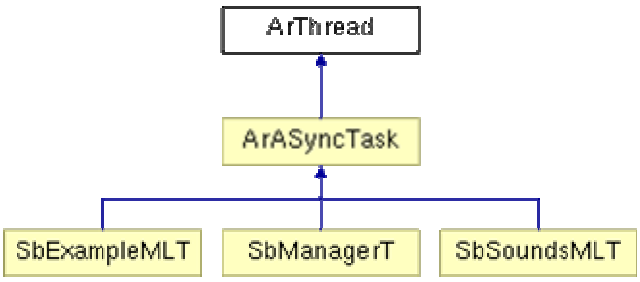


Figure 5.9 ArThread/ArAsyncTask Inheritance Diagram.

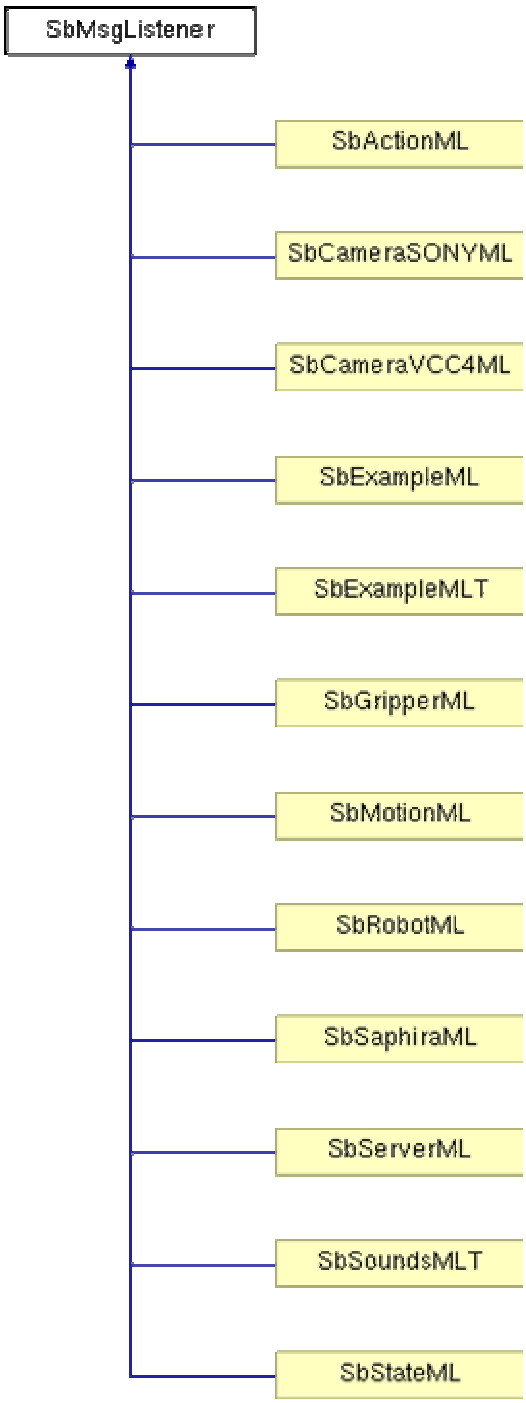


Figure 5.10 SbMsgListener Inheritance Diagram.

The following sections describe classes and their functions:

5.11.1 SbActionGoto

This action goes to a given **ArPose** although it does it very naively. Its design has been adopted from ActivMedia's **ArActionGoto**. The action stops when it gets *closeDist* away. A new goal can be set with *setGoal()*, canceled with *cancelGoal()*, and success state retrieved with *haveAchievedGoal()*. This doesn't avoid obstacles at all, an avoid routine at a higher priority may be used. For real and intelligent looking navigation something like **Saphira's** Gradient navigation should be used.

5.11.2 SbActionML

Manages **ARIA** and custom (**Sb**) actions that combine and control robot behavior. **SbActionML** listens for queries to control actions and manages their initialization. The more important actions are the **SbActionMove** and **SbActionRotate** which allow the user to combine the popular functions of **MOVE** and **ROTATE** to be used in combination with action limiting functions such as **ArActionAvoidFront**. Direct or motion command may conflict with controls from actions or other upper level process and lead to unexpected consequences. *robot->clearDirectMotion()* will cancel the direct motion so an action can get the control back. The other way to automatically do it is using *robot->setDirectMotionPrecedenceTime()* but this may block actions forever if set too high.

5.11.3 SbActionMove

Action that moves the robot forward and backward in a straight line. **SbActionMove** drives straight by a given distance. The action stops when it gets *closeDist* away. The new goal is given with *setGoal()*, the action is canceled with *cancelGoal()*, and its success returned by *haveAchievedGoal()*. This doesn't avoid obstacles, unless used in conjunction with other actions. This action was created to be used with higher priority actions to replace direct motion **MOVE** command.

5.11.4 SbActionRotate

Action that rotates the robot. **SbActionRotate** was required because the direct motion **ROTATE** command rotates in the direction of the smallest possible motion to achieve the required heading; specifically in the case when the turn angle is greater than 180 or less than -180. This prevented the user to rotate the robot in one direction continuously until goal is achieved. I.e. 750 degrees. This action is created so it can be combined with limiters of different kinds and types while using fundamental control of the robot.

5.11.5 SbCameraSONYML

Manages Sony EVI-D30 PTZ (Pan Tilt Zoom) camera. This class controls all the functions available for the camera (Pan/Tilt/Zoom etc) attached to the serial port on the robot.

5.11.6 SbCameraVCC4ML

Manages Cannon VC-C4 PTZ (Pan Tilt Zoom) camera. This class controls functions available for the camera (Pan/Tilt/Zoom etc) attached to the serial port on the robot. Cannon camera needs to be initialized prior to sending commands to it.

5.11.7 SbExampleML

SbExampleML is a template implementation of a class that inherits **SbMsgListener**. New code should be placed in the fire() function. *sendMsgToHandler()* will fire messages back to the handler.

5.11.8 SbExampleMLT

SbExampleMLT is an example implementation that inherits **SbMsgListener** and is threaded. This class inherits **ARIA**'s threaded **ArSyncTask**. It displays a possible way to use a separate thread to do something (calculation, control another process, etc) while the robot is running. Attention should be given to locking and unlocking the thread. This class is threaded and therefore should not use *sendMsgToHandler()* to fire messages back to the handler because handler is not thread safe unless it is called from **SbManagerT**.

5.11.9 SbGripperML

Manages Gripper functions. **SbGripperML** controls the gripper (Performance PeopleBot) if present. It controls the lift, gripper arms and returns the state of the gripper (open, close, liftmax, etc)

5.11.10 SbManagerT

Main thread manager class of **RCCI.** This is a threaded class which is the managing body of the software, supervising all the high level functions of **RCCI**. It takes care of timed TCP server updates, checking the robot state, since shutting down **RCCI** and registering **SbMsgListeners** with **SbMsgHandlerSingleton**. Singleton class needs to be destroyed by the last thread alive by calling *removeInstance()*, and is accomplished in this class.

5.11.11 SbMessage

Message structure class used for communications. **SbMessage** class stores message information in an organized string format. Each message is characterized by "TYPE" which inherently sorts the messages for processing in **SbMsgHandlerSingleton**. Following the type, arguments (strings) can be added and removed using functions, i.e. *addArgument()*. **SbMessages** can also be created using a formatted string delimited by a unique character ('|'). Each message is timestamped. Example: “CAM|zoom|56||”.

‘|’ is the default delimiter -> MSGTYPE|ARG1|ARG2|ARG3|ARGn||

5.11.12 SbMotionML

Manages robot motion functions. This listener parses motion requests. It commands the robot to "MOVE", "ROTATE", sets the "VEL", "LRVEL", "ROTVEL", "HEADING" and takes care of other motion related functions. Direct or motion command may conflict with controls from actions or other upper level process and lead to unexpected consequences. *robot->clearDirectMotion()* will cancel the direct motion so an action can get the control back. The other way to automatically do it is using *robot->setDirectMotionPrecedenceTime()* but this may block actions forever if set too high.

5.11.13 SbMsgHandlerSingleton

Manager for the messaging structure. Any class of type **MsgListener** can register with the handler along with a key (type). When *fireMsg()* function is called, it will "fire" (send) the message to all the listeners registered with the particular message type. Each listener implements a fire function which accepts the message and interprets it in any way it wants. Singleton concept was implemented for accessing **SbMsgHanglerSingleton**. It is required because some of the listeners need to send a message back to the handler. Singleton class assures a maximum of one object of its type at a given time and provides a global access point to this object; although, it may be problematic if threading is used. In that case, **ArMutex** should be implemented. This was avoided here for gain in speed. In the future, instead of making **SbMsgHandlerSingleton** as the singleton, the manager class should be the singleton and the reference to it should be possessed by all **SbMsgListeners**. However, these relations would be forced since listener is far away in function from manager (listener->handler->manager). Another way is to add lock/unlock feature to the handler.Singleton class needs to be destroyed by the last thread alive by calling *removeInstance()*.

5.11.14 SbMsgListener

Abstract class for the messaging system. An instance of an abstract class type can not be created; but pointers and references to abstract class types can exist. A class that contains at least one pure virtual function (i.e. *fire()*) is considered an abstract class. Classes derived from the abstract class must implement the pure virtual function or they, too, are abstract classes. This class provides messaging functionality; it stores the listener types that the listener wants to be registered with. Since the **MsgListener** can send messages back to the handler using *sendMsgToHandler()* via the pointer to the handler *SbMsgHandlerSingleton::getInstance()*. Since only one message handler exists, the listener can't register with more than one handler. In the future this could be improved using an array of pointers to different handlers.

5.11.15 SbRobotML

Manages robot connection and checks for errors (bumpers, stall, etc). Creates and instance of **ARIA's** robot interface. Provides connections to the real robot or simulator. It checks if the robot is connected, if any of its bumpers are triggered and if it has stalled. It also provides an interface for activating sonar and the motors. Returns some robot parameters: name, type, etc.

5.11.16 SbSaphiraML

Listener/wrapper for **Saphira** - intelligent avoidance software. This class calls **Saphira** function that avoids objects and intelligently maneuvers around ("IGOTO"). At each sync cycle (100 ms), the Gradient module calculates the lowest-cost path from a goal point or set of goal points to the robot. The algorithm starts by considering a local neighborhood connecting the robot and the goal or goals, and then expands its search if no path is found. There is a user-settable limit on the size of the neighborhood considered. Gradient uses a square-cell grid as a cost field for determining good paths. You can set the grid resolution; a typical resolution is 10 mm. The avoidance part can be turned off allowing for the robot to naively go to the desired position - "GOTO." The distance when the robot assumes it has successfully reached the goal can be set using "GETSETDONE", while "GOTOSETCLOSE" sets the distance to goal that the robot starts to slow down. "GOTOWINDOW" describes the size of the window that **Saphira** will look. Exact motion patterns of "GOTO" and "IGOTO" may be different every time due to sonar tolerances and electronic interference.

5.11.17 SbServerML

Manages **TCP/IP** server: multiple socket connections, send/receive **ASCII** data. This class uses **SbSockets** to listen for incoming connections. It allows multiple clients to connect. It will broadcast any message sent via *fire()* to all connected sockets. It does not automatically check for incoming sockets/messages, but can do so via the *cycleOnce()* function. Clients can disconnect by sending "QUIT" message and all clients are disconnected when "QUITALL" message type is fired. The "SEND" command broadcasts the data embedded in the message less the SEND type SEND is

stripped and the rest is sent to all clients.) The server is started with the *open()* function; commands are added with the *addCommand()* function and removed with *remCommand()*. The server is shutdown with *close()* function. **SbServerML** was modeled from ActivMedia's **ArNetServer**. The *sendMsgToHandler()* is used to fire messages back to the handler.

5.11.18 SbSocket

UNIX **TCP/IP** socket communication wrapper. **SbSocket** is a layer which allows people to use the sockets networking interface in *UNIX* operating system. All of the standard socket functions are implemented. This class also contains the file descriptor which identifies the socket to the operating system. It has been adopted from two ActivMedia's classes, **ArSocket.cpp** and **ArScoket_LIN.cpp** The ActivMedia original socket class was undocumented and limited the transmission of large data packets. This class was also implemented strictly for *Linux*.

5.11.19 SbSoundsMLT

Listener for audio playback including speech synthesis and digital audio file playback. **SbSoundsT** class inherits **ARIA**'s threaded **ArSyncTask** to take care of playing *wavs* and synthesizing without locking the robot cycling. It also allows for sending many consecutive requests. First use of the *fire()* function will initialize **Botspeak**.

5.11.20 SbStateML

Manages requests for robot's state. **SbStateML** gathers requested information and sends most recent state data back to the user. The information includes current motion/position, battery voltage, sonar states, table sensors, motor activity.

5.12 Command Interface Syntax

The messages are structured in a flexible format. Short message such as the STOP command is only composed of one element: the message type. (**Figure 5.11**) The **MOVE** command has two elements, the first one is the message type (command type) and the second is the argument of the distance to

move, negative means reverse. For more complex commands, two elements specify the type of message as in the **CAM** directive. First element specifies the type of message and the second is the subcommand, in this case specifying tilt. The argument follows. All **RCCI** responses are of type **MSG**, and follow with the type of information sent, and the arguments; Illustrated in **Figure 5.11** with **CLOSESTSONAR** message.

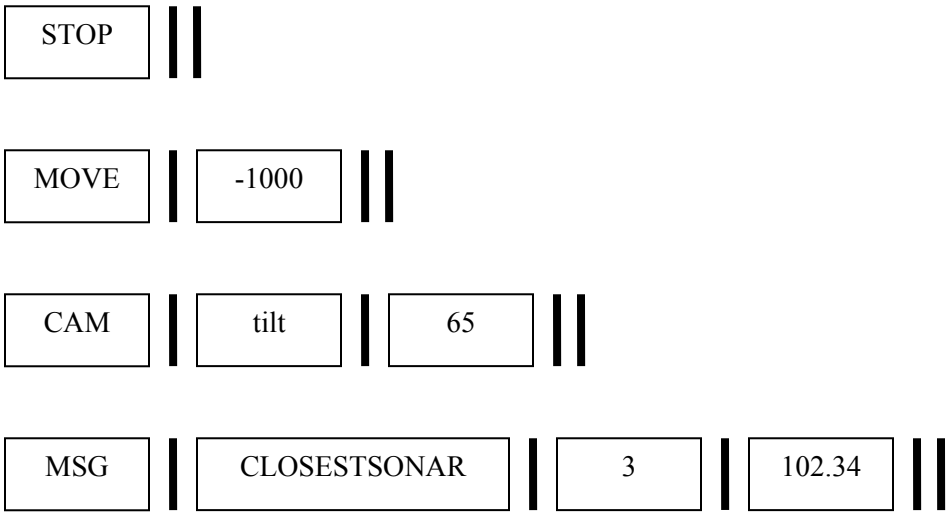


Figure 5.11 Sample Message Formats.

The command message type (first element) typically specifies the module that will process the message. The following are possible commands and their usage organized by location (class name) of their implementation.

5.12.1 User Commands

5.12.1.1 SbManagerT

SHUTDOWN||

Will disconnect from the robot by firing the **DISCONNECT** message, followed by closing client connections by calling **QUITALL**. TCP server will be closed next, and finally **ARIA** is shutdown leaving **RCCI** program.

5.12.1.2 SbServerML

QUIT||

Disconnects the client that has sent **QUIT** from the server.

QUITALL||
Disconnects all the clients from the server.

SEND|arg1|arg2|...||
Used to send messages back to the TCP clients. The SEND is removed and the rest of the message is broadcast to all clients. For example, message "SEND|MSG|left|23||" will be converted and sent to the client out as: "MSG|left|23||"

5.12.1.3 SbMotionML

MOVE|double||
Commands the robot to move forward(+) or in reverse(-) by a given distance in millimeters. The robot's direct motion move command actually uses short for the movement, therefore the input limits are + - 32767 mm.

VEL|double||
Sets constant velocity in mm/sec, forward(+) and reverse(-).

LRVEL|double|double||
Sets the constant velocity of each wheel (LEFT|RIGHT) in mm/sec, forward(+) and reverse(-).

ROTATE|double||
Rotates the robot by given amount in degrees. Positive is counterclockwise. This works similar to the heading where a rotation greater than 180 (or smaller than -180) the shortest method. For example, the robot will turn clockwise 170 degrees if 190 is entered.

ROTVEL|double||
Sets the rotational velocity in degrees/sec. Directions same as **ROTATE**.

HEADING|double||

Will rotate the robot to the desired heading (degrees) in the shortest possible manner. For example if the robot is at 0 degree heading, an input of 350 will turn the robot to the left 10 degrees, so a full turn is not possible with one call to this command.

STOP||
Stops all motion commands. Actions will be blocked until the precedence time is exceeded or CLEAR is called.

CLEAR||
Clears direct motion (move, rotate, etc) commands. For example if **VEL** command is active, **CLEAR** will end the **VEL** command in 2 or 3 seconds. It comes useful when robot's motors are shut off while it attempts to complete a motion command producing continuous beep. Firing **CLEAR** will end it.

MAXTRANSVEL|double||
Sets maximum velocity in mm/sec.

MAXTROTVEL|double||
Sets maximum rotational velocity degrees/sec.

TRANSACCEL|double||
Sets translational acceleration in mm/sec^2.

TRANSDECEL|double||
Sets translational deceleration in mm/sec^2.

ROTACCEL|double||
Sets rotational acceleration in deg/sec^2.

ROTDECEL|double||
Sets rotational acceleration in deg/sec^2.

5.12.1.4 SbActionML

AMOVE|double||

The action move command was created to replace the direct motion command so it can be used in combination with other actions (i.e. forward limiter). Typical direct motion commands disable all actions. The **AMOVE** works like **MOVE**, except for action blocking.

AROTATE|double||

The action rotate command was created to replace the direct motion command so it can be used in combination with other actions (i.e. forward limiter). It works differently than **ROTATE** when a large angle is entered, where it attempts to reach the desired rotation angle by rotating in the direction described by the sign of the input. For example, if 300 is entered the robot will rotate counterclockwise until it has rotated 300 degrees.

STOP||

Disables the above actions.

5.12.1.5 SbCameraSONYML/SbCameraVCC4ML

All the camera control commands begin with **CAM** and follow up with a subcommand as the first argument.

CAM|init||

Powers up and connects to the camera (only **SbCameraVCC4ML**).

CAM|zoom|double||

Zooms in and out from 0 - 100, 0 is no zoom, 100 is the camera's maximum zoom.

CAM|zoomrel|double||

Zooms relative to the current zoom level (+/- 100).

CAM|pan|double||

Pans the camera left and right (+/- 100), positive 100 is maximum clockwise pan position. Negative 100 is maximum counterclockwise pan position.

CAM|panrel|double||

Pans relative to current pan position (+/- 100).

CAM|tilt|double||

Tilts the camera +/- 100, positive 100 is maximum position up, while negative 100 is maximum position down.

CAM|tiltrel|double||

Tilts camera relative to the current camera tilt position (+/- 100).

CAM|stop||

Stops all camera motions (only **SbCameraVCC4ML**).

CAM|panslew|double||

Sets the speed of panning (only **SbCameraVCC4ML**).

CAM|tiltslew|double||

Sets the speed of tilting (only **SbCameraVCC4ML**).

CAM|backlight|int||

Sets automatic backlighting on/off. (only **SbCameraSONYML**).

5.12.1.6 SbExampleML

EXAMPLE||

Example command that returns: **MSG|EXAMPLE|X|Y|TH|VOLTAGE|| .**

5.12.1.7 SbExampleMLT

EXAMPLETHREAD||

An example of how a threaded message listener behaves. It will print out a simple message locally.

5.12.1.8 SbGripperML

GRIPPER|int|| or GRIPPER|string||

Stops, stores or readies the whole gripper system. It can accept a string or an integer: 0 = “halt”, 1 = “store”, 2 = “ready”.

GRIP|int|| or GRIP|string||

Closes and opens the gripper. It can accept a string or an integer:

0 = “stop”, 1 = “open”, 2 = “close”.

LIFT|int|| or LIFT|string||

Lifts the gripper up and down. It can accept a string or an integer:

0 = “stop”, 1 = “up”, 2 = “down”.

GRIPPERSTATE||

Returns information about the gripper:

outer	- object crossed the first beam between paddles (0,1).
inner	- object crossed the second beam between paddles (0,1).
openclose	- gripper is opened or closed or in between (0,1,2).
liftMaxed	- the lift is in one of the limit positions (0,1).
leftPaddle	- pressure sensor in the left paddle triggered (0,1).
rightPaddle	- pressure sensor in the right paddle triggered (0,1).

The format of the returned message:

GRIPPERSTATE|*outer*|*inner*|*openclose*|*liftMaxed*|*leftPaddle*|*rightPaddle*||
0=between, 1=open, 2=closed; 1=yes, 0=no.

5.12.1.9 SbRobotML

CONNECT|string|string|string||

Connects to a robot or the simulator. If the first argument is blank (0) or “sim”, **RCCI** will attempt to connect to default simulator location. If the first argument is “sim” and the rest of arguments are non blank, second argument will be the IP address and third one will be the port of the simulator: *CONNECT*|*sim*|*address*|*port*||

If the first argument is “robot” **RCCI** will attempt to connect to the robot on default serial port.

DISCONNECT||

Disconnects from the robot or simulator.

SETPOS|double|double|double||

Sets the apparent position of the robot in the world. *SETPOS|X|Y|HEADING||*

MOTORS|int||

Turns the robot motors on and off: 0=off 1=on.

SONAR|int||

Turns the sonar off and on: 0=off, 1=on.

ROBOTPARAMS||

Returns robot parameters as doubles:

velmax	- maximum translational velocity.
rotvelmax	- maximum rotational velocity.
accel	- maximum translational acceleration.
decal	- maximum translation deceleration.
rotaccel	- maximum rotational acceleration.
absmaxvel	- absolute maximum velocity.

ROBOTPARAMS|velmax|rotvelmax|accel|decel|rotaccel|absmaxvel||

ROBOTNAME||

Returns robot name. **ROBOTNAME|name||**

ROBOTTYPE||

Returns robot type. **ROBOTTYPE|type||**

5.12.1.10 SbSaphiraML

GOTO|double|double||

Activates gradient path planning. Turns and goes to the goal without avoidance. Accepts X,Y coordinates (positive X direction is straight ahead while positive Y is 90 degrees to the left.

IGOTO|double|double||

Activates gradient path planning. Goes to the given goal and uses sonar data to avoid obstacles. Accepts X, Y coordinates (positive X axis is straight ahead, while positive Y is 90 degrees to the left.

STOP||

Stops the **Saphira** gradient path planning.

GOTOSETDONE|double||

Tolerance on reaching the goal. Set how close we need to be to a goal to slow down or be done in millimeters.

GOTOSETCLOSE|double||

Distance to goal in millimeters where the robot starts slowing down.

GOTOWINDOW|double|double||

Area in which the robot will look for possible paths to goal, X, Y in millimeters.

5.12.1.11 SbSoundsMLT

SPEAK|string||

Will use **Botspeak** to synthesize speech, initializes if it has not done so. Synthesis customization is done via special codes that are placed in the input text (string argument). The following is an example of *Botspeak* (*ViaVoice*) annotations:

`vbN	Pitch, N in range 0 to 100.
`vhN	Head size, N in range 0 (tiny head) to 100 (huge head).
`vrN	Roughness, N in range 0 (smooth) to 100 (rough).
`vyN	Breathiness, N in range 0 (not breathy) to 100 (breathy whisper).
`vfN	Pitch fluctuation, N in range 0 (monotone) to 100 (wide fluctuation).
`vsN	Speed, N in range 0 (slow) to 250 (Fast).
`vvN	Volume, N in range 0 (soft) to 100 (loud).
`vg0	Set voice to male.
`vg1	Set voice to female.
`pN	Create a pause n milliseconds long.
`00	Reduced emphasis.
`0	No emphasis.
`1	Normal emphasis.
`2	Added emphasis.
`3	Heavy emphasis.
`4	Very heavy emphasis.

Example of speak message:

SPEAK|`vg0 When are you going to be `3 finished with this thesis Marcin?||

PLAY|string||

Will use **Botspeak** to play a WAV file. Initializes if it has not done so. The string is the location and name of the file to be played, relative to the directory where the **RCCI** executable file is located. Absolute paths are also possible.

Example: PLAY|/home/marcin/music/song.wav||

AMIGOSOUND|int||

Plays a given sound number on AmigoBot.

AMIGOSOUNDTOG|int||

Turns the AmigoBot sounds on or off. 0=off, 1=on.

5.12.1.12 SbStateML

The STATE command has a number of subcommands under it.

STATE|string|int||

STATE||

If the message has no arguments or first one is 0, broadcast vital state information:

X	- position, positive X is parallel to forward motion.
Y	- position, positive Y is 90 degrees to the right of X axis.
Heading	- the direction the robot is facing -180 to 180 in degrees.
Vel	- translational velocity of the robot in mm/sec.
RotVel	- rotational velocity of the robot in degrees/sec.
LeftWheelVel	- velocity of the left wheel in mm/sec.
RightWheelVel	- velocity of the right wheel in mm/sec.
Volt	- voltage level of the robot's batteries 9.0-13.0 volts.

The message broadcast is in the following format:

MSG|STATE|X|Y|Heading|Vel|RotVel|LeftWheelVel|RightWheelVel|Volt||

STATE|sonar|int||

Returns values of a set of sonar in millimeters and will be -1 if no data was received from the sonar (either no sonar or sonar did not return a reading. If the first argument is **SONAR** and second is "0", sonar values for all available sonar (up to 32) will be returned. If the second argument is 1, 2, 3 or 4 one of the following sets of sonar will be returned.

0: all, 1: 0-7, 2: 8-15, 3: 16-23, 4: 24-31.

MSG|SONAR|double|double|double|double|double|…|double||

STATE|closestsonar|int|int||
Will return the number of the sonar that has the closest current reading in the given range. Requires start and end angles.

MSG|CLOSESTSONAR|sonarnumber|distance||

STATE|sonaron||
Returns whether the **SONAR** are active or not. 0=off, 1=on.

MSG|SONARON|int||

STATE|tablesensors||
Checks if the table sensors are triggered. Returns LEFT|RIGHT with 0=off 1=on.

MSG|TABLESENSORS|left|right||

STATE|motors||
Send whether the motors are activated or not. 0=off, 1=on.

MSG|MOTORS|int||

5.12.2 Automatic Response Messages

The following messages are automatically generated and sent out to the connected clients when events occur.

5.12.2.1 SbRobotML

MSG|CONNECTED||

This message is sent after a successful connection to either robot or simulator.

MSG|DISCONNECTED||

This message is sent when a connection is lost between robot or simulator and **RCCI**. Disconnection can be user initialized or unexpected.

MSG|CONNECTION FAILED - unknown connection method||
Warning when an unknown connection type is specified via the **CONNECT** command. (i.e. **CONNECT|rbt||**)

MSG|CONNECTION FAILED - serial comm problem||
Refers to an instance of a connecting problem related to opening a serial port for connecting with the robot.

MSG|CONNECTION FAILED - simulator not available||
Refers to an instance of connecting problem related to opening a socket for TCP robot simulator.

MSG|CONNECTION FAILED - trying again||
If the first attempt to connect to the simulator or robot fails, this message will be broadcast.

MSG|STALLED|int|int||
Will send which motor is currently stalled (left/right.)

MSG|BUMPERS|0|0|0|1|0|1|0|...||
This message is sent when one or more bumpers are activated. Checks if back or the front bumpers have been activated. If they are pressed, a message is generated displaying all the bumper states where 0 is default state and 1 stands for active.

MSG|BYE||
Message sent to all clients before they are disconnected.

MSG|NO ARIA - discard connection attempt||
This message will be broadcast if **ARIA** is not initialized.

5.12.2.2 SbServerML

MSG|WELCOME TO StudioBlue RCCI SERVER||

Upon successful connection the above message is sent to the client.

MSG|SHUTTING DOWN TCP SERVER||

This message is broadcast when the TCP server is being shutdown.

MSG|MALFORMED MESSAGE||

Parser failed to create a proper **SbMessage**. (i.e. |||)

5.12.2.3 SbMsgHandlerSingleton

MSG|UNKOWN MESSAGE TYPE -> ::TYPE::||

No listeners registered to fire the particular message type. The message will be discarded.

5.12.2.4 SbManagerT

MSG|SHUTTING DOWN RCCI!||

Broadcast before **RCCI** shutdown.

5.12.2.5 SbGripperML

MSG|NOGRIPPER||

Sent when the gripper commands are used on a robot without a gripper.

5.13 Installation and Operation

Installing and updating the software requires familiarity with *UNIX* programming practices (*makefile, gcc*) and knowledge of the operating system. Utilization of **RCCI** is fairly straightforward.

5.13.1 Installation

The compiled program should work on any *Intel* based *Linux* machine with **ARIA**, **Saphira**, and if speech is desired, *ViaVoice* and **Botspeak** installed. If recompilation is required *GNU GCC* compiler version 2.95.3 is necessary. It is imperative that all the paths are correct and that the machine is setup with Internet access. For detailed directions consult the appendix.

5.13.2 Operation

Initially, software testing was done using a common *Telnet* client such as the one provided by the latest versions of *Microsoft Windows* or any *UNIX* machines. Before any testing can take place, the **RCCI** server needs to be activated. This is rather complicated due to the fact that *ViaVoice* engine requires *X Windows* and attempts to bring itself up on the client side (user side). Since *MS Windows* machines do not have *X Windows* a *Linux* machine is used to connect to the robot and start up **RCCI**. Before that is possible the client machine needs to allow all clients or just the robot's IP to open *X* application. This is done by allowing all clients to connect to the user's machine by typing in the terminal:

```
xhost +
```

or just one by specifying particular clients DNS name:

```
xhost robo4.me.cooper.edu
```

Following, *Telnet* to the robot and log in as a guest (default: enter guest when prompted for login and nothing for password):

```
telnet robo4.me.cooper.edu
```

In the directory where the latest version of **RCCI** is located invoke the start script with the desired PORT:

```
./start 5555
```

RCCI will run until it is shutdown cleanly, runs into an error, or is killed by the user.

PORT represents a socket port that clients are allowed to use to connect to the **RCCI** server. Port should be a number from *5001* to *49151*, since ports *1024* to *5000* are typically reserved for operating system's client programs. It is suggested that a port number that does not contain patterns or have a known meaning is chosen to avoid a conflict because it is used by others for its ease of memorization. Pots *1* to *1023* should not be used since these ports are reserved for use by the Internet

Assigned Numbers Authority (IANA). Avoid using ports *49152* through *65535* which are dynamic ports that operating systems use randomly [51,35].

Any number (system resources permitting) of clients can connect to the **RCCI** server using a *Telnet* client or their own software with a socket connection. They have to connect to the robot or computer that is running **RCCI** on the *PORT* specified when **RCCI** was started up. If *Telnet* is used the following command can be used from the command prompt in *Linux* or *Windows*:

Besides using SHUTDOWN|| command, there are two other methods to stop **RCCI** on the robot.

```
telnet robo4.me.cooper.edu 5555
```

Typical method of stopping running applications on *Linux* is using the following combination of keys:

```
Control-C
```

Occasionally this does not work a more severe method is applied using the killall command:

```
killall -9 rcci
```

As the software became more complex with many commands to test, a graphical user interface client was created using Java programming language [19]; it is discussed in detail later.

5.14 Java GUI Test Application

5.14.1 Sample Java Client

Towards the end of the development process, a more rapid and flexible application testing method was desired. A graphical user interface client was created in Java to communicate via user defined message. It simply allows the user to connect and disconnect and send control commands to **RCCI.** When send button is pressed four of the arguments from the input boxes are structured into a Message and sent to the robot. Robot replies are displayed in the communication box.

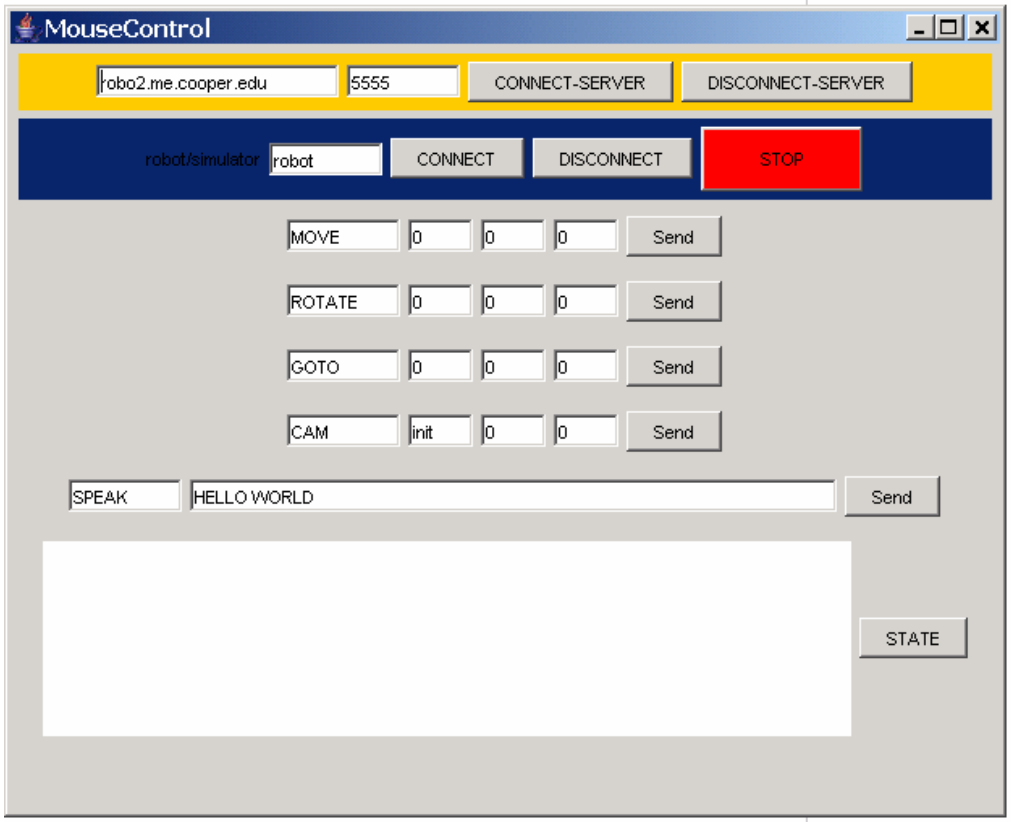


Figure 5.12 Flexible Test Application Screenshot.

5.14.2 Mouse Based Navigation Interface

Mouse seems to be a good control device for video games why not robots? What if the mouse motion could be mapped to robot's coordinate system with adjustable scaling: moving the mouse would move the robot in the same pattern. Moving the robot using "go to" statements is sufficient for reproducing large simple motions; however, it is very limiting to straight and rotation motions. A direct motion translated directly from user's interaction with the mouse yields puppet like robot control that can be useful in queuing the robot, testing maneuvers and instant direct gratification. There of course would be a delay but that may be acceptable with practice. As for the interface a left mouse button could initiate forward motion while the wheel could adjust top speed (control key for reverse). Right button could control camera movement while the middle button could be linked to the stop command. The Caps Lock could engage and disengage robot motion control via the mouse. This application is in early development stages and hopefully will be included in **GRASP** in the future.

5.14.3 Suggested Navigation Interface

The following figure represents the possible application that could be useful addition to the lab. It can use services provided by **RCCI** with additional video technology. The application is similar to one used by some recent I-Robot CoWorker, could be cool. [53]

Figure 5.13 Navigation Application Suggestion (Mock up).

The navigation interface work with the video feed from the robots camera. By calculating the angles (pan, tilt, zoom) of the camera and accounting for the camera mount height one can project the approximate distance and direction of desired travel after the user clicks on the image. Physics model of the optical system could decrease the optical error created by the curvature of the lens.

6 CONCLUSIONS

Throughout the development of the theater, Marcin Balicki and James Cruickshanks have closely worked with artists to improve **StudioBlue's** functionality and to produce exciting works of art. Adrianne Wortzel and Huy Truong were the first artists in residence to produce video art pieces using the facility and the software created by Marcin Balicki and James Cruickshanks. Although both creations were film productions, they sufficiently tested all the technologies involved, exposing unforeseen problems and potential improvements.

The tele-robotic aspect of the lab has been slow to materialize; however, with all the underlying technologies in place, the team is hoping to have a real time interactive online installation by the end of May, 2004. "Eliza Redux" by Adrianne Wortzel, is an online interactive psychoanalysis session with a Flash based interface that incorporates Eliza-type (pseudo intelligent) conversation engine [54] and audio/video feedback. The office space is virtual and the roll of the psychotherapist is performed by one of the PeopleBots. This work is a collaboration with graduate students from Parsons School of Design.

6.1 StudioBlue

The creation of **StudioBlue** has fulfilled the objective of the **NSF** grant by creating a space where a variety of disciplines unite to produce innovative technologies and demonstrate artistic expression. The theater is outfitted with autonomous and semi-autonomous research robots, supporting computers, audio/video recording equipment, and theatrical stage with a turntable floor, ample lighting and bluescreen technology. In addition, the laboratory posses a Turnkey Video Editing System making **StudioBlue** an independent production and post production facility.

The theater has received much publicity, even appearing on *Martha Stewart Living*'s special on The Cooper Union. There is obvious public interest in this type of research and many are eager to see continued results.

6.1.1 Suggested Improvements

In terms of the lab layout, current space is inadequate for more than three performers on the stage at one time. Furthermore, the robots navigation accuracy is very low, and can exceed a foot difference from the commanded distance; the error is between 5-10 percent of the stage size. A larger stage area would give more flexibility to the directors, diminish the obvious motion errors and allow the actors to compensate for robot's inaccuracy. A larger studio would also improve the video recording methods by allowing for more camera placements and would create more room for off stage actors, support crew, and audience. The lab requires constant supervision during operation because the lack of fencing on the turntable stage permits the robots to "jump off stage". A mechanized "lazy Suzan" stage that is flush with the rest of the floor would be an optimal solution. Due to the lab's proximity to 9th street (in the middle of New York City), the outside, street level noise has often impeded production. The performance space needs to be acoustically insulated and preferably be relocated to a higher floor. The audio system consisted solely of a pair of computer speakers needs to be upgraded to a more professional, surround sound system. The lighting system is comprehensive; however a few sets of colored gels for Fresnel spot lights and large reflectors would only improve the lighting atmosphere. More fixture mounting points, on the walls would also prove useful.

One way to improve some of the navigation issues is to implement a tracking system that updates all robot positions and orientations on the stage. A number of different localization technologies are available (i.e. image processing) and are highly recommended to improve robot control and position accuracy.

Even though the theater was designed with Chromakey operation in mind, the system is very finicky and requires practice. **StudioBlue** only works with flat backgrounds and motionless camera shots. Professional Chromakey systems are capable of not only replacing flat backgrounds with still images or video but also solid primitives (cube, rectangle, etc.). Such a system would add another exciting visual dimension to the lab.

More development would also help on the Internet audio/video and tele-operation elements of the laboratory. Due to strict Internet network safety precautions at the Cooper Union serving websites and sending out high bandwidth audio/video streams may be difficult. At some point outsourcing becomes a possibility; or to bypass the firewall issues, a proxy server could be instituted. Upgrading the wireless network technologies to the newest standards would improve security and reliability.

A collection of costumes and props is part of any theater group and should be a part of **StudioBlue**. Besides the classical props, accessories such as LED lights, small LCD or touch screen displays, robotic arms, and remote controlled gizmos would benefit the theater as well. This requires a space to store materials and tools for creating costumes and robot accessories

The biggest asset of **StudioBlue** is the personnel, and increasing the size of the technical support, artists in residents and able students will greatly improve the functionality of the lab as well as quality of academic research.

6.2 RCCI

The Remote Control and Communication Interface has been successfully used by James Cruickshanks in his **GRASP** application that was instituted by Huy Truong during his residency at the **StudioBlue**. Truong translated his script into **GRASP** and used it exclusively to command *Woody* in a play involving two human actors. Prior to the creation of **GRASP**, a simple *Telnet* client was used in the lab's first major production (by Adrianne Wortzel) to send commands to **RCCI**. It was found that this method was rather cumbersome and slow. However, both of these instances proved valuable as real life applications of **RCCI**. The software went through a number of iterations, such as the addition of multiple client connections, and internal feedback system. The latest stable release is version 1.0.

Through trial and error, relying on precision of the robot location feedback system is not possible. Besides the typical dead reckoning errors involved in robotics, changes in temperature and wheel diameter associated with two inflatable rubber tires amplifies this error. The tires were also

inflated to an optimal pressure to reduce the noise produced by the interaction of rubber and floor surface. The change in the size of the tires is not accounted for in the control code which results in incorrect rotation or distance traveled calculations. Familiarity with the robot behavior through rehearsals greatly improves prediction of robots response to motion commands.

Due to unforeseen developments that caused James Cole's software to become obsolete, maintainability was a significant design objective for **RCCI**. Modular architecture was implemented to accept new **ARIA** updates. Additionally, the software model is designed so it can be implemented with other robot application development interfaces since **ARIA** utilities used in **RCCI** are open source. Robustness was another major factor in designing **RCCI** which operates days at a time without crashing that is caused by memory leaks or thread locks. There exists an external factor (shorts/low voltage) that may influence unpredicted robot behavior, as a result the robot should always be watched or have the emergency break switch active when it is not performing.

Typically software response using **RCCI** will never be as fast as the software utilizing the robot API running on the same machine; unless the controlling application requires great processing resources in which case it is favorable to keep the control application on a remote processor, i.e. desktop machine. Even though the robot may be upgraded to a high performance processor, the power consumption directly increases with processor speed. This is evident in the difference between the charge life of two PeopleBots. Performance PeopleBot has more peripherals and operates for 6 hours (75% that of Kiru's operating time).

Communication frequency (number of datagrams sent to **RCCI**) can effect the performance of the software. Since some of the commands retrieve robot information they use up system resources and can tax the processor. If many messages are sent to **RCCI**, the robot will slow down and attempt to fulfill all requests, although eventually some datagrams will be dropped off when the buffer fills up. However, it is very difficult to overrun the server. This was not fully tested because messaging at 10Hz was more than sufficient at the moment.

Minimal usage of the *X windows System* is suggested because it produces heavy network traffic and causes interference with some of the **ARIA** functions in the form of lost communication packets from the controller. This may be due to electromagnetic interference from the network adapter.

There is also an occasional issue with "Buffer Overrun" error believed to be linked to *ViaVoice*/**Botspeak**. This does not directly affect the performance of **RCCI**.

The main **RCCI** usage issue is the inaccuracy of positioning which was mentioned earlier. **RCCI** has implemented a function for updating the robot's current position and orientation which will hopefully be used in the future.

6.2.1 Suggested Improvements

RCCI requires a user to login on to the robot computer and initiate the application which could be considered a superfluous step; an automatic start on boot up as a daemon may be desirable. However, this is currently problematic due to an odd dependence on *X Windows* system by *ViaVoice* engine, which exports display to the current user's desktop. **RCCI** consistently reports its activities through print statements; it may be beneficial to include an option that would report to a log file. Although multithreading is very useful, it taxes operating system resources especially when using locks and unlocks. There may exist more efficient way of sharing data between threads. Parameter file for command specification may be useful to easily customize the messaging syntax removing the need for altering source code and recompiling which may not be possible on all systems. In the future the may be a need to synchronize time among communicating parties which would require some type of request for system time command which responds to all the clients with snapshot of the OS time. Furthermore, each **SbMessage** has a timestamp, which may be useful if it is included in the **TCP/IP** message structure.

As the processing power increases, artificial intelligence will become an integral part of theatrics by incorporating intelligent positioning systems, queuing mechanisms and intelligent

human-robot actor interaction. Furthermore, speech synthesis should be expanded to included accents and other languages. Voice recognition could be integrated for smoother interaction between real actors and the robots.

6.3 Remarks

Working on **StudioBlue** and other robotics projects at Cooper Union has been most enjoyable, satisfying, and unforgettable learning experience: from learning business practices, institutional politics to writing skills and research techniques in unknown disciplines. Meeting amazingly creative people and establishing life long friendships added to the experience. Besides a comprehensive engineering education, many valuable skills were attained. The topics varied from *UNIX* software development, to lighting design, to managing large projects.

Often the actors become skeptical about the robots place in theater. Whether the robots will become the stars or just another tool to help directors express their ideas, they will never replace humans. Nonetheless, it is hoped that the theater will serve the Cooper Union community as and interdisciplinary crossroad for art, architecture and engineering students as well as the general public.

7 Bibliography

1. Faver , Cheryl "Toward a Digital Stage Architecture: A Long-Term Research Agenda in Digitally Enabled Theater." *IEEE Multimedia October-December 2001* (Vol. 8, No. 4) 2001.

2. Weiman, C.F.R., Wortzel, A. Wei,S. "*Robotic Renaissance: Bridging Engineering, Art, and Science via Web Robotics , **NSF** Award Abstract - #9980873*" January, 2000.

3. "*The Robotic Renaissance Project*." Gateway Coalition, Cooper Union.
   < http://doc1.cooper.edu:8080/gateway/robotics/site/> (April 5, 2004).

4. Wortzel, Adrianne "Endeavors in Robotics and Theater"
   < http://www.artnetweb.com/wortzel/robottheater > (February 23, 2004.)

5. ActivMedia Robotics. 2003
   < http://robots.activmedia.com > (March 2004).

6. Cruickshanks, James "*Graphical Robot Action Scripting Platform (GRASP) for ActivMedia Robots.*" Master's Thesis – Cooper Union. May 2004.

7. "ActivMedia PeopleBot Web Site" Activmedia Robotics.
   < http://www.activrobots.com/ROBOTS/peoplebot.html > (January 5, 2001.)

8. *Pioneer2/PeopleBot Operation Manual v 1.1* Activmedia Robotics.
   < http://robots.activmedia.com/docs/all_docs/p2opman11.pdf > (February 5, 2004.)

9. iRobot Incorporated.
   < http://www.irobot.com > (March 20, 2004.)

10. Wortzel, Adrianne "Camouflage Town"
    < http://www.camouflagetown.tv > (February 22, 2004.)

11. Cole, James, "A Web-Enabled Communication Platform For The ActivMedia PeopleBot". Master's Thesis – Cooper Union. October 2002.

12. Balicki, Marcin. "*Computer, Multimedia Systems, and Communication Infrastructure for Implementation of Camouflage Town Exhibit at The Whitney Museum of American Art.*" Senior Design Project, Spring 2001.

13. Wortzel, Adrianne "*An Interactive Tele-robotic Installation*" End of The year Show – The Cooper Union. 2002.
    < http://www.artnetweb.com/wortzel/robottheater/endofyear.html > (April 2, 2004.)

14. Arsham, Daniel. "*COLORAID."* The Cooper Union. Fall of 2000,
    < http://doc1.cooper.edu:8080/gateway/robotics/site/ashram.html > (April 5, 2004.)

15. The Forrest Wade Rapid Prototyping Laboratory. Cooper Union
    < http://www.cooper.edu/engineering/me/facilities/proto.html > (April 6, 2004.)

16. Gerkey, B.P. Vaughan, R. T. Howard, A. "Player User Manual Version 1.4rc2. December 6, 2003."
< http://playerstage.sourceforge.net/doc/Player-manual-1.4.pdf > (April 2, 2004)

17. **Saphira** *Operations and Programming Manual Version 6.2*, ActivMedia Robotics. ,August 1999.

18. *ARIA Reference Manual.* Activmedia Robotics.
< http://robots.activmedia.com/ARIA/Aria-Reference.pdf > (February 5 , 2004.)

19. Java Technology. Sun Microsystems Inc.
< http://java.sun.com/ > (April 6, 2004.)

20. "**Saphira** Web Site" Activmedia Robotics.
< http://www.ai.sri.com/~konolige/**Saphira**/index.html > (January 6, 2001.)

21. **Saphira** *8.1 Manual.* Activmedia Robotics.
< http://robots.activmedia.com/**Saphira**/**Saphira**-Reference.pdf > (February 5, 2004.)

22. Kessler, Gary. "*An Overview of TCP/IP Protocols and the Internet*".
< http://www.garykessler.net/library/tcpip.html >. September 15, 2003.

23. "*Ullanta Performance Robotic*"
< http://www.*Ullanta*.com/*Ullanta*/ > (January 4, 2003.)

24. Garvey, Frank. "*OmniCircus: Junkyard Cabaret and Robot Ensemble.*"
< http://www.*OmniCircus*.com > (April 1, 2004.)

25. Garvey, Frank. The Center for Robotic and Synthetic Performance - The Robotics Institute, Carnegie Mellon University. March 31, 2004.
< http://www.ri.cmu.edu/centers/crp/ > (April, 2004)

26. "*Synthetic Characters Group.* " MIT
< http://characters.media.mit.edu/> (April 1, 2004.)

27. "*The Virtual Theater project.*" May 2001.
< http://ksl-web.stanford.edu/projects/cait > (March 23, 2004.)

28. *PTZ Robotic Cameras for Pioneer 2.* Activmedia Robotics.
< http://robots.activmedia.com/docs/all_docs/PTZcameras3.pdf> (February 5, 2004.)

29. *Pioneer 2 Gripper Manual.* Activmedia Robotics.
< http://robots.activmedia.com/docs/all_docs/gripmanP2_3.pdf > (February 5, 2004.)

30. AmigoBot. ActivMedia Robotics. 2003.
< http://www.amigobot.com/ > (April 6, 2004.)

31. The Acoustics Laboratory. The Cooper Union.
< http://www.cooper.edu/engineering/me/facilities/acustics.html > (April 6, 2004.)

32. Wortzel, Adrianne. "Eliza Redux: Veils Of Transference." **StudioBlue**, 2003.

33. Porvin, Dan "*Lighting Tips For Chromakey - Video or Photography*", Cooper Union School of Art Video Dept. 2002

34. Kim, Yoongeu Yusuf, Mohammed. "*Design and Construction of the Tele-Robotic Theatre.*" Capstone Senior Design Project. May 8, 2003

35. Goerzen, John. *Linux Programming Bible.* Foster City, CA: IDG Books Worldwire Inc., 2000.

36. Stroustrup, Bjarne. *The C++ Programming Language, Special Edition.* Reading, MA: Addison-Wesley, 2000.

37. Wiener, Richard. Pinson, Lewis J. *Fundamentals of OOP and Data Structures in Java.* New York, NY. Cambridge University Press, 2000.

38. "The Player/Stage Project"
< http://playerstage.sourceforge.net > (December 22, 2002.)

39. " *MissionLab v6.0 - multiagent robotics mission specification and control software*"
< http://www.cc.gatech.edu/aimosaic/robot-lab/research/MissionLab > February, 2004.

40. *User Manual for MissionLab version 6.0.* Georgia Tech Mobile Robot Laboratory. April 6, 2003
< http://www.cc.gatech.edu/aimosaic/robot-lab/research/MissionLab/mlab_manual-6.0.pdf > (April 2, 2004)

41. "*The GNU General Public License.*"
< www.gnu.org/licenses/licenses.html > (March 23, 2004.)

42. " ActivMedia Robotics Interface for Applications. " ActivMedia Robotics. 2002
< http://robots.activmedia.com/ARIA/ > (April 6, 2004.)

43. "*K AFT Radio- Robot Theater*"
< http://www.aftimes.com/actionradio/nav/robot.radio/ > February, 2004.

44. *IBM ViaVoice Outloud API Reference Version 5.0 – Beta* IBM, 1999.

45. Carasik, Anne H. *LINUX System Administration.* Foster City, CA: M&T Books, 1999.

46. "GCC home page. " March 28, 2004.
< http://www.gnu.org/software/gcc/gcc.html > (April 6, 2004.)

47. "Vim Online" The VIM text editor.
< http://www.vim.org/ > (April 6, 2004.)

48. " KDE Home Page." K Desktop Environment.
< http://www.kde.org/ > (April 6, 2004.)

49. *Doxygen Documentation*
< www.doxygen.org > (March 22, 2004.)

50. Kalev, Danny. "*Implementing the Singleton Design Pattern*"
< http://gethelp.devx.com/techtips/cpp_pro/10min/10min0200.asp > (April 2, 2004.)

51. "Link for ModelSim: Choosing **TCP/IP** Socket Ports "The MathWorks, 2004.
<http://www.mathworks.com/access/helpdesk/help/toolbox/modelsim/a1057689278b4.html >
(April 7, 2004.)

52. Arciniegas, Fabio *C++ XML.* Indianapolis, IN: New Riders, 2002.

53. "CoWorker Robot." I-Robot
< *http://www.irobot.com/industrial/coworker.asp* > (March 24, 2004.)

54. Tangorra, Joanne. "The personal Computer as Therapist." Digital Deli.
< http://www.atariarchives.org/deli/therapist.php > (April 6, 2004.)

8 APPENDIX

8.1 Contents of Included CD

- This document
- ActivMedia software and documentation
- RCCI Source Code - Raw, tar and zip files
- Start up script
- Makefile
- User bash scripts
- Code Documentation in HTML
- Test Application
- Videos, Images
- Relevant Student Reports
- James Cruickshanks' Thesis
- GRASP
- Thesis Defense Presentation Slides
- Command List

8.2 Command List

USER COMMANDS	
SHUTDOWN||	Disconnects all clients and shutdown RCCI and ARIA.
QUIT||	Disconnects the client
QUITALL||	Disconnects all clients
SEND|**arg1**|**arg2**|**...**||	Will send **arg1**|**arg2**|**…**| to all connected clients.
MOVE|**double**||	Move + - 32767 mm
VEL|**double**||	Speed + - mm/sec
LRVEL|**double**|**double**||	Constant velocity of each wheel (LEFT|RIGHT) in mm/sec
ROTATE|**double**||	Rotate + - 180 Degrees (Positive is counterclockwise)
ROTVEL|**double**||	rotational velocity in degrees/sec
HEADING|**double**||	Rotate to a heading + - 180.
STOP||	Stops Robot motions
CLEAR||	Clears direct motion (move, rotate, etc) commands
MAXTRANSVEL|**double**||	Sets maximum velocity in mm/sec.
MAXTROTVEL|**double**||	Sets maximum rotational velocity degrees/sec.
TRANSACCEL|**double**||	Sets translational acceleration in mm/sec^2.
TRANSDECEL|**double**||	Sets translational deceleration in mm/sec^2.
ROTACCEL|**double**||	Sets rotational acceleration in deg/sec^2.
ROTDECEL|**double**||	Sets rotational acceleration in deg/sec^2.
AMOVE|**double**||	Move + - 32767 mm
AROTATE|**double**||	+ - in Degrees (Positive is counterclockwise)
STOP||	Disables the above actions.
CAM|**init**||	Powers up and connects to the camera (only **SbCameraVCC4ML**).
CAM|**zoom**|**double**||	0 is no zoom, 100 is the camera's maximum zoom.
CAM|**zoomrel**|**double**||	Zooms relative to the current zoom level (+/- 100).
CAM|**pan**|**double**||	Pans the camera left and right (+/- 100), positive 100 is maximum clockwise pan position. Negative 100 is maximum counterclockwise pan position.
CAM|**panrel**|**double**||	Pans relative to current pan position (+/- 100).
CAM|**tilt**|**double**||	Tilts the camera +/- 100, positive 100 is maximum position up, while negative 100 is maximum position down.
CAM|**tiltrel**|**double**||	Tilts camera relative to the current camera tilt position (+/- 100).
CAM|**stop**||	Stops all camera motions (only **SbCameraVCC4ML**).
CAM|**panslew**|**double**||	Sets the speed of panning (only **SbCameraVCC4ML**).
CAM|**tiltslew**|**double**||	Sets the speed of tilting (only **SbCameraVCC4ML**).
CAM|**backlight**|**int**||	Sets automatic backlighting on/off. (only **SbCameraSONYML**).
EXAMPLE||	Example command that returns: **MSG**|**EXAMPLE**|**X**|**Y**|**TH**|**VOLTAGE**|| **.**
EXAMPLETHREAD||	An example of how a threaded message listener behaves. It will print out a simple message locally.
GRIPPER|**int**|| **or** **GRIPPER**|**string**||	Stops, stores or readies the whole gripper system. It can accept a string or an integer: 0 = "halt", 1 = "store", 2 = "ready".
GRIP|**int**|| **or GRIP**|**string**||	Closes and opens the gripper. It can accept a string or an integer: 0 = "stop", 1 = "open", 2 = "close".
LIFT|**int**|| **or LIFT**|**string**||	Lifts the gripper up and down. It can accept a string or an integer: 0 = "stop", 1 = "up", 2 = "down".
GRIPPERSTATE||	Returns the state of the gripper in the following format: *GRIPPERSTATE*|*outer*|*inner*|*openclose*|*liftMaxed*|*leftPaddle*|*rightPaddle*||
	outer - object crossed the first beam between paddles (0,1).

	inner - object crossed the second beam between paddles (0,1). openclose - gripper is opened or closed or in between (0,1,2). liftMaxed - the lift is in one of the limit positions (0,1). leftPaddle - pressure sensor in the left paddle triggered (0,1). rightPaddle - pressure sensor in the right paddle triggered (0,1).
CONNECT||	Connects to the Simulator on the pc where the RCCI is running. (default port)
CONNECT|**sim**||	Connects to the Simulator on the pc where the RCCI is running. (default port)
CONNECT|**sim**|**address**|**port**||	Connects to the Simulator at the given address and port.
CONNECT|**robot**||	Connects to the robot on default serial port.
DISCONNECT||	Disconnects from the robot or simulator.
SETPOS|**double**|**double**|**double**||	Sets the apparent position of the robot in the world. *SETPOS*|*X*|*Y*|*HEADING*||
MOTORS|**int**||	Turns the robot motors on and off: 0=off 1=on.
SONAR|**int**||	Turns the sonar off and on: 0=off, 1=on.
ROBOTPARAMS||	*ROBOTPARAMS*|*velmax*|*rotvelmax*|*accel*|*decel*|*rotaccel*|*absmaxvel*||
	velmax - maximum translational velocity. rotvelmax - maximum rotational velocity. accel - maximum translational acceleration. decal - maximum translation deceleration. rotaccel - maximum rotational acceleration. absmaxvel - absolute maximum velocity.
ROBOTNAME||	Returns robot name. **ROBOTNAME**|**name**||
ROBOTTYPE||	Returns robot type. **ROBOTTYPE**|**type**||
GOTO|**double**|**double**||	Turns and goes to the goal without avoidance. Accepts X,Y coordinates (positive X direction is straight ahead while positive Y is 90 degrees to the left.)
IGOTO|**double**|**double**||	Same as GOTO but with obstacle avoidance using sonar data and gradient path planning from Saphira
STOP||	Stops the **Saphira** gradient path planning.
GOTOSETDONE|**double**||	Tolerance on reaching the goal. (mm)
GOTOSETCLOSE|**double**||	Distance to goal in millimeters where the robot starts slowing down. (mm)
GOTOWINDOW|**double**|**double**| ||	Area in which the robot will look for possible paths to goal, X, Y in mm.
SPEAK|**string**||	Synthesis customization is done via special codes that are placed in the input text (string argument).
	`vbN - Pitch, N in range 0 to 100. `vhN - Head size, N in range 0 (tiny head) to 100 (huge head). `vrN - Roughness, N in range 0 (smooth) to 100 (rough). `vyN - Breathiness, N in range 0 (not breathy) to 100 (breathy whisper). `vfN - Pitch fluctuation, N in range 0 (monotone) to 100 (wide fluctuation). `vsN - Speed, N in range 0 (slow) to 250 (Fast). `vvN - Volume, N in range 0 (soft) to 100 (loud). `vg0 - Set voice to male. `vg1 - Set voice to female. `pN - Create a pause n milliseconds long. `00 - Reduced emphasis. `0 - No emphasis. `1 - Normal emphasis.

	`2 - Added emphasis. `3 - Heavy emphasis. `4 - Very heavy emphasis.
PLAY|string||	The string is the location and name of the file to be played, relative to the directory where the **RCCI** executable file is located.
AMIGOSOUND|int||	Plays a given sound number on AmigoBot.
AMIGOSOUNDTOG|int||	Turns the AmigoBot sounds on or off. 0=off, 1=on.
STATE||	***MSG|STATE|X|Y|Heading|Vel|RotVel|LeftWheelVel|RightWheelVel|Volt||***
	X - position, positive X is parallel to forward motion. Y - position, positive Y is 90 degrees to the right of X axis. Heading - the direction the robot is facing -180 to 180 in degrees. Vel - translational velocity of the robot in mm/sec. RotVel - rotational velocity of the robot in degrees/sec. LeftWheelVel - velocity of the left wheel in mm/sec. RightWheelVel - velocity of the right wheel in mm/sec. Volt - voltage level of the robot's batteries 9.0-13.0 volts.
STATE|sonar|int||	Returns values of sonar set in mm (-1 no sonar, or max). 0: All sonar, 1: 0-7, 2: 8-15, 3: 16-23, 4: 24-31.
STATE|closestsonar|int|int||	Will return the number of the sonar that has the closest current reading in the given range. Requires start and end angles.
STATE|sonaron||	Returns whether the **SONAR** are active or not. 0=off, 1=on.
STATE|tablesensors||	Checks if the table sensors are triggered. Returns LEFT|RIGHT with 0=off 1=on.
STATE|motors||	Send whether the motors are activated or not. 0=off, 1=on.

AUTOMATIC RESPONSE MESSAGES	
MSG|CONNECTED||	This message is sent after a successful connection to either robot or simulator.
MSG|DISCONNECTED||	This message is sent when a robot/sim and RCCI connection is lost.
MSG|CONNECTION FAILED - unknown connection method||	Warning when an unknown connection type is specified via the **CONNECT**.
MSG|CONNECTION FAILED - serial comm problem||	Refers to an instance of a connecting problem related to opening a serial port for connecting with the robot.
MSG|CONNECTION FAILED - simulator not available||	Refers to an instance of connecting problem related to opening a socket for TCP robot simulator.
MSG|CONNECTION FAILED - trying again||	If the first attempt to connect to the simulator or robot fails, this message will be broadcast.
MSG|STALLED|int|int||	Will send which motor is currently stalled (left/right.)
MSG|BUMPERS|0|0|0|1|0|1|0|...||	This message is sent when one or more bumpers is activated. If they are pressed, a message is generated displaying all the bumper states where 0 is default state and 1 stands for active.
MSG|WELCOME TO StudioBlue RCCI SERVER||	Upon successful connection the above message is sent to the client.
MSG|BYE||	Message sent to all clients before they are disconnected
MSG|NO ARIA - discard connection attempt||	This message will be broadcast if **ARIA** is not initialized.
MSG|SHUTTING DOWN TCP SERVER||	This message is broadcast when the TCP server is being shutdown.

MSG|MALFORMED MESSAGE||	Parser failed to create a proper **SbMessage**. (i.e. |||)
MSG|UNKOWN MESSAGE TYPE -> ::TYPE::||	No listeners registered to fire the particular message type. The message will be discarded.
MSG|SHUTTING DOWN RCCI!||	Broadcast before **RCCI** shutdown.
MSG|NOGRIPPER||	Sent when the gripper commands are used on a robot without a gripper.

8.3 NSF Abstract

NSF Award Abstract - #9980873
FLAWS008

Robotic Renaissance: Bridging Engineering, Art, and Science via Web Robotics

Investigators:
Carl F. R. Weiman Weiman@pcnet.com (Principal Investigator current)
Chih-Shing Wei (Co-Principal Investigator current)
Adrianne Wortzel (Co-Principal Investigator current)
Sponsor
The Cooper Union for the Advancement of Science and Art
41 Cooper Square
New York, NY 100037136 212/254-6300
NSF Program
7428 CCLI-ADAPTATION AND IMPLEMENTA
Field Application
0000099 Other Applications NEC

Abstract

Engineering - Electrical (55)

The essence of this project is to adopt mobile robotic technology from engineering and implement it in a fundamentally interdisciplinary education environment involving the performing arts and design. The project is adapting the proven educational use of off-the-shelf robots and their previous successful use of custom robots in theatrical performance. Traditional robotic pedagogy progresses from mechanisms and software to functional vehicles. Little time is left for application development, and many talented non-engineering students cannot participate. This innovative undergraduate robotics application environment by using off-the-shelf mobile robots, controlled via Web page interfaces requires no specialized systems experience. It is providing productive cross-disciplinary participation and mutual enrichment of engineering and arts students. Users access HTML control panels via standard Web browser such as Netscape or Internet Explorer. Mobile robots, wireless communication links, and PC computers with suitable server and software is being used. The educational aim is to provide an open development environment - free of operating system or platform expertise requirements - for students and faculty in all academic disciplines to develop creative applications for mobile robots. By analogy, this is what browsers did for the Internet, revolutionizing both commerce and culture while fueling further technical progress. Outcomes are enrichment of cross-disciplinary curricula in engineering and arts, faculty development, and exciting applications including WWW robotic theatrical performances directed by a professional artist. Results are being disseminated via Web and publications.

8.4 StudioBlue Productions

8.4.1 Eliza Redux: Veils of Transference

In the summer of 2003, a video was produced at StudioBlue Cooper Union by the artist Adrianne Wortzel. Eliza Redux: Veils Of Transference depicts a fictive pre-scripted psychoanalytic session between a human actor and a robot. Two versions were shot, one with the human as psychoanalyst and robot as patient and one with the roles reversed. The script, along with gestures, the movement of the robot's camera and the robot itself, was entered into GRASP software by James Cruickshanks. StudioBlue is a Chromakey telerobotic theater at Cooper Union, with a turntable set, allowing for background display of the "patient" or "therapist's" subtext (as interpreted by the artist) as the environment in which the session takes place.

Script and Production Design: Adrianne Wortzel
Cinematography: Huy Truong
Scripting Software: James Cruickshanks
Puppeteers: Chris Simon and Mike Sudano
Audio: Chris Simon

8.4.2 Shakespeare Robots

'Shakespeare Robots' is an interactive cinematic experience about artificial life in a post-robotic world. Woody, one of the last few classically trained Shakespearean robot actors, 'of the highest caliber' as he would say, is not interested in the quantum advancements in artificial intelligence or self organizing systems of artificial life, he just wants to perform contemporary plays. Plays that speak to him and his vanishing generation. Plays about their common 'Robot Condition'. He has no idea his ambitions are about to trigger the emergence of the first native life form in cyberspace. Initiating not only an era of the post-robotic but a post-human world as well.

Script and Production Design: Huy Truong
Cinematography: Huy Truong
Scripting Software: James Cruickshanks

8.5 Request for Service (Wall/Electrical)

REQUEST FOR SERVICES

The Cooper Union for the Advancement of Science
School of Engineering
Department of Mechanical Engineering
Telerobotics Theatre
Room 232

31 January 2002

Department Chair	**Prof. Jean LeMee**
Advisor	**Prof. Stan Wei**
Directors	**James Cruickshanks, ME Masters**
	Marcin Balicki, ME Masters

Introduction

The telerobotics theatre is still in its infancy. We have just finished the acquisition of the larger part of the hardware needed to perform the myriad of tasks necessary to meet the needs of such a theatre. However, in order get to the point of full operation there are still several technical issues that must be resolved. These include primarily the electrical system currently present in the theatre along with one or two structural changes.

Electrical

Several operations are needed to adequately wire the theatre. These operations as listed below.

Load Capacity

The current electrical system within the room is inadequate to power all of the devices and hardware needed for the theatre. Since the room is being divided into two separate labs there is currently only one outlet available for use. In addition, the one outlet that is available to us seems to be shared with appliances in the adjacent room (Materials Lab). Here is a list of the devices that need powering:

(5) Computers (CPU and Monitor)
(2) Video Monitors
(12) 250 Watt Stage Lights
(1) Air Conditioner
(5) Robot Power Supplies
and other Extraneous Hardware

In Figure 1 there is a layout of the theatre and the general locations of the aforementioned equipment.

Computer Outlets

There is an existing power strip located on the back wall of the alcove that is no longer powered. Powering this to support the majority of the computers and some of the equipment would be convenient.

Removal and Replacement of Fixtures

All but one of the existing fluorescents are to be removed. Only the florescent light above the workstations and the incandescent lights located on the floor beams are needed. The one remaining fluorescent is already independent from the rest of the existing lighting and requires no attention. The other fluorescents (a total of six sections of lights) must be removed (see Figure 1). The incandescent lights (2), are controlled by a switch located at the entrance to the room. Control of these lights must also be rerouted to the location noted in Figure 1.
In addition, boxes containing 4 outlets each will be used to power the (12) lights and several CCTV cameras. They currently do not exist and need to be installed. Control of these outlets must be routed to the location shown in Figure 1, by the other two switches, where a security switch will be placed.

Mounting Hooks

In addition to removing the lighting fixtures, hooks must be anchored in the ceiling to support the theatrical lighting. There will be X's marked at about 20 locations on the ceiling were supports are needed to hang the trusses. (See Figure 1)

Painting Paper

The floor is currently covered with "blue screen" flooring. The flooring (which is not always in use) quickly gets dirty and is in danger of being damaged do to relatively heavy foot traffic. Something is needed (perhaps painting paper) to cover the "blue screened" floor for protection.

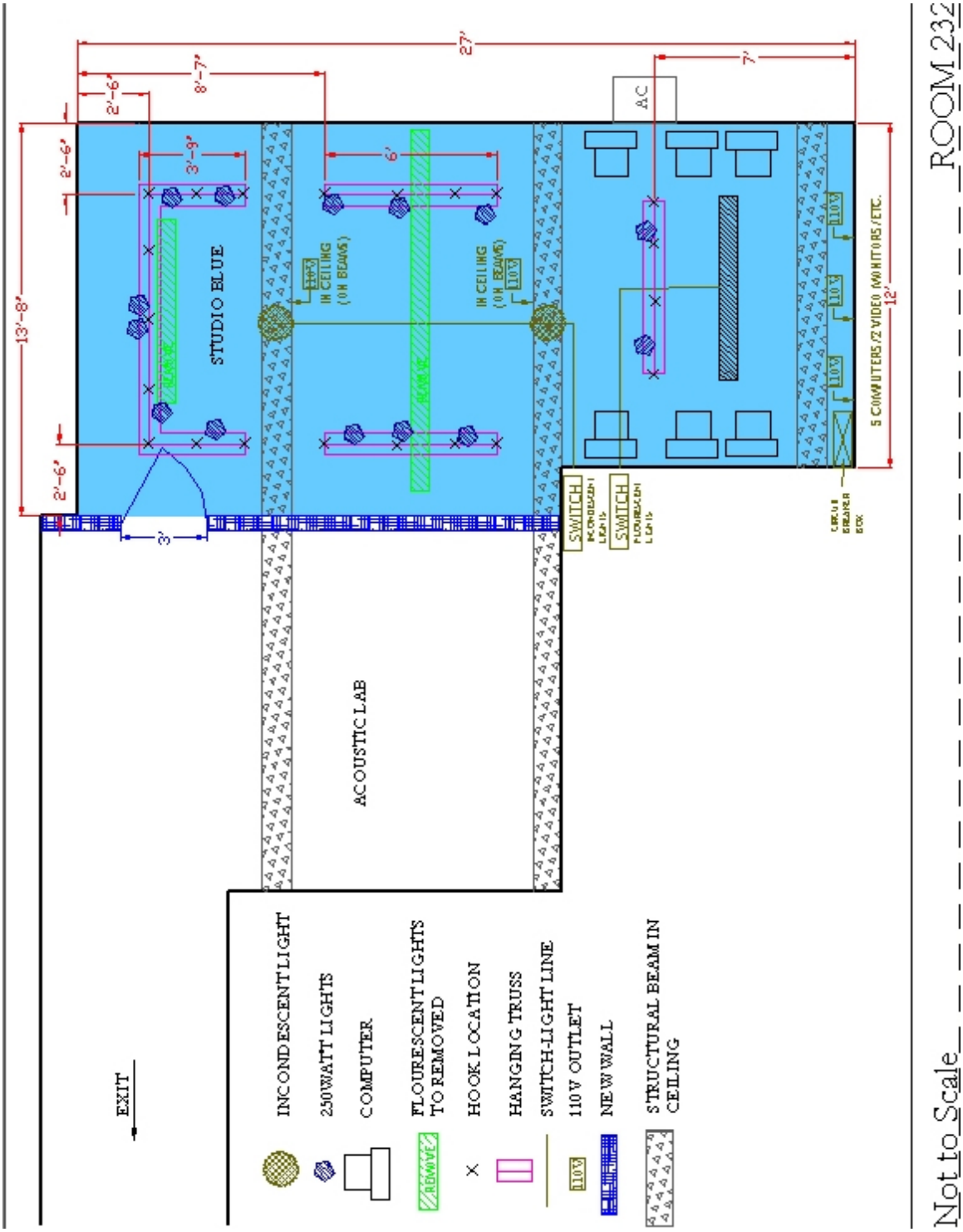
Dividing Wall

A dividing wall is necessary to separate the two labs that Room 232 has been broken up into: the telerobotics theatre and the acoustics lab. The proposed location is at the point on the west wall where it protrudes approximately 15". This can be seen in Figure 1.

Air Conditioner

The summer will be coming soon and it will be necessary to keep all of the equipment cool. The lighting and computers will be producing a lot of heat.

FIGURE 1


ROOM 232
Not to Scale
STUDIO BLUE
ACOUSTIC LAB
AC
27'
8'-7"
2'-6"
13'-8"
3'-9"
6'
7'
12'
3'
IN CEILING (ON BEAMS)
SWITCH
5 COMPUTERS /2 VIDEO MONITORS/ETC.
EXIT
INCONDESCENT LIGHT
250 WATT LIGHTS
COMPUTER
FLOURESCENT LIGHTS TO REMOVED
HOOK LOCATION
HANGING TRUSS
SWITCH-LIGHT LINE
110 V OUTLET
NEW WALL
STRUCTURAL BEAM IN CEILING

8.6 Chromakeyer (TBC –6000) User Guide

Connecting the TBC –6000.

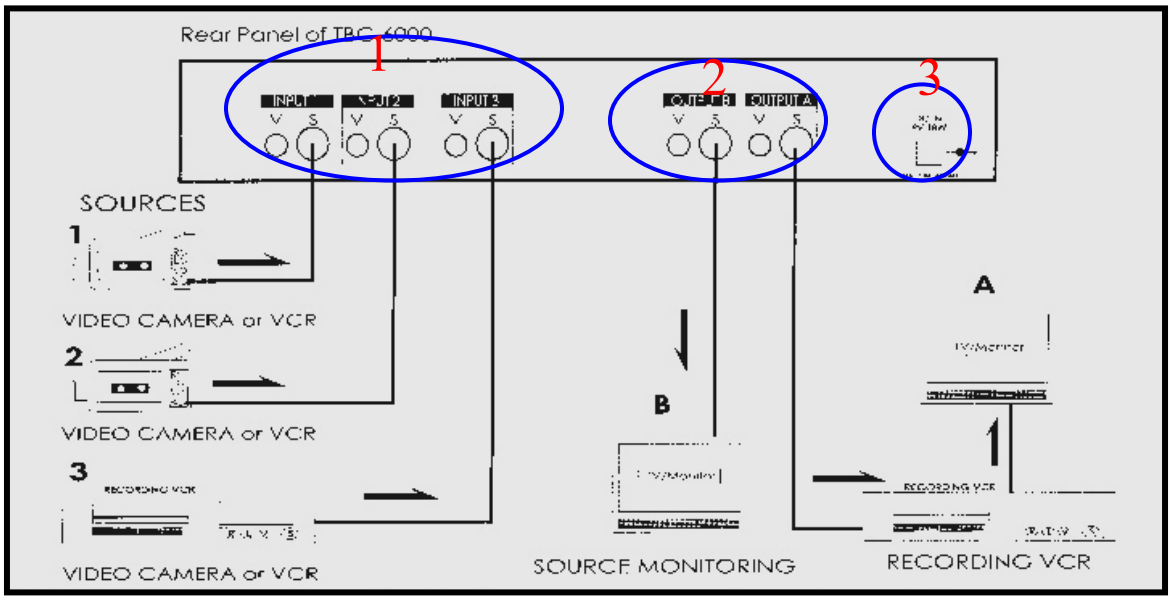


Figure 1 - Rear view of the TBC-6000.

1. The S-VHS input jacks (Input 1-2-3) allow connections for (a) two camera sources and a background VCR or (b) one camera and two VCR background sources.
2. The S-VHS output jack A is attached to a VCR and monitor to display the output with Chromakey effect. Output jack B is for monitoring the source video.
3. Power Supply: A 9V 18W AC-DC tra**NSF**ormer is connected in the rear to supply power.

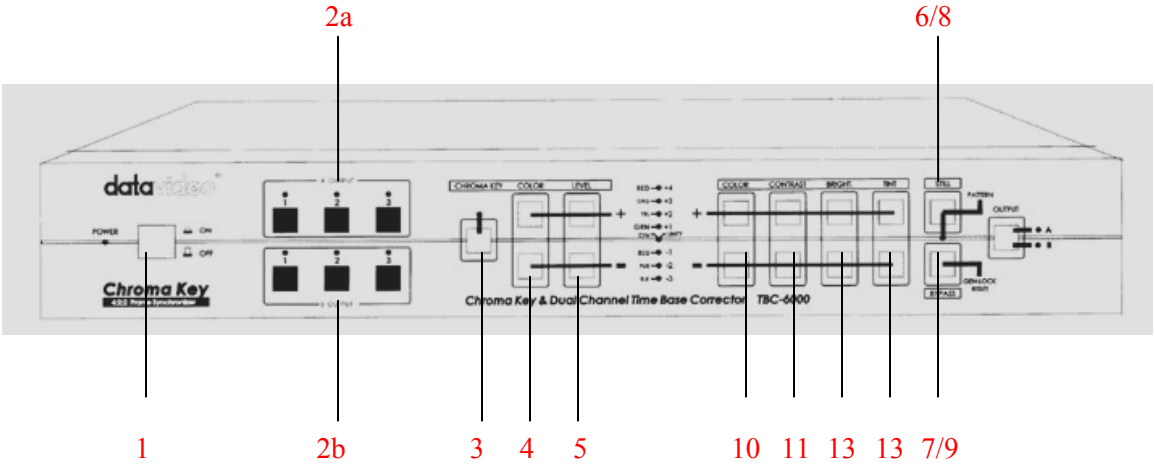


Figure 2 – Front View of the TBC-6000.

1. Power ON/OFF switch: (That's a hard one!!!)

2. (a) OUTPUT A SOURCE SWITCH: Use this to select the source channel 1,2 or 3 that the "chroma key" effect should "keyout" and "overlay" onto the video source. (b) OUTPUT B SOURCE SWITCH: Use this to select the source channel 1,2,or 3 for the background video.

3. CHROMA KEY SWITCH: press to activate or deactivate the chroma key effect.

4. COLOR SWITCHES: (+ and -) select the key color from 16 colors. (There are only 8 displayed, so you might be able to combine 2 at a time)

5. LEVEL SWITCHES: (+ and -) adjust the key level from 15-step level.

6. PATTERN SWITCHES: The TBC-6000 can generate a color bar test pattern on 4 source channels A to D.??? To generate a test pattern, press the pattern key and hold it for 2 seconds. To remove test pattern, press the PATTERN switch. ??????

7. GEN_LOCK Reset Switch: If the color in the video-image appears to be incorrect when you switch and Hold for 2 seconds to reset general-lock and correct the color.

8. STILL SWITCH: Pressing this switch "freezes" the source video image.

9. BYPASS SWITCH: Bypassing all color processing adjustment made to the source video image.

10. COLOR SWITCHES: (+ and -) adjust the Chroma level of the video source.

11. CONTRAST SWITCHES: (+ and -) adjust the luminance level of the source video.

12. BRIGHTNESS SWITCHES: (+ and -) the black level of the source video.

TINT SWITCHES: (+ and -) adjust the Chroma Tint of the source video (applies to NTSC only)

The above guide is part of the following work:
Kim, Yoongeu Yusuf, Mohammed. "Design and Construction of the Tele-Robotic Theatre."
Capstone Senior Design Project. May 8, 2003

8.7 Lighting Tips For Chromakey - Video or Photography

Dan Porvin - Cooper Union School of Art Video Deptartment

First, you will want to have plenty of space to work with and quite a few lights. Here are the 2 most common problems when using a chromakey backdrop:

- Uneven backdrop lighting.
- Reflected spill onto the foreground subject.

Here is how I recommend you set up your lighting.

Step 1 Light The Chromakey Backdrop:

When lighting a chromakey backdrop, you want the lighting on the backdrop to be as evenly as possible, with no shadows. Your backdrop lights should be behind or even with the foreground subject.

Step 2 Light The Foreground Subject

You will want to light the subject with its own set of lights.
You want as much separation between the foreground subject and the chromakey backdrop as possible. The more separation you have between the foreground subject and the chromakey backdrop, the better. This reduces the chance of the foreground shadows hitting the chromakey backdrop. It will help minimizes the "blue or green fringe" from light that is reflected off the chromakey backdrop. (called reflective spill)

Step 3 Neutralize Any Color-Cast

To neutralize any color-cast from light reflected off the backing. I use a soft back-light from each side to light up each side of the foreground subject, helping to neutralizing the reflected spill from the chromakey backdrop.

On non-Ultimatte Keyers, a standard technique to combat spill is to gel the backlight with a color opposite the chroma key color. That's magenta for green, straw for blue. DON'T do this with Ultimatte, screws up the spill correction.

Note If you take the time to light your foreground and chromakey backdrop correctly. No "blue or green fringe" on the object or the persons clothing, hair or skin, then even the most basic chromakey hardware or software will give you good results.

Rules For A Good Key

- Make sure your lighting is even and well balanced on the chromakey backdrop.
- Keep the subject as far forward of the chromakey backdrop as possible.
- Light the subject separately.
- Prevent shadows from hitting the chromakey backdrop.
- Make sure the subject does not contain the same color as your chromakey backdrop!

8.8 Linux Command Guide

This was compiled from man pages, random websites, reference books, and is considered common knowledge of *Linux*.

**********************PRINTING**

lpr -P Printer file
a2ps fin ascii to postscript
psnup -2 -d fin fout ps file into ps file of 2 pages per page
lpstat
cancel

**********************NAVIGATION**

cd - takes you to the last directory
cntrl a jump to beg of line
cntrl e end
cntrl k delete from current cursor to end.
cntrl k delete from current word
cntrl r cmd line history search
history | grep ln works on any command that returns text

**********************TEXT***

grep searches in text files
grep - i boo filename find boo in filename no matter the case
awk
tail -f logfilename shows last few lines of log files/updates
less
more
cat /etc/fstab
cat /etc/conf.modules
sed a text editor, makes editing changes according to a script

**********************JOBS**

jobs jobs in the current directory.
kill %# where # is the job number
ps aux | less
ps -aef
killall -9 netscape will kill netscape softly
kill -9 PID to kill a process
kill -l list of options
top (ps and kill in one) h for help
whereis -command locate the binary, source, and manual page files for a command
which -command locate a command; display its pathname or alias
fuser identify processes using a file or file structure
pstree how the processes are related
xload cpu load meter
free memory

**********************SHELLS**

Bash startup
For paths etc/profile seems to only work for all but root. Root has its own profile loaded (.bashrc) which is located in /root/

/etc/profile login shells
.bash_profile login shells

.profile login if no .bash_profile file is present
.bashrc interactive non-login shells
$ENV non-interactive shells
.bash_logout runs on termination
aliases should be placed in .profile
alias to see all of the aliases
aliases in '/etc/bashrc' for working
with CD drives in a console.
alias mcd="mount /dev/cdrom"
alias ecd="eject /dev/cdrom"
to restart a shell session
. .profile
If your shell is bash, and you have created a .bashrc file, you should also enter:
. .bashrc

**********************FILE SYSTEM***************************************
/opt/security security progams
/opt/install programs to be installed
/usr/local/bin/ . /usr/bin, /bin local binaries
/usr/sbin, /sbin, /usr/local/sbin system admin binaries
/var/log
/var/adm/kernel
/var/adm/syslog
/var/adm/messages
/sbin/liloconfig
**********************FILE OPERATIONS*************************************
rm -fr filename or directory (force recursive)
cp -i interactive copy
rm -i interactive remove
mv -i interactive move (use this to rename)
file filename classifies type of file
sum calculates and prints a 16-bit checksum
find / -name .rhosts -exec ;ls -l {}; (find all .rhost and ls them or you can rm or any other command
tar -cv tarball.tar files create and view
tar -cvf tarball.tar /usr create tar of directory and contents
tar -xvf tarball.tar extract
to add compression to an archive use the -z option for gzip or -Z for compress.
to decompress .tar.gz use -z to or to decompress .tar.Z use -Z
rpm -i file.rpm redhat package manager
rpm -ihv --force file.rpm forces installation
chown user/file change owner of file/dir
chgrp group/file change group of file/dir
chmod Absolute form
The other way to use the chmod command is the absolute form. In this case, you specify a set of three numbers that together determine all the access classes and types. Rather than being able to change only particular attributes, you must specify the entire state of the file's permissions. The three numbers are specified in the order: user (or owner), group, other. Each number is the sum of values that specify read (4), write (2), and execute (1) access, with 0 (zero) meaning no access. For example, if you wanted to have read, write, and execute permission on myfile, users in your group to have read and execute permission, and others to only have execute permission, the appropriate number would be calculated as (4+2+1)(4+0+1)(0+0+1) for the three digits 751. You would then enter the command as:
chmod 751 myfile

As another example, to give only yourself read, write, and execute permission on the current directory, you would calculate the digits as (4+2+1)(0+0+0)(0+0+0) for the sequence 700, and enter the command:
chmod 700 *
If it seems clearer to you, you can also think of the three digit sequence as the sum of attributes you select from the following table:

400 read by owner	040 read by group	004 read by others
200 write by owner	020 write by group	002 write by others
100 execute by owner	010 execute by group	001 execute by others

Some other frequently used examples are:
777 anyone can do anything (read, write, or execute)
755 you can do anything; others can only read and execute
711 you can do anything; others can only execute
644 you can read and write; others can only read
umask 022 (sets your default file permissions at creation 7-0,7-2,7-2)

**********NETWORK**********

rup uptime for remote machines
strobe Scan program
xauth used to edit and display the authorization information used in connecting to the X server
iptraf
ipfw firewall
ipfwadm
xfwp firewall proxy
samba *Linux* implementation of smb/session messege block aka netbios
majordomo mailing list daemon
traceroute
nslookup Query Internet domain name servers.

**********APPLICATIONS**********

xclock
cal calendar
xpdf pdf reader
acroread pdf reader
gv
mpg123 mp3 player
date
play wav player
aumix
man -k ssh looks for ssh in all man pages

**********SYSTEM **********

uname -a display system name
cat /proc/cpuinfo display cpu information
set shows settings
crontab -e edit crontab
du -sk fileorDirname summarize disk usage in Kilobytes 1023 KB= Megabyte
e2fsck -options device

**********USER **********

w shows all people logged in
who shows all people logged in
groups belongs to groups

**********VIM options**********

Create a .vimrc file in home directory

number of spaces.

set tabstop=4

set shiftwidth=4

set expandtab

**********Exporting Display**********

xhost +
xon servername
rsh servername remote shell login
on the machine you are telneting from:
"xhost + IP "
IP=machine that you are telneting to.
On the remote machine (one telneted to) :
export DISPLAY="IP:0.0"
export DISPLAY=roboweb.me.cooper.edu:0.0

A simple shell script :
==========
set-display.sh:
#!/bin/sh
echo Enter the IP you've logged in from:
read IP
export DISPLAY="$IP:0.0"

To run it every time you login include it in .profile:
==========
.profile:
exec set-display.sh
==========
Note also that there are ways of automating this, so you can have it automatically grab your IP rather than having you type it in, but I don't know the program to do it.

**********SCRIPTS**********

To start a program at startup in .etc/rc.d/rc4.file include the following:
echo "trying to start program"
exec /usr/local/myprog -nodaemon

If sockets prevent the restart of the server they could possibly be removed.
Although the are usually cleaned up by the system...
# remove stale hunt sockets so the game can start
if [-r /tmp/hunt -o -r /tmp/hunt.starts]; then
echo "removing your stale hunt sockets from /tmp..."
/bin/rm -f /tmp/hunt*
fi

8.9 Robot Issues and How to Correct Them

ROBOT CRASHED AND DOESN'T BOOT

This happens occasionally, a forced hard disk scan is required.
Use the keyboard and monitor to log in, if hard disk failed, it needs to be repaired.
As root type in:
e2fsck -options device
To force device check :
e2fsck -f devicewhere the device is the device that has errors ie '/dev/hda'.

dumpe2fs /dev/hda prints the super block and blocks group information for the file system present on **/dev/hda**

Excellent guide for *Linux* operation:
http://www.ee.surrey.ac.uk/Teaching/Unix/index.html

NO SOUND

Jumpers on the board did not match the ones in the modules.conf file so there was no sound. I also changed the irq to 9 instead of 5 due to a conflict, although ActivMedia suggested 5. I added privileges to /dev/dsp and /dev/mixer to everyone. Irq 7 works

Use play and aumixer to change props.

LOST ROOT PASSWORD
The robots are generally shipped without passwords for root or guest.
If root does seem to require a password, startup in single user mode.
Press [Ctrl]+[x] if prompted to start text mode. At the Boot: prompt type "*Linux* single" then at the shell prompt type passwd root to change the root password.

*8.10 Aria/*Saphira*/Botspeak Installation Instructions*

Installation:

Download Aria and **Saphira** from robots.activmedia.com.
Aria is freely distributed while **Saphira** is a licensed product requiring purchasing.
Follow their instructions for installation.
ViaVoice and Botspeak for *Linux* are not longer supported, however **StudioBlue** has two copies of it that were included in the earlier robots. After ViaVoice is installed, install botspeak from a file called botspeak.tar which can be found in older ActivMedia Installation CDs.

In the root directory
add these lines to .bashrc

```
# Robot directories
ARIA=/usr/local/Aria
SAPHIRA=/usr/local/Saphira
BOTSPEAK=/usr/local/botspeak
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$SAPHIRA/lib:$ARIA/lib
ILU_BINDING_DIRECTORY=$BOTSPEAK/binding
export BOTSPEAK LD_LIBRARY_PATH ILU_BINDING_DIRECTORY
export ARIA SAPHIRA BOTSPEAK
```

after the following line:

```
PATH=/sbin:/usr/sbin:/bin:/usr/bin:/usr/X11R6/bin:/usr/local/bin:/usr/loca
l/sbin
```

add **Saphira** and aria to the path:

```
PATH=$PATH:$ARIA/bin:$SAPHIRA/bin
```

As root add the same lines to /etc/profile :

```
# Robot directories
ARIA=/usr/local/Aria
SAPHIRA=/usr/local/Saphira
BOTSPEAK=/usr/local/botspeak
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$SAPHIRA/lib:$ARIA/lib
ILU_BINDING_DIRECTORY=$BOTSPEAK/binding
export BOTSPEAK LD_LIBRARY_PATH ILU_BINDING_DIRECTORY
export ARIA SAPHIRA BOTSPEAK
PATH=$PATH:$ARIA/bin:$SAPHIRA/bin
----
```

Install ViaVoice for *Linux*! (from the Original CD)
Restart the system

After **Saphira**, Botspeak (**ViaVoice**) and especially **ARIA** are installed **RCCI** installation can proceed.

From the included CD copy rcci1.0 directory (includes source, compiled code and documentation) via FTP to your user directory.
The code should already be complied into an executable file called *rcci.*

Execution:

Startup script should start the required services and rcci with the desired port (i.e. 5555):

./start 5555

Note: Occasionally ftp will change file permissions on tra**NSF**ers.
To make sure that the files can execute change their permission via:
chmod 744 *

Compilation:

If recompilation is required in rcci1.0 directory (file permissions permitting) type:

make

This makefile script will compile and link all the code and libraries.
Note: Aria should be compiled with the same compiler as RCCI.

If there is a linking error it is most likely due to incompatible compiler versions.
Download and install gcc version 2.95.3 from http://gcc.gnu.org/.
Add the new compiler to the path before anything else so when GCC is called, the 2.95.3 version is used.

This next line should be located in both of the aforementioned files (.bashrc, .profile)

```
PATH=/usr/local/gcc-2.95.3/bin:$PATH
```

Recompile Aria and then ***rcci***.

Running:

Once the RCCI (server) is running any number of clients can connect to it via *Telnet*:

Telnet **ADDRESS PORT**

ADDRESS is the IP or DNS name of the computer running RCCI.
PORT is the port number that RCCI is listening to.

After connection is establish user can send formatted commands to RCCI:

CONNECT|robot||

8.11 Startup Shell Script

To start the shell script :

./start 5555

“start” script file:

```
#!/bin/bash
echo "RUNNING RCCI: "
if [ $# -ne 1 ]; then
         echo 1>&2 Usage: $0 PORT
         exit 127
    fi
#echo "Adjusting volume and muting microphone"
aumix -w 50
aumix -v 50
aumix -m 0
while ((1>0))
do
echo "Killing ViaVoice engine, Botspeak and RCCI"
killall -9 rcci
killall -9 botspeak.srvr
killall -9 engine
sleep 2
#restart the ViaVoice engine
/usr/lib/ViaVoice/bin/engine
sleep 3
echo "Attempting to RCCI Server on port: " $1
./rcci $1
sleep 10
echo "TRYING AGAIN"
done
```

8.12 C++ Compilation Makefile

```
#
# Makefile for Saphira applications
#

SHELL = /bin/sh

############################################################

SRCD =./
OBJD =./obj/
INCD =./include/
INCSAPH =/usr/local/Saphira/ohandler/include
LIBD =/usr/local/Saphira/lib/
ARIAD = /usr/local/Aria
INCSPEECH=$(ARIAD)/ArSpeech/include/
BOTSPEAK= /usr/local/botspeak
# find out which OS we have
include $(INCSAPH)/os.h

CFLAGS =  -g -D$(CONFIG) $(PICFLAG) $(REENTRANT)
CC = gcc
CPP = g++
INCLUDE2= -I$(BOTSPEAK)/include -I$(INCSAPH) -I$(INCSPEECH)
INCLUDE= -I$(INCD) -I$(ARIAD)/include -I$(INCSAPH)
############################################################
all: $(BIND)rcci
      touch all

#
# Test
#
#the : -o file.o file.cpp : allows for specifiying the name and location
#of .o file.

$(OBJD)main.o: $(SRCD)main.cpp
      $(CPP) $(CFLAGS) -c -o $(OBJD)main.o $(SRCD)main.cpp $(INCLUDE)

$(OBJD)SbMessage.o: $(SRCD)SbMessage.cpp $(INCD)SbMessage.h
      $(CPP) $(CFLAGS) -c -o $(OBJD)SbMessage.o $(SRCD)SbMessage.cpp
$(INCLUDE)

$(OBJD)SbServerML.o: $(SRCD)SbServerML.cpp $(INCD)SbServerML.h
      $(CPP) $(CFLAGS) -c -o $(OBJD)SbServerML.o $(SRCD)SbServerML.cpp
$(INCLUDE)

$(OBJD)SbSocket.o: $(SRCD)SbSocket.cpp $(INCD)SbSocket.h
      $(CPP) $(CFLAGS) -c -o $(OBJD)SbSocket.o $(SRCD)SbSocket.cpp
$(INCLUDE)

$(OBJD)SbMsgListener.o: $(SRCD)SbMsgListener.cpp $(INCD)SbMsgListener.h
      $(CPP) $(CFLAGS) -c -o $(OBJD)SbMsgListener.o
$(SRCD)SbMsgListener.cpp $(INCLUDE)
```

```
$(OBJD)SbMsgHandlerSingleton.o: $(SRCD)SbMsgHandlerSingleton.cpp
$(INCD)SbMsgHandlerSingleton.h
	$(CPP) $(CFLAGS) -c -o $(OBJD)SbMsgHandlerSingleton.o
$(SRCD)SbMsgHandlerSingleton.cpp $(INCLUDE)

$(OBJD)SbRobotML.o: $(SRCD)SbRobotML.cpp $(INCD)SbRobotML.h
	$(CPP) $(CFLAGS) -c -o $(OBJD)SbRobotML.o $(SRCD)SbRobotML.cpp
$(INCLUDE)

$(OBJD)SbManagerT.o: $(SRCD)SbManagerT.cpp $(INCD)SbManagerT.h
	$(CPP) $(CFLAGS) -c -o $(OBJD)SbManagerT.o $(SRCD)SbManagerT.cpp
$(INCLUDE)  $(INCLUDE2)

$(OBJD)SbSoundsMLT.o: $(SRCD)SbSoundsMLT.cpp $(INCD)SbSoundsMLT.h
	$(CPP) $(CFLAGS) -c -o $(OBJD)SbSoundsMLT.o $(SRCD)SbSoundsMLT.cpp
$(INCLUDE) $(INCLUDE2)

$(OBJD)SbMotionML.o: $(SRCD)SbMotionML.cpp $(INCD)SbMotionML.h
	$(CPP) $(CFLAGS) -c -o $(OBJD)SbMotionML.o $(SRCD)SbMotionML.cpp
$(INCLUDE)

$(OBJD)SbActionML.o: $(SRCD)SbActionML.cpp $(INCD)SbActionML.h
	$(CPP) $(CFLAGS) -c -o $(OBJD)SbActionML.o $(SRCD)SbActionML.cpp
$(INCLUDE)

$(OBJD)SbGripperML.o: $(SRCD)SbGripperML.cpp $(INCD)SbGripperML.h
	$(CPP) $(CFLAGS) -c -o $(OBJD)SbGripperML.o $(SRCD)SbGripperML.cpp
$(INCLUDE)

$(OBJD)SbStateML.o: $(SRCD)SbStateML.cpp $(INCD)SbStateML.h
	$(CPP) $(CFLAGS) -c -o $(OBJD)SbStateML.o $(SRCD)SbStateML.cpp
$(INCLUDE)

$(OBJD)SbCameraSONYML.o: $(SRCD)SbCameraSONYML.cpp $(INCD)SbCameraSONYML.h
	$(CPP) $(CFLAGS) -c -o $(OBJD)SbCameraSONYML.o
$(SRCD)SbCameraSONYML.cpp $(INCLUDE)

$(OBJD)SbCameraVCC4ML.o: $(SRCD)SbCameraVCC4ML.cpp $(INCD)SbCameraVCC4ML.h
	$(CPP) $(CFLAGS) -c -o $(OBJD)SbCameraVCC4ML.o
$(SRCD)SbCameraVCC4ML.cpp $(INCLUDE)

$(OBJD)SbActionGoto.o: $(SRCD)SbActionGoto.cpp $(INCD)SbActionGoto.h
	$(CPP) $(CFLAGS) -c -o $(OBJD)SbActionGoto.o $(SRCD)SbActionGoto.cpp
$(INCLUDE)

$(OBJD)SbActionRotate.o: $(SRCD)SbActionRotate.cpp $(INCD)SbActionRotate.h
	$(CPP) $(CFLAGS) -c -o $(OBJD)SbActionRotate.o
$(SRCD)SbActionRotate.cpp $(INCLUDE)

$(OBJD)SbActionMove.o: $(SRCD)SbActionMove.cpp $(INCD)SbActionMove.h
	$(CPP) $(CFLAGS) -c -o $(OBJD)SbActionMove.o $(SRCD)SbActionMove.cpp
$(INCLUDE)

$(OBJD)SbSaphiraML.o: $(SRCD)SbSaphiraML.cpp $(INCD)SbSaphiraML.h
	$(CPP) $(CFLAGS) -c -o $(OBJD)SbSaphiraML.o $(SRCD)SbSaphiraML.cpp
$(INCLUDE)
```

```
$(OBJD)SbExampleML.o: $(SRCD)SbExampleML.cpp $(INCD)SbExampleML.h
	$(CPP) $(CFLAGS) -c -o $(OBJD)SbExampleML.o $(SRCD)SbExampleML.cpp
$(INCLUDE)

$(OBJD)SbExampleMLT.o: $(SRCD)SbExampleMLT.cpp $(INCD)SbExampleMLT.h
	$(CPP) $(CFLAGS) -c -o $(OBJD)SbExampleMLT.o $(SRCD)SbExampleMLT.cpp
$(INCLUDE)

$(BIND)rcci: $(OBJD)main.o $(OBJD)SbMessage.o  $(OBJD)SbServerML.o
$(OBJD)SbSocket.o  \
	$(OBJD)SbMsgListener.o $(OBJD)SbMsgHandlerSingleton.o
$(OBJD)SbRobotML.o  \
	$(OBJD)SbSoundsMLT.o $(OBJD)SbManagerT.o  $(OBJD)SbMotionML.o
$(OBJD)SbActionML.o \
	$(OBJD)SbGripperML.o  $(OBJD)SbStateML.o $(OBJD)SbCameraSONYML.o
$(OBJD)SbActionGoto.o \
	$(OBJD)SbActionRotate.o $(OBJD)SbActionMove.o $(OBJD)SbSaphiraML.o
$(OBJD)SbExampleML.o \
	  $(OBJD)SbExampleMLT.o $(OBJD)SbCameraVCC4ML.o
	$(CPP) $(OBJD)main.o $(OBJD)SbMessage.o $(OBJD)SbServerML.o
$(OBJD)SbSocket.o \
	$(OBJD)SbMsgListener.o $(OBJD)SbMsgHandlerSingleton.o
$(OBJD)SbRobotML.o \
	$(OBJD)SbSoundsMLT.o $(OBJD)SbManagerT.o $(OBJD)SbMotionML.o
$(OBJD)SbActionML.o \
	$(OBJD)SbGripperML.o $(OBJD)SbStateML.o $(OBJD)SbCameraSONYML.o
$(OBJD)SbActionGoto.o \
	$(OBJD)SbActionMove.o $(OBJD)SbActionRotate.o $(OBJD)SbSaphiraML.o
$(OBJD)SbExampleML.o \
	  $(OBJD)SbExampleMLT.o $(OBJD)SbCameraVCC4ML.o\
	-o $(BIND)rcci \
	-L$(LIBD) -L$(ARIAD)/lib -lAria $(LLIBSX) \
	-L$(BOTSPEAK)/lib -lbotspeak -lsf -lsfGrad
```

8.13 RCCI Documentation

Robot Control and Communication Interface

Documentation

Marcin Balicki
Version v 1.0
5/11/2004

Table of Contents

Robot Control and Communication Interface

Introduction

Robot Control and Communication Interface (RCCI) is a software package (C++) that works with ARIA robot API to provide a user friendly and versatile command interface over TCP/IP to control robot functions and can be utilized from practically any computer platform. It is an integral part of StudioBlue; the Robotic Theater at Cooper Union.

Installation

Download Aria and Saphira from robots.activmedia.com. Aria is freely distributed while Saphira is a licensed product requiring purchasing. Follow their instructions for installation.

ViaVoice and Botspeak for Linux are not longer supported, however StudioBlue has two copies of it that were included in the earlier robots. After ViaVoice is installed, install Botspeak from a file called botspeak.tar which can be found in older ActivMedia Installation CDs.

In the root directory add these lines to .bashrc

```
# Robot directories
 ARIA=/usr/local/Aria
 SAPHIRA=/usr/local/Saphira
 BOTSPEAK=/usr/local/botspeak
 LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$SAPHIRA/lib:$ARIA/lib
 ILU_BINDING_DIRECTORY=$BOTSPEAK/binding
 export BOTSPEAK LD_LIBRARY_PATH ILU_BINDING_DIRECTORY
 export ARIA SAPHIRA BOTSPEAK
 after the following line:
 PATH=/sbin:/usr/sbin:/bin:/usr/bin:/usr/X11R6/bin:/usr/local/bin:/usr/local/sbin
 add saphira and aria to the path:
 PATH=$PATH:$ARIA/bin:$SAPHIRA/bin
```

As root add the same lines to /etc/profile :

```
# Robot directories
 ARIA=/usr/local/Aria
 SAPHIRA=/usr/local/Saphira
 BOTSPEAK=/usr/local/botspeak
 LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$SAPHIRA/lib:$ARIA/lib
```

ILU_BINDING_DIRECTORY=$BOTSPEAK/binding

export BOTSPEAK LD_LIBRARY_PATH ILU_BINDING_DIRECTORY

export ARIA SAPHIRA BOTSPEAK

PATH=$PATH:$ARIA/bin:$SAPHIRA/bin

Install ViaVoice for Linux! (from the Original CD) and Restart the system After Saphira, Botspeak (ViaVoice) and especially ARIA are installed RCCI installation can proceed. From the included CD copy rcci1.0 directory (includes source, compiled code and documentation) via FTP to your user directory.

Execution

ExecutionThe code should already be complied into an executable file called rcci.

Startup script should start the required services and rcci with the desired port (i.e. 5555):

./start 5555

Note: Occasionally ftp will change file permissions on transfers.

To make sure that the files can execute change their permission via:

chmod 744 *

Compilation

CompilationIf recompilation is required in rcci1.0 directory (file permissions permitting) type:

make

This makefile script will compile and link all the code and libraries.

Note: Aria should be compiled with the same compiler as RCCI.

If there is a linking error it is most likely due to incompatible compiler versions.

Download and install gcc version 2.95.3 from http://gcc.gnu.org/.

Add the new compiler to the path before anything else so when GCC is called, the 2.95.3 version is used.

This next line should be located in both of the aforementioned files (.bashrc, .profile)

PATH=/usr/local/gcc-2.95.3/bin:$PATH

Recompile Aria and then rcci.

Running

main c++ file is the entrance point into the program. It checks if the command line PORT argument was entered correctly. If true, then it initializes ARIA in threaded mode. Once the RCCI (server) is running Telnet can be used to connect to RCCI.

telnet ADDRESS PORT

ADDRESS is the IP or DNS name of the computer running RCCI.

PORT is the port number that RCCI is listening to.

After connection is establish user can send formatted commands to RCCI:

CONNECT|robot||

Robot Control and Communication Interface Hierarchical Index

Robot Control and Communication Interface Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Robot Control and Communication Interface Class Index

Robot Control and Communication Interface Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Robot Control and Communication Interface Page Index

Robot Control and Communication Interface Related Pages

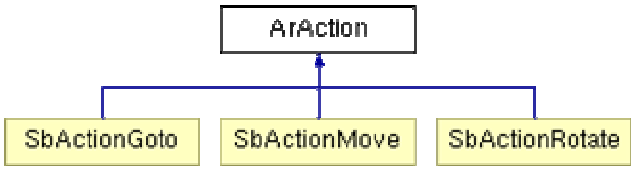
Here is a list of all related documentation pages:

Robot Control and Communication Interface Class Documentation

ArAction Class Reference

ArAction`#include <ArAction.h>`

Inheritance diagram for ArAction:



Detailed Description

Action class, what typically makes the robot move.

Definition at line 39 of file ArAction.h.

Public Member Functions

- AREXPORT ArAction (const char *name, const char *description="")
  Constructor.
- virtual AREXPORT ~ArAction ()
  Desructor.
- virtual AREXPORT bool isActive (void) const
  Finds out whether the action is active or not.
- virtual AREXPORT void activate (void)
  Activate the action.
- virtual AREXPORT void deactivate (void)
  Deactivate the action.
- virtual AREXPORT ArActionDesired * fire (ArActionDesired currentDesired)=0
- virtual AREXPORT void setRobot (ArRobot *robot)
  Sets the robot this action is driving.
- virtual AREXPORT int getNumArgs (void) const
  Find the number of arguments this action takes.
- virtual AREXPORT const ArArg * getArg (int number) const
  Gets the numbered argument.
- virtual AREXPORT ArArg * getArg (int number)
  Gets the numbered argument.
- virtual AREXPORT const char * getName (void) const
  Gets the name of the action.
- virtual AREXPORT const char * getDescription (void) const
  Gets the long description of the action.

- virtual AREXPORT ArActionDesired * getDesired (void)
  Gets what this action wants to do (for display purposes).
- virtual AREXPORT void log (bool verbose=true) const
  ArLog::log s the actions stats.

Protected Member Functions

- AREXPORT void setNextArgument (ArArg const &arg)
  Sets the argument type for the next argument (only use in constructor).

Protected Attributes

- ArRobot * **myRobot**
- bool **myIsActive**
- int **myNumArgs**
- std::map< int, ArArg > **myArgumentMap**
- std::string **myName**
- std::string **myDescription**

Member Function Documentation

virtual AREXPORT ArActionDesired* ArAction::fire (ArActionDesired *currentDesired*) [pure virtual]

Parameters:

currentDesired this is what the current resolver has for its desired, this is SOLELY for the purpose of giving information to the action

Returns:

pointer to what this action wants to do, NULL if it wants to do nothing

Implemented in SbActionGoto, SbActionMove, and SbActionRotate.

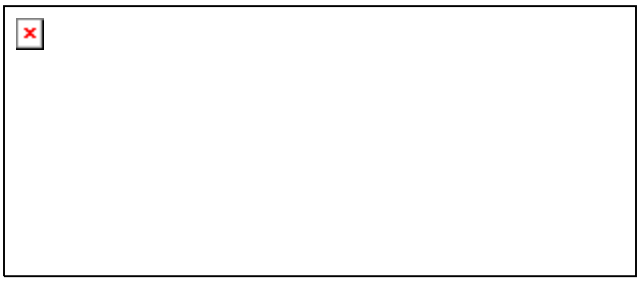
The documentation for this class was generated from the following file:

- ArAction.h

ArASyncTask Class Reference

ArASyncTask#include <ArASyncTask.h>

Inheritance diagram for ArASyncTask:



Detailed Description

The ArAsynTask is a task that runs in its own thread.

This is a rather simple class. The user simply needs to derive their own class from ArAsyncTask and define the runThread() function. They then need to create an instance of their task and call run or runAsync. The standard way to stop a task is to call stopRunning() which sets ArThread::myRunning to false. In their run loop, they should pay attention to the getRunning() or the ArThread::myRunning variable. If this value goes to false, the task should clean up after itself and exit its runThread() function.

Definition at line 49 of file ArASyncTask.h.

Public Types

- typedef pthread_t **ThreadType**
- typedef std::map< ThreadType, ArThread * > **MapType**
- enum Status { STATUS_FAILED = 1, STATUS_NORESOURCE, STATUS_NO_SUCH_THREAD, STATUS_INVALID, STATUS_JOIN_SELF, STATUS_ALREADY_DETATCHED }

Public Member Functions

- AREXPORT ArASyncTask ()
  Constructor.
- virtual AREXPORT ~ArASyncTask ()
  Destructor.
- virtual AREXPORT void * runThread (void *arg)=0
  Override this function and put your taskes run loop here.
- virtual AREXPORT void run (void)
  Run in this thread.
- virtual AREXPORT void runAsync (void)
  Run in its own thread.
- virtual AREXPORT void stopRunning (void)
  Stop the thread.

- virtual AREXPORT int create (bool joinable=true, bool lowerPriority=true)
  Create the task and start it going.
- virtual AREXPORT void * runInThisThread (void *arg=0)
  Run the code of the task syncronously.
- virtual AREXPORT int join (void **ret=NULL)
  Join on the thread.
- virtual AREXPORT int detach (void)
  Detatch the thread so it cant be joined.
- virtual AREXPORT void cancel (void)
  Cancel the thread.
- virtual AREXPORT bool getRunning (void) const
  Get the running status of the thread.
- virtual AREXPORT bool getRunningWithLock (void)
  Get the running status of the thread, locking around the variable.
- virtual AREXPORT bool getJoinable (void) const
  Get the joinable status of the thread.
- virtual AREXPORT const ThreadType * getThread (void) const
  Get the underlying thread type.
- virtual AREXPORT ArFunctor * getFunc (void) const
  Get the functor that the thread runs.
- virtual AREXPORT void setRunning (bool running)
  Set the running value on the thread.
- AREXPORT int lock (void)
  Lock the thread instance.
- AREXPORT int tryLock (void)
  Try to lock the thread instance without blocking.
- AREXPORT int unlock (void)
  Unlock the thread instance.
- bool getBlockAllSignals (void)
  Do we block all process signals at startup?

Static Public Member Functions

- AREXPORT void init (void)
  Initialize the internal book keeping structures.
- AREXPORT ArThread * self (void)
  Returns the instance of your own thread.
- AREXPORT void stopAll ()
  Stop all threads.
- AREXPORT void cancelAll (void)
  Cancel all threads.
- AREXPORT void joinAll (void)
  Join on all threads.
- AREXPORT void yieldProcessor (void)
  Yield the processor to another thread.

Protected Member Functions

- virtual AREXPORT int **doJoin** (void **ret=NULL)

Protected Attributes

- ArMutex **myMutex**
- bool myRunning
  State variable to denote when the thread should continue or exit.
- bool **myJoinable**
- bool **myBlockAllSignals**
- ThreadType **myThread**
- ArStrMap **myStrMap**

Static Protected Attributes

- ArMutex **ourThreadsMutex**
- MapType **ourThreads**

Private Member Functions

- virtual int create (ArFunctor *func, bool joinable=true, bool lowerPriority=true)
  Create and start the thread.

Private Attributes

- ArRetFunctor1C< void *, ArASyncTask, void * > **myFunc**

Member Enumeration Documentation

enum ArThread::Status [inherited]

Enumeration values:

STATUS_FAILED Failed to create the thread.
STATUS_NORESOURCE Not enough system resources to create the thread.
STATUS_NO_SUCH_THREAD The thread can no longer be found.
STATUS_INVALID Thread is detached or another thread is joining on it.
STATUS_JOIN_SELF Thread is your own thread. Can't join on self.
STATUS_ALREADY_DETATCHED Thread is already detatched.

Definition at line 63 of file ArThread.h.

Member Function Documentation

virtual AREXPORT void* ArASyncTask::runThread (void * *arg*) [pure virtual]

Override this function and put your taskes run loop here.

Check the value of getRunning() or myRunning periodicly in your loop. If the value goes false, the loop should exit and runThread() should return.

Implemented in SbExampleMLT, SbManagerT, and SbSoundsMLT.

The documentation for this class was generated from the following file:

- ArASyncTask.h

ArMutex Class Reference

ArMutex#include <ArMutex.h>

Detailed Description

This class wraps the operating systems mutex functions.

It allows mutualy exclusive access to a critical section. This is extremely usefull for multiple threads which want to use the same variable. ArMutex simply uses the POSIX pthread interface in an object oriented manner. It also applies the same concept to Windows using Windows own abilities to restrict access to critical sections.

Definition at line 47 of file ArMutex.h.

Public Types

- typedef pthread_mutex_t **MutexType**
- enum Status { STATUS_FAILED_INIT = 1, STATUS_FAILED, STATUS_ALREADY_LOCKED }

Public Member Functions

- AREXPORT ArMutex ()
  Constructor.
- virtual AREXPORT ~ArMutex ()
  Destructor.
- virtual AREXPORT int lock ()
  Lock the mutex.
- virtual AREXPORT int tryLock ()
  Try to lock the mutex, but do not block.
- virtual AREXPORT int unlock ()
  Unlock the mutex, allowing another thread to obtain the lock.
- virtual AREXPORT const char * getError (int messageNumber) const
  Get a human readable error message from an error code.
- virtual AREXPORT MutexType & getMutex ()
  Get a reference to the underlying mutex variable.

Protected Attributes

- bool **myFailedInit**
- MutexType **myMutex**
- ArStrMap **myStrMap**

Member Enumeration Documentation

enum ArMutex::Status

Enumeration values:

STATUS_FAILED_INIT Failed to initialize.
STATUS_FAILED General failure.

STATUS_ALREADY_LOCKED Mutex already locked.

Definition at line 57 of file ArMutex.h.

Member Function Documentation

int ArMutex::lock (void) [virtual]

Lock the mutex.

This function will block until no other thread has this mutex locked. If it returns 0, then it obtained the lock and the thread is free to use the critical section that this mutex protects. Else it returns an error code. See getError().

Definition at line 68 of file ArMutex_LIN.cpp.

References STATUS_ALREADY_LOCKED, STATUS_FAILED, and STATUS_FAILED_INIT.

Referenced by ArThread::lock().

int ArMutex::tryLock (void) [virtual]

Try to lock the mutex, but do not block.

This function will not block if another thread has the mutex locked. It will return instantly if that is the case. It will return STATUS_ALREADY_LOCKED if another thread has the mutex locked. If it obtains the lock, it will return 0.

Definition at line 99 of file ArMutex_LIN.cpp.

References STATUS_ALREADY_LOCKED, STATUS_FAILED, and STATUS_FAILED_INIT.

Referenced by ArThread::tryLock().

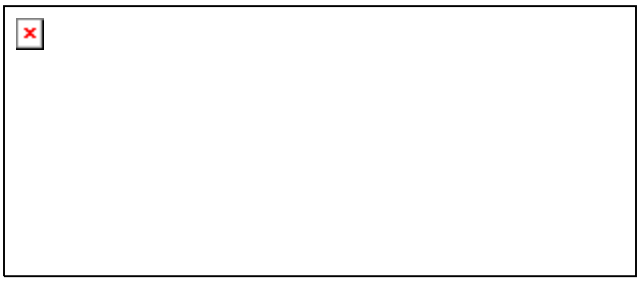
The documentation for this class was generated from the following files:

- ArMutex.h
- ArMutex_LIN.cpp

ArThread Class Reference

ArThread#include <ArThread.h>

Inheritance diagram for ArThread:



Detailed Description

POSIX/WIN32 thread wrapper class.

create() will create the thread. That thread will run the given Functor. A thread can either be in a detached state or a joinable state. If the thread is in a detached state, that thread can not be join()'ed upon. The thread will simply run until the program exits, or its function exits. A joinable thread means that another thread and call join() upon it. If this function is called, the caller will block until the thread exits its function. This gives a way to synchronize upon the lifespan of threads. Calling cancel() will cancel the thread. The static function self() will return a thread

Definition at line 52 of file ArThread.h.

Public Types

- typedef pthread_t **ThreadType**
- typedef std::map< ThreadType, ArThread * > **MapType**
- enum Status { STATUS_FAILED = 1, STATUS_NORESOURCE, STATUS_NO_SUCH_THREAD, STATUS_INVALID, STATUS_JOIN_SELF, STATUS_ALREADY_DETATCHED }

Public Member Functions

- AREXPORT ArThread (bool blockAllSignals=true)
  Constructor.
- AREXPORT ArThread (ThreadType thread, bool joinable, bool blockAllSignals=true)
  Constructor - starts the thread.
- AREXPORT ArThread (ArFunctor *func, bool joinable=true, bool blockAllSignals=true)
  Constructor - starts the thread.
- virtual AREXPORT ~ArThread ()
  Destructor.
- virtual AREXPORT int create (ArFunctor *func, bool joinable=true, bool lowerPriority=true)
  Create and start the thread.
- virtual AREXPORT void stopRunning (void)
  Stop the thread.
- virtual AREXPORT int join (void **ret=NULL)

Join on the thread.

- virtual AREXPORT int detach (void)
  Detatch the thread so it cant be joined.
- virtual AREXPORT void cancel (void)
  Cancel the thread.
- virtual AREXPORT bool getRunning (void) const
  Get the running status of the thread.
- virtual AREXPORT bool getRunningWithLock (void)
  Get the running status of the thread, locking around the variable.
- virtual AREXPORT bool getJoinable (void) const
  Get the joinable status of the thread.
- virtual AREXPORT const ThreadType * getThread (void) const
  Get the underlying thread type.
- virtual AREXPORT ArFunctor * getFunc (void) const
  Get the functor that the thread runs.
- virtual AREXPORT void setRunning (bool running)
  Set the running value on the thread.
- AREXPORT int lock (void)
  Lock the thread instance.
- AREXPORT int tryLock (void)
  Try to lock the thread instance without blocking.
- AREXPORT int unlock (void)
  Unlock the thread instance.
- bool getBlockAllSignals (void)
  Do we block all process signals at startup?

Static Public Member Functions

- AREXPORT void init (void)
  Initialize the internal book keeping structures.
- AREXPORT ArThread * self (void)
  Returns the instance of your own thread.
- AREXPORT void stopAll ()
  Stop all threads.
- AREXPORT void cancelAll (void)
  Cancel all threads.
- AREXPORT void joinAll (void)
  Join on all threads.
- AREXPORT void yieldProcessor (void)
  Yield the processor to another thread.

Protected Member Functions

- virtual AREXPORT int **doJoin** (void **ret=NULL)

Protected Attributes

- ArMutex **myMutex**

- bool myRunning
  State variable to denote when the thread should continue or exit.
- bool **myJoinable**
- bool **myBlockAllSignals**
- ArFunctor * **myFunc**
- ThreadType **myThread**
- ArStrMap **myStrMap**

Static Protected Attributes

- ArMutex **ourThreadsMutex**
- MapType **ourThreads**

Member Enumeration Documentation

enum ArThread::Status

Enumeration values:

STATUS_FAILED Failed to create the thread.
STATUS_NORESOURCE Not enough system resources to create the thread.
STATUS_NO_SUCH_THREAD The thread can no longer be found.
STATUS_INVALID Thread is detached or another thread is joining on it.
STATUS_JOIN_SELF Thread is your own thread. Can't join on self.
STATUS_ALREADY_DETATCHED Thread is already detatched.

Definition at line 63 of file ArThread.h.

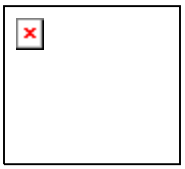
The documentation for this class was generated from the following file:

- ArThread.h

SbActionGoto Class Reference

SbActionGoto`#include <SbActionGoto.h>`

Inheritance diagram for SbActionGoto:



Detailed Description

This action goes to a given ArPose very naively.

This class has been adopted from ActivMedia's ArActionGoto. The action stops when it gets closeDist away. You can give it a new goal with setGoal(), cancel its movement with cancelGoal(), and see if it got there with haveAchievedGoal(). This doesn't avoid obstacles or anything, you could have an avoid routine at a higher priority to avoid on the way there, but for real and intelligent looking navigation you should use something like Saphira's Gradient navigation.

Author:

Marcin Balicki : Cooper Union - Mechanical Engineering

Date:

March 2004

Bug:

Doesn't work as expected...used Saphira instead.

Definition at line 22 of file SbActionGoto.h.

Public Member Functions

- SbActionGoto (const char *name="goto", double closeDist=200)
  Constructor.
- virtual ~SbActionGoto ()
  Destructor.
- bool haveAchievedGoal (void)
  Sees if the goal has been achieved.
- void cancelGoal (void)
  Cancels the goal the robot has.
- void setGoal (ArPose goal)
  Sets a new goal and sets the action to go there.
- ArPose getGoal (void)
  Gets the goal the action has.
- void setCloseDist (double closeDist)
  Set the distance which is close enough to the goal (mm);.

- double getCloseDist (void)
  Gets the distance which is close enough to the goal (mm).
- void setSpeed (double speed)
  Sets the speed the action will travel to the goal at (mm/sec).
- double getSpeed (void)
  Gets the speed the action will travel to the goal at (mm/sec).
- void go (ArPose)
  this should be used by the used to set new goal
- virtual ArActionDesired * fire (ArActionDesired currentDesired)
  gets fired every cycle.
- virtual ArActionDesired * getDesired (void)
  accesor for action system
- virtual AREXPORT bool isActive (void) const
  Finds out whether the action is active or not.
- virtual AREXPORT void activate (void)
  Activate the action.
- virtual AREXPORT void deactivate (void)
  Deactivate the action.
- virtual AREXPORT void setRobot (ArRobot *robot)
  Sets the robot this action is driving.
- virtual AREXPORT int getNumArgs (void) const
  Find the number of arguments this action takes.
- virtual AREXPORT const ArArg * getArg (int number) const
  Gets the numbered argument.
- virtual AREXPORT ArArg * getArg (int number)
  Gets the numbered argument.
- virtual AREXPORT const char * getName (void) const
  Gets the name of the action.
- virtual AREXPORT const char * getDescription (void) const
  Gets the long description of the action.
- virtual AREXPORT void log (bool verbose=true) const
  ArLog::log s the actions stats.

Protected Types

- enum State { **STATE_NO_GOAL**, **STATE_ACHIEVED_GOAL**, **STATE_GOING_TO_GOAL** }
  enumerates the current state of this action

Protected Member Functions

- AREXPORT void setNextArgument (ArArg const &arg)
  Sets the argument type for the next argument (only use in constructor).

Protected Attributes

- ArPose myGoal
  current goal

- double myCloseDist
  allow distance within the goal as success
- double mySpeed
  desired speed
- double myTurnAmount
  increment amount to turn to achieve heading
- double myOldDist
  previous goal distance
- ArActionDesired myDesired
  internal action
- ArPose myOldGoal
  old position
- State myState
  current state of the goto function
- ArRobot * **myRobot**
- bool **myIsActive**
- int **myNumArgs**
- std::map< int, ArArg > **myArgumentMap**
- std::string **myName**
- std::string **myDescription**

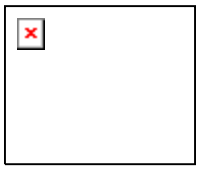
The documentation for this class was generated from the following files:

- SbActionGoto.h
- SbActionGoto.cpp

SbActionML Class Reference

SbActionML#include <SbActionML.h>

Inheritance diagram for SbActionML:



Detailed Description

Manages Aria and custom (Sb) actions that combine and control robot behavior.

SbActionML listens for queries to control actions and manages their initialization. The more important actions are the SbActionMove and SbActionRotate which allow the user to combine the popular functions of move and rotate to be used in combination with action limiting functions such as ArActionAvoidFront.

Author:

Marcin Balicki : Cooper Union - Mechanical Engineering

Date:

March 2004

Warning:

Direct Motion Commands. Be aware that direct or motion command may conflict with controls from
Actions or other upper level process and lead to unexpected consequences.
Robot->clearDirectMotion() will cancel the direct Motion so action can
get the control back there is some other way to automatically do it using
robot->setDirectMotionPrecedenceTime();
may block actions forever if too high
robot->getDirectMotionPrecedencetTime();
be careful direct motion commands may prevent actions forever
may need to set robot->clearDirectMotion(); maybe I should use actions instead of direct motion commmands.
because direct motions do not work with actions well.

Note:

You can use the sendMsgToHandler to fire messages back to the handler.

Bug:

None

Definition at line 39 of file SbActionML.h.

Public Member Functions

- SbActionML (void)
  Constructor.
- ~SbActionML (void)
  Destructor.

- virtual void fire (const SbMessage &msg)
  implements the SbMsgListener pure virtual function.
- vector< string > getMsgTypes (void)
  returns a pointer to the vector containg types of messages that this listener wants to listen to
- virtual void sendMsgToHandler (const SbMessage &msg)
  call back for sending messages back to the message handler.
- virtual void sendMsgToHandler (const string &str)
  call back for sending messages back to the message handler.

Protected Member Functions

- bool checkForRobot (void)
  checks if the robot exists and is connected
- void init (void)
  initialize the actions
- virtual void addMsgType (const string m)
  adds the message type that this listener should listen to.

Protected Attributes

- ArRobot * robot
  internal robot pointer
- string type
  message type.
- SbActionGoto goTo
  Non Aria goto action.
- SbActionRotate rotate
  Non Aria rotate action.
- SbActionMove move
  Non action move action.
- ArActionAvoidFront avoidFront
  front avoid
- ArActionAvoidFront avoidSide
  side avoid
- ArActionBumpers bumpers
  bumper action
- ArActionConstantVelocity constVel
  keeps velocity constant
- ArActionLimiterBackwards limitBackwards
  limits objects in the back
- ArActionLimiterForwards limitForwards
  limits objects in the front
- ArActionLimiterTableSensor tableLimiter
  table sensor
- ArActionStallRecover recover
  recovers from a stall
- ArActionStop stop
  stops all actions

- ArActionTurn turn
  Aria's turn action.
- ArActionGroupStop * gStop
  groups
- ArActionGroupWander * gWander
  groups
- vector< string > msgTypeVector
  vector containing all the message types that this listener is interested in registering with the handler.

Constructor & Destructor Documentation

SbActionML::~SbActionML (void)

Destructor.

if (!(gWander==NULL && gStop==NULL)){

Definition at line 18 of file SbActionML.cpp.

References bumpers, goTo, move, recover, robot, rotate, tableLimiter, and turn.

Member Function Documentation

void SbActionML::fire (const SbMessage & *msg*) [virtual]

implements the SbMsgListener pure virtual function.

Parameters:

&msg SbMessage which caused the listener to fire.

Implements SbMsgListener.

Definition at line 96 of file SbActionML.cpp.

References ArAction::activate(), bumpers, SbActionGoto::cancelGoal(), SbActionMove::cancelGoal(), checkForRobot(), fire(), SbMessage::getArgAsDbl(), SbMessage::getArgAsInt(), SbMessage::getAsStr(), SbMessage::getType(), SbActionMove::go(), SbActionRotate::go(), goTo, move, robot, rotate, and type.

Referenced by fire().

vector<string > SbMsgListener::getMsgTypes (void) [inherited]

returns a pointer to the vector containg types of messages that this listener wants to listen to

Returns:

the pointer to vector of message types

Definition at line 38 of file SbMsgListener.h.

References SbMsgListener::msgTypeVector.

Referenced by SbMsgHandlerSingleton::addListener().

void SbMsgListener::sendMsgToHandler (const SbMessage & *msg*) [virtual, inherited]

call back for sending messages back to the message handler.

Parameters:

&msg SbMessage that should be sent to the handler

Definition at line 12 of file SbMsgListener.cpp.

References SbMsgHandlerSingleton::getInstance(), and SbMsgListener::sendMsgToHandler().

Referenced by SbMsgListener::sendMsgToHandler().

void SbMsgListener::sendMsgToHandler (const string & *str*) [virtual, inherited]

call back for sending messages back to the message handler.

Parameters:

&str String that should be sent to the handler

Definition at line 17 of file SbMsgListener.cpp.

References SbMsgHandlerSingleton::getInstance(), and SbMsgListener::sendMsgToHandler().

virtual void SbMsgListener::addMsgType (const string *m*) [protected, virtual, inherited]

adds the message type that this listener should listen to.

msgHandler will read the vector containing these and register the types for messaging

Parameters:

&m string as a message type ie "SEND"

Definition at line 56 of file SbMsgListener.h.

References SbMsgListener::addMsgType(), and SbMsgListener::msgTypeVector.

Referenced by SbMsgListener::addMsgType().

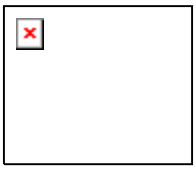
The documentation for this class was generated from the following files:

- SbActionML.h
- SbActionML.cpp

SbActionMove Class Reference

SbActionMove#include <SbActionMove.h>

Inheritance diagram for SbActionMove:



Detailed Description

Action that moves the robot forward and backward.

SbActionMove drives straight by a given distance. The action stops when it gets closeDist away. You can give it a new goal with setGoal(), cancel its movement with cancelGoal(), and see if it got there with haveAchievedGoal(). This doesn't avoid obstacles, you could have an avoid routine at a higher priority to avoid on the way there. This action was created to be used with other actions to replace direct motion MOVE command. It basically made so that you can just have a ton of limiters of different kinds and types while using fundamental control of the robot.

Author:

Marcin Balicki : Cooper Union - Mechanical Engineering

Date:

March 2004

Bug:

None

Definition at line 22 of file SbActionMove.h.

Public Member Functions

- SbActionMove (const char *name="goto", double closeDist=200)
  Constructor.
- bool haveAchievedGoal (void)
  Sees if the goal has been achieved.
- void cancelGoal (void)
  Cancels the goal the robot has.
- void setGoal (ArPose goal)
  Sets a new goal and sets the action to go there.
- ArPose getGoal (void)
  Gets the goal the action has.
- void setCloseDist (double closeDist)
  Set the distance which is close enough to the goal (mm);.
- double getCloseDist (void)
  Gets the distance which is close enough to the goal (mm).
- void setSpeed (double speed)

Sets the speed the action will travel to the goal at (mm/sec).

- double getSpeed (void)
  Gets the speed the action will travel to the goal at (mm/sec).
- void go (double dist)
  go
- virtual ArActionDesired * fire (ArActionDesired currentDesired)
  implementation of base fire function
- virtual ArActionDesired * getDesired (void)
  implementation for the base getDesired action
- virtual AREXPORT bool isActive (void) const
  Finds out whether the action is active or not.
- virtual AREXPORT void activate (void)
  Activate the action.
- virtual AREXPORT void deactivate (void)
  Deactivate the action.
- virtual AREXPORT void setRobot (ArRobot *robot)
  Sets the robot this action is driving.
- virtual AREXPORT int getNumArgs (void) const
  Find the number of arguments this action takes.
- virtual AREXPORT const ArArg * getArg (int number) const
  Gets the numbered argument.
- virtual AREXPORT ArArg * getArg (int number)
  Gets the numbered argument.
- virtual AREXPORT const char * getName (void) const
  Gets the name of the action.
- virtual AREXPORT const char * getDescription (void) const
  Gets the long description of the action.
- virtual AREXPORT void log (bool verbose=true) const
  ArLog::log s the actions stats.

Protected Types

- enum State { **STATE_NO_GOAL**, **STATE_ACHIEVED_GOAL**, **STATE_GOING_TO_GOAL** }
  enumerates the current state of this action

Protected Member Functions

- AREXPORT void setNextArgument (ArArg const &arg)
  Sets the argument type for the next argument (only use in constructor).

Protected Attributes

- double myCloseDist
  allow distance within the goal as success
- double mySpeed
  desired speed
- double myDirectionToTurn
  incrememnt amount to turn to achieve heading

- double myCurDir
  the current direction
- double myOldDist
  the last distance to goal.
- bool myTurnedBack
  turn back flag
- ArActionDesired myDesired
  internal action
- ArPose myOldGoal
  old position
- ArPose origGoal
  requested goal
- ArPose myGoal
  goal
- State myState
  instance of the state enumerator
- ArRobot * **myRobot**
- bool **myIsActive**
- int **myNumArgs**
- std::map< int, ArArg > **myArgumentMap**
- std::string **myName**
- std::string **myDescription**

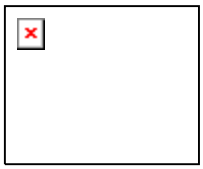
The documentation for this class was generated from the following files:

- SbActionMove.h
- SbActionMove.cpp

SbActionRotate Class Reference

SbActionRotate#include <SbActionRotate.h>

Inheritance diagram for SbActionRotate:



Detailed Description

Action that rotates the robot.

SbActionRotate was required because the direct motion ROTATE command rotates in the direction of the smallest possible motion to achieve the required heading; specifically in the case the angle is greater then 180 or less than -180. This prevented the user to rotate the robot in one direction continuesly until goal is achieved. , ie 750 degrees. This action is basically made so that you can just have a ton of limiters of different kinds and types while using fundamental control of the robot.

Author:

Marcin Balicki : Cooper Union - Mechanical Engineering

Date:

March 2004

Bug:

None

Definition at line 19 of file SbActionRotate.h.

Public Member Functions

- SbActionRotate (const char *name="turn", double turnAmount=15)
  Constructor.
- virtual ~SbActionRotate ()
  Destructor.
- virtual ArActionDesired * fire (ArActionDesired currentDesired)
  implementation of base fire function, gets fired every cycle.
- virtual ArActionDesired * getDesired (void)
  accesor for action system
- void go (double)
  this function is what users uses to fire this action. Arg: Degrees.
- virtual AREXPORT bool isActive (void) const
  Finds out whether the action is active or not.
- virtual AREXPORT void activate (void)
  Activate the action.
- virtual AREXPORT void deactivate (void)

Deactivate the action.

- virtual AREXPORT void setRobot (ArRobot *robot)
  Sets the robot this action is driving.
- virtual AREXPORT int getNumArgs (void) const
  Find the number of arguments this action takes.
- virtual AREXPORT const ArArg * getArg (int number) const
  Gets the numbered argument.
- virtual AREXPORT ArArg * getArg (int number)
  Gets the numbered argument.
- virtual AREXPORT const char * getName (void) const
  Gets the name of the action.
- virtual AREXPORT const char * getDescription (void) const
  Gets the long description of the action.
- virtual AREXPORT void log (bool verbose=true) const
  ArLog::log s the actions stats.

Protected Member Functions

- AREXPORT void setNextArgument (ArArg const &arg)
  Sets the argument type for the next argument (only use in constructor).

Protected Attributes

- double myTurnAmount
  the incremental rotation amount
- ArActionDesired myDesired
  internal
- double goalAngle
  current angle to goal
- double oldAngle
  older angle to goal.
- ArRobot * **myRobot**
- bool **myIsActive**
- int **myNumArgs**
- std::map< int, ArArg > **myArgumentMap**
- std::string **myName**
- std::string **myDescription**

The documentation for this class was generated from the following files:

- SbActionRotate.h
- SbActionRotate.cpp

SbCameraSONYML Class Reference

SbCameraSONYML#include <SbCameraSONYML.h>

Inheritance diagram for SbCameraSONYML:

Detailed Description

Manages Sony EVI-D30 PTZ (Pan Tilt Zoom) camera.

This class controls functions of the camera attached to the serial port on the robot. It controls all the functions available for the camera. (Pan/Tilt/Zoom etc).

Author:

Marcin Balicki : Cooper Union - Mechanical Engineering

Date:

March 2004

Warning:

Uses SbMsgListener's SbMsgListener::sendMsgToHandler()

Todo:

Would be nice to figure out how to test which type of camera is connected to this robot and then create the appropriate control class.

Bug:

None

Definition at line 24 of file SbCameraSONYML.h.

Public Member Functions

- SbCameraSONYML (void)
  Constructor.
- ~SbCameraSONYML (void)
  Destructor.
- virtual void fire (const SbMessage &msg)
  implements the SbMsgListener pure virtual function.
- vector< string > getMsgTypes (void)
  returns a pointer to the vector containg types of messages that this listener wants to listen to
- virtual void sendMsgToHandler (const SbMessage &msg)
  call back for sending messages back to the message handler.
- virtual void sendMsgToHandler (const string &str)
  call back for sending messages back to the message handler.

Protected Member Functions

- bool init (void)
  set the max / min variables.
- bool checkForRobot (void)
  check if the robot exists and is connected.
- void tilt (double)
  tilts the camera +- 100.0
- void pan (double)
  pans the camera + - 100.0
- void panRel (double)
  pans relateviely to current position + - 100.0
- void tiltRel (double)
  tilts relateviely to current position + - 100.0
- void zoom (double)
  zoom position + - 100.0
- void zoomRel (double)
  zoom relatively to current position + - 100.0
- double getState (int f)
  returns the current value of Pan/Tilt/Zoom pan(0) tilt(1) zoom(2)
- virtual void addMsgType (const string m)
  adds the message type that this listener should listen to.

Protected Attributes

- ArRobot * robot
  internal robot pointer
- string type
  message type
- ArSonyPTZ * camera
  SONY camera control class.
- int maxPanPos
  maximum Pan position
- int maxTiltPos
  maximum Tilt position
- int minPanPos
  minimum Pan position
- int minTiltPos
  minimum Tilt position
- int minZoom
  minimum Zoom value
- int maxZoom
  maximum Zoom value
- int minPanSlew
  minimum Pan speed
- int maxPanSlew
  maximum Pan speed

- int minTiltSlew
  minimum Tilt speed
- int maxTiltSlew
  maximum Titl speed
- vector< string > msgTypeVector
  vector containing all the message types that this listener is interested in registering with the handler.

Member Function Documentation

void SbCameraSONYML::fire (const SbMessage & *msg*) [virtual]

implements the SbMsgListener pure virtual function.

Parameters:

msg SbMessage which caused the listener to fire.

Implements SbMsgListener.

Definition at line 193 of file SbCameraSONYML.cpp.

References camera, checkForRobot(), fire(), SbMessage::getArgAsDbl(), SbMessage::getArgAsInt(), SbMessage::getArgAsStr(), SbMessage::getAsStr(), SbMessage::getType(), pan(), panRel(), robot, tilt(), tiltRel(), type, zoom(), and zoomRel().

Referenced by fire().

vector<string > SbMsgListener::getMsgTypes (void) [inherited]

returns a pointer to the vector containg types of messages that this listener wants to listen to

Returns:

the pointer to vector of message types

Definition at line 38 of file SbMsgListener.h.

References SbMsgListener::msgTypeVector.

Referenced by SbMsgHandlerSingleton::addListener().

void SbMsgListener::sendMsgToHandler (const SbMessage & *msg*) [virtual, inherited]

call back for sending messages back to the message handler.

Parameters:

&msg SbMessage that should be sent to the handler

Definition at line 12 of file SbMsgListener.cpp.

References SbMsgHandlerSingleton::getInstance(), and SbMsgListener::sendMsgToHandler().

Referenced by SbMsgListener::sendMsgToHandler().

void SbMsgListener::sendMsgToHandler (const string & *str*) [virtual, inherited]

call back for sending messages back to the message handler.

Parameters:

&str String that should be sent to the handler

Definition at line 17 of file SbMsgListener.cpp.

References SbMsgHandlerSingleton::getInstance(), and SbMsgListener::sendMsgToHandler().

virtual void SbMsgListener::addMsgType (const string *m*) [protected, virtual, inherited]

adds the message type that this listener should listen to.

msgHandler will read the vector containing these and register the types for messaging

Parameters:

&m string as a message type ie "SEND"

Definition at line 56 of file SbMsgListener.h.

References SbMsgListener::addMsgType(), and SbMsgListener::msgTypeVector.

Referenced by SbMsgListener::addMsgType().

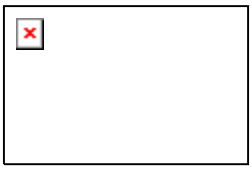
The documentation for this class was generated from the following files:

- SbCameraSONYML.h
- SbCameraSONYML.cpp

SbCameraVCC4ML Class Reference

SbCameraVCC4ML#include <SbCameraVCC4ML.h>

Inheritance diagram for SbCameraVCC4ML:



Detailed Description

Manages Cannon VC-C4 PTZ (Pan Tilt Zoom) camera.

This class controls functions of the camera attached to the serial port on the robot. It controls all the functions available for the camera. (Pan/Tilt/Zoom etc). Cannon needs to be initilized proir to sending commands to it.

Author:

Marcin Balicki : Cooper Union - Mechanical Engineering

Date:

March 2004

Warning:

Uses SbMsgListener's SbMsgListener::sendMsgToHandler()

Todo:

Would be nice to figure out how to test which type of camera is connected to this robot and then create the appropriate control class.

Bug:

None

Definition at line 24 of file SbCameraVCC4ML.h.

Public Member Functions

- SbCameraVCC4ML (void)
  Constructor.
- ~SbCameraVCC4ML (void)
  Destructor.
- virtual void fire (const SbMessage &msg)
  implements the SbMsgListener pure virtual function.
- vector< string > getMsgTypes (void)
  returns a pointer to the vector containg types of messages that this listener wants to listen to
- virtual void sendMsgToHandler (const SbMessage &msg)
  call back for sending messages back to the message handler.
- virtual void sendMsgToHandler (const string &str)
  call back for sending messages back to the message handler.

Protected Member Functions

- bool init (void)
  init the camera and set the max min variables.
- bool checkForRobot (void)
  check if the robot exists and is connected.
- void tilt (double)
  tilts the camera +- 100.0
- void pan (double)
  pans the camera + - 100.0
- void panRel (double)
  pans relatively to current position + - 100.0
- void tiltRel (double)
  tilts relatively to current position + - 100.0
- void zoom (double)
  zoom position + - 100.0
- void zoomRel (double)
  zooms relatively to current position + - 100.0
- void panSlew (double)
  pan speed
- void tiltSlew (double)
  tilt speed
- double getState (int)
  returns the current value of Pan/Tilt/Zoom pan(0) tilt(1) zoom(2)
- virtual void addMsgType (const string m)
  adds the message type that this listener should listen to.

Protected Attributes

- ArRobot * robot
  internal robot pointer
- string type
  message type
- ArVCC4 * camera
  VCC4 canon camera control class.
- int maxPanPos
  maximum Pan position
- int maxTiltPos
  maximum Tilt position
- int minPanPos
  minimum Pan position
- int minTiltPos
  minimum Tilt position
- int minZoom
  minimum Zoom value
- int maxZoom
  maximum Zoom value

- int minPanSlew
  minimum Pan speed
- int maxPanSlew
  maximum Pan speed
- int minTiltSlew
  minimum Tilt speed
- int maxTiltSlew
  maximum Tilt speed
- vector< string > msgTypeVector
  vector containing all the message types that this listener is interested in registering with the handler.

Member Function Documentation

void SbCameraVCC4ML::fire (const SbMessage & *msg*) [virtual]

implements the SbMsgListener pure virtual function.

Parameters:

msg SbMessage which caused the listener to fire.

Implements SbMsgListener.

Definition at line 238 of file SbCameraVCC4ML.cpp.

References camera, checkForRobot(), fire(), SbMessage::getArgAsDbl(), SbMessage::getArgAsStr(), SbMessage::getAsStr(), SbMessage::getType(), init(), pan(), panRel(), panSlew(), robot, tilt(), tiltRel(), tiltSlew(), type, zoom(), and zoomRel().

Referenced by fire().

vector<string > SbMsgListener::getMsgTypes (void) [inherited]

returns a pointer to the vector containg types of messages that this listener wants to listen to

Returns:

the pointer to vector of message types

Definition at line 38 of file SbMsgListener.h.

References SbMsgListener::msgTypeVector.

Referenced by SbMsgHandlerSingleton::addListener().

void SbMsgListener::sendMsgToHandler (const SbMessage & *msg*) [virtual, inherited]

call back for sending messages back to the message handler.

Parameters:

&msg SbMessage that should be sent to the handler

Definition at line 12 of file SbMsgListener.cpp.

References SbMsgHandlerSingleton::getInstance(), and SbMsgListener::sendMsgToHandler().

Referenced by SbMsgListener::sendMsgToHandler().

void SbMsgListener::sendMsgToHandler (const string & *str*) [virtual, inherited]

call back for sending messages back to the message handler.

Parameters:

&str String that should be sent to the handler

Definition at line 17 of file SbMsgListener.cpp.

References SbMsgHandlerSingleton::getInstance(), and SbMsgListener::sendMsgToHandler().

virtual void SbMsgListener::addMsgType (const string *m*) [protected, virtual, inherited]

adds the message type that this listener should listen to.

msgHandler will read the vector containing these and register the types for messaging

Parameters:

&m string as a message type ie "SEND"

Definition at line 56 of file SbMsgListener.h.

References SbMsgListener::addMsgType(), and SbMsgListener::msgTypeVector.

Referenced by SbMsgListener::addMsgType().

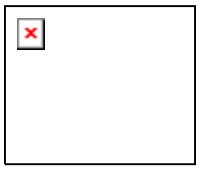
The documentation for this class was generated from the following files:

- SbCameraVCC4ML.h
- SbCameraVCC4ML.cpp

SbExampleML Class Reference

SbExampleML#include <SbExampleML.h>

Inheritance diagram for SbExampleML:



Detailed Description

Example implementation of a class that inherits SbMsgListener.

A template example, just add your code in the fire() function. You can use the sendMsgToHandler() to fire messages back to the handler.

Author:

Marcin Balicki : Cooper Union - Mechanical Engineering

Date:

March 2004

Warning:

Uses SbMsgListener's SbMsgListener::sendMsgToHandler()

Note:

You can use the sendMsgToHandler to fire messages back to the handler.

Bug:

None

Definition at line 21 of file SbExampleML.h.

Public Member Functions

- SbExampleML (void)
  Constructor.
- ~SbExampleML (void)
  Destructor.
- string toString (double d)
  converts double to a string.
- virtual void fire (const SbMessage &msg)
  implements the SbMsgListener pure virtual function.
- vector< string > getMsgTypes (void)
  returns a pointer to the vector containg types of messages that this listener wants to listen to
- virtual void sendMsgToHandler (const SbMessage &msg)
  call back for sending messages back to the message handler.
- virtual void sendMsgToHandler (const string &str)
  call back for sending messages back to the message handler.

Protected Member Functions

- bool checkForRobot (void)
  checks if the robot (ARIA instance) exists and is connected
- virtual void addMsgType (const string m)
  adds the message type that this listener should listen to.

Protected Attributes

- ArRobot * robot
  robot pointer
- string type
  type of message
- SbMessage msgExample
  internal message
- ostringstream oss
  stream instance used to convert double to string
- vector< string > msgTypeVector
  vector containing all the message types that this listener is interested in registering with the handler.

Member Function Documentation

void SbExampleML::fire (const SbMessage & *msg*) [virtual]

implements the SbMsgListener pure virtual function.

Parameters:

&msg SbMessage which caused the listener to fire.

Implements SbMsgListener.

Definition at line 46 of file SbExampleML.cpp.

References SbMessage::addArgument(), checkForRobot(), SbMessage::clear(), fire(), SbMessage::getAsStr(), SbMessage::getType(), msgExample, robot, SbMessage::setType(), and toString().

Referenced by fire().

vector<string > SbMsgListener::getMsgTypes (void) [inherited]

returns a pointer to the vector containg types of messages that this listener wants to listen to

Returns:

the pointer to vector of message types

Definition at line 38 of file SbMsgListener.h.

References SbMsgListener::msgTypeVector.

Referenced by SbMsgHandlerSingleton::addListener().

void SbMsgListener::sendMsgToHandler (const SbMessage & *msg*) [virtual, inherited]

call back for sending messages back to the message handler.

Parameters:

&msg SbMessage that should be sent to the handler

Definition at line 12 of file SbMsgListener.cpp.

References SbMsgHandlerSingleton::getInstance(), and SbMsgListener::sendMsgToHandler().

Referenced by SbMsgListener::sendMsgToHandler().

void SbMsgListener::sendMsgToHandler (const string & *str*) [virtual, inherited]

call back for sending messages back to the message handler.

Parameters:

&str String that should be sent to the handler

Definition at line 17 of file SbMsgListener.cpp.

References SbMsgHandlerSingleton::getInstance(), and SbMsgListener::sendMsgToHandler().

virtual void SbMsgListener::addMsgType (const string *m*) [protected, virtual, inherited]

adds the message type that this listener should listen to.

msgHandler will read the vector containing these and register the types for messaging

Parameters:

&m string as a message type ie "SEND"

Definition at line 56 of file SbMsgListener.h.

References SbMsgListener::addMsgType(), and SbMsgListener::msgTypeVector.

Referenced by SbMsgListener::addMsgType().

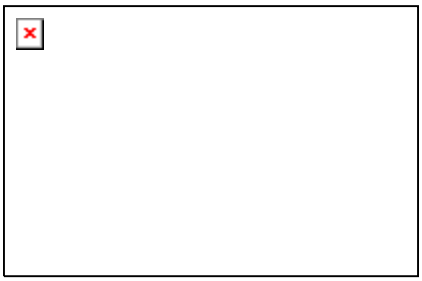
The documentation for this class was generated from the following files:

- SbExampleML.h
- SbExampleML.cpp

SbExampleMLT Class Reference

SbExampleMLT#include <SbExampleMLT.h>

Inheritance diagram for SbExampleMLT:



Detailed Description

Example implementation that inherits SbMsgListener and is threaded.

This class inherits Aria's threaded ArSyncTask. It displays a possible way to use a separate thread to do something (calculation, control another process, etc) while the robot is running.

Author:

Marcin Balicki : Cooper Union - Mechanical Engineering

Date:

March 2004

Warning:

Be careful with locking and unlocking.
This class is threaded and therefore should not use sendMsgToHandler to fire messages back to the handler because handler is not thread safe unless used from called from managerT

Note:

You can't use the sendMsgToHandler to fire messages back to the handler.

Todo:

Instead of making SbMsgHandlerSingleton as the singleton, the manager class should be one, and the reference to it should be possessed by all SbMsgListeners. However ,this relations would be forced since listener is far away in function from manager (listener->handler->manager). another way is to add lock/unlock feature to the handler.

Bug:

None

Definition at line 33 of file SbExampleMLT.h.

Public Types

- typedef pthread_t **ThreadType**
- typedef std::map< ThreadType, ArThread * > **MapType**
- enum Status { STATUS_FAILED = 1, STATUS_NORESOURCE, STATUS_NO_SUCH_THREAD, STATUS_INVALID, STATUS_JOIN_SELF, STATUS_ALREADY_DETATCHED }

Public Member Functions

- string toString (double d)
  converts double to a string.
- virtual void fire (const SbMessage &msg)
  implements the SbMsqListener pure virtual function.
- virtual void * runThread (void *arg)
  implements the ArSyncTask main thread function
- virtual AREXPORT void run (void)
  Run in this thread.
- virtual AREXPORT void runAsync (void)
  Run in its own thread.
- virtual AREXPORT void stopRunning (void)
  Stop the thread.
- virtual AREXPORT int create (bool joinable=true, bool lowerPriority=true)
  Create the task and start it going.
- virtual AREXPORT void * runInThisThread (void *arg=0)
  Run the code of the task synchronously.
- virtual AREXPORT int join (void **ret=NULL)
  Join on the thread.
- virtual AREXPORT int detach (void)
  Detach the thread so it cant be joined.
- virtual AREXPORT void cancel (void)
  Cancel the thread.
- virtual AREXPORT bool getRunning (void) const
  Get the running status of the thread.
- virtual AREXPORT bool getRunningWithLock (void)
  Get the running status of the thread, locking around the variable.
- virtual AREXPORT bool getJoinable (void) const
  Get the joinable status of the thread.
- virtual AREXPORT const ThreadType * getThread (void) const
  Get the underlying thread type.
- virtual AREXPORT ArFunctor * getFunc (void) const
  Get the functor that the thread runs.
- virtual AREXPORT void setRunning (bool running)
  Set the running value on the thread.
- AREXPORT int lock (void)
  Lock the thread instance.
- AREXPORT int tryLock (void)
  Try to lock the thread instance without blocking.
- AREXPORT int unlock (void)
  Unlock the thread instance.
- bool getBlockAllSignals (void)
  Do we block all process signals at startup?
- vector< string > getMsgTypes (void)
  returns a pointer to the vector containg types of messages that this listener wants to listen to

- virtual void sendMsgToHandler (const SbMessage &msg)
  call back for sending messages back to the message handler.
- virtual void sendMsgToHandler (const string &str)
  call back for sending messages back to the message handler.

Static Public Member Functions

- AREXPORT void init (void)
  Initialize the internal book keeping structures.
- AREXPORT ArThread * self (void)
  Returns the instance of your own thread.
- AREXPORT void stopAll ()
  Stop all threads.
- AREXPORT void cancelAll (void)
  Cancel all threads.
- AREXPORT void joinAll (void)
  Join on all threads.
- AREXPORT void yieldProcessor (void)
  Yield the processor to another thread.

Protected Member Functions

- bool checkForRobot (void)
  checks if the robot (ARIA instance) exists and is connected
- virtual AREXPORT int **doJoin** (void **ret=NULL)
- virtual void addMsgType (const string m)
  adds the message type that this listener should listen to.

Protected Attributes

- ArRobot * robot
  robot pointer
- string type
  type of message
- SbMessage msgExample
  internal most current message
- ostringstream oss
  stream instance used to convert double to string
- bool newMsg
  new message flag
- ArMutex **myMutex**
- bool myRunning
  State variable to denote when the thread should continue or exit.
- bool **myJoinable**
- bool **myBlockAllSignals**
- ThreadType **myThread**
- ArStrMap **myStrMap**
- vector< string > msgTypeVector
  vector containing all the message types that this listener is interested in registering with the handler.

Static Protected Attributes

- ArMutex **ourThreadsMutex**
- MapType **ourThreads**

Member Enumeration Documentation

enum ArThread::Status [inherited]

Enumeration values:

STATUS_FAILED Failed to create the thread.
STATUS_NORESOURCE Not enough system resources to create the thread.
STATUS_NO_SUCH_THREAD The thread can no longer be found.
STATUS_INVALID Thread is detached or another thread is joining on it.
STATUS_JOIN_SELF Thread is your own thread. Can't join on self.
STATUS_ALREADY_DETATCHED Thread is already detatched.

Definition at line 63 of file ArThread.h.

Member Function Documentation

void SbExampleMLT::fire (const SbMessage & *msg*) [virtual]

implements the SbMsgListener pure virtual function.

Parameters:

&msg SbMessage which caused the listener to fire.

Implements SbMsgListener.

Definition at line 50 of file SbExampleMLT.cpp.

References checkForRobot(), fire(), SbMessage::getAsStr(), SbMessage::getType(), ArThread::lock(), msgExample, newMsg, and ArThread::unlock().

Referenced by fire().

void * SbExampleMLT::runThread (void * *arg*) [virtual]

implements the ArSyncTask main thread function

Parameters:

**arg* that does nothing.

Implements ArASyncTask.

Definition at line 72 of file SbExampleMLT.cpp.

References SbMessage::getAsStr(), ArThread::lock(), msgExample, newMsg, runThread(), and ArThread::unlock().

Referenced by runThread().

vector<string > SbMsgListener::getMsgTypes (void) [inherited]

returns a pointer to the vector containg types of messages that this listener wants to listen to

Returns:

the pointer to vector of message types

Definition at line 38 of file SbMsgListener.h.

References SbMsgListener::msgTypeVector.

Referenced by SbMsgHandlerSingleton::addListener().

void SbMsgListener::sendMsgToHandler (const SbMessage & *msg*) [virtual, inherited]

call back for sending messages back to the message handler.

Parameters:

&msg SbMessage that should be sent to the handler

Definition at line 12 of file SbMsgListener.cpp.

References SbMsgHandlerSingleton::getInstance(), and SbMsgListener::sendMsgToHandler().

Referenced by SbMsgListener::sendMsgToHandler().

void SbMsgListener::sendMsgToHandler (const string & *str*) [virtual, inherited]

call back for sending messages back to the message handler.

Parameters:

&str String that should be sent to the handler

Definition at line 17 of file SbMsgListener.cpp.

References SbMsgHandlerSingleton::getInstance(), and SbMsgListener::sendMsgToHandler().

virtual void SbMsgListener::addMsgType (const string *m*) [protected, virtual, inherited]

adds the message type that this listener should listen to.

msgHandler will read the vector containing these and register the types for messaging

Parameters:

&m string as a message type ie "SEND"

Definition at line 56 of file SbMsgListener.h.

References SbMsgListener::addMsgType(), and SbMsgListener::msgTypeVector.

Referenced by SbMsgListener::addMsgType().

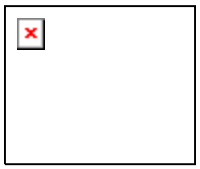
The documentation for this class was generated from the following files:

- SbExampleMLT.h
- SbExampleMLT.cpp

SbGripperML Class Reference

SbGripperML#include <SbGripperML.h>

Inheritance diagram for SbGripperML:



Detailed Description

Manages Gripper functions.

SbGripperML controls the gripper (Performance PeopleBot) if present. It controls the lift, gripper arms and returns the state of the gripper (open,close, liftmax, etc)

Author:

Marcin Balicki : Cooper Union - Mechanical Engineering

Date:

March 2004

Warning:

Uses SbMsgListener's SbMsgListener::sendMsgToHandler()

Note:

You can use the sendMsgToHandler to fire messages back to the handler.

Bug:

None

Definition at line 21 of file SbGripperML.h.

Public Member Functions

- SbGripperML (void)
  Constructor.
- ~SbGripperML (void)
  Destructor.
- virtual void fire (const SbMessage &msg)
  implements the SbMsgListener pure virtual function.
- vector< string > getMsgTypes (void)
  returns a pointer to the vector containg types of messages that this listener wants to listen to
- virtual void sendMsgToHandler (const SbMessage &msg)
  call back for sending messages back to the message handler.
- virtual void sendMsgToHandler (const string &str)
  call back for sending messages back to the message handler.

Protected Member Functions

- virtual void addMsgType (const string m)

adds the message type that this listener should listen to.

Protected Attributes

- vector< string > msgTypeVector
  vector containing all the message types that this listener is interested in registering with the handler.

Private Member Functions

- bool checkForRobot (void)
  checks if the robot exists and is connected

Private Attributes

- ArRobot * robot
  internal robot pointer
- string type
  message type
- ArGripper * gripper
  pointer to the gripper control class

Member Function Documentation

void SbGripperML::fire (const SbMessage & *msg*) [virtual]

implements the SbMsgListener pure virtual function.

Parameters:

msg SbMessage which caused the listener to fire.

Implements SbMsgListener.

Definition at line 51 of file SbGripperML.cpp.

References SbMessage::addArgument(), checkForRobot(), fire(), SbMessage::getArgAsInt(), SbMessage::getArgAsStr(), SbMessage::getAsStr(), SbMessage::getType(), gripper, robot, and type.

Referenced by fire().

vector<string > SbMsgListener::getMsgTypes (void) [inherited]

returns a pointer to the vector containg types of messages that this listener wants to listen to

Returns:

the pointer to vector of message types

Definition at line 38 of file SbMsgListener.h.

References SbMsgListener::msgTypeVector.

Referenced by SbMsgHandlerSingleton::addListener().

void SbMsgListener::sendMsgToHandler (const SbMessage & *msg*) [virtual, inherited]

call back for sending messages back to the message handler.

Parameters:

&msg SbMessage that should be sent to the handler

Definition at line 12 of file SbMsgListener.cpp.

References SbMsgHandlerSingleton::getInstance(), and SbMsgListener::sendMsgToHandler().

Referenced by SbMsgListener::sendMsgToHandler().

void SbMsgListener::sendMsgToHandler (const string & *str*) [virtual, inherited]

call back for sending messages back to the message handler.

Parameters:

&str String that should be sent to the handler

Definition at line 17 of file SbMsgListener.cpp.

References SbMsgHandlerSingleton::getInstance(), and SbMsgListener::sendMsgToHandler().

virtual void SbMsgListener::addMsgType (const string *m*) [protected, virtual, inherited]

adds the message type that this listener should listen to.

msgHandler will read the vector containing these and register the types for messaging

Parameters:

&m string as a message type ie "SEND"

Definition at line 56 of file SbMsgListener.h.

References SbMsgListener::addMsgType(), and SbMsgListener::msgTypeVector.

Referenced by SbMsgListener::addMsgType().

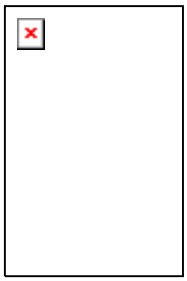
The documentation for this class was generated from the following files:

- SbGripperML.h
- SbGripperML.cpp

SbManagerT Class Reference

SbManagerT#include <SbManagerT.h>

Inheritance diagram for SbManagerT:



Detailed Description

Main manager class of RCCI (threaded).

This is a threaded class which is the managing body of the software, supervising all the high level functions of RCCI. It takes care of timed TCP server updates, checking the robot state and shutting down RCCI and registering SbMsgListeners with SbMsgHandlerSingleton.

Author:

Marcin Balicki : Cooper Union - Mechanical Engineering

Date:

March 2004

Warning:

a Singleton class needs to be destroyed by the last thread alive by calling removeInstance, which is this class.

Definition at line 22 of file SbManagerT.h.

Public Types

- typedef pthread_t **ThreadType**
- typedef std::map< ThreadType, ArThread * > **MapType**
- enum Status { STATUS_FAILED = 1, STATUS_NORESOURCE, STATUS_NO_SUCH_THREAD, STATUS_INVALID, STATUS_JOIN_SELF, STATUS_ALREADY_DETATCHED }

Public Member Functions

- SbManagerT (const string &)
  Constructor.
- ~SbManagerT ()
  Destructor deletes the objects (msgListeners) allacted for this class.
- bool init (void)
  Initilizes robot and TCP server.
- virtual void * runThread (void *arg)
  the function to run in the new thread, this just is called once, so only this gets run as soon as the object is created so this part should have the init stuff in it

- virtual AREXPORT void run (void)
  Run in this thread.
- virtual AREXPORT void runAsync (void)
  Run in its own thread.
- virtual AREXPORT void stopRunning (void)
  Stop the thread.
- virtual AREXPORT int create (bool joinable=true, bool lowerPriority=true)
  Create the task and start it going.
- virtual AREXPORT void * runInThisThread (void *arg=0)
  Run the code of the task syncronously.
- virtual AREXPORT int join (void **ret=NULL)
  Join on the thread.
- virtual AREXPORT int detach (void)
  Detatch the thread so it cant be joined.
- virtual AREXPORT void cancel (void)
  Cancel the thread.
- virtual AREXPORT bool getRunning (void) const
  Get the running status of the thread.
- virtual AREXPORT bool getRunningWithLock (void)
  Get the running status of the thread, locking around the variable.
- virtual AREXPORT bool getJoinable (void) const
  Get the joinable status of the thread.
- virtual AREXPORT const ThreadType * getThread (void) const
  Get the underlying thread type.
- virtual AREXPORT ArFunctor * getFunc (void) const
  Get the functor that the thread runs.
- virtual AREXPORT void setRunning (bool running)
  Set the running value on the thread.
- AREXPORT int lock (void)
  Lock the thread instance.
- AREXPORT int tryLock (void)
  Try to lock the thread instance without blocking.
- AREXPORT int unlock (void)
  Unlock the thread instance.
- bool getBlockAllSignals (void)
  Do we block all process signals at startup?

Static Public Member Functions

- AREXPORT ArThread * self (void)
  Returns the instance of your own thread.
- AREXPORT void stopAll ()
  Stop all threads.
- AREXPORT void cancelAll (void)
  Cancel all threads.
- AREXPORT void joinAll (void)
  Join on all threads.

- AREXPORT void yieldProcessor (void)
  Yield the processor to another thread.

Protected Types

- typedef vector< SbMsgListener * >::iterator vectIter
  iterator through the ptr vetor.

Protected Member Functions

- void registerListener (const string str, SbMsgListener *msgListenerPtr)
  registeres the listener with the msgHandler.
- void registerListener (SbMsgListener *msgListenerPtr)
  registeres the listener with the msgHandler.
- void initMsgHandler (void)
  Creates the listener objects and registers msg Types with respective listener.
- virtual AREXPORT int **doJoin** (void **ret=NULL)

Protected Attributes

- vector< SbMsgListener * > msgPtrVector
  vector of listener pointers that are registered with handler.
- string port
  port for tcp server
- SbMsgHandlerSingleton * msgHandler
  instance pointer to Singleton msgHandler
- SbServerML * tcpServer
  instance pointer to the TCP comm server
- SbMessage msg
  internal msg
- SbRobotML * sbRbPtr
  instance pointer to the robot control class.
- ArMutex **myMutex**
- bool myRunning
  State variable to denote when the thread should continue or exit.
- bool **myJoinable**
- bool **myBlockAllSignals**
- ThreadType **myThread**
- ArStrMap **myStrMap**

Static Protected Attributes

- ArMutex **ourThreadsMutex**
- MapType **ourThreads**

Member Enumeration Documentation

enum ArThread::Status [inherited]

Enumeration values:

STATUS_FAILED Failed to create the thread.
STATUS_NORESOURCE Not enough system resources to create the thread.
STATUS_NO_SUCH_THREAD The thread can no longer be found.
STATUS_INVALID Thread is detached or another thread is joining on it.
STATUS_JOIN_SELF Thread is your own thread. Can't join on self.
STATUS_ALREADY_DETATCHED Thread is already detatched.

Definition at line 63 of file ArThread.h.

Member Function Documentation

bool SbManagerT::init (void)

Initilizes robot and TCP server.

failed to start the server.

Reimplemented from ArThread.

Definition at line 70 of file SbManagerT.cpp.

References initMsgHandler(), msgPtrVector, SbServerML::open(), port, sbRbPtr, and tcpServer.

void SbManagerT::registerListener (const string *str*, SbMsgListener * *msgListenerPtr*) [protected]

registeres the listener with the msgHandler.

also registeres the new pointer with the local memory manager which uses the msgPtrVector to delete object after this class is destr.

Definition at line 32 of file SbManagerT.cpp.

References SbMsgHandlerSingleton::addListener(), msgHandler, msgPtrVector, and registerListener().

Referenced by initMsgHandler(), and registerListener().

void SbManagerT::registerListener (SbMsgListener * *msgListenerPtr*) [protected]

registeres the listener with the msgHandler.

also registeres the new pointer with the local memory manager which uses the msgPtrVector to delete object after this class is destr. this one calls the function in msgHandler that extracts types out of listener

Definition at line 51 of file SbManagerT.cpp.

References SbMsgHandlerSingleton::addListener(), msgHandler, msgPtrVector, and registerListener().

void SbManagerT::initMsgHandler (void) [protected]

Creates the listener objects and registers msg Types with respective listener.

remember to delete these in the destructor.

More than one listener can be registered with the same type.

Definition at line 98 of file SbManagerT.cpp.

References registerListener(), sbRbPtr, and tcpServer.

Referenced by init().

Member Data Documentation

vector< SbMsgListener * > SbManagerT::msgPtrVector [protected]

vector of listener pointers that are registered with handler.

This is used to delete allocated memory after class is destroyed.

Definition at line 40 of file SbManagerT.h.

Referenced by init(), registerListener(), and ~SbManagerT().

The documentation for this class was generated from the following files:

- SbManagerT.h
- SbManagerT.cpp

SbMessage Class Reference

SbMessage#include <SbMessage.h>

Detailed Description

Message structure class used for communications.

SbMessage class stores message information in an organized string format. Each message is characterized by "TYPE" which inherently sorts the messages for processing in SbMsgHandlerSingleton. Following the type, arguments (strings) can be added and removed using functions such as addArgument(). SbMessages can also be created using a formatted string delimited by a unique character (default: '|'). Each message is timestamped. Example: “CAM|zoom|56||”.

Author:

Marcin Balicki : Cooper Union - Mechanical Engineering

Date:

March 2004

Todo:

Improve speed by removing vector, try catch throw an error with indexing or trying to convert string to an incompatible type.

Note:

timestamp is set everytime the SbMessage type is set
| is the default delimiter -> MSGTYPE|ARG1|ARG2|ARG3|ARGn||

Bug:

None

Definition at line 26 of file SbMessage.h.

Public Member Functions

- SbMessage (void)
  Constructor.
- SbMessage (const string &type)
  Constructor which sets the type of message.
- SbMessage (const vector< string > &msgVctr, string delimiter)
  Constructor that creates a message from a parsed vector anda delimiter.
- SbMessage (const string &, const string &)
  constructor with unparsed message and delimiter strings
- ~SbMessage (void)
  Destructor.
- void setType (const string &type)
  Sets the type of message.
- void clear (void)
  resets the message
- void addArgument (const string &arg)
  adds on an argument to the message

- void removeArgument (int i)
  removes an argument from message
- void setDelimiter (const string &d)
  sets the delimiter for the message format
- string getDelimiter (void)
  delimiter accessor
- bool create (const string &str)
  parses and sets the type and arg fields of message
- string getType (void) const
  returns type of message.
- string getTimeStamp (void) const
  returns timestamp.
- int getNumOfArgs (void) const
  gets number of arguments in message.
- string getAsStr (void) const
  returns the message as a string (includes delimiters).
- int getArgAsInt (int i) const
  returns an argument as an integer.
- double getArgAsDbl (int i) const
  returns an argument as a double.
- string getArgAsStr (int i) const
  returns an argument as a string.

Static Public Member Functions

- string convToString (int i)
  helper function that converts integer to string
- string convToString (double d)
  helper function that converts double to string

Protected Member Functions

- void setTimeStamp ()
  sets the time of creation or recreation
- bool isNumber (string) const
  test if the string is a number

Protected Attributes

- string delimiter
  delimiter for the current message
- string timeStamp
  timeStamp string
- string type
  type of this message
- int numOfArgs
  number of arguments that are part of this message.
- vector< string > argVctr
  this is slow and may need to be replaced by an array or something like that.

Static Protected Attributes

- const int iniVctrSize = 255
  size that will improve the speed of the vector by allocating space for it

---

Member Function Documentation

string SbMessage::convToString (int *i*) `[static]`

helper function that converts integer to string

internal stream function used in toString()

Definition at line 198 of file SbMessage.cpp.

References convToString().

Referenced by convToString(), and SbRobotML::fire().

string SbMessage::convToString (double *d*) `[static]`

helper function that converts double to string

internal stream function used in toString()

Definition at line 191 of file SbMessage.cpp.

References convToString().

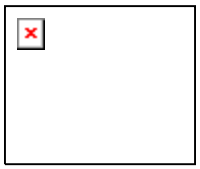
The documentation for this class was generated from the following files:

- SbMessage.h
- SbMessage.cpp

SbMotionML Class Reference

SbMotionML#include <SbMotionML.h>

Inheritance diagram for SbMotionML:



Detailed Description

Manages robot motion functions.

This listener parses motion requests. It commands the robot to "MOVE", "ROTATE", sets the "VEL", "LRVEL", "ROTVEL", "HEADING" and other motion related functions.

Author:

Marcin Balicki : Cooper Union - Mechanical Engineering

Date:

March 2004

Warning:

Aria Motion Commands Be aware that direct or motion command may conflict with controls from Actions or other upper level processes and lead to unexpected consequences. robot->clearDirectMotion() will cancel the direct Motion so action can get the control back there is some other way to automatically do it using:
robot->setDirectMotionPrecedenceTime();
may block actions forever if too high
robot->getDirectMotionPrecedencetTime();
moving the robot using actions is recommended....

Note:

You can use the sendMsgToHandler to fire messages back to the handler.

Bug:

None

Definition at line 29 of file SbMotionML.h.

Public Member Functions

- SbMotionML (void)
  Constructor.
- ~SbMotionML (void)
  Destructor.
- virtual void fire (const SbMessage &msg)
  implements the SbMsgListener pure virtual function.
- vector< string > getMsgTypes (void)
  returns a pointer to the vector containg types of messages that this listener wants to listen to
- virtual void sendMsgToHandler (const SbMessage &msg)
  call back for sending messages back to the message handler.

- virtual void sendMsgToHandler (const string &str)
  call back for sending messages back to the message handler.

Protected Member Functions

- bool checkForRobot (void)
  checks if the robot (ARIA instance) exists and is connected
- virtual void addMsgType (const string m)
  adds the message type that this listener should listen to.

Protected Attributes

- ArRobot * robot
  internal robot pointer
- string type
  type of message
- vector< string > msgTypeVector
  vector containing all the message types that this listener is interested in registering with the handler.

Member Function Documentation

void SbMotionML::fire (const SbMessage & *msg*) [virtual]

implements the SbMsgListener pure virtual function.

Parameters:

&msg SbMessage which caused the listener to fire.

Implements SbMsgListener.

Definition at line 48 of file SbMotionML.cpp.

References checkForRobot(), fire(), SbMessage::getArgAsDbl(), SbMessage::getAsStr(), SbMessage::getType(), robot, and type.

Referenced by fire().

vector<string > SbMsgListener::getMsgTypes (void) [inherited]

returns a pointer to the vector containg types of messages that this listener wants to listen to

Returns:

the pointer to vector of message types

Definition at line 38 of file SbMsgListener.h.

References SbMsgListener::msgTypeVector.

Referenced by SbMsgHandlerSingleton::addListener().

void SbMsgListener::sendMsgToHandler (const SbMessage & *msg*) [virtual, inherited]

call back for sending messages back to the message handler.

Parameters:

&msg SbMessage that should be sent to the handler

Definition at line 12 of file SbMsgListener.cpp.

References SbMsgHandlerSingleton::getInstance(), and SbMsgListener::sendMsgToHandler().

Referenced by SbMsgListener::sendMsgToHandler().

void SbMsgListener::sendMsgToHandler (const string & *str*) [virtual, inherited]

call back for sending messages back to the message handler.

Parameters:

&str String that should be sent to the handler

Definition at line 17 of file SbMsgListener.cpp.

References SbMsgHandlerSingleton::getInstance(), and SbMsgListener::sendMsgToHandler().

virtual void SbMsgListener::addMsgType (const string *m*) [protected, virtual, inherited]

adds the message type that this listener should listen to.

msgHandler will read the vector containing these and register the types for messaging

Parameters:

&m string as a message type ie "SEND"

Definition at line 56 of file SbMsgListener.h.

References SbMsgListener::addMsgType(), and SbMsgListener::msgTypeVector.

Referenced by SbMsgListener::addMsgType().

The documentation for this class was generated from the following files:

- SbMotionML.h
- SbMotionML.cpp

SbMsgHandlerSingleton Class Reference

SbMsgHandlerSingleton`#include <SbMsgHandlerSingleton.h>`

Detailed Description

Manager for the messaging structure.

Any class of type MsgListener can register with the Handler along with a key (type). When fireMsg function is called it will fire (send) the message to all the listeners registered with the particular message type. Each listener implements a fire function which accepts the message and interprets it in any way it wants. Singleton concept was implemented for accessing SbMsgHanglerSingleton. Required because some of the listeners want to send a message back to the Handler. Singleton class assures a maximum of ONE object of its type at a given time and provides a global access point to this object. It may be problematic if threading is used. In that case I suggest to lock the handler by using ArMutex. This was avoided here for gain in speed.

Author:

Marcin Balicki : Cooper Union - Mechanical Engineering

Date:

March 2004

Warning:

a Singleton class needs to be destroyed by the last thread alive by calling removeInstance.

Definition at line 30 of file SbMsgHandlerSingleton.h.

Public Member Functions

- void addListener (const string &, SbMsgListener *)
  adds a key-listener pair to the map.
- void addListener (SbMsgListener *)
  adds a listener, extracts msg types (key) from the listener class
- void removeListener (const string &, SbMsgListener *)
  only removes the msglistener from one particular type.
- void removeListenerInstances (SbMsgListener *)
  removes all instances of MsgListener from the map regarding of key
- void removeType (const string &)
  removes a key and associated listeners
- int getNumOfTypes (void)
  SbMsgListener getMsgListeners(const string&);.
- int getNumOfListeners (const string &)
  returns number of listeners for a msg type
- int getNumOfAllListeners (void)
  returns total number of listeners
- void fireMsg (const SbMessage &)
  fires a message from a SbMessage format
- void **fireMsg** (const string &)

Static Public Member Functions

- SbMsgHandlerSingleton * getInstance (void)
  returns the instance to this class, if does not exist it will be created
- void removeInstance (void)
  remove instance by deleteing it.

Protected Types

- typedef vector< SbMsgListener * >::iterator vectIter
  vector Iterator type
- typedef vector< SbMsgListener * > * listenerVectPtr
  vector Pointer type
- typedef map< string, listenerVectPtr >::iterator mapIter
  map Iterator type

Protected Member Functions

- SbMsgHandlerSingleton (void)
  protected Constructor
- ~SbMsgHandlerSingleton (void)
  protected Destructor which deletes alocated memory for vectors in the map;
- SbMsgHandlerSingleton (const SbMsgHandlerSingleton &)
  Copy constructor.
- SbMsgHandlerSingleton & operator= (const SbMsgHandlerSingleton &)
  operator overload for coping.
- bool msgTypeExists (const string &) const
  check if message type exists
- vectIter vectorFind (vector< SbMsgListener * > *, SbMsgListener *)
  finds the position of the pointer in the given vector (end if not found)

Protected Attributes

- map< string, listenerVectPtr > MsgHndlrMap
  this is the essential data structure that holds the key and the relevant pointer to the list of msgListener pointers.
- SbMessage msg
  temporary msg variable

Static Private Attributes

- SbMsgHandlerSingleton * pinstance = 0
  internal single instance of THIS class.

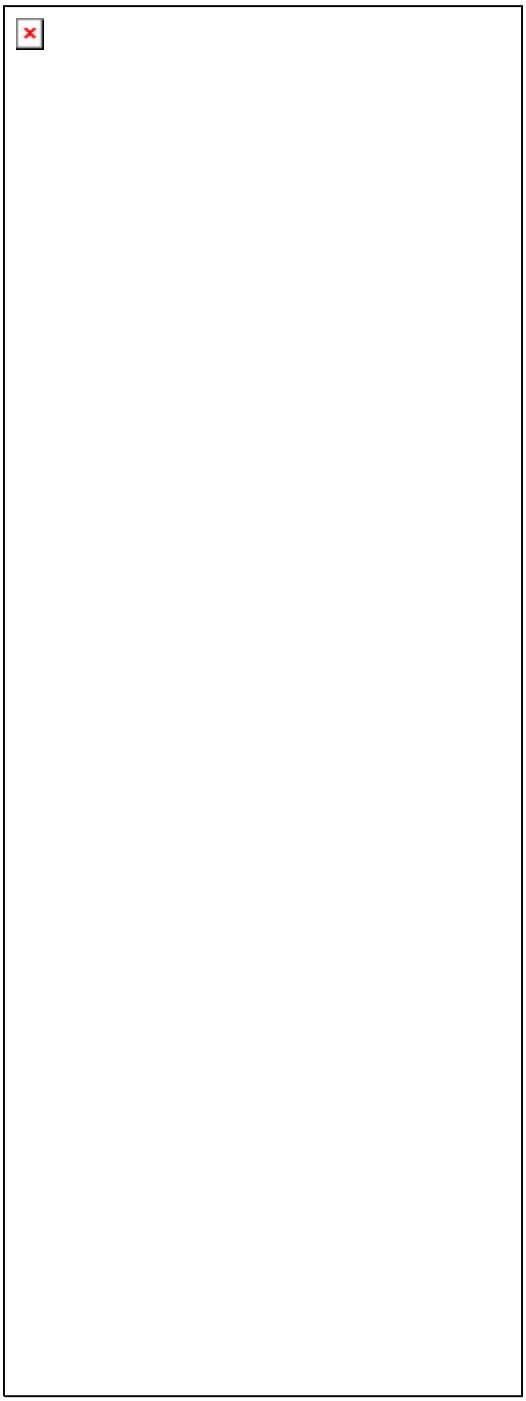
The documentation for this class was generated from the following files:

- SbMsgHandlerSingleton.h
- SbMsgHandlerSingleton.cpp

SbMsgListener Class Reference

SbMsgListener#include <SbMsgListener.h>

Inheritance diagram for SbMsgListener:



Detailed Description

Abstract class for the messaging system.

You cannot create an object of an abstract class type; however, you can use pointers and references to abstract class types. A class that contains at least one pure virtual function (fire) is considered an abstract class. Classes derived from the abstract class must implement the pure virtual function or they, too, are abstract classes. This class provides messaging functionality, it stores the listener types that the listener wants to be registered with. Since the MsgListener can send messages back to the handler sendMsgToHandler() via the pointer to the handler (SbMsgHandlerSingleton::getInstance()) it can only register with one message handler in this program.

Author:

Marcin Balicki : Cooper Union - Mechanical Engineering

Date:

March 2004

Note:

since the handler pointer exists here, the listener can't register with more than one Handler. would be nice to change this later or create an array of pointers.

Bug:

None

Definition at line 28 of file SbMsgListener.h.

Public Member Functions

- SbMsgListener ()
  Constructor.
- ~SbMsgListener ()
  Destructor.
- vector< string > getMsgTypes (void)
  returns a pointer to the vector containg types of messages that this listener wants to listen to
- virtual void fire (const SbMessage &msg)=0
  Pure virtual function (makes this class abstract).
- virtual void sendMsgToHandler (const SbMessage &msg)
  call back for sending messages back to the message handler.
- virtual void sendMsgToHandler (const string &str)
  call back for sending messages back to the message handler.

Protected Member Functions

- virtual void addMsgType (const string m)
  adds the message type that this listener should listen to.

Protected Attributes

- vector< string > msgTypeVector
  vector containing all the message types that this listener is interested in registering with the handler.

Member Function Documentation

vector<string > SbMsgListener::getMsgTypes (void)

returns a pointer to the vector containg types of messages that this listener wants to listen to

Returns:

the pointer to vector of message types

Definition at line 38 of file SbMsgListener.h.

References msgTypeVector.

Referenced by SbMsgHandlerSingleton::addListener().

virtual void SbMsgListener::fire (const SbMessage & *msg*) [pure virtual]

Pure virtual function (makes this class abstract).

Fire the listener function. This will be called by the Handler whenever this Listener is triggered. It is pure abstract needing implementation when inherited.

Parameters:

&msg the SbMessage which caused this MsgListener to fire.

Implemented in SbActionML, SbCameraSONYML, SbCameraVCC4ML, SbExampleML, SbExampleMLT, SbGripperML, SbMotionML, SbRobotML, SbSaphiraML, SbServerML, SbSoundsMLT, and SbStateML.

void SbMsgListener::sendMsgToHandler (const SbMessage & *msg*) [virtual]

call back for sending messages back to the message handler.

Parameters:

&msg SbMessage that should be sent to the handler

Definition at line 12 of file SbMsgListener.cpp.

References SbMsgHandlerSingleton::getInstance(), and sendMsgToHandler().

Referenced by sendMsgToHandler().

void SbMsgListener::sendMsgToHandler (const string & *str*) [virtual]

call back for sending messages back to the message handler.

Parameters:

&str String that should be sent to the handler

Definition at line 17 of file SbMsgListener.cpp.

References SbMsgHandlerSingleton::getInstance(), and sendMsgToHandler().

virtual void SbMsgListener::addMsgType (const string *m*) [protected, virtual]

adds the message type that this listener should listen to.

msgHandler will read the vector containing these and register the types for messaging

Parameters:

&m string as a message type ie "SEND"

Definition at line 56 of file SbMsgListener.h.

References addMsgType(), and msgTypeVector.

Referenced by addMsgType().

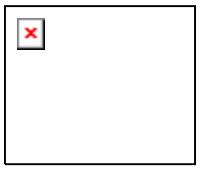
The documentation for this class was generated from the following files:

- SbMsgListener.h
- SbMsgListener.cpp

SbRobotML Class Reference

SbRobotML#include <SbRobotML.h>

Inheritance diagram for SbRobotML:



Detailed Description

Manages robot connection and checks for errors (bumpers, stall, etc).

Creates and instance of ARIA's robot interface. Provides connections to the real robot or simulator. It checks if the robot is connected, if any of its bumpers are triggered and if it has stalled. It also provides and interface for activating sonar and the motors. Returns some robot parameters: name, type, etc.

Author:

Marcin Balicki : Cooper Union - Mechanical Engineering

Date:

March 2004

Warning:

Uses SbMsgListener's SbMsgListener::sendMsgToHandler()

Bug:

None

Definition at line 22 of file SbRobotML.h.

Public Types

- enum conType { **ROBOT**, **SIM** }
  connection type.

Public Member Functions

- SbRobotML (void)
  Constructor.
- ~SbRobotML (void)
  Destructor.
- ArRobot * getRobot (void)
  robot pointer accessor function
- virtual void checkOnce (void)
  checks for any errors (stall, bumpers, etc, connection) etc.
- bool connect (conType t, const string &, const int &)
  connects to the robot or a simulator.
- bool disconnect (void)
  disconnects from the robot or a simulator.

- bool isConnected (void)
  checks if the robot is connected.
- virtual void fire (const SbMessage &msg)
  implements the SbMsgListener pure virtual function.
- vector< string > getMsgTypes (void)
  returns a pointer to the vector containg types of messages that this listener wants to listen to
- virtual void sendMsgToHandler (const SbMessage &msg)
  call back for sending messages back to the message handler.
- virtual void sendMsgToHandler (const string &str)
  call back for sending messages back to the message handler.

Protected Member Functions

- bool connectSim (const string &host, const int &port)
  Connects to the simulator.
- bool connectRobot (void)
  connects to robot connected to serial port
- virtual void addMsgType (const string m)
  adds the message type that this listener should listen to.

Protected Attributes

- bool disconnected
  disconnects from robot
- string type
  type of message
- ArRobot * robot
  internal robot pointer
- ArSonarDevice sonar
  sonar for the robot
- ArSerialConnection * serialCon
  serial connection class
- ArTcpConnection tcpCon
  tcp connection class used for connecting to simulator
- vector< string > msgTypeVector
  vector containing all the message types that this listener is interested in registering with the handler.

Member Function Documentation

void SbRobotML::checkOnce (void) [virtual]

checks for any errors (stall, bumpers, etc, connection) etc.

Checks if the robot is connected and if it is ok (ie stalled) this function returns true if everything is ok. if it is false it will fire a message with stall, bumper or disconnect

Definition at line 175 of file SbRobotML.cpp.

References SbMessage::addArgument(), and robot.

Referenced by SbManagerT::runThread().

void SbRobotML::fire (const SbMessage & *msg*) [virtual]

implements the SbMsgListener pure virtual function.

Parameters:

&msg SbMessage which caused the listener to fire.

Implements SbMsgListener.

Definition at line 256 of file SbRobotML.cpp.

References SbMessage::addArgument(), connect(), SbMessage::convToString(), disconnect(), fire(), SbMessage::getArgAsDbl(), SbMessage::getArgAsInt(), SbMessage::getArgAsStr(), SbMessage::getAsStr(), SbMessage::getType(), isConnected(), robot, and type.

Referenced by fire().

bool SbRobotML::connectSim (const string & *host*, const int & *port*) [protected]

Connects to the simulator.

Parameters:

&host DNS name of the server hosting the simulator
&port integer representing TCP port of the simulator

Definition at line 73 of file SbRobotML.cpp.

References connectSim(), isConnected(), robot, and tcpCon.

Referenced by connect(), and connectSim().

vector<string > SbMsgListener::getMsgTypes (void) [inherited]

returns a pointer to the vector containg types of messages that this listener wants to listen to

Returns:

the pointer to vector of message types

Definition at line 38 of file SbMsgListener.h.

References SbMsgListener::msgTypeVector.

Referenced by SbMsgHandlerSingleton::addListener().

void SbMsgListener::sendMsgToHandler (const SbMessage & *msg*) [virtual, inherited]

call back for sending messages back to the message handler.

Parameters:

&msg SbMessage that should be sent to the handler

Definition at line 12 of file SbMsgListener.cpp.

References SbMsgHandlerSingleton::getInstance(), and SbMsgListener::sendMsgToHandler().

Referenced by SbMsgListener::sendMsgToHandler().

void SbMsgListener::sendMsgToHandler (const string & *str*) [virtual, inherited]

call back for sending messages back to the message handler.

Parameters:

&str String that should be sent to the handler

Definition at line 17 of file SbMsgListener.cpp.

References SbMsgHandlerSingleton::getInstance(), and SbMsgListener::sendMsgToHandler().

virtual void SbMsgListener::addMsgType (const string *m*) [protected, virtual, inherited]

adds the message type that this listener should listen to.

msgHandler will read the vector containing these and register the types for messaging

Parameters:

&m string as a message type ie "SEND"

Definition at line 56 of file SbMsgListener.h.

References SbMsgListener::addMsgType(), and SbMsgListener::msgTypeVector.

Referenced by SbMsgListener::addMsgType().

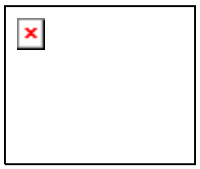
The documentation for this class was generated from the following files:

- SbRobotML.h
- SbRobotML.cpp

SbSaphiraML Class Reference

SbSaphiraML#include <SbSaphiraML.h>

Inheritance diagram for SbSaphiraML:



Detailed Description

Listener/wrapper for Saphira - intelligent avoidance software.

This class calls Saphira function that avoid objects and intelligently maneuver around ("IGOTO"). The avoidance part can be turned off allowing for the robot to naively go to the desired position - "GOTO" The distance when the robot assumes it has successfully reached the goal can be set using "GETSETDONE" while "GOTOSETCLOSE" sets the distance to goal that the robot starts to slow down. "GOTOWINDOW" describes the size of the window that Saphira will look.

Author:

Marcin Balicki : Cooper Union - Mechanical Engineering

Date:

March 2004

Warning:

Uses SbMsgListener's SbMsgListener::sendMsgToHandler().
Test the exact motion patterns of "GOTO", "IGOTO" may be different every time.

Bug:

None

Definition at line 26 of file SbSaphiraML.h.

Public Member Functions

- SbSaphiraML ()
  Constructor.
- ~SbSaphiraML ()
  Destructor.
- virtual void fire (const SbMessage &msg)
  implements the SbMsgListener pure virtual function.
- void init (void)
  initializes the gradient.
- vector< string > getMsgTypes (void)
  returns a pointer to the vector containing types of messages that this listener wants to listen to
- virtual void sendMsgToHandler (const SbMessage &msg)
  call back for sending messages back to the message handler.
- virtual void sendMsgToHandler (const string &str)

call back for sending messages back to the message handler.

Protected Member Functions

- virtual void addMsgType (const string m)
  adds the message type that this listener should listen to.

Protected Attributes

- vector< string > msgTypeVector
  vector containing all the message types that this listener is interested in registering with the handler.

Private Member Functions

- bool checkForRobot (void)
  checks if the robot exists and is connected

Private Attributes

- ArRobot * robot
  internal robot pointer
- string type
  message type variable
- bool inited
  have we initialized
- int close
  saphira variables (mm)distance away the robot is from the goal when it switches to close mode
- int done
  (mm)the distance away the robot is from the goal when the gradient decides its done.
- int winX
  (mm)tSets the size of the gradient window dimensions for gradient path planning.
- int winY
  (mm)tSets the size of the gradient window dimensions for gradient path planning.

Member Function Documentation

void SbSaphiraML::fire (const SbMessage & *msg*) [virtual]

implements the SbMsgListener pure virtual function.

Parameters:

&msg SbMessage which caused the listener to fire.

Implements SbMsgListener.

Definition at line 48 of file SbSaphiraML.cpp.

References checkForRobot(), close, done, fire(), SbMessage::getArgAsDbl(), SbMessage::getArgAsInt(), SbMessage::getAsStr(), SbMessage::getType(), robot, type, winX, and winY.

Referenced by fire().

vector<string > SbMsgListener::getMsgTypes (void) [inherited]

returns a pointer to the vector containg types of messages that this listener wants to listen to

Returns:

the pointer to vector of message types

Definition at line 38 of file SbMsgListener.h.

References SbMsgListener::msgTypeVector.

Referenced by SbMsgHandlerSingleton::addListener().

void SbMsgListener::sendMsgToHandler (const SbMessage & *msg*) [virtual, inherited]

call back for sending messages back to the message handler.

Parameters:

&msg SbMessage that should be sent to the handler

Definition at line 12 of file SbMsgListener.cpp.

References SbMsgHandlerSingleton::getInstance(), and SbMsgListener::sendMsgToHandler().

Referenced by SbMsgListener::sendMsgToHandler().

void SbMsgListener::sendMsgToHandler (const string & *str*) [virtual, inherited]

call back for sending messages back to the message handler.

Parameters:

&str String that should be sent to the handler

Definition at line 17 of file SbMsgListener.cpp.

References SbMsgHandlerSingleton::getInstance(), and SbMsgListener::sendMsgToHandler().

virtual void SbMsgListener::addMsgType (const string *m*) [protected, virtual, inherited]

adds the message type that this listener should listen to.

msgHandler will read the vector containing these and register the types for messaging

Parameters:

&m string as a message type ie "SEND"

Definition at line 56 of file SbMsgListener.h.

References SbMsgListener::addMsgType(), and SbMsgListener::msgTypeVector.

Referenced by SbMsgListener::addMsgType().

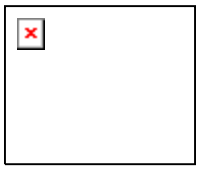
The documentation for this class was generated from the following files:

- SbSaphiraML.h
- SbSaphiraML.cpp

SbServerML Class Reference

SbServerML#include <SbServerML.h>

Inheritance diagram for SbServerML:



Detailed Description

Manages TCP/IP server: multiple socket connections, send/recv ASCII data.

This class uses SbSockets to listen for incoming connections. It allows multiple clients to connect. It will broadcast any message sent via fire() to all connected sockets. It does not automatically check for incomming sockets/messages, which happens via the cycleOnce() function. Clients can disconnect by sending "QUIT" message and all clients are disconnected when "QUITALL" message type is fired. The "SEND" type message broadcasts the message embedded in the message, (SEND is stripped) Start the server with the open() function, add commands with the addCommand function and remove commands with remCommand, and close the server with the close function. It was modeled from ActivMedia's ArNetServer.

Note:

You can use the sendMsgToHandler to fire messages back to the handler.

Author:

Marcin Balicki : Cooper Union - Mechanical Engineering

Date:

March 2004

Warning:

Uses SbMsgListener's SbMsgListener::sendMsgToHandler()

Bug:

None

Definition at line 34 of file SbServerML.h.

Public Member Functions

- SbServerML (void)
  Constructor.
- SbServerML (string port)
  Constructor creates a socket on specified port.
- ~SbServerML (void)
  Destructor.
- void setPort (string)
  set the server port
- string getPort ()

returns the port

- bool open (void)
  Initializes the server.
- bool open (string port)
  Initializes the server with port.
- void cycleOnce (void)
  Runs once through the servers functions.
- void close (void)
  Closes thegetDelimiter(); server.
- void disconnectClients (void)
  Disconnects all Clients.
- void disconnectClient (SbSocket *)
  Disconnects one client.
- string getClientAddress (SbSocket *)
  Disconnects the Client.
- bool isOpened (void)
  Sees if the server is running and open for connections.
- int numConnected (void)
  Returns number of clients connected.
- bool sendData (const SbMessage &)
  send a SbMessage object
- bool sendData (string)
  send a basic string.
- bool getMsg (SbMessage &)
  parses the databuffer for message and assigns the oldesvectort one to the object argument.
- void setDelimiter (const string &d)
  sets the delimiter for msg parsing
- string getDelimiter (void)
  gets the current delimiter
- string getData (SbSocket *)
  gets data and then clears the buffer
- bool getData (SbSocket *, string &)
  gets data and clears the buffer, but lets the user know when the buffer is not empty.
- virtual void fire (const SbMessage &msg)
  implements the SbMsgListener pure virtual function.
- vector< string > getMsgTypes (void)
  returns a pointer to the vector containg types of messages that this listener wants to listen to
- virtual void sendMsgToHandler (const SbMessage &msg)
  call back for sending messages back to the message handler.
- virtual void sendMsgToHandler (const string &str)
  call back for sending messages back to the message handler.

Protected Types

- typedef map< SbSocket *, string * >::iterator myConnsIterator
  connection map iterator

Protected Member Functions

- bool isInteger (string)
  checks if a string is an integer.
- virtual void addMsgType (const string m)
  adds the message type that this listener should listen to.

Protected Attributes

- bool opened
  server open flag
- string port
  port
- string defaultPort
  default port string
- string delimiter
  message delimiter string
- string type
  type of message string
- SbSocket serverSocket
  instance of the socket.
- map< SbSocket *, string * > myConns
  map of connected sockets(KEY) with their respective buffers (socketPointer,buffer)
- vector< SbSocket * > deleteConnsVector
  vector holding pointers to sockets that have been flagged because of a communication error and will be deleted.
- vector< string > msgTypeVector
  vector containing all the message types that this listener is interested in registering with the handler.

Member Function Documentation

void SbServerML::cycleOnce (void)

Runs once through the servers functions.

there was a problem NULL was returned, flag that socket

Definition at line 67 of file SbServerML.cpp.

References SbSocket::accept(), deleteConnsVector, SbSocket::getAddress(), SbSocket::getFD(), getPort(), isOpened(), myConns, myConnsIterator, serverSocket, SbSocket::setNonBlock(), SbSocket::transfer(), and SbSocket::writeString().

Referenced by SbManagerT::runThread().

bool SbServerML::getMsg (SbMessage &)

parses the databuffer for message and assigns the oldesvectort one to the object argument.

This only works in conjunction with cycleOnce().

Definition at line 228 of file SbServerML.cpp.

References SbMessage::create(), disconnectClient(), getDelimiter(), getMsg(), SbMessage::getType(), isOpened(), myConns, myConnsIterator, and sendData().

Referenced by getMsg(), and SbManagerT::runThread().

void SbServerML::fire (const SbMessage & *msg*) [virtual]

implements the SbMsgListener pure virtual function.

Parameters:

&msg SbMessage which caused the listener to fire.

Implements SbMsgListener.

Definition at line 310 of file SbServerML.cpp.

References SbMessage::addArgument(), disconnectClients(), fire(), SbMessage::getArgAsStr(), SbMessage::getNumOfArgs(), SbMessage::getType(), sendData(), and type.

Referenced by fire().

vector<string > SbMsgListener::getMsgTypes (void) [inherited]

returns a pointer to the vector containg types of messages that this listener wants to listen to

Returns:

the pointer to vector of message types

Definition at line 38 of file SbMsgListener.h.

References SbMsgListener::msgTypeVector.

Referenced by SbMsgHandlerSingleton::addListener().

void SbMsgListener::sendMsgToHandler (const SbMessage & *msg*) [virtual, inherited]

call back for sending messages back to the message handler.

Parameters:

&msg SbMessage that should be sent to the handler

Definition at line 12 of file SbMsgListener.cpp.

References SbMsgHandlerSingleton::getInstance(), and SbMsgListener::sendMsgToHandler().

Referenced by SbMsgListener::sendMsgToHandler().

void SbMsgListener::sendMsgToHandler (const string & *str*) [virtual, inherited]

call back for sending messages back to the message handler.

Parameters:

&str String that should be sent to the handler

Definition at line 17 of file SbMsgListener.cpp.

References SbMsgHandlerSingleton::getInstance(), and SbMsgListener::sendMsgToHandler().

virtual void SbMsgListener::addMsgType (const string *m*) [protected, virtual, inherited]

adds the message type that this listener should listen to.

msgHandler will read the vector containing these and register the types for messaging

Parameters:

&m string as a message type ie "SEND"

Definition at line 56 of file SbMsgListener.h.

References SbMsgListener::addMsgType(), and SbMsgListener::msgTypeVector.

Referenced by SbMsgListener::addMsgType().

The documentation for this class was generated from the following files:

- SbServerML.h
- SbServerML.cpp

SbSocket Class Reference

SbSocket#include <SbSocket.h>

Detailed Description

Unix Socket TCP/IP communication wrapper.

SbSocket is a layer which allows people to use the sockets networking interface in unix operating system. All of the standard commonly used socket functions are implemented. This class also contains the file descriptor which identifies the socket to the operating system. This class has been adopted from two ActivMedia's classes, ArSocket.cpp and ArScoket_LIN.cpp The ActivMedia original socket class was undocumented and limited the transmission of large data packets. This class was also implemented strictly for Linux.

Author:

Marcin Balicki : Cooper Union - Mechanical Engineering

Date:

March 2004

Bug:

Maximum message size is 5120 bytes, may work erratically with windows telnet client and crash if overrun.

Warning:

This class was adopted from ARIA and therefore needs to follow ActivMedia's redistribution terms :

Adopted from ARIA, and will follow ActivMedia's distribution terms:

Definition at line 58 of file SbSocket.h.

Public Types

- enum Type { **UDP**, **TCP**, **Unknown** }
  type of connection enumerator
- enum Error { **NoErr**, **NetFail**, **ConBadHost**, **ConNoRoute**, **ConRefused** }
  type of error enumerator

Public Member Functions

- SbSocket ()
  Constructor.
- SbSocket (const char *host, int port, Type type)
  Constructs the socket and connects it to the myConn[socket]) given host.
- SbSocket (int port, bool doClose, Type type)
  Constructs the socket and opens it as a server port.
- ~SbSocket ()
  Destructor.
- bool copy (int fd, bool doclose)
  Copy socket structures.
- void copy (SbSocket *s)
  Copy socket structures.
- void transfer (SbSocket *s)
  Transfer ownership of a socket transfer() will transfer ownership to this socket.
- string getAddress (void)
  returns the address in the string format.
- bool connect (const char *host, int port, Type type)
  Connect as a client to a server.
- bool open (int port, Type type)
  Open a server port.
- bool create (Type type)
  Simply create a port.
- bool findValidPort (int startPort)
  Find a valid unused port and bind the socket to it.
- bool connectTo (const char *host, int port)
  Connect the socket to the given address.
- bool connectTo (struct sockaddr_in *sin)
  Connect the socket to the given address.
- bool accept (SbSocket *sock)
  Accept a new connection.
- bool close ()
  Close the socket.
- int write (const void *buff, size_t len)
  Write to the socket.
- int read (void *buff, size_t len, unsigned int msWait=0)
  Read from the socket.
- int sendTo (const void *msg, int len)
  Send a message on the socket.
- int sendTo (const void *msg, int len, struct sockaddr_in *sin)
  Send a message on the socket.
- int recvFrom (void *msg, int len, sockaddr_in *sin)
  Receive a message from the socket.
- bool getSockName ()
  Get the socket name. Stored in SbSocket::mySin.

- sockaddr_in * sockAddrIn ()
  Accessor for the sockaddr.
- in_addr * inAddr ()
  Accessor for the in_addr.
- unsigned short int inPort (void)
  Accessor for the port of the sockaddr.
- bool setLinger (int time)
  Set the linger value.
- bool setBroadcast ()
  Set broadcast value.
- bool setReuseAddress ()
  Set the reuse address value.
- bool setNonBlock ()
  Set socket to nonblocking.
- void setDoClose (bool yesno)
  Change the doClose value.
- int getFD () const
  Get the file descriptor.
- Type getType () const
  Get the protocol type.
- const std::string & getErrorStr () const
  Get the last error string.
- Error getError () const
  Get the last error.
- int writeString (const char *str,...)
  Writes a string to the socket (adding end of line characters).
- int writeStringPlain (const char *str)
  Same as writeString, but no varargs, wrapper for java.
- char * readString (void)
  Reads a string from the socket.
- void setEcho (bool echo)
  Sets echoing on the readString calls this socket does.
- bool getEcho (void)
  Gets if we are echoing on the readString calls this socket does.

Static Public Member Functions

- bool hostAddr (const char *host, struct in_addr &addr)
  Convert a host string to an address structure.
- bool addrHost (struct in_addr &addr, char *host)
  Convert an address structure to a host string.
- std::string getHostName ()
  Get the localhost address.
- void inToA (struct in_addr *addr, char *buff)
  Convert addr into string numerical address.

- const size_t sockAddrLen ()
  Size of the sockaddr.
- const size_t maxHostNameLen ()
  Max host name length.
- unsigned int hostToNetOrder (int i)
  Host byte order to network byte order.
- unsigned int netToHostOrder (int i)
  Network byte order to host byte order.

Static Public Attributes

- bool ourInitialized = true
  flag whether the socket is initialized or not

Protected Member Functions

- void doStringEcho (void)
  internal function that echos strings from read string
- void **internalInit** (void)

Protected Attributes

- Type myType
  internal type of connection (udp,tcp)
- Error myError
  type of error
- std::string myErrorStr
  type of error in string format
- bool myDoClose
  flag for closing socket
- int myFD
  file descriptor
- bool myNonBlocking
  flag for blocking
- sockaddr_in mySin
  structure of socket address
- bool myStringAutoEcho
  auto echo flag
- bool myStringEcho
  echo flag
- char myStringBuf [BUFFSIZE]
  this is the buffer that stores socket data
- size_t myStringPos
  helpful in reading bytes from the buffer
- char myStringBufEmpty [1]
  empty string buffer
- size_t myStringPosLast
  keeps the last position in reading buffer
- bool myStringGotEscapeChars

flag for escape character

- bool myStringGotComplete
  flag for complete data
- bool myStringHaveEchoed
  flag for successful echo

Static Protected Attributes

- const int BUFFSIZE = 5120
  shoulod implement a variable for the string buffer size.

Constructor & Destructor Documentation

SbSocket::SbSocket (const char * *host*, int *port*, Type *type*)

Constructs the socket and connects it to the myConn[socket]) given host.

Parameters:

host hostname of the server to connect to
port port number of the server to connect to
type protocol type to use

Definition at line 170 of file SbSocket.cpp.

References connect().

SbSocket::SbSocket (int *port*, bool *doClose*, Type *type*)

Constructs the socket and opens it as a server port.

Parameters:

port port number to bind the socket to
doClose automaticaly close the port if the socket is destructed
type protocol type to use

Definition at line 183 of file SbSocket.cpp.

References open().

Member Function Documentation

bool SbSocket::copy (int *fd*, bool *doclose*)

Copy socket structures.

Copy from one Socket to another will still have the first socket close the file descripter when it is destructed.

Definition at line 539 of file SbSocket.cpp.

References copy(), myDoClose, myErrorStr, myFD, mySin, and myType.

Referenced by copy().

void SbSocket::transfer (SbSocket * s)

Transfer ownership of a socket transfer() will transfer ownership to this socket.

The input socket will no longer close the file descriptor when it is destructed.

Definition at line 102 of file SbSocket.h.

References myDoClose, myFD, mySin, myType, and transfer().

Referenced by SbServerML::cycleOnce(), and transfer().

int SbSocket::write (const void * *buff*, size_t *len*)

Write to the socket.

Parameters:

buff buffer to write from
len how many bytes to write

Returns:

number of bytes written

Definition at line 133 of file SbSocket.h.

References myFD, and write().

Referenced by doStringEcho(), write(), and writeString().

int SbSocket::read (void * *buff*, size_t *len*, unsigned int *msWait* = 0)

Read from the socket.

Parameters:

buff buffer to read into
len how many bytes to read
msWait if 0, don't block, if > 0 wait this long for data

Returns:

number of bytes read

Definition at line 151 of file SbSocket.h.

References myFD, and read().

Referenced by read(), and readString().

int SbSocket::writeString (const char * *str*, ...)

Writes a string to the socket (adding end of line characters).

this cannot write more than 5120 bytes

Parameters:

**str* the string to write to the socket

Returns:

number of bytes written

Definition at line 43 of file SbSocket.cpp.

References BUFFSIZE, write(), and writeString().

Referenced by SbServerML::cycleOnce(), SbServerML::disconnectClient(), and writeString().

char * SbSocket::readString (void)

Reads a string from the socket.

The original function could only read strings less than 512 characters as long as it read the characters into its own internal buffer (to compensate for some of the things the DOS telnet does). I have expanded this to 5120 chars to see what the issue is. Seems to work fine.

Returns:

if there was good data return string, if there was no data, a string with a "\0" first character and if the socket was bad it returns NULL

Definition at line 70 of file SbSocket.cpp.

References doStringEcho(), myStringBuf, myStringBufEmpty, myStringGotComplete, myStringGotEscapeChars, myStringPos, myStringPosLast, and read().

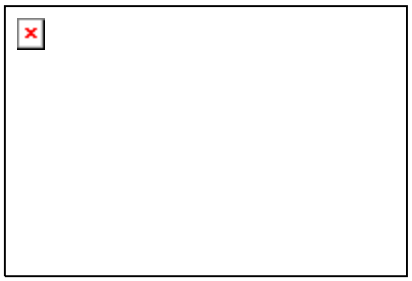
The documentation for this class was generated from the following files:

- SbSocket.h
- SbSocket.cpp

SbSoundsMLT Class Reference

SbSoundsMLT#include <SbSoundsMLT.h>

Inheritance diagram for SbSoundsMLT:



Detailed Description

Listener for Audio playback including speech synthesis and digital audio play.

SbSoundsT class inherits Aria's threaded ArSyncTask to take care of playing wavs and synthesizing without locking the robot cycling. It also allows for sending many consecutive requests. First use of speak or Play will initialize botspeak.

Author:

Marcin Balicki : Cooper Union - Mechanical Engineering

Date:

March 2004

Warning:

Be careful with locking and unlocking.
This class is threaded and therefore should not use sendMsgToHandler to fire messages back to the handler because some other thread maybe using it.

Todo:

Instead of making SbMsgHandler as the singleton, the manager class should be one, and the reference to it should be possessed by all SbMsgListeners. However, this relation would be forced since listener is far away in function from manager (listener->handler->manager).

Bug:

None

Definition at line 40 of file SbSoundsMLT.h.

Public Types

- typedef pthread_t **ThreadType**
- typedef std::map< ThreadType, ArThread * > **MapType**
- enum Status { STATUS_FAILED = 1, STATUS_NORESOURCE, STATUS_NO_SUCH_THREAD, STATUS_INVALID, STATUS_JOIN_SELF, STATUS_ALREADY_DETATCHED }

Public Member Functions

- virtual void fire (const SbMessage &msg)
  implements the SbMsgListener pure virtual function.

- virtual void * runThread (void *arg)
  implements the ArSyncTask main thread function
- vector< string > getMsgTypes (void)
  returns a pointer to the vector containg types of messages that this listener wants to listen to
- virtual void sendMsgToHandler (const SbMessage &msg)
  call back for sending messages back to the message handler.
- virtual void sendMsgToHandler (const string &str)
  call back for sending messages back to the message handler.
- virtual AREXPORT void run (void)
  Run in this thread.
- virtual AREXPORT void runAsync (void)
  Run in its own thread.
- virtual AREXPORT void stopRunning (void)
  Stop the thread.
- virtual AREXPORT int create (bool joinable=true, bool lowerPriority=true)
  Create the task and start it going.
- virtual AREXPORT void * runInThisThread (void *arg=0)
  Run the code of the task syncronously.
- virtual AREXPORT int join (void **ret=NULL)
  Join on the thread.
- virtual AREXPORT int detach (void)
  Detatch the thread so it cant be joined.
- virtual AREXPORT void cancel (void)
  Cancel the thread.
- virtual AREXPORT bool getRunning (void) const
  Get the running status of the thread.
- virtual AREXPORT bool getRunningWithLock (void)
  Get the running status of the thread, locking around the variable.
- virtual AREXPORT bool getJoinable (void) const
  Get the joinable status of the thread.
- virtual AREXPORT const ThreadType * getThread (void) const
  Get the underlying thread type.
- virtual AREXPORT ArFunctor * getFunc (void) const
  Get the functor that the thread runs.
- virtual AREXPORT void setRunning (bool running)
  Set the running value on the thread.
- AREXPORT int lock (void)
  Lock the thread instance.
- AREXPORT int tryLock (void)
  Try to lock the thread instance without blocking.
- AREXPORT int unlock (void)
  Unlock the thread instance.
- bool getBlockAllSignals (void)
  Do we block all process signals at startup?

Static Public Member Functions

- AREXPORT void init (void)
  Initialize the internal book keeping structures.
- AREXPORT ArThread * self (void)
  Returns the instance of your own thread.
- AREXPORT void stopAll ()
  Stop all threads.
- AREXPORT void cancelAll (void)
  Cancel all threads.
- AREXPORT void joinAll (void)
  Join on all threads.
- AREXPORT void yieldProcessor (void)
  Yield the processor to another thread.

Protected Member Functions

- bool checkForRobot (void)
  checks if the robot exists and is connected
- virtual void addMsgType (const string m)
  adds the message type that this listener should listen to.
- virtual AREXPORT int **doJoin** (void **ret=NULL)

Protected Attributes

- ArRobot * robot
  internal robot pointer
- SbMessage msg
  internal message
- bool newSpeakMsg
  new message for speak arrived flag
- bool newPlayMsg
  new message for play arrived flag
- bool init
  initilized botspeak flag
- vector< string > msgTypeVector
  vector containing all the message types that this listener is interested in registering with the handler.
- ArMutex **myMutex**
- bool myRunning
  State variable to denote when the thread should continue or exit.
- bool **myJoinable**
- bool **myBlockAllSignals**
- ThreadType **myThread**
- ArStrMap **myStrMap**

Static Protected Attributes

- ArMutex **ourThreadsMutex**
- MapType **ourThreads**

Member Enumeration Documentation

enum ArThread::Status [inherited]

Enumeration values:

STATUS_FAILED Failed to create the thread.
STATUS_NORESOURCE Not enough system resources to create the thread.
STATUS_NO_SUCH_THREAD The thread can no longer be found.
STATUS_INVALID Thread is detached or another thread is joining on it.
STATUS_JOIN_SELF Thread is your own thread. Can't join on self.
STATUS_ALREADY_DETATCHED Thread is already detatched.

Definition at line 63 of file ArThread.h.

Member Function Documentation

void SbSoundsMLT::fire (const SbMessage & *msg*) [virtual]

implements the SbMsgListener pure virtual function.

Parameters:

&msg SbMessage which caused the listener to fire.

Implements SbMsgListener.

Definition at line 41 of file SbSoundsMLT.cpp.

References checkForRobot(), fire(), SbMessage::getAsStr(), init, ArThread::lock(), newPlayMsg, newSpeakMsg, robot, and ArThread::unlock().

Referenced by fire().

void * SbSoundsMLT::runThread (void * *arg*) [virtual]

implements the ArSyncTask main thread function

Parameters:

**arg* argument that does nothing.

Implements ArASyncTask.

Definition at line 115 of file SbSoundsMLT.cpp.

References SbMessage::getArgAsStr(), ArThread::lock(), msg, newPlayMsg, newSpeakMsg, runThread(), and ArThread::unlock().

Referenced by runThread().

vector<string > SbMsgListener::getMsgTypes (void) [inherited]

returns a pointer to the vector containg types of messages that this listener wants to listen to

Returns:

the pointer to vector of message types

Definition at line 38 of file SbMsgListener.h.

References SbMsgListener::msgTypeVector.

Referenced by SbMsgHandlerSingleton::addListener().

void SbMsgListener::sendMsgToHandler (const SbMessage & *msg*) [virtual, inherited]

call back for sending messages back to the message handler.

Parameters:

&msg SbMessage that should be sent to the handler

Definition at line 12 of file SbMsgListener.cpp.

References SbMsgHandlerSingleton::getInstance(), and SbMsgListener::sendMsgToHandler().

Referenced by SbMsgListener::sendMsgToHandler().

void SbMsgListener::sendMsgToHandler (const string & *str*) [virtual, inherited]

call back for sending messages back to the message handler.

Parameters:

&str String that should be sent to the handler

Definition at line 17 of file SbMsgListener.cpp.

References SbMsgHandlerSingleton::getInstance(), and SbMsgListener::sendMsgToHandler().

virtual void SbMsgListener::addMsgType (const string *m*) [protected, virtual, inherited]

adds the message type that this listener should listen to.

msgHandler will read the vector containing these and register the types for messaging

Parameters:

&m string as a message type ie "SEND"

Definition at line 56 of file SbMsgListener.h.

References SbMsgListener::addMsgType(), and SbMsgListener::msgTypeVector.

Referenced by SbMsgListener::addMsgType().

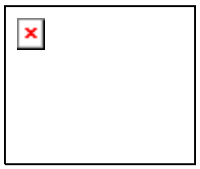
The documentation for this class was generated from the following files:

- SbSoundsMLT.h
- SbSoundsMLT.cpp

SbStateML Class Reference

SbStateML#include <SbStateML.h>

Inheritance diagram for SbStateML:



Detailed Description

Manages requests for robot's state (position,velocity, sonar, motors, etc).

SbStateML gathers requested information and sends back to the user. The information includes current motion/position, battery voltage, sonar states, table sensors, motor activity.)

Author:

Marcin Balicki : Cooper Union - Mechanical Engineering

Date:

March 2004

Note:

You can use the sendMsgToHandler to fire messages back to the handler.

Bug:

None

Definition at line 22 of file SbStateML.h.

Public Member Functions

- SbStateML (void)
  Constructor.
- ~SbStateML (void)
  Decontstructor.
- string toString (double)
  Converts double to a string.
- virtual void fire (const SbMessage &msg)
  implements the SbMsgListener pure virtual function
- vector< string > getMsgTypes (void)
  returns a pointer to the vector containg types of messages that this listener wants to listen to
- virtual void sendMsgToHandler (const SbMessage &msg)
  call back for sending messages back to the message handler.
- virtual void sendMsgToHandler (const string &str)
  call back for sending messages back to the message handler.

Protected Member Functions

- virtual void addMsgType (const string m)
  adds the message type that this listener should listen to.

Protected Attributes

- vector< string > msgTypeVector
  vector containing all the message types that this listener is interested in registering with the handler.

Private Member Functions

- bool checkForRobot (void)
  checks if the robot exists and is connected

Private Attributes

- ArRobot * robot
  internal robot pointer
- string type
  message type
- SbMessage * msgState
  internal message
- ostringstream oss
  internal stream function used in toString()

Member Function Documentation

void SbStateML::fire (const SbMessage & *msg*) [virtual]

implements the SbMsgListener pure virtual function

Parameters:

&msg SbMessage that coused the SbMsgListener to be fired.

Implements SbMsgListener.

Definition at line 39 of file SbStateML.cpp.

References SbMessage::addArgument(), checkForRobot(), SbMessage::clear(), fire(), SbMessage::getArgAsDbl(), SbMessage::getArgAsInt(), SbMessage::getArgAsStr(), SbMessage::getAsStr(), SbMessage::getType(), msgState, robot, SbMessage::setType(), toString(), and type.

Referenced by fire().

vector<string > SbMsgListener::getMsgTypes (void) [inherited]

returns a pointer to the vector containg types of messages that this listener wants to listen to

Returns:

the pointer to vector of message types

Definition at line 38 of file SbMsgListener.h.

References SbMsgListener::msgTypeVector.

Referenced by SbMsgHandlerSingleton::addListener().

void SbMsgListener::sendMsgToHandler (const SbMessage & *msg*) [virtual, inherited]

call back for sending messages back to the message handler.

Parameters:

&msg SbMessage that should be sent to the handler

Definition at line 12 of file SbMsgListener.cpp.

References SbMsgHandlerSingleton::getInstance(), and SbMsgListener::sendMsgToHandler().

Referenced by SbMsgListener::sendMsgToHandler().

void SbMsgListener::sendMsgToHandler (const string & *str*) [virtual, inherited]

call back for sending messages back to the message handler.

Parameters:

&str String that should be sent to the handler

Definition at line 17 of file SbMsgListener.cpp.

References SbMsgHandlerSingleton::getInstance(), and SbMsgListener::sendMsgToHandler().

virtual void SbMsgListener::addMsgType (const string *m*) [protected, virtual, inherited]

adds the message type that this listener should listen to.

msgHandler will read the vector containing these and register the types for messaging

Parameters:

&m string as a message type ie "SEND"

Definition at line 56 of file SbMsgListener.h.

References SbMsgListener::addMsgType(), and SbMsgListener::msgTypeVector.

Referenced by SbMsgListener::addMsgType().

The documentation for this class was generated from the following files:

- SbStateML.h
- SbStateML.cpp

Robot Control and Communication Interface Page Documentation

Bug ListBug List

Class SbActionGoto

Doesn't work as expected...used Saphira instead.

Class SbSocket

Maximum message size is 5120 bytes, may work erratically with windows telnet client and crash if overrun.

Todo ListTodo List

Class SbCameraSONYML

Would be nice to figure out how to test which type of camera is connected to this robot and then create the appropriate control class.

Class SbCameraVCC4ML

Would be nice to figure out how to test which type of camera is connected to this robot and then create the appropriate control class.

Class SbExampleMLT

Instead of making SbMsgHandlerSingleton as the singleton, the manager class should be one, and the reference to it should be possessed by all SbMsgListeners. However ,this relations would be forced since listener is far away in function from manager (listener->handler->manager). another way is to add lock/unlock feature to the handler.

Class SbMessage

Improve speed by removing vector, try catch throw an error with indexing or trying to convert string to an incompatible type.

Class SbSoundsMLT

Instead of making SbMsgHandler as the singleton, the manager class should be one, and the reference to it should be possessed by all SbMsgListeners. However, this relation would be forced since listener is far away in function from manager (listener->handler->manager).

Index

8.14 RCCI Source Code

Included Source files:
main.cpp
SbActionGoto.h
SbActionGoto.cpp
SbActionML.h
SbActionML.cpp
SbActionMove.h
SbActionMove.cpp
SbActionRotate.h
SbActionRotate.cpp
SbCameraSONYML.h
SbCameraSONYML.cpp
SbCameraVCC4ML.h
SbCameraVCC4ML.cpp
SbExampleML.h
SbExampleML.cpp
SbExampleMLT.h
SbExampleMLT.cpp
SbGripperML.h
SbGripperML.cpp
SbManagerT.h
SbManagerT.cpp
SbMessage.h
SbMessage.cpp
SbMotionML.h
SbMotionML.cpp
SbMsgHandlerSingleton.h
SbMsgHandlerSingleton.cpp
SbMsgListener.h
SbMsgListener.cpp
SbRobotML.h
SbRobotML.cpp
SbSaphiraML.h
SbSaphiraML.cpp
SbServerML.h
SbServerML.cpp
SbSocket.h
SbSocket.cpp
SbSoundsMLT.h
SbSoundsMLT.cpp
SbStateML.h
SbStateML.cpp

```
#undef TCP
#undef UDP

#include "SbManagerT.h"
#include "Aria.h"
#include <string>

/// main c++ file is the entrance point into the program.
/// it checks if the command line PORT argument was entered correctly.
/// if true, then it initializes  ARIA in threaded mode.
/// @author Marcin Balicki : Cooper Union - Mechanical Engineering
/// @date March 2004
/// @bug None

using namespace std;

int main(int argc, char * argv[])
{
  if(argc != 2)
  {
    cout << "INCORRECT NUMBER OF ARGUMENTS"<< endl;
    exit(1);
  }
  //Setup a dedicated signal handling thread.
  Aria::init(Aria::SIGHANDLE_THREAD);
  // creates manager instance which starts its own thread.
  SbManagerT mngr(argv[1]); //pass the port number.
  //if can't initialize (ie can't bind to port) just quit.
  if(!mngr.init())
  {
    cout << "Exiting Program due to Init failure. (main)"<< endl;
    Aria::shutdown();
    return 1;
  }
  ArUtil::sleep(100);
 // this runs until aria gets an exit signal.
  while(Aria::getRunning())
  {
    ArUtil::sleep(300);
    // cout<<"Alive"<<endl;
  }
  return 0;
}
```

```
#ifndef SbActionGoto_h
#define SbActionGoto_h

#include "ariaTypedefs.h"
#include "ariaUtil.h"
#include "ArAction.h"

/// This action goes to a given ArPose very naively.
/// This class has been adopted from ActivMedia's ArActionGoto. The action
/// stops when it gets closeDist away.
/// You can give it a new goal with setGoal(), cancel its movement with
/// cancelGoal(), and see if it got there with haveAchievedGoal().
/// This doesn't avoid obstacles or anything, you could have an avoid
/// routine at a higher priority to avoid on the way there,  but for
/// real and intelligent looking navigation you should use something like
/// Saphira's Gradient navigation.
/// @author Marcin Balicki : Cooper Union - Mechanical Engineering
/// @date March 2004
/// @bug Doesn't work as expected...used Saphira instead.


class SbActionGoto : public ArAction
{
public:
  ///Constructor
  SbActionGoto(const char * name = "goto", double closeDist = 200);
  ///Destructor
  virtual ~SbActionGoto();
  /// Sees if the goal has been achieved
  bool haveAchievedGoal(void);
  /// Cancels the goal the robot has
  void cancelGoal(void);
  /// Sets a new goal and sets the action to go there
  void setGoal(ArPose goal);

  /// Gets the goal the action has
  ArPose getGoal(void)
  { return myGoal; }

  /// Set the distance which is close enough to the goal (mm);
  void setCloseDist(double closeDist)
  { myCloseDist = closeDist; }

  /// Gets the distance which is close enough to the goal (mm)
  double getCloseDist(void)
  { return myCloseDist; }

  /// Sets the speed the action will travel to the goal at (mm/sec)
  void setSpeed(double speed)
  { mySpeed = speed; }

  /// Gets the speed the action will travel to the goal at (mm/sec)
  double getSpeed(void)
  { return mySpeed; }

  /// this should be used by the used to set new goal
  void go(ArPose);
  /// gets fired every cycle.
  virtual ArActionDesired * fire(ArActionDesired currentDesired);

  /// accesor for action system
  virtual ArActionDesired * getDesired(void)
  { return & myDesired; }

protected:
  /// current goal
  ArPose myGoal;
  /// allow distance within the goal as success
  double myCloseDist;
  /// desired speed
  double mySpeed;
  /// incrememnt amount to turn to achieve heading
  double myTurnAmount;
  /// previous goal distance
  double myOldDist;
  /// internal action
  ArActionDesired myDesired;
  /// old position
  ArPose myOldGoal;

  /// enumerates the current state of this action
  enum State
  { STATE_NO_GOAL, STATE_ACHIEVED_GOAL, STATE_GOING_TO_GOAL};

  /// current state of the goto function
  State myState;
};

#endif // SbActionGoto
```

```
/* This class was addopted from SbActionGoto created by activmedia. */

#include "SbActionGoto.h"
#include "ArRobot.h"

SbActionGoto::SbActionGoto(const char * name, double closeDist) :
     ArAction(name, "Goes to the given goal.")
     {
       setNextArgument(
            ArArg("close dist", & myCloseDist,
            "Distance that is close enough to goal. (mm)"));
       myCloseDist = closeDist;
       setGoal(ArPose(0.0, 0.0, 0.0));
       myTurnAmount = 7;
       mySpeed = 2000;
       myOldDist = 10000000;
}

SbActionGoto::~SbActionGoto()
{ }

bool SbActionGoto::haveAchievedGoal(void)
{
  if(myState == STATE_ACHIEVED_GOAL)
    return true;
  else
    return false;
}

void SbActionGoto::cancelGoal(void)
{ myState = STATE_NO_GOAL; }

void SbActionGoto::setGoal(ArPose goal)
{
  myState = STATE_GOING_TO_GOAL;
  myGoal = goal;
  myOldGoal = myGoal;
}

void SbActionGoto::go(ArPose goal)
{
  myOldDist = 10000000; //reset it.
  ArPose g = myRobot->getPose();
  goal.setX(goal.getX() + g.getX());
  goal.setY(goal.getY() + g.getY());
  goal.setTh(goal.getTh() + g.getTh());
  //printf("goal : %.0f  ", goal.getX());
  setGoal(goal);
}

ArActionDesired * SbActionGoto::fire(ArActionDesired currentDesired)
{
  double angle;
  double dist;
  double vel;

  //this checks if the new goal set is just from
  if(myGoal.findDistanceTo(myOldGoal) > 5)
    setGoal(myGoal);
  // if we're there we don't do anything
  if(myState == STATE_ACHIEVED_GOAL|| myState == STATE_NO_GOAL)
    return NULL;
  printf("#OldDistance %.0f  ", myOldDist);
  dist = myRobot->getPose().findDistanceTo(myGoal);
  printf("Dist : %.0f  ", dist);
  // see where we want to point
  angle = myRobot->getPose().findAngleTo(myGoal);
  printf("AngleToGoal : %.0f  ", angle);
  //this is in case we over shoot.
  //if the difference is positive than we are moving towards the goal.
  //else we should stop.
  if((myOldDist - dist) < 0)
  {
    printf("Achieved goal\n");
    myState = STATE_ACHIEVED_GOAL;
    myDesired.setVel(0);
    myDesired.setDeltaHeading(0);
    myOldDist = 10000000; //reset it.
    return & myDesired;
  }
  //do it again...but this time to turn..if we overshot do not turn.
  else
    myDesired.setHeading(angle);
  if(dist < (myCloseDist * 0.25) && ArMath::fabs(myRobot->getVel() < 5))
  {
    printf("#achieved\n");
    myState = STATE_ACHIEVED_GOAL;
    myDesired.setVel(0);
    myDesired.setDeltaHeading(0);
    myOldDist = 10000000; //reset it.
  }
  else if(dist < myCloseDist)
  {
    //if (dist < 60){
    //    printf("#Velocity %.0f\n", vel);
    //    myDesired.setVel(0);
    //    }
    // else
    //{
```

```
      vel = sqrt(dist * 10);
      printf("#Velocity %.0f\n", vel);
      myDesired.setVel(vel);
      //}
    }

    else
    {
      vel = sqrt(dist * 200 * 2);
      if(vel > mySpeed)
        vel = mySpeed;
      printf("#Velocity %.0f\n", vel);
      myDesired.setVel(vel);
    }
    myOldDist = dist;

    return & myDesired;
  }
```

```
#ifndef SbActionML_h
#define SbActionML_h

#include "SbMsgListener.h"
#include "SbMessage.h"
#include "SbActionRotate.h"
#include "SbActionGoto.h"
#include "SbActionMove.h"
#include "Aria.h"

#include <string>
using namespace std;


/// Manages Aria and custom (Sb) actions that combine and control robot
/// behavior.
/// SbActionML listens for queries to control actions and manages their
/// initialization. The more important actions are the SbActionMove and
/// SbActionRotate which allow the user to combine the popular functions of
/// move and rotate to be used in combination with action limiting functions
/// such as ArActionAvoidFront.
/// @author Marcin Balicki : Cooper Union - Mechanical Engineering
/// @date March 2004
/// @warning Direct Motion Commands.
/// Be aware that direct or motion command may conflict with controls from \n
/// Actions or other upper level process and lead to unexpected consequences.\n
/// Robot->clearDirectMotion() will cancel the direct Motion so action can \n
/// get the control back there is some other way to automatically do it using \n
/// robot->setDirectMotionPrecedenceTime();  \n
/// may block actions forever if too high \n
/// robot->getDirectMotionPrecedencetTime(); \n
/// be careful direct motion commands may prevent actions forever\n
/// may need to set robot->clearDirectMotion();
/// maybe I should use actions instead of direct motion commmands.\n
/// because direct motions do not work with actions well.\n
/// @note You can use the sendMsgToHandler to fire messages back to the handler.
/// @bug None

class SbActionML : public SbMsgListener
{
public:
  ///Constructor
  SbActionML( void );
  ///Destructor
  ~SbActionML( void );
  /// implements the SbMsgListener pure virtual function.
  /// @param &msg SbMessage which caused the listener to fire.
  virtual void fire( const SbMessage & msg);
protected:
  ///internal robot pointer
  ArRobot * robot;
  ///checks if the robot exists and is connected
  bool checkForRobot( void );
  ///initialize the actions
  void init( void );
  ///message type.
  string type;
  /// Non Aria goto action
  SbActionGoto goTo;
  /// Non Aria rotate action
  SbActionRotate rotate;
  /// Non action move action
  SbActionMove move;
  /// front avoid
  ArActionAvoidFront avoidFront;
  /// side avoid
  ArActionAvoidFront avoidSide;
  /// bumper action
  ArActionBumpers bumpers;
  /// keeps velocity constant
  ArActionConstantVelocity constVel;
  /// limits objects in the back
  ArActionLimiterBackwards limitBackwards;
  /// limits objects in the front
  ArActionLimiterForwards limitForwards;
  /// table sensor
  ArActionLimiterTableSensor tableLimiter;
  /// recovers from a stall
  ArActionStallRecover recover;
  /// stops all actions
  ArActionStop stop;
  /// Aria's turn action
  ArActionTurn turn;
  /// groups
  ArActionGroupStop* gStop;
  /// groups
  ArActionGroupWander* gWander;

};

#endif
```

```
#include "SbActionML.h"


SbActionML::SbActionML(void)
{
  robot = NULL;
  //gWander=NULL;
  //gStop=NULL;
  //new ArActionStallRecover
  //adds message types to the msgListener (base) vector for registering with
  //msg handler.
  addMsgType("AROTATE");
  addMsgType("AMOVE");
  addMsgType("STOP");

}

SbActionML::~SbActionML(void)
{
  ///  if (!(gWander==NULL && gStop==NULL)){
  //    delete gStop;
  //    delete gWander;
  robot->remAction(& goTo);
  robot->remAction(& move);
  //robot->remAction(&stop);
  robot->remAction(& turn);
  robot->remAction(& recover);
  robot->remAction(& bumpers);
  robot->remAction(& tableLimiter);
  robot->remAction(& rotate);
}

bool SbActionML::checkForRobot(void)
{
  //if it does and its connected than return true;
  //it it exists and its the same as the old one just return the
  //Check if the robot exists if not return false
  ArRobot * r;
  if((r = Aria::findRobot("SBROBOT")) == NULL)
    return false;
  //if it exists check if it is different than the last definition of the robot.
  //if it is update the robot pointer and then initilize
  else if(r != robot)
  {
    robot = r;
    init();
  }
  //lock the robot se we can get data from it
  //check if it is connected
  robot->lock();
  if(robot->isConnected())
  {
    //if we are connected unlock and quit function
    robot->unlock();
    return true;
  }
  //we are not connected so unlock the robot and return false.
  robot->unlock();
  return false;
}

void SbActionML::init(void)
{
  robot->lock();
  //if the groups are not instantiated then do not delete
  // if (!(gWander==NULL && gStop==NULL)){
  //    delete gStop;
  //   delete gWander;
  // }
  //cout<<"INIT ACTIONS!"<<endl;
  // add our actions in a good order, the integer here is the priority,
  // with higher priority actions going first
  //add all desired actions and deactivate them until user activates them.
  //remember to enablemotors..and turn on the sonars if needed.
  //robot->comInt(ArCommands::SETO, 1);
  //robot->comInt(ArCommands::ENABLE, 1);
  robot->addAction(& goTo, 90);
  robot->addAction(& move, 91);
  move.deactivate();
  //robot->addAction(&stop, 100);
  robot->addAction(& rotate, 90);
  goTo.deactivate();
  //robot->addAction(&turn, 49);
  //turn.deactivate();
  robot->addAction(& recover, 100);
  recover.deactivate();
  robot->addAction(& bumpers, 80);
  bumpers.deactivate();
  robot->addAction(& tableLimiter, 90);
  tableLimiter.deactivate();
  //gWander = new ArActionGroupWander(robot);
  //gStop= new ArActionGroupStop(robot);
  robot->unlock();
}

void SbActionML::fire(const SbMessage & msg)
{
  //after adding all the possible actions  i can activate and deactivate them.
  if(!checkForRobot())
```

```
100     {
          cout << "Robot doesn't exist or is not connected:"
          <<" Will not fire command. (SbActionML)"
          << endl;
          sendMsgToHandler("SEND|MSG|DISCONNECTED||");
105       return;
        }
        // add our actions in a good order, the integer here is the priority,
        // with higher priority actions going first
        // check type of msg
110     type = msg.getType();
        robot->lock();
        if(type == "AGOTO")
        {
          robot->clearDirectMotion();
115       //robot->moveTo(ArPose(0,0,0));
          //goTo.activate();
          //goTo.cancelGoal();
          //goTo.go(ArPose(msg.getArgAsDbl(0), msg.getArgAsDbl(1),
          //msg.getArgAsDbl(2)));
120     }
        else if(type == "AROTATE")
        {
          //robot->move(msg.getArgAsDbl(0));
          robot->clearDirectMotion();
125       rotate.go(msg.getArgAsDbl(0));
        }

        else if(type == "AMOVE")
        {
130       robot->clearDirectMotion();
          //amove moves along the X axis.
          //robot->moveTo(ArPose(0,0,0));
          move.activate();
          move.cancelGoal();
135       move.go(msg.getArgAsDbl(0));
        }
        else if(type == "STOP")
        {
          rotate.go(0);
140       goTo.cancelGoal();
          move.cancelGoal();
        }
        //add the actions for wander
        // if we're stalled we want to back up and recover
145     // gWander->addAction(new ArActionStallRecover, 100);
        // react to bumpers
        // gWander->addAction(new ArActionBumpers, 75);
        // avoid things closer to us
        else if(type == "ABUMPER")
150     {
          if(msg.getArgAsInt(0))
            robot->addAction(& bumpers, 90);
          else
            robot->remAction(& bumpers);
155     }

        robot->unlock();
        cout << "Firing Action with msg : "<< msg.getAsStr() << endl;
      }
160
```

```
#ifndef SbActionMove_h
#define SbActionMove_h

#include "ariaUtil.h"
#include "ArAction.h"

/// Action that moves the robot forward and backward.
/// SbActionMove drives straight by a given distance. The action stops when it
/// gets closeDist away. You can give it a new goal with setGoal(), cancel its
/// movement with cancelGoal(), and see if it got there with haveAchievedGoal().
/// This doesn't avoid obstacles, you could have an avoid routine at a higher
/// priority to avoid on the way there.
/// This action was created to be used with other actions to replace
/// direct motion MOVE command.It's basically made so that you can just have a
/// ton of limiters of different kinds and types while using fundamental
/// control of the robot.
/// @author Marcin Balicki : Cooper Union - Mechanical Engineering
/// @date March 2004
/// @bug None


class SbActionMove : public ArAction
{
public:
  ///Constructor
  SbActionMove(const char * name = "goto", double closeDist = 200);
  virtual ~SbActionMove();
  /// Sees if the goal has been achieved
  bool haveAchievedGoal(void);
  /// Cancels the goal the robot has
  void cancelGoal(void);
  /// Sets a new goal and sets the action to go there
  void setGoal(ArPose goal);

  /// Gets the goal the action has
  ArPose getGoal(void)
  { return myGoal; }

  /// Set the distance which is close enough to the goal (mm);
  void setCloseDist(double closeDist)
  { myCloseDist = closeDist; }

  /// Gets the distance which is close enough to the goal (mm)
  double getCloseDist(void)
  { return myCloseDist; }

  /// Sets the speed the action will travel to the goal at (mm/sec)
  void setSpeed(double speed)
  { mySpeed = speed; }

  /// Gets the speed the action will travel to the goal at (mm/sec)
  double getSpeed(void)
  { return mySpeed; }

  /// go
  void go(double dist);
  /// implementation of base fire function
  virtual ArActionDesired * fire(ArActionDesired currentDesired);

  /// implementation for the base getDesired action
  virtual ArActionDesired * getDesired(void)
  { return & myDesired; }

protected:
  /// allow distance within the goal as success
  double myCloseDist;
  /// desired speed
  double mySpeed;
  /// incrememnt amount to turn to achieve heading
  double myDirectionToTurn;
  /// the current direction
  double myCurDir;
  /// the last distance to goal.
  double myOldDist;
  /// turn back flag
  bool myTurnedBack;
  /// internal action
  ArActionDesired myDesired;
  /// old position
  ArPose myOldGoal;
  /// requested goal
  ArPose origGoal;
  ///  goal
  ArPose myGoal;

  /// enumerates the current state of this action
  enum State
  { STATE_NO_GOAL, STATE_ACHIEVED_GOAL, STATE_GOING_TO_GOAL};

  /// instance of the state enumerator
  State myState;
};

#endif // SbActionMove
```

```
/* This class was addopted from SbActionMove created by activmedia. */

#include "SbActionMove.h"
#include "ArRobot.h"

SbActionMove::SbActionMove( const char * name, double closeDist ) : ArAction( name
, "Goes to the given goal.")
{
  myDirectionToTurn= 1;

  setNextArgument( ArArg( "close dist", & myCloseDist, "Distance that is close
enough to goal. (mm)") );
  myCloseDist = closeDist;
  setGoal( ArPose( 0.0, 0.0, 0.0 ) );
  mySpeed = 2000;
  myOldDist = 10000000;
}

SbActionMove::~SbActionMove()
{
}

bool SbActionMove::haveAchievedGoal( void )
{
  if ( myState == STATE_ACHIEVED_GOAL)
    return true;
  else
    return false;
}

void SbActionMove::cancelGoal( void )
{
  myState = STATE_NO_GOAL;
}

void SbActionMove::go( double dist )
{

  myOldDist = 10000000; //reset it.
  if ( dist >= 0 )
    myCurDir = 1;
  else
    myCurDir = -1;
  ArPose goal = myRobot->getPose();
  goal.setX( goal.getX() + dist );
  //printf("goal : %.0f  ", goal.getX());
  setGoal( goal );

}

void SbActionMove::setGoal( ArPose goal )
{
  myState = STATE_GOING_TO_GOAL;
  myGoal = goal;
  myOldGoal = myGoal;
}

ArActionDesired * SbActionMove::fire( ArActionDesired currentDesired )
{
  double dist;
  double vel;
  double angle;

  //this checks if the new goal set is just from
  if ( myGoal.findDistanceTo( myOldGoal ) > 5 )
    setGoal( myGoal );

  // if we're there we don't do anything
  if ( myState == STATE_ACHIEVED_GOAL || myState == STATE_NO_GOAL )
    return NULL;
  printf( "#OldDistance %.0f  ", myOldDist );

  dist = myRobot->getPose().findDistanceTo( myGoal );
  printf( "Dist : %.0f  ", dist );
  // see where we want to point
  angle = myRobot->getPose().findAngleTo( myGoal );
  printf( "AngleToGoal : %.0f  ", angle );
  //this is in case we over shoot.
  //if the difference is positive than we are moving towards the goal.
  //else we should stop.
  if ( ( myOldDist - dist ) < 0 )
  {
    printf( "Achieved goal\n");
    myState = STATE_ACHIEVED_GOAL;
    myDesired.setVel( 0 );
    myDesired.setDeltaHeading( 0 );
    myOldDist = 10000000; //reset it.
    return & myDesired;
  }
  //do it again...but this time to turn..if we overshot do not turn.
  //else
  //myDesired.setHeading(angle);
  if ( dist < ( myCloseDist * 0.25 ) && ArMath::fabs( myRobot->getVel() < 5 ) )
  {
    printf( "#achieved\n" );
    myState = STATE_ACHIEVED_GOAL;
    myDesired.setVel( 0 );
    myDesired.setDeltaHeading( 0 );
    myOldDist = 10000000; //reset it.
```

```
      }
      else if ( dist < myCloseDist )
100   {
        // if (dist < 60){
        //     printf("#Velocity %.0f\n", vel);
        //    myDesired.setVel(0);
        //     }
105     // else
        // {
        vel = sqrt( dist * 10 );
        printf( "#Velocity %.0f\n", vel );
        myDesired.setVel( vel * myCurDir );
110     //}
      }
      else
      {
        vel = sqrt( dist * 200 * 2 );
115     if ( vel > mySpeed )
          vel = mySpeed;
        printf( "#Velocity %.0f\n", vel );
        myDesired.setVel( vel * myCurDir );
      }
120   myOldDist = dist;
      return & myDesired;
    }
```

```
#ifndef SbActionRotate_h
#define SbActionRotate_h

#include "ArAction.h"

/// Action that rotates the robot.
/// SbActionRotate was required because the direct motion ROTATE command
/// rotates in the direction of the smallest possible motion to achieve the
/// required heading; specifically in the case the angle is greater then 180
/// or less than -180. This prevented the user to rotate the robot in one
/// direction continuesly until goal is achieved. , ie 750 degrees.
/// This action is basically made so that you can just have a ton of
/// limiters of different kinds and types while using fundamental control of
/// the robot.
/// @author Marcin Balicki : Cooper Union - Mechanical Engineering
/// @date March 2004
/// @bug None

class SbActionRotate : public ArAction
{
public:
  /// Constructor
  SbActionRotate( const char * name = "turn", double turnAmount = 15 );
  /// Destructor
  virtual ~SbActionRotate();
  /// implementation of base fire function, gets fired every cycle.
  virtual ArActionDesired * fire( ArActionDesired currentDesired );
  /// accesor for action system
  virtual ArActionDesired * getDesired( void )
  {
    return & myDesired;
  }
  /// this function is what users uses to fire this action. Arg: Degrees.
  void go( double );
protected:
  /// the incremental rotation amount
  double myTurnAmount;
  /// internal
  ArActionDesired myDesired;
  /// current angle to goal
  double goalAngle;
  /// older angle to goal.
  double oldAngle;
};

#endif // SbActionRotate
```

```
#include "SbActionRotate.h"
#include "ArRobot.h"
#include <string>
using namespace std;

SbActionRotate::SbActionRotate( const char * name, double turnAmount ) :
     ArAction( name, "Rotates the robot.")
     {
       setNextArgument( ArArg( "turn amount", & myTurnAmount, " turning" ) );
       myTurnAmount = turnAmount;
}
SbActionRotate::~SbActionRotate()
{
}
void SbActionRotate::go( double amount )
{
  goalAngle = amount;
  // myRobot->lock();
  oldAngle = myRobot->getTh();
  //myRobot->unlock();
}
ArActionDesired * SbActionRotate::fire( ArActionDesired currentDesired )
{
  myDesired.reset();
  //reduce the goal angle by the amount moved, and try to move another 15
  // unless the goal angle is less than 15 then
  //move that left over angle.
  //is positive turn +10
  //if negative turn -10
  double ang = 0;
  //if 0 no need to do anything.
  if ( goalAngle == 0 ) return NULL;

  if ( ArMath::fabs( goalAngle ) < 180 )
  {
    myDesired.setDeltaHeading( goalAngle );
    goalAngle = 0;
  }
  // if the angle is positive
  //keep turning until we get below 180
  else if ( goalAngle >= 180 )
  {
    //if same sign just subract
    if ( myRobot->getTh() >= 0 && oldAngle >= 0 )
      ang = myRobot->getTh() - oldAngle;
    else if ( myRobot->getTh() <= 0 && oldAngle <= 0 )
      ang = myRobot->getTh() - oldAngle;
    else if ( myRobot->getTh() < 0 && oldAngle > 0 )
      ang = 360 + myRobot->getTh() - oldAngle;
    else if ( myRobot->getTh() >= 0 && oldAngle <= 0 )
      ang = myRobot->getTh() - oldAngle;
    goalAngle -= ang;
    cout << goalAngle << endl;
    oldAngle = myRobot->getTh();
    myDesired.setDeltaHeading( 100 );
  }
  //want to rotate in negative direction now.
  else // (goalAngle <=-180)
  {
    if ( myRobot->getTh() >= 0 && oldAngle >= 0 )
    {
      cout << "POS POS " << endl;
      ang = oldAngle - myRobot->getTh();
    }
    else if ( myRobot->getTh() <= 0 && oldAngle <= 0 )
    {
      cout << "NEG NEG " << endl;
      ang = -myRobot->getTh() + oldAngle;
    }
    else if ( myRobot->getTh() <= 0 && oldAngle >= 0 )
    {
      cout << "POS NEG " << endl;
      ang = oldAngle - myRobot->getTh();
    }
    else if ( myRobot->getTh() >= 0 && oldAngle <= 0 )
    {
      cout << "NEG POS " << endl;
      ang = 360 - myRobot->getTh() + oldAngle;
    }
    goalAngle += ang;
    cout << goalAngle << endl;
    oldAngle = myRobot->getTh();
    myDesired.setDeltaHeading( -100 );
  }

  return & myDesired;
}
```

```
#ifndef SbCameraSONYML_h
#define SbCameraSONYML_h

#include "SbMsgListener.h"
#include "SbMessage.h"
#include "Aria.h"

#include <string>

using namespace std;

/// Manages Sony EVI-D30 PTZ (Pan Tilt Zoom) camera.
/// This class controls functions of the camera attached to the serial port on
/// the robot. It controls all the functions available for the camera.
/// (Pan/Tilt/Zoom etc).
/// @author Marcin Balicki : Cooper Union - Mechanical Engineering
/// @date March 2004
/// @warning Uses SbMsgListener's SbMsgListener::sendMsgToHandler()
/// @todo Would be nice to figure out how to test which type of camera is
/// connected to this robot and then create the appropriate control class.
/// @bug None


class SbCameraSONYML : public SbMsgListener
{
public:
  ///Constructor
  SbCameraSONYML( void );
  ///Destructor
  ~SbCameraSONYML( void );
  /// implements the SbMsgListener pure virtual function.
  /// @param msg SbMessage which caused the listener to fire.
  virtual void fire( const SbMessage & msg);
protected:
  /// set the max / min variables.
  bool init( void );
  /// check if the robot exists and is connected.
  bool checkForRobot( void );
  /// tilts the camera +- 100.0
  void tilt( double );
  /// pans the camera + - 100.0
  void pan( double );
  /// pans relateviely to current position + - 100.0
  void panRel( double );
  /// tilts relateviely to current position + - 100.0
  void tiltRel( double );
  /// zoom position + - 100.0
  void zoom( double );
  /// zoom relateviely to current position + - 100.0
  void zoomRel( double );
  /// returns the current value of Pan/Tilt/Zoom
  /// pan(0) tilt(1) zoom(2)
  double getState( int f);
  /// internal robot pointer
  ArRobot * robot;
  /// message type
  string type;
  /// SONY camera control class
  ArSonyPTZ * camera;
  /// maximum Pan position
  int maxPanPos;
  /// maximum Tilt position
  int maxTiltPos;
  /// minimum Pan position
  int minPanPos;
  /// minimum Tilt position
  int minTiltPos;
  /// minimum Zoom value
  int minZoom;
  /// maximum Zoom value
  int maxZoom;
  /// minimum Pan speed
  int minPanSlew;
  /// maximum Pan speed
  int maxPanSlew;
  /// minimum Tilt speed
  int minTiltSlew;
  /// maximum Titl speed
  int maxTiltSlew;
};

#endif
```

```
#include "SbCameraSONYML.h"
//should not sleep while the robot is locked!!!
SbCameraSONYML::SbCameraSONYML(void)
{   camera = NULL;
  //CAM|INIT,ZOOM,PAN,TILT|+-100||
  addMsgType("CAM");
 }

SbCameraSONYML::~SbCameraSONYML(void)
{ }

bool SbCameraSONYML::checkForRobot(void)
{
  //if it does and its connected than return true;
  //it it exists and its the same as the old one just return the
  //Check if the robot exists if not return false
  ArRobot * r;
  if((r = Aria::findRobot("SBROBOT")) == NULL)
    return false;
  //if it exists check if it is different than the last definition of the robot.
  //if it is update the robot pointer and then initilize
  else if(r != robot)
  {
    robot = r;
    //has to be connected first.
    robot->lock();
    //if the camera not instantiated then do not delete
    if(!(camera == NULL))
      delete camera;
    camera = new ArSonyPTZ(robot);
    init();
    robot->unlock();
  }
  robot->lock();
  if(robot->isConnected())
  {
    robot->unlock();
    return true;
  }
  robot->unlock();
  return false;
}

bool SbCameraSONYML::init(void)
{
//  if(!checkForRobot())
//    return false;
  //should I lock the robot??
  cout << "Initializing Camera (SbCameraSONYML)" << endl;
  //ArUtil::sleep(10);
//  robot->lock();
  //need to get the max negative tilt and pan also.
  maxPanPos = camera->getMaxPosPan();
  minPanPos = camera->getMaxNegPan();
  minTiltPos = camera->getMaxNegTilt();
  maxTiltPos = camera->getMaxPosTilt();
  maxZoom = camera->getMaxZoom();
  minZoom = camera->getMinZoom();
  camera->zoom(0);
  camera->panTilt(0, 0);
//  robot->unlock();
  return true;
}

double SbCameraSONYML::getState(int f)
{
  double result = 0;
  double scale = 100;
  double tilt = 0;
  double pan = 0;
  robot->lock();
  switch(f)
  {

    case 0:
      pan = camera->getPan();
      if(pan >= 0)
        result = ((pan / maxPanPos) * scale);
      else
        result = ((pan / (-minPanPos)) * scale);
      break;
    case 1:
     tilt = camera->getTilt();
     if(tilt >= 0)
       result = ((tilt / maxTiltPos) * scale);
     else
       result = ((tilt / (-minTiltPos)) * scale);
     result = -result;
     break;
    case 2:
      result = camera->getZoom();
      break;
    default:
      result = 0;
  }
  robot->unlock();
  return result;
}
```

```
100  void SbCameraSONYML::pan(double x)
     {
       double hor = 0;
       double scale = 100.0;
       //if greater then zero
105    //what about if it is greater than scale??
       if(x > scale) x = scale;
       if(x < -scale) x = -scale;
       if(x >= 0) hor = (x / scale) * maxPanPos;
       else
110      hor = (x / scale) * (-minPanPos);
       robot->lock();
       camera->pan(hor);
       robot->unlock();
     }
115
     void SbCameraSONYML::tilt(double y)
     {
       double ver = 0;
       double scale = 100.0;
120    //if greater then zero
       //what about if it is greater than scale??
       if(y > scale) y = scale;
       else if(y < -scale) y = -scale;
       //cout<<"MAX: "<<maxTiltPos<<"MIN: "<<-minTiltPos <<endl;
125    if(y >= 0) ver = (y / scale) * maxTiltPos;
       else
         ver = (y / scale) * (-minTiltPos);
       //        cout<<endl<<"Tilt="<<ver<<endl;
       robot->lock();
130    camera->tilt(ver);
       robot->unlock();
     }

     void SbCameraSONYML::panRel(double x)
135  {
       double hor = 0;
       double scale = 100.0;
       //if greater then zero
       //what about if it is greater than scale??
140    if(x > scale) x = scale;
       if(x < -scale) x = -scale;
       if(x >= 0) hor = (x / scale) * maxPanPos;
       else
         hor = (x / scale) * (-minPanPos);
145    robot->lock();
       camera->panRel(hor);
       robot->unlock();
     }

150  void SbCameraSONYML::tiltRel(double y)
     {
       double ver = 0;
       double scale = 100.0;
       //if greater then zero
155    //what about if it is greater than scale??
       if(y > scale) y = scale;
       if(y < -scale) y = -scale;
       if(y >= 0) ver = (y / scale) * maxTiltPos;
       else
160      ver = (y / scale) * (-minTiltPos);
       robot->lock();
       camera->tiltRel(ver);
       robot->unlock();
     }
165
     void SbCameraSONYML::zoom(double z)
     {
       double scale = 100.0;
       if(z > scale) z = scale;
170    if(z < 0) z = 0;
       z = (z / scale) * maxZoom;
       robot->lock();
       camera->zoom(z);
       robot->unlock();
175  }

     void SbCameraSONYML::zoomRel(double z)
     {
       double scale = 100.0;
180    double zoom = 0;
       if(z > scale) z = scale;
       if(z < 0) z = 0;
       z = (z / scale) * maxZoom;
       robot->lock();
185    zoom = (camera->getZoom());
       zoom += z;
       if(zoom > maxZoom) zoom = maxZoom;
       else if(zoom < minZoom) zoom = minZoom;
       camera->zoom(zoom);
190    robot->unlock();
     }

     void SbCameraSONYML::fire(const SbMessage & msg)
     {
195    //all values should be scaled 0-100 or -100->100
       //should I make the user intialize the Camera?
       //or should it be automatic when the first command is called?
       //has to be connected first.
```

```
      if(!checkForRobot())
200   {
        cout << "Robot doesn't exist or is not connected:"
        <<" Will not fire command. (SbCameraSONYML)"
        << endl;
        sendMsgToHandler("SEND|MSG|DISCONNECTED||");
205     return;
      }
      type = msg.getType();
      if (!(type == "CAM"))
         return;
210
      //CAM|zoom|double|| zoom 0-100
      if(msg.getArgAsStr(0) == "zoom" )
        zoom(msg.getArgAsDbl(1));
      //CAM|zoomrel|double|| zooms relative to current position  +- 100
215   else if(msg.getArgAsStr(0)== "zoomrel" )
        zoomRel(msg.getArgAsDbl(1));
      //CAM|pan|double|| pans +- 100
      else if( msg.getArgAsStr(0) == "pan" )
        pan(msg.getArgAsDbl(1));
220   //CAM|panrel|double|| pans relative to current position  +- 100
      else if( msg.getArgAsStr(0) == "panrel" )
        panRel(msg.getArgAsDbl(1));
      //CAM|tilt|double|| tilts +- 100
      else if(msg.getArgAsStr(0)  ==  "tilt")
225     tilt(msg.getArgAsDbl(1));
      //CAM|tiltrel|double|| +- 100
      else if(msg.getArgAsStr(0)  == "tiltrel" )
        tiltRel(msg.getArgAsDbl(1));
      //CAM|backlight|int|| +- 100
230   else if("backlight" == msg.getArgAsStr(0))
      {
            robot->lock();
        switch(msg.getArgAsInt(1))
        {
235       case 1:
           camera->backLightingOn();
            break;
          case 0:
          default:
240        camera-> backLightingOff();
            break;
        }
        robot->unlock();
      }
245   cout << "Firing Camera with msg : "<< msg.getAsStr() << endl;
    }
```

```
#ifndef SbCameraVCC4ML_h
#define SbCameraVCC4ML_h

#include "SbMsgListener.h"
#include "SbMessage.h"
#include "Aria.h"

#include <string>

using namespace std;

/// Manages Cannon VC-C4 PTZ (Pan Tilt Zoom) camera.
/// This class controls functions of the camera attached to the serial port on
/// the robot. It controls all the functions available for the camera.
/// (Pan/Tilt/Zoom etc).
/// Cannon needs to be initilized proir to sending commands to it.
/// @author Marcin Balicki : Cooper Union - Mechanical Engineering
/// @date March 2004
/// @warning Uses SbMsgListener's SbMsgListener::sendMsgToHandler()
/// @todo Would be nice to figure out how to test which type of camera is
/// connected to this robot and then create the appropriate control class.
/// @bug None

class SbCameraVCC4ML : public SbMsgListener
{
public:
  ///Constructor
  SbCameraVCC4ML( void );
  ///Destructor
  ~SbCameraVCC4ML( void );
  /// implements the SbMsgListener pure virtual function.
  /// @param msg SbMessage which caused the listener to fire.
  virtual void fire( const SbMessage & msg);
protected:
  ///init the camera and set the max min variables.
  bool init( void );
  /// check if the robot exists and is connected.
  bool checkForRobot( void );
  /// tilts the camera +- 100.0
  void tilt( double );
  /// pans the camera + - 100.0
  void pan( double );
  /// pans relateviely to current position + - 100.0
  void panRel( double );
  /// tilts relateviely to current position + - 100.0
  void tiltRel( double );
  /// zoom position + - 100.0
  void zoom( double );
  /// zooms relateviely to current position + - 100.0
  void zoomRel( double );
  /// pan speed
  void panSlew( double );
  /// tilt speed
  void tiltSlew( double );
  /// returns the current value of Pan/Tilt/Zoom
  /// pan(0) tilt(1) zoom(2)
  double getState( int );
  /// internal robot pointer
  ArRobot * robot;
  /// message type
  string type;
  /// VCC4 canon camera control class
  ArVCC4 * camera;
  /// maximum Pan position
  int maxPanPos;
  /// maximum Tilt position
  int maxTiltPos;
  /// minimum Pan position
  int minPanPos;
  /// minimum Tilt position
  int minTiltPos;
  /// minimum Zoom value
  int minZoom;
  /// maximum Zoom value
  int maxZoom;
  /// minimum Pan speed
  int minPanSlew;
  /// maximum Pan speed
  int maxPanSlew;
  /// minimum Tilt speed
  int minTiltSlew;
  /// maximum Tilt speed
  int maxTiltSlew;
};

#endif
```

```
#include "SbCameraVCC4ML.h"
//should not sleep while the robot is locked!!!
SbCameraVCC4ML::SbCameraVCC4ML(void)
{
  camera = NULL;
  //CAM|INIT,ZOOM,PAN,TILT|+-100||
  addMsgType("CAM");
}

SbCameraVCC4ML::~SbCameraVCC4ML(void)
{ }

bool SbCameraVCC4ML::checkForRobot(void)
{
  //if it does and its connected than return true;
  //it it exists and its the same as the old one just return the
  //Check if the robot exists if not return false
  ArRobot * r;
  if((r = Aria::findRobot("SBROBOT")) == NULL)
    return false;
  //if it exists check if it is different than the last definition of the robot.
  //if it is update the robot pointer and then initilize
  else if(r != robot)
  {
    robot = r;
    //has to be connected first.
    robot->lock();
    //if camera not instantiated then do not delete
    if(!(camera == NULL))
      delete camera;
    // true sets the fact that the camera is inverted.
    camera = new ArVCC4(robot, true);
    robot->unlock();
  }
  robot->lock();
  if(robot->isConnected())
  {
    robot->unlock();
    return true;
  }
  robot->unlock();
  return false;
}

bool SbCameraVCC4ML::init(void)
{
  if(!checkForRobot())
    return false;
  //should I lock the robot??
  cout << "Initializing Camera (SbCameraVCC4ML)" << endl;
  robot->lock();
  camera->power(1);
  robot->unlock();
  ArUtil::sleep(2000); //should not sleep while the robot is locked!!!
  robot->lock();
  camera->init();
  robot->unlock();
  //cout<<"PTU ini"<<endl;
  ArUtil::sleep(1000);
  robot->lock();
  //need to get the max negative tilt and pan also.
  maxPanPos = camera->getMaxPosPan();
  minPanPos = camera->getMaxNegPan();
  minTiltPos = camera->getMaxNegTilt();
  maxTiltPos = camera->getMaxPosTilt();
  maxZoom = camera->getMaxZoom();
  minZoom = camera->getMinZoom();
  minPanSlew = camera->getMinPanSlew();
  maxPanSlew = camera->getMaxPanSlew();
  minTiltSlew = camera->getMinTiltSlew();
  maxTiltSlew = camera->getMaxTiltSlew();
  camera->zoom(0);
  camera->panTilt(0, 0);
  bool ret = (camera->isInitted());
  robot->unlock();
  return ret;
}

double SbCameraVCC4ML::getState(int f)
{
  double result = 0;
  double scale = 100;
  double tilt = 0;
  double pan = 0;
  robot->lock();
  switch(f)
  {
    case 1:
      tilt = camera->getTilt();
      if(tilt >= 0)
        result = ((tilt / maxTiltPos) * scale);
      else
        result = ((tilt / (-minTiltPos)) * scale);
      result = -result;
      break;
    case 0:
      pan = camera->getPan();
      if(pan >= 0)
        result = ((pan / maxPanPos) * scale);
```

```
        else
          result = ((pan / (-minPanPos)) * scale);
        break;
      case 2:
        result = camera->getZoom();
        break;
      default:
        result = 0;
    }
    robot->unlock();
    return result;
}

void SbCameraVCC4ML::pan(double x)
{
  double hor = 0;
  double scale = 100.0;
  //if greater then zero
  //what about if it is greater than scale??
  if(x > scale) x = scale;
  if(x < -scale) x = -scale;
  if(x >= 0) hor = (x / scale) * maxPanPos;
  else
    hor = (x / scale) * (-minPanPos);
  robot->lock();
  camera->pan(hor);
  robot->unlock();
}

void SbCameraVCC4ML::tilt(double y)
{
  double ver = 0;
  double scale = 100.0;
  //if greater then zero
  //what about if it is greater than scale??
  if(y > scale) y = scale;
  else if(y < -scale) y = -scale;
  //cout<<"MAX: "<<maxTiltPos<<"MIN: "<<-minTiltPos <<endl;
  if(y >= 0) ver = (y / scale) * maxTiltPos;
  else
    ver = (y / scale) * (-minTiltPos);
  //        cout<<endl<<"Tilt="<<ver<<endl;
  robot->lock();
  camera->tilt(ver);
  robot->unlock();
}

void SbCameraVCC4ML::panRel(double x)
{
  double hor = 0;
  double scale = 100.0;
  //if greater then zero
  //what about if it is greater than scale??
  if(x > scale) x = scale;
  if(x < -scale) x = -scale;
  if(x >= 0) hor = (x / scale) * maxPanPos;
  else
    hor = (x / scale) * (-minPanPos);
  robot->lock();
  camera->panRel(hor);
  robot->unlock();
}

void SbCameraVCC4ML::tiltRel(double y)
{
  double ver = 0;
  double scale = 100.0;
  //if greater then zero
  //what about if it is greater than scale??
  if(y > scale) y = scale;
  if(y < -scale) y = -scale;
  if(y >= 0) ver = (y / scale) * maxTiltPos;
  else
    ver = (y / scale) * (-minTiltPos);
  robot->lock();
  camera->tiltRel(ver);
  robot->unlock();
}

void SbCameraVCC4ML::zoom(double z)
{
  double scale = 100.0;
  if(z > scale) z = scale;
  if(z < 0) z = 0;
  z = (z / scale) * maxZoom;
  robot->lock();
  camera->zoom(z);
  robot->unlock();
}

void SbCameraVCC4ML::zoomRel(double z)
{
  double scale = 100.0;
  double zoom = 0;
  if(z > scale) z = scale;
  if(z < 0) z = 0;
  z = (z / scale) * maxZoom;
  robot->lock();
  zoom = (camera->getZoom());
```

```
    zoom += z;
    if(zoom > maxZoom) zoom = maxZoom;
    else if(zoom < minZoom) zoom = minZoom;
    camera->zoom(zoom);
    robot->unlock();
  }

  void SbCameraVCC4ML::panSlew(double s)
  {
    double slew = 0;
    double scale = 100.0;
    //if greater then zero
    //what about if it is greater than scale??
    if(s > scale) s = scale;
    if(s < -scale) s = -scale;
    if(s >= 0) slew = (s / scale) * maxPanSlew;
    else
      slew = (s / scale) * (-minPanSlew);
    robot->lock();
    camera->panSlew(slew);
    robot->unlock();
  }

  void SbCameraVCC4ML::tiltSlew(double s)
  {
    double slew = 0;
    double scale = 100.0;
    //if greater then zero
    //what about if it is greater than scale??
    if(s > scale) s = scale;
    if(s < -scale) s = -scale;
    if(s >= 0) slew = (s / scale) * maxPanSlew;
    else
      slew = (s / scale) * (-minPanSlew);
    robot->lock();
    camera->tiltSlew(slew);
    robot->unlock();
  }

  void SbCameraVCC4ML::fire(const SbMessage & msg)
  {
    //all values should be scaled 0-100 or -100->100
    //should I make the user intialize the Camera?
    //or should it be automatic when the first command is called?
    //has to be connected first.
    if(!checkForRobot())
    {
      cout << "Robot doesn't exist or is not connected:"
      <<" Will not fire command. (SbCameraVCC4ML)"
      << endl;
      sendMsgToHandler("SEND|MSG|DISCONNECTED||");
      return;
    }
    //check type of msg

    type = msg.getType();
    if(!(type == "CAM"))
      return;
    //CAM|init|| initializes the camera
    if(msg.getArgAsStr(0) == "init")
      init();
    //CAM|zoom|double|| zoom 0-100
    else if(msg.getArgAsStr(0) == "zoom")
      zoom(msg.getArgAsDbl(1));
    //CAM|zoomrel|double|| zooms relative to current position  +- 100
    else if(msg.getArgAsStr(0) == "zoomrel")
      zoomRel(msg.getArgAsDbl(1));
    //CAM|pan|double|| pans +- 100
    else if(msg.getArgAsStr(0) == "pan")
      pan(msg.getArgAsDbl(1));
    //CAM|panrel|double|| pans relative to current position  +- 100
    else if(msg.getArgAsStr(0) == "panrel")
      panRel(msg.getArgAsDbl(1));
    //CAM|tilt|double|| tilts +- 100
    else if(msg.getArgAsStr(0) == "tilt")
      tilt(msg.getArgAsDbl(1));
    //CAM|tiltrel|double|| +- 100
    else if(msg.getArgAsStr(0) == "tiltrel")
      tiltRel(msg.getArgAsDbl(1));
    //CAM|stop||
    else if(msg.getArgAsStr(0) == "stop")
    {
      robot->lock();
      camera->haltPanTilt();
      camera->haltZoom();
      robot->unlock();
    }
    // CAM|panslew|double|| speed of panning
    else if(msg.getArgAsStr(0) == "panslew")
      panSlew(msg.getArgAsDbl(1));
    // CAM|tiltslew|double|| speed of tilting
    else if(msg.getArgAsStr(0) == "tiltslew")
      tiltSlew(msg.getArgAsDbl(1));
    cout << "Firing Camera with msg : "<< msg.getAsStr() << endl;
  }
```

```
#ifndef SbExampleML_h
#define SbExampleML_h

#include "SbMsgListener.h"
#include "SbMessage.h"
#include "Aria.h"

#include <sstream>
#include <string>

using namespace std;

/// Example implementation of a class that inherits SbMsgListener.
/// A template example, just add your code in the fire() function.
/// You can use the sendMsgToHandler() to fire messages back to the handler.
/// @author Marcin Balicki : Cooper Union - Mechanical Engineering
/// @date March 2004
/// @warning Uses SbMsgListener's SbMsgListener::sendMsgToHandler()
/// @note You can use the sendMsgToHandler to fire messages back to the handler.
/// @bug None
class SbExampleML : public SbMsgListener
{
public:
  ///Constructor
  SbExampleML(void);
  ///Destructor
  ~SbExampleML(void);
  ///converts double to a string.
  string toString(double d);
  /// implements the SbMsgListener pure virtual function.
  /// @param &msg SbMessage which caused the listener to fire.
  virtual void fire(const SbMessage & msg);
protected:
  ///robot pointer
  ArRobot * robot;
  ///checks if the robot (ARIA instance) exists and is connected
  bool checkForRobot(void);
  ///type of message
  string type;
  ///internal message
  SbMessage msgExample;
  ///stream instance used to convert double to string
  ostringstream oss;
};

#endif
```

```
#include "SbExampleML.h"

SbExampleML::SbExampleML(void)
{
  //adds message types to the msgListener (base) vector for registering with
  //msg handler.
  addMsgType("EXAMPLE");

}

SbExampleML::~SbExampleML(void)
{ }

//Checks if an instance of the robot exists and if it is connected.
//this is necessary if we are going to use
bool SbExampleML::checkForRobot(void)
{
  //Check if the robot(SBROBOT ) exists, if yes and its connected
  //than return true;
  if((robot = Aria::findRobot("SBROBOT")) == NULL)
    return false; //quit this function because of error

  //lock the robot se we can get data from it
  robot->lock();
  if(robot->isConnected())
  {
    //if we are connected unlock and quit function
    robot->unlock();
    return true;
  }
  //we are not connected so unlock the robot and return false.
  robot->unlock();
  return false;
}

//convert a double to a string
string SbExampleML::toString(double d)
{
  oss.str("");
  oss << d; // insert double into stream
  //return the double as a string.
  return oss.str();
}

//this is the handler fire function
void SbExampleML::fire(const SbMessage & msg)
{
  cout << "Firing Example with msg : "<< msg.getAsStr() << endl;
  //has to be connected first.
  if(!checkForRobot())
  {
    cout << "Robot doesn't exist or is not connected:"
    << " Will not fire command. (SbExampleML)"
    << endl;
    sendMsgToHandler("SEND|MSG|DISCONNECTED||");
    return;
  }

  //now test the type of msg that the listener was associated with
  //and fire appropriate function.
  // EXAMPLE will send EXAMPLE back to the handler with some robot state values
  if(msg.getType() == "EXAMPLE")
  {
    msgExample.clear();
    //set the msg type as a SEND which fires a SEND event...
    msgExample.setType("SEND");
    //send will strip "SEND" and create EXAMPLE as a new msg
    msgExample.addArgument("EXAMPLE");
    //with the following arguments:
    //get some data from the robot
    robot->lock();
    msgExample.addArgument(toString(robot->getX()));
    msgExample.addArgument(toString(robot->getY()));
    msgExample.addArgument(toString(robot->getTh()));
    msgExample.addArgument(toString(robot->getBatteryVoltage()));
    robot->unlock();
  }
  //send the new msg to the msgHandler to be fired it
  sendMsgToHandler(msgExample);
}
```

```
#ifndef SbExampleMLT_h
#define SbExampleMLT_h

#include "SbMsgListener.h"
#include "SbMessage.h"
//#include "ArSyncTask.h"
#include "Aria.h"

#include <string>
#include <sstream>
using namespace std;

/// Example implementation that inherits SbMsgListener and is threaded.
/// This class inherits Aria's threaded ArSyncTask. It displays a possible
/// way to use a separate thread to do something (calculation, control another
/// process, etc) while the robot is running.
/// @author Marcin Balicki : Cooper Union - Mechanical Engineering
/// @date March 2004
/// @warning Be careful with locking and unlocking.
/// @warning This class is threaded and therefore should not use
/// sendMsgToHandler to fire messages back to the handler because handler
/// is not thread safe unless used from called from managerT
/// @note You can't use the sendMsgToHandler to fire messages back to the
/// handler.
/// @todo Instead of making SbMsgHandlerSingleton as the singleton, the manager
/// class should be one, and the reference to it should be possessed by all
/// SbMsgListeners. However ,this relations would be forced since listener
/// is far away in function from manager (listener->handler->manager).
/// another way is to add lock/unlock feature to the handler.
/// @bug None


class SbExampleMLT : public ArASyncTask , public SbMsgListener
{
public:
  //Constructor
  SbExampleMLT(void);
  //Destructor
  ~SbExampleMLT(void);
  ///converts double to a string.
  string toString(double d);
  /// implements the SbMsgListener pure virtual function.
  /// @param &msg SbMessage which caused the listener to fire.
  virtual void fire(const SbMessage &msg);
  ///implements the ArSyncTask main thread function
  /// @param *arg that does nothing.
  virtual void * runThread(void * arg);

protected:
  ///robot pointer
  ArRobot * robot;
  ///checks if the robot (ARIA instance) exists and is connected
  bool checkForRobot(void);
  ///type of message
  string type;
  ///internal most current message
  SbMessage msgExample;
  ///stream instance used to convert double to string
  ostringstream oss;
  /// new message flag
  bool newMsg;

};
#endif
```

```cpp
#include "SbExampleMLT.h"

SbExampleMLT::SbExampleMLT()
{
  //start the thread
  create( true, true );
  newMsg = false;
  //adds message types to the msgListener (base) vector for registering with
  // msg handler.
  addMsgType("EXAMPLETHREAD");

}

SbExampleMLT::~SbExampleMLT()
{
}
//Checks if an instance of the robot exists and if it is connected.
//this is necessary if we are going to use
bool SbExampleMLT::checkForRobot(void)
{
  //Check if the robot(SBROBOT ) exists, if yes and its connected
  //than return true;
  if((robot = Aria::findRobot("SBROBOT")) == NULL)
    return false; //quit this function because of error

  //lock the robot se we can get data from it
  robot->lock();
  if(robot->isConnected())
  {
    //if we are connected unlock and quit function
    robot->unlock();
    return true;
  }
  //we are not connected so unlock the robot and return false.
  robot->unlock();
  return false;
}

//convert a double to a string
string SbExampleMLT::toString(double d)
{
  oss.str("");
  oss << d; // insert double into stream
  //return the double as a string.
  return oss.str();
}

//this class will not be called by the local thread so we can
//acccess it as long as we use luck/unlock within it to access local data.
void SbExampleMLT::fire( const SbMessage & msg )
{
  msgExample=msg;
  cout << "Firing STATE with msg : "<< msg.getAsStr() << endl;
   //has to be connected first.
   if(!checkForRobot())
   {
     cout << "Robot doesn't exist or is not connected:"
    << "Will not fire command. (SbExampleMLT)"
     << endl;
     sendMsgToHandler("SEND|MSG|DISCONNECTED||");
     return;
   }

 //locks this class data so we can get anything we want.
  lock();
  //EXAMPLETHREAD will just let the thread now that we have received a msg.
  if ( msgExample.getType() == "EXAMPLETHREAD" )
     newMsg = true;
  //done
  unlock();
}
void * SbExampleMLT::runThread( void * arg )
{
   while ( myRunning )
  {
    //could check if the robot exists here if we are using it.
    ArUtil::sleep( 500 );
    //locks this class data so we can get anything we want.
    lock();
    //Check if there is a new msg
    if ( newMsg )
    {
      cout << "Thread found this message waiting for it:"
                                  << msgExample.getAsStr()<< endl;
      newMsg = false;
    }
    unlock();
  }
  return NULL;
}
```

```
#ifndef SbGripperML_H
#define SbGripperML_H

#include "SbMsgListener.h"
#include "SbMessage.h"
#include "Aria.h"

#include <string>
using namespace std;

/// Manages Gripper functions.
/// SbGripperML controls the gripper (Performance PeopleBot) if present.
/// It controls the lift, gripper arms and returns the state of the gripper
/// (open,close, liftmax, etc)
/// @author Marcin Balicki : Cooper Union - Mechanical Engineering
/// @date March 2004
/// @warning Uses SbMsgListener's SbMsgListener::sendMsgToHandler()
/// @note You can use the sendMsgToHandler to fire messages back to the handler.
/// @bug None

class SbGripperML : public SbMsgListener
{
public:
  ///Constructor
  SbGripperML(void);
  ///Destructor
  ~SbGripperML(void);
  ///implements the SbMsgListener pure virtual function.
  ///@param msg SbMessage which caused the listener to fire.
  virtual void fire(const SbMessage& msg );
private:
  ///internal robot pointer
  ArRobot * robot;
  ///message type
  string type;
  ///pointer to the gripper control class
  ArGripper * gripper;
  ///checks if the robot exists and is connected
  bool checkForRobot(void);
};

#endif
```

```cpp
#include "SbGripperML.h"

SbGripperML::SbGripperML(void)
{
  gripper = NULL;
  //adds message types to the msgListener (base) vector for registering with
  //msg handler.
  addMsgType("GRIPPER");
  addMsgType("GRIP");
  addMsgType("LIFT");
}

SbGripperML::~SbGripperML(void)
{
  //if the gripper is  not null than delete
  if(!(gripper == NULL))
    delete gripper;
}

bool SbGripperML::checkForRobot(void)
{
  //if it does and its connected than return true;
  //it it exists and its the same as the old one just return the
  //Check if the robot exists if not return false
  ArRobot * r;
  if((r = Aria::findRobot("SBROBOT")) == NULL)
    return false;
  //if it exists check if it is different than the last definition of the robot.
  //if it is update the robot pointer and then initilize
  else if(r != robot)
  {
    robot = r;
    //has to be connected first.
    robot->lock();
    //if the gripper is  not null than delete
    if(!(gripper == NULL))
      delete gripper;
    gripper = new ArGripper(robot);
    robot->unlock();
  }
  robot->lock();
  if(robot->isConnected())
  {
    robot->unlock();
    return true;
  }
  robot->unlock();
  return false;
}

void SbGripperML::fire(const SbMessage & msg)
{
  cout << "Firing Gripper with msg : " << msg.getAsStr() << endl;

  //should I make the user intialize the gripper?
  //or should it be automatic when the first command is called?
  //need to work on this because the if the user sends a grip
  //command and the pointer is null program crashses
  //or hangs rather.
  //need to have a safer method here.
  //has to be connected first.
  if(!checkForRobot())
  {
    cout << "Robot doesn't exist or is not connected:"
    <<" Will not fire command. (SbGripperML)"
    << endl;
    sendMsgToHandler("SEND|MSG|DISCONNECTED||");
    return;
  }
  //bool getHasGripper (void) const
  //Gets the gripper value (whether or not the robot has a gripper).
  robot->lock();
  if(gripper->getType() == ArGripper::NOGRIPPER)
  {
    cout << "No Gripper: Will not fire command. (SbGripperML)" << endl;
    sendMsgToHandler("SEND|MSG|NOGRIPPER||");
    robot->unlock();
    return;
  }
  //check type of msg
  type = msg.getType();

  // LIFT|int|| or LIFT|string||
  // lifts the gripper up and down  0=stop,1=up,2=down either int or string can
  // used
  if(type == "LIFT")
  {
    if("stop" == msg.getArgAsStr(0) || 1 == msg.getArgAsInt(0))
      gripper->liftStop();
    else if("up" == msg.getArgAsStr(0) || 2 == msg.getArgAsInt(0))
      gripper->liftUp();
    else if("down" == msg.getArgAsStr(0) || 3 == msg.getArgAsInt(0))
      gripper->liftDown();
  }
  // GRIP|int|| or GRIP|string|| closes and opens the gripper.
  // 0=stop,1=open,2=close either int or string can be  used
  else if(type == "GRIP")
  {
    if("stop" == msg.getArgAsStr(0) || 1 == msg.getArgAsInt(0))
```

```
      gripper->gripStop();
    else if("open" == msg.getArgAsStr(0) || 2 == msg.getArgAsInt(0))
      gripper->gripOpen();
    else if("close" == msg.getArgAsStr(0) || 3 == msg.getArgAsInt(0))
      gripper->gripClose();
  }
  // GRIPPER|int|| or GRIPPER|string|| stops, stores the whole gripper arm.
  // 0=halt,1=store,2=ready either int or string can be  used
  else if(type == "GRIPPER")
  {
    if("halt" == msg.getArgAsStr(0) || 1 == msg.getArgAsInt(0))
      gripper->gripperHalt();
    else if("store" == msg.getArgAsStr(0) || 2 == msg.getArgAsInt(0))
      gripper->gripperStore();
    else if("ready" == msg.getArgAsStr(0) || 3 == msg.getArgAsInt(0))
      gripper->gripperDeploy();
  }
  // GRIPPERSTATE returns information about the gripper
  // outerBreakBeamState|innerBreakBeamState|
  // openclosestate|isMaxed|leftPaddletrig|rightPaddletrig||
  // 0=between   1=open  2=closed
  // 1=yes 0=no
  else if(type == "GRIPPERSTATE")
  {
    SbMessage msg("SEND");
    msg.addArgument("MSG");
    msg.addArgument("GRIPPERSTATE");

    if(gripper->getBreakBeamState() & 2) // outer
          msg.addArgument("1");
    else
      msg.addArgument("0");

    if(gripper->getBreakBeamState() & 1) // inner
          msg.addArgument("1");
    else
      msg.addArgument("0");
    //    0=between   1=open  2=closed
    msg.addArgument(SbMessage::convToString(gripper->getGripState()));

    if(gripper->isLiftMaxed()) // lift
          msg.addArgument("1"); //max
    else
      msg.addArgument("0");

    int val = gripper->getPaddleState(); // paddle section
    if(val & 1) // left paddle
          msg.addArgument("1");
    else
      msg.addArgument("0");

    if(val & 2) // right paddle
          msg.addArgument("1");
    else
      msg.addArgument("0");

    robot->unlock();
    sendMsgToHandler(msg);
    return;
   }
   robot->unlock();

}
```

```
#ifndef SbManagerT_h
#define SbManagerT_h

#include "SbRobotML.h"
#include "SbServerML.h"
#include "SbMsgListener.h"
#include "SbMsgHandlerSingleton.h"
#include "SbMessage.h"

#include "Aria.h"

/// Main manager class of RCCI (threaded).
/// This is a threaded class which is the managing body of the software,
/// supervising all the high level functions of RCCI. It takes care of timed
/// TCP server updates, checking the robot state and shutting down RCCI and
/// registering SbMsgListeners with SbMsgHandlerSingleton.
/// @author Marcin Balicki : Cooper Union - Mechanical Engineering
/// @date March 2004
/// @warning a Singleton class needs to be destroyed by the last thread
/// alive by calling removeInstance, which is this class.

class SbManagerT : public ArASyncTask
{
public:
  /// Constructor
  SbManagerT(const string &);
  /// Destructor
  /// deletes the objects (msgListeners) allacted for this class.
  ~SbManagerT();
  /// Initilizes robot and TCP server
  bool init(void);
  ///the function to run in the new thread, this just is called once, so only
  ///this gets run as soon as the object is created
  ///so this part should have the init stuff in it
  virtual void * runThread(void * arg);

protected:
  /// vector of listener pointers that are registered with handler.
  /// This is used to delete allocated memory after class is destroyed.
  vector < SbMsgListener * > msgPtrVector;
  /// registeres the listener with the msgHandler.
  /// also registeres the new pointer with the local memory manager
  /// which uses the msgPtrVector to delete object after this class is destr.
  void registerListener(const string str, SbMsgListener *msgListenerPtr);
  /// registeres the listener with the msgHandler.
  /// also registeres the new pointer with the local memory manager
  /// which uses the msgPtrVector to delete object after this class is destr.
  /// this one calls the function in msgHandler
  /// that extracts types out of listener
  void registerListener(SbMsgListener *msgListenerPtr);
  /// iterator through the ptr vetor.
  typedef vector < SbMsgListener * >::iterator vectIter;
  /// Creates the listener objects and registers msg Types with
  /// respective listener.
  /// remember to delete these in the destructor. \n
  /// More than one listener can be registered with the same type.
  void initMsgHandler(void);
  /// port for tcp server
  string port;
  /// instance pointer to Singleton msgHandler
  SbMsgHandlerSingleton*msgHandler;
  /// instance pointer to the TCP comm server
  SbServerML * tcpServer;
  /// internal msg
  SbMessage msg;
  /// instance pointer to the robot control class.
  SbRobotML * sbRbPtr;
};
#endif // SbManagerT_h
```

```
#include "SbManagerT.h"
#include "SbRobotML.h"
#include "SbSoundsMLT.h"
#include "SbMotionML.h"
#include "SbGripperML.h"
#include "SbCameraSONYML.h"
#include "SbCameraVCC4ML.h"
#include "SbActionML.h"
#include "SbStateML.h"
#include "SbSaphiraML.h"
#include "SbExampleML.h"
#include "SbExampleMLT.h"
#include <algorithm>

SbManagerT::SbManagerT(const string & port)
{
  this->port = port;
  //create an instance of singleton.
  msgHandler = SbMsgHandlerSingleton::getInstance();
}

SbManagerT::~SbManagerT()
{
  //Delete some of the allocated objects when I am done.
  for(vectIter i = msgPtrVector.begin(); i != msgPtrVector.end(); ++i)
  { delete(* i); }

  //delete the only instance of the singleton handler.
  msgHandler->removeInstance();
}

void SbManagerT::registerListener(const string str,
SbMsgListener * msgListenerPtr)
{
  // add it to the msgHanlder
  // this should add the listener pointer to our vector
  // first check if it exists already...
  msgHandler->addListener(str, msgListenerPtr);
  // and search to see if the pointer already exists
  // if not add sbMsgListener pointer.
  // don't want to register the same one twice...
  if(std::find(msgPtrVector.begin(), msgPtrVector.end(), msgListenerPtr)
  == msgPtrVector.end())
  {
    //didn't find it so add it.
    msgPtrVector.push_back(msgListenerPtr);
    return;
  }
}

void SbManagerT::registerListener(SbMsgListener * msgListenerPtr)
{
  // add it to the msgHanlder
  // this should add the listener pointer to our vector
  // first check if it exists already...
  msgHandler->addListener(msgListenerPtr);
  // and search to see if the pointer already exists
  // if not add sbMsgListener pointer.
  // don't want to register the same one twice...
  if(std::find(msgPtrVector.begin(), msgPtrVector.end(), msgListenerPtr)
  == msgPtrVector.end())
  {
    //didn't find it so add it.
    msgPtrVector.push_back(msgListenerPtr);
    return;
  }
}

// Init fuction starts the server, creates robot, and new thread.
bool SbManagerT::init(void)
{
  //we should never shutdown the robot since it will disconnect
  //Aria::init(Aria::SIGHANDLE_THREAD);
  //init server
  tcpServer = new SbServerML();
  // this is sneaky because addMsgAndPointer was not used.
  msgPtrVector.push_back(tcpServer);
  //try to start up the server
  if(tcpServer->open(port))
  {
    //work only if the server is started.
    sbRbPtr = new SbRobotML();
    //add the new msgListener object to the list
    msgPtrVector.push_back(sbRbPtr);
    initMsgHandler();
    //start the thread.
    create();
    return true;
  }
  ///failed to start the server.
  return false;
}

// Creates the listener objects and registers msg Types with respective listene
// remember to delete these in the destructor.!
// More than one listener can be registered with the same type.
// SbRobotML is of MsgListener type
void SbManagerT::initMsgHandler(void)
{
```

```
100     //LISTENER COMMANDS:
        registerListener(tcpServer);
        registerListener(sbRbPtr);

        //EXAMPLE
105     SbExampleML * example = new SbExampleML();
        registerListener(example);
        //EXAMPLET
        SbExampleMLT * exampleThread = new SbExampleMLT();
        registerListener(exampleThread);
110     //STATE REFLECTOR
        SbMsgListener * state = new SbStateML();
        registerListener(state);
        //MOTION
        SbMsgListener * motion = new SbMotionML();
115     registerListener(motion);
        //GRIPPER
        SbMsgListener * gripper = new SbGripperML();
        registerListener(gripper);
        //CAMERA need to specify what kind of camera this system has.
120     //choose the right camera for the current robot
        // SbMsgListener * camera = new SbCameraSONYML();
        SbMsgListener * camera = new SbCameraVCC4ML();
        registerListener(camera);
        //ACTIONS
125     SbMsgListener * action = new SbActionML();
        registerListener(action);
        //SAPHIRA intelligent motion commands.
        SbMsgListener * sbSaphira = new SbSaphiraML();
        registerListener(sbSaphira);
130     //SOUNDS
        SbMsgListener * snd = new SbSoundsMLT();
        registerListener(snd);

    }
135
    // the function to run in the new thread, this just is called once, so only
    // return when you want the thread to exit.
    // this gets run as soon as the object is created so this part should have the
    // init stuff in it
140 void * SbManagerT::runThread(void * arg)
    {
      while(myRunning)
      {
        // call the SbRobot check once function which checks if the robot is
145     // connected and if it has any problems (ie stall, bumbpers)
        // here I should first check if the robot is connected if not send a msg.
        sbRbPtr->checkOnce();
        // now take a little nap, let the robot use the CPU
        ArUtil::sleep(50);
150     //check if anything is going on, on the server.
        tcpServer->cycleOnce();
        //get the message in the server if it exists than send it to handler
        //getMsg returns bool, and sets msg.
        if(tcpServer->getMsg(msg))
155     {
          //SHUTDOWN||  disconnects from robot, quits allconnections, exits program
          if(msg.getType() == "SHUTDOWN")
          {
            //disconnect from the robot;
160         msgHandler->fireMsg("DISCONNECT||");
            //send message to everyone.
            msgHandler->fireMsg("SEND|MSG|SHUTTING DOWN RCCI!||");
            //disconnect all clients.
            msgHandler->fireMsg("QUITALL||");
165         //wait till above get processed.
            ArUtil::sleep(100);
            //shutdown tcpServer
            tcpServer->close();
            //wait a little for the tcpServer to stop.
170         ArUtil::sleep(1500);
            cout << "SHUTTING DOWN - USER COMMAND (SbManagerT)" << endl;
            //and finally shutdown the whole program.
            Aria::shutdown();
          }
175       //if the message was not SHUTDOWN let thhe msgHandler distribute it.
          else
            msgHandler->fireMsg(msg);
        }
      }
180   // return out here, means the thread is done
      return NULL;
    }
```

```
#ifndef SbMessage_h
#define SbMessage_h

#include <string>
#include <vector>
#include <sstream>

using namespace std;

/// Message structure class used for communications.
/// SbMessage class stores message information in an organized string format.
/// Each message is characterized by "TYPE" which inherently sorts the messages
/// for processing in SbMsgHandlerSingleton. Following the type, arguments
/// (strings) can be added and removed using functions such as addArgument().
/// SbMessages can also be created using a formatted string delimited by a
/// unique character (default: '|'). Each message is timestamped.
/// Example: “CAM|zoom|56||”.
/// @author Marcin Balicki : Cooper Union - Mechanical Engineering
/// @date March 2004
/// @todo Improve speed by removing vector, try catch throw an error with
/// indexing or trying to convert string to an incompatible type.
/// @note timestamp is set everytime the SbMessage type is set
/// @note | is the default delimiter -> MSGTYPE|ARG1|ARG2|ARG3|ARGn||
/// @bug None

class SbMessage
{
public:
  ///Constructor
  SbMessage(void);
  ///Constructor which sets the type of message
  SbMessage(const string & type);
  ///Constructor that creates a message from a parsed vector anda delimiter
  SbMessage(const vector < string > & msgVctr, string delimiter);
  ///constructor with unparsed message and delimiter strings
  SbMessage(const string &, const string &);
  ///Destructor
  ~SbMessage(void);
  ///Sets the type of message
  void setType(const string & type);
  /// resets the message
  void clear(void);
  /// adds on an argument to the message
  void addArgument(const string & arg);
  /// removes an argument from message
  void removeArgument(int i);
  /// sets the delimiter for the message format
  void setDelimiter(const string & d)
  { delimiter = d; };
  ///delimiter accessor
  string getDelimiter(void)
  { return delimiter; };
  /// parses and sets the type and arg fields of message
  bool create(const string & str);
  /// returns type of message.
  string getType(void) const
  { return type; };
  ///returns timestamp.
  string getTimeStamp(void)const
  { return timeStamp; };
  ///gets number of arguments in message.
  int getNumOfArgs(void)const
  { return numOfArgs; };
  ///returns the message as a string (includes delimiters).
  string getAsStr(void)const;
  ///returns an argument as an integer.
  int getArgAsInt(int i)const;
  ///returns an argument as a double.
  double getArgAsDbl(int i)const;
  ///returns an argument as a string.
  string getArgAsStr(int i)const;
  ///helper function that converts integer to string
  static string convToString( int i );
  /// helper function that converts double  to string
  static string convToString( double d );
protected:
  ///sets the time of creation or recreation
  void setTimeStamp();
  ///test if the string is a number
  bool isNumber(string) const;
  ///delimiter for the current message
  string delimiter;
  ///timeStamp string
  string timeStamp;
  ///type of this message
  string type;
  ///number of arguments that are part of this message.
  int numOfArgs;
  ///size that will improve the speed of the vector by allocating space for it
  const static int iniVctrSize = 255;
  ///this is slow and may need to be replaced by an array or something like tha
  vector < string > argVctr;

};

#endif
```

```cpp
#include "SbMessage.h"
#include <iostream>
#include <sys/time.h>
#include <unistd.h>
#include <iomanip>


SbMessage::SbMessage(void)
{
  setType("");
  setDelimiter("|");
  setTimeStamp();
  numOfArgs = 0;
  //allocate spaces for args.
  argVctr.resize(iniVctrSize);
}

//The constructor that has type of SbMessage passed to it.
SbMessage::SbMessage(const string & type)
{
  setType(type);
  setDelimiter("|");
  setTimeStamp();
  numOfArgs = 0;
  //allocate spaces for args.
  argVctr.resize(iniVctrSize);




}

SbMessage::SbMessage(const vector < string > & msgVctr, string delimiter)
{
  setDelimiter(delimiter);
  setTimeStamp();
  numOfArgs = 0;
  //allocate spaces for args.
  argVctr.resize(iniVctrSize);
  if(!msgVctr.empty())
  {
    setType(msgVctr[0]);
    for(unsigned int i = 1; i < msgVctr.size(); i++)
      addArgument(msgVctr[i]);
  }
}

SbMessage::SbMessage(const string & str, const string & d)
{
  setDelimiter(d);
  //allocate spaces for args.
  argVctr.resize(iniVctrSize);

  create(str);
}

SbMessage::~SbMessage(void)
{ }

bool SbMessage::create(const string & str)
{
  //this will just look until the first double deliminator.
  //rest will be dropped.
  clear();
  setTimeStamp();
  string d(getDelimiter());
  string dd(d + d);
  //if the delimtor is not found exit
  if(str.find(dd) == str.npos)
  {
    cout << "ERROR: Can't create message (SbMessage)" << endl;
    return false;
  }
  //get the first msg in the string (rest gets thrown away
  // including two end of msg characters)
  string s = str.substr(0, str.find(dd));
  //problem what if the msg has no arguments just a type ???
  //if there is no type specified just make it blank.
  if(s.empty())
  {
    setType("");
    return true;
  }
  //if the type is the only part of the msg:
  //becuase no delimiter was found
  else if(s.find(d) == s.npos)
  {
    setType(s);
    return true;
  }
  //just find the first string up to delimiter and set it as a type
  else
  {
    setType(s.substr(0, s.find(d)));
    s.erase(0, (s.find(d) + 1));
  }
  //add arguments
  //while the message is not completed.
  while(!s.empty())
  {
```

```
100       if(!(s.find(d) == s.npos))
          {
            addArgument(s.substr(0, s.find(d)));
            s.erase(0, s.find(d) + 1);
          }
105       else
          {
            addArgument(s);
            s.erase();
          }
110     }
        return true;
      }

      void SbMessage::setType(const string & type)
115   {
        this->type = type;
        setTimeStamp();
      }

120   void SbMessage::clear(void)
      {
        setType("");
        timeStamp = "";
        argVctr.clear();
125     //allocate spaces for args.
        argVctr.resize(iniVctrSize);
        numOfArgs = 0;
      }

130   void SbMessage::addArgument(const string & arg)
      {
        argVctr[numOfArgs] = arg;
        numOfArgs++;
        if(numOfArgs == iniVctrSize)
135     {
          argVctr.resize(argVctr.size() + iniVctrSize);
          cout << "Increasing argVctrSize, (SbMessage)" << endl;
        }
      }
140
      void SbMessage::removeArgument(int i)
      {
        //check to see if the I is out of range
        if(i >= numOfArgs || i > argVctr.size() || i < 0)
145       cout << "Index out of range, can't  remove argument(SbMessage)" << endl;
        else
        {
          argVctr.erase(argVctr.begin() + i);
          numOfArgs--;
150     }
      }

      string SbMessage::getAsStr(void)const
      {
155     string msg(getType());
        //if i is out fo argVctr.size() bounds
        for(unsigned int i = 0; i < (getNumOfArgs()); i++)
          msg = msg + delimiter + argVctr[i];
        msg += (delimiter + delimiter);
160     return msg;
      }

      double SbMessage::getArgAsDbl(int i) const
      {
165     //if out of bounds
        //need to test if it is a number
        if(argVctr.empty() || i < 0 || i >= (getNumOfArgs())
        || (!(isNumber(argVctr[i]))))
        {
170       cout << "Index out of range or the arg is not a number,"
          << "returning zero as the argDbl (SbMessage)"
          << endl;
          return 0.0;
        }
175     return atof(argVctr[i].c_str());
      }

      string SbMessage::getArgAsStr(int i) const
      {
180     //if i is out fo argVctr.size() bounds
        if(i < 0 || argVctr.size() < i || i >= (getNumOfArgs()))
        {
          cout << "Index out of range, returning nothing as the argStr (SbMessage)"
          << endl;
185       return "";
        }
        else
          return argVctr[i];
      }
190
      string SbMessage::convToString( double d )
      {
          ///internal stream function used in toString()
        ostringstream oss( "" );
195     oss <<setprecision(4)<<d; // insert double into stream
        return oss.str();
      }
      string SbMessage::convToString( int i )
```

```
    {
200    ///internal stream function used in toString()
       ostringstream oss( "" );
      //oss.str( "" );
      oss << i; // insert int into stream
      return oss.str();
205 }


    int SbMessage::getArgAsInt(int i) const
    {
210   //if out of bounds
      if(argVctr.empty() || i < 0 || i >= (getNumOfArgs())
      || (!(isNumber(argVctr[i]))))
      {
        cout << "Index out of range or the arg is not a number,"
215     << "returning zero as the argDbl (SbMessage)"
        << endl;

        return 0;
      }
220   return int(atof(argVctr[i].c_str()));
    }

    void SbMessage::setTimeStamp(void)
    {
225   //    string time;
      char buf[50];
      time_t t = time(0);
      strftime(buf, 50, "%c", localtime(& t));
      timeStamp = buf;
230 }

    bool SbMessage::isNumber(string numStr) const
    {
      bool result = true;
235   //checks for number of periods
      //need to look at this closely.
      int period = 0;
      for(unsigned int i = 0; i < numStr.size(); i++)
      {
240     char Ch = numStr[i];
        if(Ch == '.') period++;
        //cout<<"ALPHA: "<<isalpha(Ch)<<endl;
        if(!((Ch >= '0' && Ch <= '9') || Ch == '.' || numStr[0] == '-')
        || period > 1)
245     {
          //throw InvalidKeyword();
          result = false;
        }
      }
250   return result;
    }
```

```
#ifndef SbMotionML_h
#define SbMotionML_h

#include "SbMsgListener.h"
#include "SbMessage.h"
#include "Aria.h"
#include <string>

using namespace std;

/// Manages robot motion functions.
/// This listener parses motion requests. It commands the robot to "MOVE",
/// "ROTATE", sets the "VEL",  "LRVEL", "ROTVEL", "HEADING" and other motion
/// related functions.
/// @author Marcin Balicki : Cooper Union - Mechanical Engineering
/// @date March 2004
/// @warning Aria Motion Commands
/// Be aware that direct or motion command may conflict with controls from
/// Actions or other upper level processes and lead to unexpected consequences.
/// robot->clearDirectMotion() will cancel the direct Motion so action can get
/// the control back there is some other way to automatically do it using:\n
/// robot->setDirectMotionPrecedenceTime();\n
/// may block actions forever if too high\n
/// robot->getDirectMotionPrecedencetTime();\n
/// moving the robot using actions is recommended....
/// @note You can use the sendMsgToHandler to fire messages back to the handler.
/// @bug None

class SbMotionML : public SbMsgListener
{
public:
  ///Constructor
  SbMotionML( void );
  ///Destructor
  ~SbMotionML( void );
  /// implements the SbMsgListener pure virtual function.
  /// @param &msg SbMessage which caused the listener to fire.
  virtual void fire( const SbMessage &msg );
protected:
  ///internal robot pointer
  ArRobot * robot;
  ///checks if the robot (ARIA instance) exists and is connected
  bool checkForRobot( void );
  ///type of message
  string type;
};

#endif
```

```
#include "SbMotionML.h"

SbMotionML::SbMotionML(void) {

//adds message types to the msgListener (base) vector for registering with
// msg handler.
addMsgType("MOVE");
addMsgType("VEL");
addMsgType("LRVEL");
addMsgType("ROTVEL");
addMsgType("HEADING");
addMsgType("ROTATE");
addMsgType("STOP");
addMsgType("CLEAR");
addMsgType("MAXTRANSVEL");
addMsgType("MAXROTVEL");
addMsgType("TRANSACCEL");
addMsgType("TRANSDECEL");
addMsgType("ROTACCEL");
addMsgType("ROTDECEL");
}

SbMotionML::~SbMotionML(void) { }

//Checks if an instance of the robot exists and if it is connected.
//this is necessary if we are going to use
bool SbMotionML::checkForRobot(void)
{
  //Check if the robot(SBROBOT ) exists, if yes and its connected
  //than return true;
  if((robot = Aria::findRobot("SBROBOT")) == NULL)
    return false; //quit this function because of error

  //lock the robot se we can get data from it
  robot->lock();
  if(robot->isConnected())
  {
    //if we are connected unluck and quit function
    robot->unlock();
    return true;
  }
  //we are not connected so unlock the robot and return false.
  robot->unlock();
  return false;
}


void SbMotionML::fire(const SbMessage & msg)
{
  //has to be connected first.
  if(!checkForRobot())
  {
    cout << "Robot doesn't exist or is not connected:"
    <<" Will not fire command. (SbMotionML)"
    << endl;
    sendMsgToHandler("SEND|MSG|DISCONNECTED||");
    return;
  }
  //check type of msg
  type = msg.getType();
  robot->lock();
  // i think that the move command actually uses short for the movement
  // so +- 32767 mm are the limits
  // MOVE|double|| moves the robot (distance)
  // forward(+) or backward(-) in straight line (mm)
  if(type == "MOVE")
    robot->move(msg.getArgAsDbl(0));
  //VEL|double|| sets robots velocity (mm/sec) - is backwards
  else if(type == "VEL")
    robot->setVel(msg.getArgAsDbl(0));
  //LRVEL|double|double|| sets the velocity of each wheel (LEFT|RIGHT) in mm/se
  else if(type == "LRVEL")
    robot->setVel2(msg.getArgAsDbl(0), msg.getArgAsDbl(1));
  //ROTVEL|double|| sets the rotacional velocity deg/sec
  else if(type == "ROTVEL")
    robot->setRotVel(msg.getArgAsDbl(0));
  // HEADING|double|| will rotate the robot to the desired heading (deg) in the
  // shortest possible manner (350) will turn -10 so full turn is not possible.
  else if(type == "HEADING")
    robot->setHeading(msg.getArgAsDbl(0));
  // ROTATE|double|| rotates the robot by given amount in Degrees
  // POSITIVE IS going ccw.   NEGATIVE is cw (right.)
  // need to look into this further because the robot goes to achieves rot
  // greater than 180 in the shortest method.
  // ie it will go right if the 190 is placed.
  else if(type == "ROTATE")
    robot->setDeltaHeading(msg.getArgAsDbl(0));
  // STOP|| this wills top all motions and actions (actions will be blocked
  // until the precidence time
  // is exceeded or cleardirectmotion is called.
  else if(type == "STOP")
    robot->stop();

  // CLEAR clears direct motion (move, rotate,etc) robot will not keep trying
  // to finish. if vel is on, and this command is fired,
  // the robot will stop in 2 or 3 seconds.
  // it  is useful when robots motors are shut off and it tries to complete
  // a motion command producing an annoying beep....
  else if(type == "CLEAR")
```

```
      robot->clearDirectMotion();
    // MAXTRANSVEL|double|| sets maximum velocity mm/sec
    else if(type == "MAXTRANSVEL")
      robot->setTransVelMax(msg.getArgAsDbl(0));
    // MAXTROTVEL|double|| sets maximum rotational velocity deg/sec
    else if(type == "MAXROTVEL")
      robot->setRotVelMax(msg.getArgAsDbl(0));
    // TRANSACCEL|double|| sets acceleration
    else if(type == "TRANSACCEL")
      robot->setTransAccel(msg.getArgAsDbl(0));
    // TRANSDECEL|double|| sets deceleration
    else if(type == "TRANSDECEL")
      robot->setTransDecel(msg.getArgAsDbl(0));
    // ROTACCEL|double||  sets rotational acceleration
    else if(type == "ROTACCEL")
      robot->setRotAccel(msg.getArgAsDbl(0));
    // ROTDECEL|double||  sets rotational deceleration
    else if(type == "ROTDECEL")
      robot->setRotDecel(msg.getArgAsDbl(0));
    robot->unlock();
    cout << "Firing motion with msg : "<< msg.getAsStr() << endl;
  }
```

```
#ifndef SbMsgHandlerSingleton_h
#define SbMsgHandlerSingleton_h

#include "SbMessage.h"
#include "SbMsgListener.h"
#include <vector>
#include <map>
#include <string>

using namespace std;

/// Manager for the messaging structure.
/// Any class of type MsgListener can register with the Handler along with a
/// key (type). When fireMsg function is called it will fire (send) the message
/// to all the listeners registered with the particular message type. Each
/// listener implements a fire function which accepts the message and
/// interprets it in any way it wants.
/// Singleton concept was implemented for accessing SbMsgHanglerSingleton.
/// Required because some of the listeners want to send a message back to the
/// Handler. Singleton class assures a maximum of ONE object of its type
/// at a given time and provides a global access point to this object.
/// It may be problematic if threading is used. In that case I suggest
/// to lock the handler by using ArMutex. This was avoided here for gain
/// in speed.
/// @author Marcin Balicki : Cooper Union - Mechanical Engineering
/// @date March 2004
/// @warning a Singleton class needs to be destroyed by the last thread
/// alive by calling removeInstance.

class SbMsgHandlerSingleton
{
public:
  ///returns the instance to this class, if does not exist it will be created
  static SbMsgHandlerSingleton* getInstance(void);
  ///remove instance by deleteing it.
  static void removeInstance(void);
  ///adds a key-listener pair to the map.
  void addListener(const string &, SbMsgListener *);
  /// adds a listener, extracts msg types (key) from the listener class
  void addListener(SbMsgListener *);
  ///only removes the msglistener from one particular type.
  void removeListener(const string &, SbMsgListener *);
  ///removes all instances of MsgListener from the map regarding of key
  void removeListenerInstances(SbMsgListener *);
  ///removes a key and associated listeners
  void removeType(const string &);
  ///SbMsgListener getMsgListeners(const string&);
  int getNumOfTypes(void);
  ///returns number of listeners for a msg type
  int getNumOfListeners(const string &);
  ///returns total number of listeners
  int getNumOfAllListeners(void);
  ///fires a message from a SbMessage format
  void fireMsg(const SbMessage &);
  //creates a SbMessage from string and fires it.
  void fireMsg(const string &);

protected:
  ///protected Constructor
  SbMsgHandlerSingleton(void);
  ///protected Destructor which deletes alocated memory for vectors in the map;
  ~SbMsgHandlerSingleton(void);
  ///Copy constructor
  SbMsgHandlerSingleton(const SbMsgHandlerSingleton&);
  /// operator overload for coping.
  SbMsgHandlerSingleton& operator = (const SbMsgHandlerSingleton&);
  ///check if message type exists
  bool msgTypeExists(const string &) const;
  /// vector Iterator type
  typedef vector < SbMsgListener * >::iterator vectIter;
  /// vector Pointer type
  typedef vector < SbMsgListener * > * listenerVectPtr;
  /// map Iterator type
  typedef map < string, listenerVectPtr >::iterator mapIter;
  /// finds the position of the pointer in the given vector (end if not found)
  vectIter vectorFind( vector < SbMsgListener * > *, SbMsgListener *);
  /// this is the essential data structure that holds the key and the relevant
  /// pointer to the list of msgListener pointers.
  map < string, listenerVectPtr > MsgHndlrMap;
  ///temporary msg variable
  SbMessage msg;
private:
  ///internal single instance of THIS class.
  static SbMsgHandlerSingleton* pinstance;
};

#endif
```

```cpp
#include "SbMsgHandlerSingleton.h"
#include <algorithm>

/// initialize pointer to zero...
SbMsgHandlerSingleton* SbMsgHandlerSingleton::pinstance = 0;

SbMsgHandlerSingleton::SbMsgHandlerSingleton(void) { }

SbMsgHandlerSingleton::~SbMsgHandlerSingleton(void)
{
  //this gest called everytime delete is called
  //have to delete each vector in the map;
  for(mapIter p = MsgHndlrMap.begin(); p != MsgHndlrMap.end(); ++p)
    delete p->second;

  //cout<<"SINGLETON DESTRUCTOR CALLED"<<endl;

}

SbMsgHandlerSingleton* SbMsgHandlerSingleton::getInstance(void)
{
  if(pinstance == 0) // is it the first call?
  {
    pinstance = new SbMsgHandlerSingleton; // create sole instance
  }
  return pinstance; // address of sole instance
}

///deletes the object instance
//this is not very safe but I think it is ok to try for the moment.
void SbMsgHandlerSingleton::removeInstance(void)
{
  //if instance exists delete it.
  if(!(pinstance == 0))
  { delete pinstance; }
}

void SbMsgHandlerSingleton::addListener(const string & t,
SbMsgListener * msgListPtr)
{
  //vector gets a pushback, the msgListPtr
  // of type MsgListener.
  //check if the msg type exists,
  if(!msgTypeExists(t))
  {
    //doesn't exist create new vector for that key and add the msgListPtr to it.
    MsgHndlrMap[t] = new vector < SbMsgListener * >;
    MsgHndlrMap[t]->push_back(msgListPtr);
  }
  //type exists so get the vector corresponding to key type
  //search to see if the pointer exists if not add sbMsgListener pointer.
  //don't want to register the same one twice...
  else if(vectorFind(MsgHndlrMap[t], msgListPtr)
  == (MsgHndlrMap[t]->end()))
  { MsgHndlrMap[t]->push_back(msgListPtr); }
}

void SbMsgHandlerSingleton::addListener(SbMsgListener * msgListPtr)
{
  // for each type in the listener type vector
  // check if the type is already registered
  // if no create a new vector and add this listener to it
  // otherwise just add the listener (if it is not there yet)
  //check if the msg type exists,
  vector < string > types = msgListPtr->getMsgTypes();
  vector < string >::iterator vIt;
  string t;
  //iterate to the vector
  for(vIt = types.begin(); vIt != types.end(); ++vIt)
  {
    t = * vIt;
    if(!msgTypeExists(t))
    {
      //doesn't exist create a new vector for that key and
      //add the msgListPtr to it.
      MsgHndlrMap[t] = new vector < SbMsgListener * >;
      MsgHndlrMap[t]->push_back(msgListPtr);
    }
    //type exists so get the vector corresponding to key type
    //and search to see if the pointer already exists if not
    // add sbMsgListener pointer. Don't want to register the same one twice...
    else if(vectorFind(MsgHndlrMap[t], msgListPtr)
    == (MsgHndlrMap[t]->end()))
    { MsgHndlrMap[t]->push_back(msgListPtr); }
  }
}

void SbMsgHandlerSingleton::removeListener(const string & t,
SbMsgListener * msgListPtr)
{
  //check if the type exists than if it does get the vector associated with
  //key type and remove the particular pointer from it.
  //if there are no items in the vector delete it, and remove the type from map
  //first makes sure that the listener is in this vector.
  if(msgTypeExists(t)
  && (vectorFind(MsgHndlrMap[t], msgListPtr)
  != (MsgHndlrMap[t]->end())))
  {
    MsgHndlrMap[t]->erase(vectorFind(MsgHndlrMap[t], msgListPtr));
```

```
100       if(MsgHndlrMap[t]->size() == 0)
          {
            delete MsgHndlrMap[t];
            int count = MsgHndlrMap.erase(t);
            //returns number of deletions
105       }

        }
      }

110   void SbMsgHandlerSingleton::removeListenerInstance(SbMsgListener * msgListPtr)
      {
        //iterate through all keys and then through all the vectors to find
        //the pointer
        //second referes to the second item in the key-item pair.
115     //get the pointer to vector with the key, just delete the whole vector
        for(mapIter p = MsgHndlrMap.begin(); p != MsgHndlrMap.end(); ++p)
          removeListener(p->first, msgListPtr);
      }

120   void SbMsgHandlerSingleton::removeType(const string & t)
      {
        //get the pointer to vector with the key, just delete the whole vector
        //check if it exists, if yes delete the vector
        if(msgTypeExists(t))
125     {
          delete MsgHndlrMap[t];
          int count = MsgHndlrMap.erase(t);
          //returns number of deletions
        }
130   }

      //SbMesssageListener SbMsgHandlerSingleton::getMsgListener(void )
      //{
      //
135   //}
      int SbMsgHandlerSingleton::getNumOfTypes(void)
      {
        return MsgHndlrMap.size();
        // MsgHndlrMap.size
140   }

      int SbMsgHandlerSingleton::getNumOfListeners(const string & t)
      {
        if(msgTypeExists(t))
145       return MsgHndlrMap[t]->size();
        return 0;
        //
      }
```

```
150   int SbMsgHandlerSingleton::getNumOfAllListeners(void)
      {
        int num = 0;
        for(mapIter p = MsgHndlrMap.begin(); p != MsgHndlrMap.end(); ++p)
          num += (p->second)->size();
155     return num;
      }

      bool SbMsgHandlerSingleton::msgTypeExists(const string & t) const
      {
160     if(MsgHndlrMap.find(t) == MsgHndlrMap.end())
          return false;
        return true;
      }

165   SbMsgHandlerSingleton::vectIter
      SbMsgHandlerSingleton::vectorFind(listenerVectPtr v,
      SbMsgListener * msgPtr)
      {
        //finds the first element equal to msgPtr
170     // contained in the vector pointed by v
        return std::find(v->begin(), v->end(), msgPtr);

        //old method
        //
175     //    for(SbMsgHandlerSingleton::vectIter i = v->begin(); i != v->end(); ++i)
        //     {
        //       if((* i) == msgPtr)
        //           return i;
        //       }
180     //       return v->end();
      }

      void SbMsgHandlerSingleton::fireMsg(const SbMessage & ms)
      {
185     //find the type of message in the map, get the vector and
        //fire each element in the vector
        if(msgTypeExists(ms.getType()))
        {
          listenerVectPtr p = MsgHndlrMap[ms.getType()];
190       //temp variable
          for(vectIter vI = p->begin(); vI != p->end(); ++vI)
            (* vI)->fire(ms);
          //(*vI) is dereferencing the iterator. to a pointer
        }
195     else
        {
          //if it is not found just see if there is a SEND listener if so create and
          // send the msg to all.
```

```
        //this is not very good programming but otherwise the user doesn't know
200     // that his msg was incorrect.
        if(msgTypeExists("SEND"))
        {
          fireMsg("SEND|MSG|UNKOWN MESSAGE TYPE -> ::"+ ms.getType()+"::||");
        }
205     cout << "WARNING: Message does not have any listeners! type ='"
        << ms.getType() << "' (SbMsgHandlerSingleton)"<< endl;
      }
    }

210 void SbMsgHandlerSingleton::fireMsg(const string & str)
    {
      //create a message from the string then fire it.
      // delimiter is the default |
      msg.create(str);
215   //clears and creates a new msg
      if(msgTypeExists(msg.getType()))
      {
        listenerVectPtr p = MsgHndlrMap[msg.getType()];
        //temp variable
220     for(vectIter vI = p->begin(); vI != p->end(); ++vI)
          (* vI)->fire(msg);
        //(*vI) is dereferencing the iterator. to a pointer
        //creates a message with the t as the type and forces fire.
      }
225   else
      {
        //if it is not found just see if there is a SEND listener if so create and
        // send the msg to all.
        //this is not very good programming but otherwise the user doesn't know
230     // that his msg was incorrect.
        if(msgTypeExists("SEND"))
        {
          fireMsg("SEND|MSG|UNKOWN MESSAGE TYPE -> ::"+ msg.getType()+"::||");
        }
235     cout << "WARNING: Message does not have any listeners! type ='"
        << msg.getType() << "' (SbMsgHandlerSingleton)"<< endl;
      }
    }
```

```
#ifndef SbMsgListener_h
#define SbMsgListener_h

#include "SbMessage.h"
#include <string>

using namespace std;

/// Abstract class for the messaging system.
/// You cannot create an object of an abstract class type; however, you can use
/// pointers and references to abstract class types.
/// A class that contains at least one pure virtual function (fire) is
/// considered an abstract class. Classes derived from the abstract class must
/// implement the pure virtual function or they, too, are abstract classes.
/// This class provides messaging functionality, it stores the listener types
/// that the listener wants to be registered with. Since the MsgListener can
/// send messages back to the handler sendMsgToHandler() via the pointer to the
/// handler (SbMsgHandlerSingleton::getInstance()) it can only register
/// with one message handler in this program.
/// @author Marcin Balicki : Cooper Union - Mechanical Engineering
/// @date March 2004
/// @note since the handler pointer exists here, the listener can't register
/// with more than one Handler.
/// would be nice to change this later or create an array of pointers.
/// @bug None


class SbMsgListener
{
public:
  ///Constructor
  SbMsgListener();
  ///Destructor
  ~SbMsgListener();
  /// returns a pointer to the vector containg types of messages that this
  /// listener wants to listen to
  /// @return the pointer to vector  of message types
  vector <string > getMsgTypes(void){ return msgTypeVector;};
  /// Pure virtual function (makes this class abstract).
  /// Fire the listener function.
  /// This will be called by the Handler whenever this Listener is triggered.
  /// It is pure abstract needing implementation when inherited.
  /// @param &msg the SbMessage which caused this MsgListener to fire.
  virtual void fire(const SbMessage & msg) = 0;
  /// call back for sending messages back to the message handler.
  /// @param &msg SbMessage that should be sent to the handler
  virtual void sendMsgToHandler(const SbMessage &msg);
  /// call back for sending messages back to the message handler.
  /// @param &str String that should be sent to the handler
  virtual void sendMsgToHandler(const string &str);
protected:
  /// adds the message type that this listener should listen to.
  /// msgHandler will read the vector containing these and register the types
  /// for messaging
  /// @param &m string as a message type ie "SEND"
  virtual void addMsgType(const string m) {msgTypeVector.push_back(m);};
  /// vector containing all the message types that this listener is interested
  /// in registering with the handler.
  vector < string  > msgTypeVector;
  //vector < string  > msgTypeVector::iterator vectIter;
};

#endif
```

```cpp
#include "SbMsgListener.h"
#include "SbMsgHandlerSingleton.h"
using namespace std;

SbMsgListener::SbMsgListener()
{
}
SbMsgListener::~SbMsgListener( void )
{
}

void SbMsgListener::sendMsgToHandler(const SbMessage & msg)
{
  // get the instance of the handler and fire the message
  (SbMsgHandlerSingleton::getInstance())->fireMsg(msg);
}
void SbMsgListener::sendMsgToHandler(const string & str)
{
  //get the instance of the handler and fire the message via string.
  (SbMsgHandlerSingleton::getInstance())->fireMsg(str);

}
```

```
#ifndef SbRobotML_h
#define SbRobotML_h

#include "SbMsgListener.h"
#include "SbMessage.h"
#include "Aria.h"

#include <string>
using namespace std;

/// Manages robot connection and checks for errors (bumpers, stall, etc).
/// Creates and instance of ARIA's robot interface. Provides connections to
/// the real robot or simulator. It checks if the robot is connected, if any of
/// its bumpers are triggered and if it has stalled. It also
/// provides and interface for activating sonar and the motors. Returns some
/// robot parameters: name, type, etc.
/// @author Marcin Balicki : Cooper Union - Mechanical Engineering
/// @date March 2004
/// @warning Uses SbMsgListener's SbMsgListener::sendMsgToHandler()
/// @bug None

class SbRobotML : public SbMsgListener
{
public:
  ///Constructor
  SbRobotML(void);
  ///Destructor
  ~SbRobotML(void);
  ///robot pointer accessor function
  ArRobot * getRobot(void) { return robot; };
  /// connection type.
  enum conType { ROBOT, SIM };
  ///checks for any errors (stall, bumpers, etc, connection) etc.
  ///Checks if the robot is connected and if it is ok (ie stalled)
  ///this function returns true if everything is ok.
  ///if it is false it will fire a message with stall, bumper or disconnect
  virtual void checkOnce(void);
  ///connects to the robot or a simulator.
  bool connect(conType t, const string &, const int &);
  ///disconnects from the robot or a simulator.
  bool disconnect(void);
  ///checks if the robot is connected.
  bool isConnected(void);
  ///implements the SbMsgListener pure virtual function.
  ///@param &msg SbMessage which caused the listener to fire.
  virtual void fire(const SbMessage &msg);

protected:
  ///Connects to the simulator.
  ///@param &host DNS name of the server hosting the simulator
  ///@param &port integer representing TCP port of the simulator
  bool connectSim(const string &host, const int &port);
  /// connects to robot connected to serial port
  bool connectRobot(void);
  /// disconnects from robot
  bool disconnected;
  /// type of message
  string type;
  /// internal robot pointer
  ArRobot * robot;
  /// sonar for the robot
  ArSonarDevice sonar;
  /// serial connection class
  ArSerialConnection* serialCon;
  /// tcp connection class used for connecting to simulator
  ArTcpConnection tcpCon;
};
#endif  //SbRobotML_h
```

```
#include "SbRobotML.h"

using namespace std;

SbRobotML::SbRobotML()
{
  //name the robot so an instance get be retrieved later.
  robot = new ArRobot("SBROBOT");
  robot->runAsync(false);
  // start up the robot, don't need a connection
  serialCon = new ArSerialConnection();
  // add the sonar to the robot
  robot->addRangeDevice(& sonar);
  //adds message types to the msgListener (base) vector for registering with
  //msg handler.
  //robot connection and control

  addMsgType("CONNECT");
  addMsgType("CONNECT");
  addMsgType("DISCONNECT");
  addMsgType("SETPOS");
  addMsgType("MOTORS");
  addMsgType("SONAR");
  addMsgType("ROBOTPARAMS");
  addMsgType("ROBOTNAME");
  addMsgType("ROBOTTYPE");
}

SbRobotML::~SbRobotML()
{
  //someone else deletes the robot.
  delete serialCon;
}

bool SbRobotML::connect(conType t, const string & host, const int & port)
{
  //this part doesn't seem to work too well when another init is called
  //in the same program.
  //getRunning should return the time between init and shutdown or exit.
  //but it doesn't, its good if no init was called then it works.
  //anyway this connect function may be eliminated
  //ArUtil::sleep(50);
  if(!(Aria::getRunning()))
  {
    sendMsgToHandler("SEND|MSG|NO ARIA - discard connection attempt||");
    cout << "ARIA is not running so don't attempt to connect (SbRobotML) "
         << endl;
    return false;
  }
  if(!isConnected())
  {
    switch(t)
    {
      case SIM:
        return connectSim(host, port);
      case ROBOT:
        return connectRobot();
      default:
        sendMsgToHandler(
        "SEND|MSG|CONNECTION FAILED - unknown connection method||");
        //cout<<"Don't know this connection method (SbRobotML)"<<endl;
        return false;
    }
  }
  else
  {
    //already connected
    sendMsgToHandler("SEND|MSG|CONNECTED||");
    return true;
  }
}

bool SbRobotML::connectSim(const string & host, const int & port)
{
  int ret;
  string str;
  if((ret = tcpCon.open(host.c_str(), port)) != 0)
  {
    str = tcpCon.getOpenMessage(ret);
    sendMsgToHandler("SEND|MSG|CONNECTION FAILED - simulator not available||");
    cout << "Open failed: "<< str.c_str() << " (SbRobotML)"<< endl;
    return false;
  }
  // set the robots connection
  robot->setDeviceConnection(& tcpCon);
  // try to connect, if we fail exit
  int count = 0;
  while(!isConnected())
  {
    cout << "Connecting..."<< endl;
    if(!robot->blockingConnect())
    {
      sendMsgToHandler("SEND|MSG|CONNECTION FAILED - trying again||");
      cout << "Could not connect to Simulator, ...trying again in 1 second. "
      << (9 - count) << " attempts left."<< endl;
      ArUtil::sleep(1000);
      count++;
      if(count == 10) return false;
    }
```

```
      else
      {
        cout << "Connection established (SbRobotML) " << endl;
        sendMsgToHandler("SEND|MSG|CONNECTED||");
        robot->comInt(ArCommands::ENABLE, 1);
        robot->comInt(ArCommands::SETO, 1);
        //resets the robot origin.
        return true;
      }
    }
}

bool SbRobotML::connectRobot(void)
{
  int ret = 0;
  string str;
  delete serialCon;
  //delete the old one and create a new one...hack around an old problem.
  serialCon = new ArSerialConnection();
  if((ret = serialCon->open()) != 0)
  {
    str = serialCon->getOpenMessage(ret);
    sendMsgToHandler("SEND|MSG|CONNECTION FAILED - serial comm problem||");
    cout << "Open failed: "<< str.c_str() << endl;
    return false;
  }
  // set the robots connection
  robot->setDeviceConnection(serialCon);
  // try to connect, if we fail exit
  int count = 0;
  while(!isConnected())
  {
    if(!robot->blockingConnect())
    {
      sendMsgToHandler("SEND|MSG|CONNECTION FAILED - trying again||");
      cout << "Could not connect to robot, ...trying again in 3 seconds.  "
      << (9 - count) << " attempts left."<< endl;
      ArUtil::sleep(3000);
      count++;
      if(count == 10) return false;
    }
    else
    {
      cout << "Connection established (SbRobotML) " << endl;
      sendMsgToHandler("SEND|MSG|CONNECTED||");
      robot->comInt(ArCommands::ENABLE, 1);
      robot->comInt(ArCommands::SETO, 1);
      //resets the robot origin.
      return true;
    }
  }
}

bool SbRobotML::disconnect(void)
{
  //make sure that ARIA is running first.
  if(!Aria::getRunning())
  {
    sendMsgToHandler("SEND|MSG|NO ARIA - discard connection attempt||");
    cout << "Aria not running will not try to disconnect (SbRobotML)" << endl;
    return false;
  }
  if(robot->disconnect()) { return true; }
  else
    return false;
}

bool SbRobotML::isConnected(void)
{
  robot->lock();
  bool ret = (robot->isConnected());
  robot->unlock();
  return ret;
}

void SbRobotML::checkOnce(void)
{
  //if everything is correct than return true, else false and a message.
  //lock the robot before touching it
  static bool conFlag = false;
  robot->lock();

  //DISCONNECTED checks if the roboot has disconnected.
  if(!(robot->isConnected()))
  {
    if(conFlag)
    {
      sendMsgToHandler("SEND|MSG|DISCONNECTED||");
      conFlag = false;
    }
    robot->unlock();
    return;
  }
  else
    conFlag = true;

  // BUMPERS checks if the back or the front
  // If the bumpers have been hit it will send the readout of all the
  // bumpers MSG|BUMPERS|0|0|0|1|0|1|0|...||
```

```
  unsigned int i;
  int val;
  int bit;
  SbMessage bumpMsg("SEND");
  bumpMsg.addArgument("MSG");
  bumpMsg.addArgument("BUMPERHIT");
  bool bumphit = false;
  //front bumpers
  val = ((robot->getStallValue() & 0xff00) >> 8);
  for(i = 0, bit = 2; i < robot->getNumFrontBumpers(); i++, bit *= 2)
  {
    if(val & bit)
    {
      bumpMsg.addArgument("1");
      printf("%6s", "trig");
      bumphit = true;
    }
    else
      bumpMsg.addArgument("0");
  }
  //rear bumpers
  val = ((robot->getStallValue() & 0xff));
  for(i = 0, bit = 2; i < robot->getNumRearBumpers(); i++, bit *= 2)
  {
    if(val & bit)
    {
      bumpMsg.addArgument("1");
      printf("%6s", "trig");
      bumphit = true;
    }
    else
      bumpMsg.addArgument("0");
  }
  //if any of the bumpers were hit send a message to the handler.
  if(bumphit)
    sendMsgToHandler(bumpMsg);

  //STALLED|int|int|| will send MSG|STALLED|int|int||
  //send which motor was stalled....left/right.
  if((robot->isLeftMotorStalled()) || (robot->isRightMotorStalled()))
  {
    SbMessage msg("MSG");
    msg.addArgument("STALLED");
    if(robot->isLeftMotorStalled())
      msg.addArgument("1");
    else
      msg.addArgument("0");
    if(robot->isRightMotorStalled())
      msg.addArgument("1");
    else
      msg.addArgument("0");
  }

  robot->unlock();
  return;
}

void SbRobotML::fire(const SbMessage & msg)
{
  type = msg.getType();
  // Parse the message, the Type should tell me what to do.

  // CONNECT|string|string|string|| connects to a robot
  //  if the first argument is blank (0) or sim it will attempt to
  // connect to default simulator location, if it is sim and the rest are
  // not blank it will take in address and port params CONNECT|sim|add|port||
  // if first arguemnt is robot it will attempt to connect to the robot on
  // default port
  if(type == "CONNECT")
  {
    if(msg.getArgAsStr(0) == "")
      connect(SIM, "localhost", 8101);
    else if(msg.getArgAsStr(0) == "sim")
      if(msg.getArgAsDbl(1) == 0)
        connect(SIM, "localhost", 8101);
      else
        connect(SIM, msg.getArgAsStr(1), msg.getArgAsInt(2));
    else if(msg.getArgAsStr(0) == "robot")
      connect(ROBOT, "", 0);
    return;
  }
  // DISCONNECT|| disconnects from the robot or simulator
  else if(type == "DISCONNECT")
  {
    disconnect();
    return;
  }
  //if the robot is connnected we can do these:
  if(!isConnected())
    return;
  //connected so the commands can be sent to the robot
  robot->lock();
  // SETPOS|double|double|double||
  // sets the apparent position of the robot in the world.
  // X|Y|HEADING|
  if(type == "SETPOS")
  {
    robot->moveTo(ArPose(msg.getArgAsDbl(0), msg.getArgAsDbl(1),
    msg.getArgAsDbl(2)));
```

```
      }
      // MOTORS|int|| turns the motors off and on 0=OFF 1=ON.
300   else if(type == "MOTORS")
      {
        switch(msg.getArgAsInt(0))
        {
          case 0:
305         robot->comInt(ArCommands::ENABLE, 0);
            break;
          case 1:
            robot->comInt(ArCommands::ENABLE, 1);
        }
310   }
      // SONARS|int|| turns the sonar off and on  0=OFF 1=ON.
      else if(type == "SONAR")
      {
        switch(msg.getArgAsInt(0))
315     {
          case 0:
            robot->comInt(ArCommands::SONAR, 0);
            break;
          case 1:
320         robot->comInt(ArCommands::SONAR, 1);
        }
      }
      // ROBOTPARAMS robot parameters which doesn't seem to work too well
      // use doubles
325   // transVELmax|rotVELmax|transACCEL|transDECEL|rotACCEL|absMaxVEL||
      else if(type == "ROBOTPARAMS")
      {
        //turns out that the robot does not have good max and min values
        const ArRobotParams * params;
330     params = robot->getRobotParams();
        robot->unlock();

        SbMessage m("SEND");
        m.addArgument("MSG");
335     m.addArgument("ROBOTPARAMS");
        m.addArgument(m.convToString(params->getTransVelMax()));
        m.addArgument(m.convToString(params->getRotVelMax()));
        m.addArgument(m.convToString(params->getTransAccel()));
        m.addArgument(m.convToString(params->getTransDecel()));
340     m.addArgument(m.convToString(params->getRotAccel()));
        m.addArgument(m.convToString(params->getAbsoluteMaxVelocity()));
        sendMsgToHandler(m);
        return;
      }
345   // ROBOTNAME|| Returns Robot Name
      else if(type == "ROBOTNAME")
      {
        string name(robot->getRobotName());
        robot->unlock();
350     SbMessage m("SEND");
        m.addArgument("MSG");

        m.addArgument("ROBOTTYPE");
        m.addArgument(name);
355     sendMsgToHandler(m);
        return;
      }
      // ROBOTTYPE|| Returns Robot Type
      else if(msg.getArgAsStr(0) == "ROBOTTYPE")
360   {
        string type(robot->getRobotType());
        robot->unlock();
        SbMessage m("SEND");
        m.addArgument("MSG");
365     m.addArgument("ROBOTTYPE");
        m.addArgument(type);
        sendMsgToHandler(m);
        return;
      }
370

      robot->unlock();
      cout << "Firing SbRobotML with msg : " << msg.getAsStr() << endl;
    }
```

```
#ifndef SbSaphiraML_h
#define SbSaphiraML_h

#include "SbMsgListener.h"
#include "SbMessage.h"
#include "Aria.h"

#include <string>
using namespace std;

/// Listener/wrapper for Saphira - intelligent avoidance software.
/// This class calls Saphira function that avoid objects and intelligently
/// maneuver around ("IGOTO"). The avoidance part can be turned off allowing
/// for the robot to naively go to the desired position - "GOTO"
/// The distance when the robot assumes it has successfully reached the goal
/// can be set using "GETSETDONE" while "GOTOSETCLOSE" sets the distance to
/// goal that the robot starts to slow down. "GOTOWINDOW" describes the size
/// of the window that Saphira will look.
/// @author Marcin Balicki : Cooper Union - Mechanical Engineering
/// @date March 2004
/// @warning Uses SbMsgListener's SbMsgListener::sendMsgToHandler().
/// @warning Test the exact motion patterns of "GOTO", "IGOTO" may be different
/// every time.
/// @bug None

class SbSaphiraML : public SbMsgListener
{
public:
  ///Constructor
  SbSaphiraML();
  ///Destructor
  ~SbSaphiraML();
  ///implements the SbMsgListener pure virtual function.
  ///@param &msg SbMessage which caused the listener to fire.
  virtual void fire(const SbMessage &msg);
  ///initilizes the gradient.
  void init(void);
private:
  ///internal robot pointer
  ArRobot * robot;
  ///checks if the robot exists and is connected
  bool checkForRobot(void);
  ///message type variable
  string type;
  ///have we initilized
  bool inited;
  ///saphira variables
  ///(mm)distance away the robot is from the goal when it switches to close mode
  int close;
  ///(mm)the distance away the robot is from the
  ///goal when the gradient decides its done.
  int done;
  ///(mm)tSets the size of the gradient window
  ///dimensions for gradient path planning.
  int winX;
  ///(mm)tSets the size of the gradient window
  ///dimensions for gradient path planning.
  int winY;
};

#endif
```

```
#undef TCP
#undef UDP
#include "SbSaphiraML.h"
#include "Saphira.h"
#include "SfGrad.h"
#include "Aria.h"

SbSaphiraML::SbSaphiraML()
{
  inited = false;
  robot = NULL;
  done = 50;
  close = 200;
  winX = 20000;
  winY = 20000;
  //adds message types to the msgListener (base) vector for registering with
  // msg handler.
  addMsgType("GOTO");
  addMsgType("IGOTO");
  //getSetDone accepts done distance
  addMsgType("GOTOSETDONE");
  //gotoSetDone accepts close distance
  addMsgType("GOTOSETCLOSE");
  //gotoSetDone accepts close distance
  addMsgType("GOTOWINDOW");
  addMsgType("STOP");
}

SbSaphiraML::~SbSaphiraML() { }

///Checks if an instance of the robot exists and if it is connected.
bool SbSaphiraML::checkForRobot(void)
{
  //Check if the robot exists, if it does and its connected than return true;
  if((robot = Aria::findRobot("SBROBOT")) == NULL)
    return false;
  robot->lock();
  if(robot->isConnected())
  {
    if(!inited) init();
    robot->unlock();
    return true;
  }
  robot->unlock();
  return false;
}

void SbSaphiraML::fire(const SbMessage & msg)
{
  //has to be connected first.
  if(!checkForRobot())
  {
    cout << "Robot doesn't exist or is not connected:"
    <<" Will not fire command. (SbSaphiraML)"
    << endl;
    sendMsgToHandler("SEND|MSG|DISCONNECTED||");
    return;
  }
  //check type of msg
  type = msg.getType();
  robot->lock();

  // GOTO|double|double|| just turns and goes to the goal without avoidance.
  // accepts x,y coordinates (+ x is straight ahead and 90 deg to the left.
  if(type == "GOTO")
  {
     //this is required otherwise previous direct motion commands will prevent
     // internal action implementation of Saphira from activating.
    robot->clearDirectMotion();
    // for some reason saphira's heading does not get updated after the last
    // time so it does some odd rotating before it goes to goal.
    robot->moveTo(ArPose(robot->getX(),robot->getY(), 0.0));
    //turns off sonar data so it just goes naively straight ot the goal.
    sfGradUseSonar(0);
    sfGradSetGoal(msg.getArgAsDbl(0) + robot->getX(),
    msg.getArgAsDbl(1) + robot->getY());
    //cout<<"SFGOTO: " <<msg.getArgAsDbl(0)+robot->getX()<<" "
    // <<msg.getArgAsDbl(1)+robot->getY()<<endl;
  }
  // IGOTO|double|double|| goes to the given goal but uses sonar data to
  /// avoid obsticles.
  // accepts x,y coordinates (+ x is straight ahead and 90 deg to the left.
  else if(type == "IGOTO")
  {
    //this is required otherwise previous direct motion commands will prevent
    // internal action implementation of Saphira from activating.
         robot->clearDirectMotion();
    //turns on sonar data so it just goes naively straight ot the goal.
    sfGradUseSonar(1);
    sfGradSetGoal(msg.getArgAsDbl(0) + robot->getX(),
    msg.getArgAsDbl(1) + robot->getY());
  }
  // STOP|| stops the Saphira gradiant path planning.
  else if(type == "STOP")
  {
    //this wills top all motions and actions (actions will be blocked until
    //the precidence time
    // is exceeded or cleardirectmotion is called.
    sfGradStop();
```

```
100     }
        // GOTOSETDONE|double|| tolerance on reaching the goal within X
        // Set how close we need to be to a goal to slow done or be done.
        else if(type == "GOTOSETDONE")
        {
105
          done = msg.getArgAsInt(0);
          sfGradSetDone(close, done);
        }
        // GOTOSETCLOSE|double||  distance to goal where the robot starts slowingdown
110     // double
        else if(type == "GOTOSETCLOSE")
        {
          close = msg.getArgAsInt(0);
          sfGradSetDone(close, done);
115     }
        // GOTOWINDOW|double|double|| area in which the robot will look
        // for possible paths to goal x,y
        else if(type == "GOTOWINDOW")
        {
120       winX = msg.getArgAsInt(0);
          winY = msg.getArgAsInt(1);
          sfGradSetMax(winX, winY);
        }
        robot->unlock();
125     cout << "Firing SbSaphira with msg : "<< msg.getAsStr() << endl;
      }

      void SbSaphiraML::init(void)
      {
130     //cast the ArRobot into SfRobot so Sf knows what we are working with.
        Sf::ourRobot = (SfRobot *) robot;
        Sf::init();
        sfGradInit();
        sfGradSetMax(winX, winY);
135     // max size of window we search
        sfGradSetDone(close, done);
        inited = true;
      }
      /* void sfGradSetTurnRadius (int turnRadius)
140   Sets the turn radius needed to let it turn instead of back, make it back with
      sfGradSetCanBack.
      sfGradGetTurnRadius (void) Gets the turn radius needed to let it turn instead
      of back, make sure it
      is allowed to back up with with sfGradGetCanBack.
145   sfGradSetCanBack (int canBack)
      Gets if the grad action can back up, 0 = no,
      1 = when appropriate, 2 = always. int sfGradGetCanBack (void) Gets if the grad
      action can back up, 0 = no, 1 = when appropriate, 2 = always.
      void sfGradUseFinalApproach (int which)
150   void sfGradSetFinalApproachSpeed (int speed) */
      //other commands
      //sfDoGoal()
      //sfGradIsActive()
      //theGradAction (a global from Saphira). sfGrad is an activity.
155   /* Exerpt from Saphira PDF file. For efficiient movement based on local
      obstacles and world maps, Saphira has
      a realtime path planner based on the gradient method [Konolige, IROS 2000].
      For planning paths and moving in a world map, the gradient module is typically
      used with Markov Localization to keep the robot registered with a map as it
160   moves (sfLoc, sfLocLaser, and sfLocFl libraries).
      Gradient Path Planning is a process for determining optimal
      paths for the robot,
      in real time. These paths can take into account both local obstacles, sensed by
      sonars and/or laser range- nder devices; and global map information such as
165   the location of walls and other structural obstacles. At each sync cycle (100
      ms), the Gradient module calculates the lowest-cost path from a goal point or
      set of goal points to the robot. The algorithm starts by considering a local
      neighborhood connecting the robot and the goal or goals, and then expands
      its search if no path is found. There is a user-settable limit on the size of
170   the neighborhood considered.
      Gradient uses a square-cell grid as a cost field for
      determining good paths. You
      can set the grid resolution; a typical resolution is 10 mm. The maximum size
      of the grid can also be set.
175   Costs are calculated from a set of obstacles, obtained from pre-existing maps
      and from local sensor readings. Here are the obstacles sources:
      1. Artifacts in a world map. Load a world map, and call
      sfGradUse- Artifacts(true).
      2. A grid map created from the laser navigation software. A grid map
180   is typically
      loaded into the localization system using mcLoadScanMap(). To
      access this map from the gradient module, use sfGradSetMap(mcGetObject()
      Finally, turn on grid map use by calling sfGradUseMap(true). Grid maps and
      world maps may both be used at the same time. In
185   this case, usually the grid map will contain the basic geometric information
      about the
      world, while the artifacts are special areas for the robot, e.g., keep-out
      areas.
      3. Laser and sonar readings. These can be turned on and off with
190   sfGradUse- Laser() (p. 10) and sfGradUseSonar() (p. 15).
      The cost field radiates outward from obstacles, to create a safety cushion for
      the robot. You can adjust this cushion using sfGradObsParams() (p. ??).
      There is a local controller, implemented as an action (SfGradAction), that
      drives
195   the robot along the gradient path. This action controls the speed of the robot
      in a parameterized fashion, and can be set with sfGradSetSpeed. A further
      refinement for rectangular robots is the ability to back up when appropriate;
      this behavior is controlled by sfGradSetCanBack and sfGradSetTurnRadius. If
```

```
    the robot needs to turn more than 45 degrees to follow the path, it will see if
200 its turn radius is clear. If it isn't, it will try to back up if possible,
    and turn around when it is clear of obstacles.
    Some relevant sample Colbert load les: grad.act - basic gradient, without
    localization for localization with respect to a world, look at loc.act
    scan.act - localization and gradient using a grid map (from the Laser
205 Localization/ Navigation module).
    You can also use the gradient for a final approach to a given position, to do
    this call sfGradUseFinalApproach... you will also want to adjust the gradient
    obstacle parameters with sfGradObsParams and the done distance with sfGrad-
    SetDone (the close dist is ignored for the final approach). You may want to
210 set the speed with sfGradSetFinalApproachSpeed.
    Another refinement on gradient is to use a global path. In this approach, a
    first
    global path is calculated to the goal, and then it is fixed. The robot
    calculates
215 a local gradient to a goal point at some distance ahead on the path. Compute
    time is saved, because only the local path is updated on each cycle.
    Set the global path with the boolean sfGradSetLocalPath(int dist) (p. 14).
    When a new goal is given, the global path is calculated and stored, then used
    as above. dist is the distance to look ahead on the global path; if 0,
220 no global path is used. */
```

```
#ifndef SbServerML_h
#define SbServerML_h


#include "SbMessage.h"
#include "SbSocket.h"

#include <string>
#include <vector>
#include <map>
#include "SbMsgListener.h"

using namespace std;


/// Manages TCP/IP server: multiple socket connections, send/recv ASCII data.
/// This class uses SbSockets to listen for incoming connections. It allows
/// multiple clients to connect. It will broadcast any message sent via fire()
/// to all connected sockets. It does not automatically check for incomming
/// sockets/messages, which happens via the cycleOnce() function.
/// Clients can  disconnect by sending "QUIT" message and all clients are
/// disconnected when "QUITALL" message type is fired. The "SEND" type
/// message broadcasts the message embedded in the message, (SEND is stripped)
/// Start the server with the open() function, add commands with the
/// addCommand function and remove commands with remCommand, and close
/// the server with the close function.
/// It was modeled from ActivMedia's ArNetServer.
/// @note You can use the sendMsgToHandler to fire messages back to the handler.
/// @author Marcin Balicki : Cooper Union - Mechanical Engineering
/// @date March 2004
/// @warning Uses SbMsgListener's SbMsgListener::sendMsgToHandler()
/// @bug None

class SbServerML : public SbMsgListener
{
public:
  /// Constructor
  SbServerML(void);
  ///Constructor creates a socket on specified port.
  SbServerML(string port);
  /// Destructor
  ~SbServerML(void);
  ///set the server port
  void setPort(string);
  ///returns the port
  string getPort()
  { return port; };
  /// Initializes the server
  bool open(void);
  /// Initializes the server with port
  bool open(string port);
  /// Runs once through the servers functions
  void cycleOnce(void);
  /// Closes thegetDelimiter(); server
  void close(void);
  /// Disconnects all Clients
  void disconnectClients(void);
  /// Disconnects one client.
  void disconnectClient(SbSocket *);
  /// Disconnects the Client
  string getClientAddress(SbSocket *);
  /// Sees if the server is running and open for connections
  bool isOpened(void)
  { return opened; };
  ///Returns number of clients connected.
  int numConnected(void)
  { return myConns.size(); };
  ///send a SbMessage object
  bool sendData(const SbMessage &);
  ///send a basic string.
  bool sendData(string);
  ///parses the databuffer for message and assigns the oldesvectort one to
  ///the object argument. This only works in conjunction with cycleOnce().
  bool getMsg(SbMessage &);
  ///sets the delimiter for msg parsing
  void setDelimiter(const string & d)
  { delimiter = d; };
  ///gets the current delimiter
  string getDelimiter(void)
  { return delimiter; };
  ///gets data and then clears the buffer
  string getData(SbSocket *);
  ///gets data and clears the buffer,
  ///but lets the user know when the buffer is not empty.
  bool getData(SbSocket *, string &);
  /// implements the SbMsgListener pure virtual function.
  /// @param &msg SbMessage which caused the listener to fire.
  virtual void fire(const SbMessage &msg);
protected:
  /// checks if a string is an integer.
  bool isInteger(string);
  /// server open flag
  bool opened;
  /// port
  string port;
  /// default port string
  string defaultPort;
  /// message delimiter string
  string delimiter;
```

```
100     /// type of message string
        string type;
        /// instance of the socket.
        SbSocket serverSocket;
        /// map of connected sockets(KEY) with their respective buffers
105     /// (socketPointer,buffer)
        map < SbSocket *, string * > myConns;
        /// connection map iterator
        typedef map < SbSocket *, string * >::iterator myConnsIterator;
        /// vector holding pointers to sockets that have been flagged because
110     /// of a communication error and will be deleted.
        vector < SbSocket * > deleteConnsVector;
    };

    #endif
115
```

```
#include "SbServerML.h"
#include <arpa/inet.h>
#include <iostream>


SbServerML::SbServerML(void)
{
  defaultPort = "5555";
  setPort(defaultPort);
  setDelimiter(string("|"));
  opened = false;
  //adds message types to the Listener (base) vector for registering with
  // msg handler.
  addMsgType("QUIT");
  addMsgType("SEND");
  addMsgType("QUITALL");
}

SbServerML::SbServerML(string port)
{
  setPort(port);
  setDelimiter(string("|"));
  opened = false;
}

SbServerML::~SbServerML(void)
{ close(); }

void SbServerML::setPort(string p)
{
  if(isInteger(p.c_str()))
    port = p;
  else
    port = defaultPort;
}

bool SbServerML::open(string port)
{
  setPort(port);
  return open();
}

bool SbServerML::open(void)
{
  if(isOpened())
  {
    cout << "Server is already open, and running (SbServerML)" << endl;
    return false;
  }
  if(serverSocket.open(atoi(getPort().c_str()), SbSocket::TCP))
  {
    serverSocket.setLinger(0);
    serverSocket.setNonBlock();
    cout << "Server opened on port: "<< getPort() << " (SbServerML)"<< endl;
    opened = true;
  }
  else
  {
    cout << "Socket Error: "<< serverSocket.getErrorStr()
    << "(SbServerML)"<< endl;
    opened = false;
    return false;
  }
  return true;
}

void SbServerML::cycleOnce(void)
{
  SbSocket * socket;
  unsigned char * bytes;
  char * str = NULL;
  //temmporary socket for incoming connections
  SbSocket incomingConn;

  if(!isOpened())
    return;

  // get any new sockets that want to connect
  // if there are new clients transfer it, and store it in the connection map.
  while(serverSocket.accept(& incomingConn) && incomingConn.getFD() >= 0)
  {
    incomingConn.setNonBlock();
    socket = new SbSocket;
    socket->transfer(& incomingConn);
    //add a socket as a key and a new string to match it.
    myConns[socket] = new string("");
    //send a message to new client
    socket->writeString("MSG|WELCOME TO StudioBlue RCCI SERVER||");
    //someone has connected
    cout << "Connection established with : "<< socket->getAddress() << ":"
    << getPort() << endl;
  }

  //Read data now.
  //cycle through all of clients in the map, read in data,
  //store it with the corresponding socket(key) in the map.

  for(myConnsIterator it = myConns.begin(); it != myConns.end(); ++it)
  {
```

```
100       //Now for each client read the string.
          while((str = (it->first)->readString()) != NULL)
          {
            // if this is null terminating char then there wasn't anything said
            if(str[0] == '\0')
105           break;
            // make sure we read something
            // if yes save the data to a string corresponding to the socket
            (* it->second) += str;
          }
110       ///there was a problem NULL was returned, flag that socket
          if(str == NULL)
          {
            //disconnected
            cout << "Lost Connection with Client : " << (it->first)->getAddress()
115         << " (SbServerML)" << endl;
            //close the client connection
            (it->first)->close();
            //remove the socket by adding to delete socket list
            deleteConnsVector.push_back(it->first);
120       }
        }
        //now delete the connections that were flaged as disconnected
        //remove from memory if there are any.
        if(deleteConnsVector.size() > 0)
125     {
          vector < SbSocket * >::iterator vIt;
          //iterate to the vector of toDelete iterators
          for(vIt = deleteConnsVector.begin(); vIt != deleteConnsVector.end(); ++vIt)
          {
130         //delete the string buffer
            // corresponding to socket pointer to by vIt iterator
            delete myConns[(* vIt)];
            //delete the socket.
            delete(* vIt);
135         // or is it *(vIt)->second)
            //remove the socket instance from the map of Socket-String pair.
            int count = myConns.erase((* vIt)); //returns number of deletions
          }
          //reset.
140       deleteConnsVector.clear();
        }

      }

145   void SbServerML::close(void)
      {
        if(!isOpened())
          return;
```

```
        cout << "Shutting down server." << endl;
150     sendData("MSG|SHUTTING DOWN TCP SERVER||");
        opened = false;
        disconnectClients();
        serverSocket.close();
      }
155
      void SbServerML::disconnectClients(void)
      {
        for(myConnsIterator it = myConns.begin(); it != myConns.end(); ++it)
        {
160       //need to cycle through all clients, disconnect, and remove.
          (it->first)->close();
          //delete the string buffer corresponding
          //to socket pointer to by map iterator
          delete it->second;
165       //delete the socket.
          delete it->first;
        }
        //clear the list
        myConns.clear();
170   }

      void SbServerML::disconnectClient(SbSocket * socket)
      {
        cout << "Client: " << socket->getAddress()
175     << " requested Disconnect (SbServerML)" << endl;
        socket->writeString("MSG|BYE||");
        socket->close();
        //not sure if I can do this (itertator pointing to iterator)
        //delete the string buffer corresponding to socket pointer to by map iterator
180     delete myConns[socket];
        //delete the socket.
        delete socket;
        //erase an instance of it in the map.
        myConns.erase(socket);
185   }

      string SbServerML::getClientAddress(SbSocket * socket)
      { return socket->getAddress(); }

190   bool SbServerML::sendData(const SbMessage & msg)
      {
        //cout<<"sending"<<endl;
        return (sendData(msg.getAsStr()));
      }
195
      bool SbServerML::sendData(string str)
      {
        // have to be careful because SbSocket does some
```

```
    // funny stuff with messages (512 bytes max?)
200 if(!isOpened() || (myConns.size() == 0))
    {
      cout << "(SbServerML) No client connected or server not running:"
      << " will not send this msg: "<< str << endl;
      //  cout<<"Can't send message no client connected or server not running
205   // (SbServerML)"<<endl;
      return false;
    }

    for(myConnsIterator it = myConns.begin(); it != myConns.end(); ++it)
210 {
      //char buf[2049];
      //may need to limit the size of the string or break them up.
      //512 bytes is the limit so if the message is large send it in parts.
      //basically less than 512 characters.
215   //(string size doesn't mean in bytes)
      string tempStr = str;
      while(tempStr.size() > 255)
      {
        (it->first)->writeString((tempStr.substr(0, 254)).c_str());
220     tempStr.erase(0, 254);
      }
      //send the last few chars.
      (it->first)->writeString(tempStr.c_str());
    }
225 return true;
  }

  bool SbServerML::getMsg(SbMessage & msg)
  {
230 //have to be careful because SbSocket does some funny
    // stuff with messages (512 bytes max?)
    if(!(isOpened()) || myConns.size() == 0)
    {
      // cout<<"Can't get message no client connectd
235   // or server not running (SbServerML)"<<endl;
      return false;
    }
    //if the delimtor is not found exit
    string dd = getDelimiter() + getDelimiter();
240 bool ret = false;
    //iterate through clients data buffers when the first message
    //is found return it and exit.
    //the first client in the list has inherent priority.
    for(myConnsIterator it = myConns.begin(); it != myConns.end(); ++it)
245 {
      //if can't find the end delimiter : no msg so check the next socket.
      if((it->second)->find(dd) == (it->second)->npos)
      {
        ret = false;
250     continue;
      }
      //if the creation is not successful than erase
      // the bad string and return false
      else if(!msg.create((it->second)->substr(0, (it->second)->find(dd) + 2)))
255   {
        //there was a problem, discard this part of the message
        (it->second)->erase(0, (it->second)->find(dd) + 2);
        sendData("MSG|MALFORMED MESSAGE||");
        cout << "Bad Message: discarded the string SbServerML" << endl;
260     ret = false;
        continue;
      }
      //otherwise it is new msg is created,
      // just clean up the buffer and return true this leaves the for loop.
265   (it->second)->erase(0, (it->second)->find(dd) + 2);
      ret = true;
      //check if the msg is a quit message if yes,
      // than disconnect its owner(client).
      if(msg.getType() == "QUIT")
270     disconnectClient(it->first);
      return ret;
    }
    return ret;
  }
275
  string SbServerML::getData(SbSocket * socket)
  {
    string str = (* myConns[socket]);
    myConns[socket]->erase();
280 return str;
  }

  bool SbServerML::getData(SbSocket * socket, string & str)
  {
285 str = (* myConns[socket]);
    if(str.empty())
      return false;
    myConns[socket]->erase();
    return true;
290 }

  bool SbServerML::isInteger(string numStr)
  {
    bool result = true;
295 //checks for number of periods
    //need to look at this closely.
    int period = 0;
```

```
      for(unsigned int i = 0; i < numStr.size(); i++)
      {
300     char Ch = numStr[i];
        if(Ch == '.')
          period++;
        //cout<<"ALPHA: "<<isalpha(Ch)<<endl;
        if(!(Ch >= '0' && Ch <= '9'))
305       result = false;
      }
      return result;
    }

310 void SbServerML::fire(const SbMessage & msg)
    {
      //check type of msg
      type = msg.getType();
      // SEND|anything|anything|...|| this function strips the SEND
315   // takes the first argument
      // and sets it as a new message type and appends the rest ...sends new msg out
      // ie SEND|MSG|left|23||
      // will be sent out as MSG|left|23||

320   if(type == "SEND")
      {
        //removes SEND and sends
        //the rest of the message which includes type and so on.
        //create a type of message
325     SbMessage m(msg.getArgAsStr(0));
        for(unsigned int i = 1; i < msg.getNumOfArgs(); i++)
          m.addArgument(msg.getArgAsStr(i));
        //cout<<m.getAsStr()<<endl;
        sendData(m);
330   }
      // QUITALL|| will disconnect all the clients from the server.
      else if(type == "QUITALL")
      {
        cout << "Client request Disconnect (SbServerML)" << endl;
335     sendData(SbMessage("MSG|BYE||"));
        disconnectClients();
      }
    }
```

```
#ifndef SbSocket_h
#define SbSocket_h

#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/param.h>
#include <fcntl.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <errno.h>
#include <stdarg.h>
#include <string>

/// Unix Socket TCP/IP communication wrapper.
/// SbSocket is a layer which allows people to use the sockets networking
/// interface in unix operating system. All of the standard
/// commonly used socket functions are implemented. This class also contains
/// the file descriptor which identifies the socket to the operating system.
/// This class has been adopted from two ActivMedia's classes, ArSocket.cpp
/// and ArScoket_LIN.cpp
/// The ActivMedia original socket class was undocumented and limited the
/// transmission of large data packets. This class was also implemented
/// strictly for Linux.
/// @author Marcin Balicki : Cooper Union - Mechanical Engineering
/// @date March 2004
/// @bug Maximum message size is 5120 bytes, may work erratically with
/// windows telnet client and crash if overrun.
/// @warning This class was adopted from ARIA and therefore needs to follow
/// ActivMedia's redistribution terms :
/// @par Adopted from ARIA, and will follow ActivMedia's distribution terms:
/// @par
///   ActivMedia Robotics Interface for Applications (ARIA)
///   Copyright (C) 2002, ActivMedia Robotics, LLC
///     This program is free software; you can redistribute it and/or modify
///     it under the terms of the GNU General Public License as published by
///     the Free Software Foundation; either version 2 of the License, or
///    (at your option) any later version.
///@par
///     This program is distributed in the hope that it will be useful,
///     but WITHOUT ANY WARRANTY; without even the implied warranty of
///     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
///    GNU General Public License for more details.
///@par
///     You should have received a copy of the GNU General Public License
///     along with this program; if not, write to the Free Software
///     Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307 US
///@par
///If you wish to redistribute ARIA under different terms, contact
///ActivMedia Robotics for information about a commercial version of ARIA at
///robots@activmedia.com or
///ActivMedia Robotics, 19 Columbia Drive, Amherst, NH 03031; 800-639-9481

class SbSocket
{
public:

  /// type of connection enumerator
  enum Type
  { UDP, TCP, Unknown };
  /// type of error enumerator
  enum Error
  { NoErr, NetFail, ConBadHost, ConNoRoute, ConRefused };
  /// Constructor
  SbSocket();
  ///Constructs the socket and connects it to the myConn[socket]) given host.
  ///@param host hostname of the server to connect to
  ///@param port port number of the server to connect to
  ///@param type protocol type to use
  SbSocket(const char * host, int port, Type type);

  ///Constructs the socket and opens it as a server port.
  ///@param port port number to bind the socket to
  ///@param doClose automaticaly close the port if the socket is destructed
  ///@param type protocol type to use
  SbSocket(int port, bool doClose, Type type);
  /// Destructor
  ~SbSocket();
  /// flag whether the socket is initialized or not
  static bool ourInitialized;

  ///Copy socket structures.
  ///Copy from one Socket to another will still have
  ///the first socket close the file descripter when it is destructed.
  bool copy(int fd, bool doclose);

  /// Copy socket structures
  void copy(SbSocket * s)
  {
    myFD = s->myFD;
    myDoClose = false;
    mySin = s->mySin;
  }

  /// Transfer ownership of a socket
```

```
100     /// transfer() will transfer ownership to this socket. The input socket
        /// will no longer close the file descriptor when it is destructed.
        void transfer(SbSocket * s)
        {
          myFD = s->myFD;
105       myDoClose = true;
          s->myDoClose = false;
          mySin = s->mySin;
          myType = s->myType;
        }
110     /// returns the address in the string format.
        string getAddress(void);
        /// Connect as a client to a server
        bool connect(const char * host, int port, Type type);
        /// Open a server port
115     bool open(int port, Type type);
        /// Simply create a port.
        bool create(Type type);
        /// Find a valid unused port and bind the socket to it.
        bool findValidPort(int startPort);
120     /// Connect the socket to the given address
        bool connectTo(const char * host, int port);
        /// Connect the socket to the given address
        bool connectTo(struct sockaddr_in * sin);
        /// Accept a new connection
125     bool accept(SbSocket * sock);
        /// Close the socket
        bool close();

        /// Write to the socket
130     /// @param buff buffer to write from
        ///@param len how many bytes to write
        ///@return number of bytes written
        int write(const void * buff, size_t len)
        {
135       struct timeval tval;
          fd_set fdSet;
          tval.tv_sec = 0;
          tval.tv_usec = 0;
          FD_ZERO(& fdSet);
140       FD_SET(myFD, & fdSet);
          if(select(myFD + 1, NULL, & fdSet, NULL, & tval) <= 0)
            return 0;
          return (::write(myFD, (char *) buff, len));
        }
145
        ///Read from the socket
        ///@param buff buffer to read into
        ///@param len how many bytes to read
        ///@param msWait if 0, don't block, if > 0 wait this long for data
150     ///@return number of bytes read
        int read(void * buff, size_t len, unsigned int msWait = 0)
        {
          if(msWait != 0)
          {
155         struct timeval tval;
            fd_set fdSet;
            tval.tv_sec = msWait / 1000;
            tval.tv_usec = (msWait % 1000) * 1000;
            FD_ZERO(& fdSet);
160         FD_SET(myFD, & fdSet);
            if(select(myFD + 1, & fdSet, NULL, NULL, & tval) <= 0)
              return 0;
          }
          return (::recv(myFD, (char *) buff, len, 0));
165     }

        /// Send a message on the socket
        int sendTo(const void * msg, int len)
        {
170       return (::sendto(myFD, (char *) msg, len, 0, (struct sockaddr *) & mySin,
          sizeof(mySin)));
        }

        /// Send a message on the socket
175     int sendTo(const void * msg, int len, struct sockaddr_in * sin)
        {
          return (::sendto(myFD, (char *) msg, len, 0, (struct sockaddr *) sin,
          sizeof(struct sockaddr_in)));
        }
180
        /// Receive a message from the socket
        int recvFrom(void * msg, int len, sockaddr_in * sin)
        {
          unsigned int i = sizeof(sockaddr_in);
185       return (::recvfrom(myFD, (char *) msg, len, 0,
          (struct sockaddr *) sin, & i));
        }

        /// Convert a host string to an address structure
190     static bool hostAddr(const char * host, struct in_addr & addr);
        /// Convert an address structure to a host string
        static bool addrHost(struct in_addr & addr, char * host);
        /// Get the localhost address
        static std::string getHostName();
195     /// Get the socket name. Stored in SbSocket::mySin
        bool getSockName();

        /// Accessor for the sockaddr
```

```
    struct sockaddr_in * sockAddrIn()
200 { return (& mySin); }
    /// Accessor for the in_addr
    struct  in_addr * inAddr()
    { return (& mySin.sin_addr); }
    /// Accessor for the port of the sockaddr
205 unsigned short int inPort(void)
    { return (mySin.sin_port); }

    /// Convert addr into string numerical address
    static void inToA(struct in_addr * addr, char * buff);
210
    /// Size of the sockaddr
    static const size_t sockAddrLen()
    { return (sizeof(struct sockaddr_in)); }

215 /// Max host name length
    static const size_t maxHostNameLen()
    { return (MAXHOSTNAMELEN); }

    /// Host byte order to network byte order
220 static unsigned int hostToNetOrder(int i);
    /// Network byte order to host byte order
    static unsigned int netToHostOrder(int i);
    /// Set the linger value
    bool setLinger(int time);
225 /// Set broadcast value
    bool setBroadcast();
    /// Set the reuse address value
    bool setReuseAddress();
    /// Set socket to nonblocking
230 bool setNonBlock();

    /// Change the doClose value
    void setDoClose(bool yesno)
    { myDoClose = yesno; }
235
    /// Get the file descriptor
    int getFD() const
    { return (myFD); }

240 /// Get the protocol type
    Type getType() const
    { return (myType); }

    /// Get the last error string
245 const std::string & getErrorStr() const
    { return (myErrorStr); }
```

```
    /// Get the last error
    Error getError() const
250 { return (myError); }

    ///Writes a string to the socket (adding end of line characters).
    ///this cannot write more than 5120 bytes
    ///@param *str the string to write to the socket
255 ///@return number of bytes written
    int writeString(const char * str,...);

    /// Same as writeString, but no varargs, wrapper for java
    int writeStringPlain(const char * str)
260 { return writeString(str); }

    /// Reads a string from the socket.
    /// The original function could only read strings less than 512 characters
    /// as long as it read the characters into its own internal
265 /// buffer (to compensate for some of the things the DOS telnet does).
    /// I have expanded this to 5120 chars to see what the issue is.
    /// Seems to work fine.
    /// @return if there was good data return string, if there was no
    /// data, a string with a "\0" first character and if the
270 /// socket was bad it returns NULL
    char * readString(void);

    /// Sets echoing on the readString calls this socket does
    void setEcho(bool echo)
275 {
      myStringAutoEcho = false;
      myStringEcho = echo;
    }

280 /// Gets if we are echoing on the readString calls this socket does
    bool getEcho(void)
    { return myStringEcho; }

  protected:
285 /// internal function that echos strings from read string
    void doStringEcho(void);
    // internal crossplatform init (mostly for string reading stuff)
    void internalInit(void);
    /// internal type of connection (udp,tcp)
290 Type myType;
    /// type of error
    Error myError;
    /// type of error in string format
    std::string myErrorStr;
295 /// flag for closing socket
    bool myDoClose;
    ///  file descriptor
```

```
      int myFD;
      /// flag for blocking
300   bool myNonBlocking;
      /// structure of socket address
      struct sockaddr_in mySin;
      /// auto echo flag
      bool myStringAutoEcho;
305   /// echo flag
      bool myStringEcho;
      ///shoulod implement a variable for the string buffer size.
      const static int BUFFSIZE = 5120;
      /// this is the buffer that stores socket data
310   char myStringBuf[BUFFSIZE];
      /// helpful in reading bytes from the buffer
      size_t myStringPos;
      /// empty string buffer
      char myStringBufEmpty[1];
315   /// keeps the last position in reading buffer
      size_t myStringPosLast;
      /// flag for escape character
      bool myStringGotEscapeChar;
      /// flag for complete data
320   bool myStringGotComplete;
      /// flag for successful echo
      bool myStringHaveEchoed;
    };

325 #endif // SbSocket_h
```

```
//ActivMedia Robotics Interface for Applications (ARIA)
//Copyright (C) 2002, ActivMedia Robotics, LLC
//
//
//     This program is free software; you can redistribute it and/or modify
//     it under the terms of the GNU General Public License as published by
//     the Free Software Foundation; either version 2 of the License, or
//     (at your option) any later version.
//
//     This program is distributed in the hope that it will be useful,
//     but WITHOUT ANY WARRANTY; without even the implied warranty of
//     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//     GNU General Public License for more details.
//
//     You should have received a copy of the GNU General Public License
//     along with this program; if not, write to the Free Software
//     Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307 USA
//
//If you wish to redistribute ARIA under different terms, contact
//ActivMedia Robotics for information about a commercial version of ARIA at
//robots@activmedia.com or
//ActivMedia Robotics, 19 Columbia Drive, Amherst, NH 03031; 800-639-9481


#include <errno.h>
#include <stdio.h>
#include <netdb.h>
#include <arpa/inet.h>
#include "SbSocket.h"

void SbSocket::internalInit(void)
{
  myStringAutoEcho= true;
  myStringEcho = false;
  myStringPosLast = 0;
  myStringPos = 0;
  myStringGotComplete= false;
  myStringBufEmpty[0] = '\0';
  myStringGotEscapeChars= false;
  myStringHaveEchoed= false;
}

int SbSocket::writeString(const char * str,...)
{
  char buf[BUFFSIZE];
  int len;
  va_list ptr;
  va_start(ptr, str);
  vsprintf(buf, str, ptr);
  va_end(ptr);
  len = strlen(buf);
  buf[len] = '\n';
  len++;
  buf[len] = '\r';
  len++;
  return write(buf, len);
}


// Reads a string from the socket
// The original function could only read strings less than 512 characters
// as long as it read the characters into its own internal
// buffer (to compensate for some of the things the DOS telnet does).
// I have expanded this to 5120 chars to see what the issue is.
// Seems to work fine.
// @return if there was good data return string, if there was no
// data, a string with a '\0' first character and if the
// socket was bad it returns NULL

char * SbSocket::readString(void)
{
  size_t i;
  int n;
  myStringBufEmpty[0] = '\0';

  // read one byte at a time
  for(i = myStringPos; i < sizeof(myStringBuf); i++)
  {

    n = read(& myStringBuf[i], 1);
    if(n > 0)
    {
      if(i == 0 && myStringBuf[i] < 0)
      { myStringGotEscapeChars= true; }
      if(myStringBuf[i] == '\n' || myStringBuf[i] == '\r')
      {
        if(i != 0)
          myStringGotComplete= true;
        myStringBuf[i] = '\0';
        myStringPos = 0;
        myStringPosLast = 0;
        // if we have leading escape characters get rid of them
        if(myStringBuf[0] < 0)
        {
          int ei;
          myStringGotEscapeChars= true;
          // increment out the escape chars
          for(ei = 0; myStringBuf[ei] < 0 || (ei > 0 && myStringBuf[ei-1] < 0);
          ei++);
```

```
100           // okay now return the good stuff
              doStringEcho();
              return & myStringBuf[ei];
            }
            // if we don't return what we got
105         doStringEcho();
            return myStringBuf;
          }
          // if its not an ending character but was good keep going
          else
110         continue;
        }
        // failed SbMsgReader
        else if(n == 0)
        { return NULL; }
115     else // which means (n < 0)
        {
          if(errno == EAGAIN)
          {
            myStringPos = i;
120         doStringEcho();
            return myStringBufEmpty;
          }
          perror("Error in reading from network");
          return NULL;
125     }
      }
      // if they want a 0 length string
      cout << "Some trouble in SbSocket::readString" << endl;
      return NULL;
130   }

      void SbSocket::doStringEcho(void)
      {
        size_t to;
135     if(!myStringAutoEcho && !myStringEcho)
          return;
        // if we're echoing complete thel ines
        if(myStringHaveEchoed && myStringGotComplete)
        {
140       write("\n\r", 2);
          myStringGotComplete = false;
        }

        // if there's nothing to send we don't need to send it
145     if(myStringPosLast == myStringPos)
          return;

        // we probably don't need it if its doing escape chars
        if(myStringAutoEcho && myStringGotEscapeChars)
150       return;

        myStringHaveEchoed = true;
        to = strchr(myStringBuf, '\0') - myStringBuf;
        write(& myStringBuf[myStringPosLast], myStringPos - myStringPosLast);
155     myStringPosLast = myStringPos;
      }

      /// We're always initialized in Linux
      bool SbSocket::ourInitialized = true;
160

      SbSocket::SbSocket() : myType(Unknown), myError(NoErr), myErrorStr(),
      myDoClose(true), myFD(-1),
      myNonBlocking(false), mySin()
      { internalInit(); }
165

      //Constructs the socket and connects it to themyConn[socket]) given host.
      //@param host hostname of the server to connect to
      //@param port port number of the server to connect to
      //@param type protocol type to use
170   SbSocket::SbSocket(const char * host, int port, Type type) : myType(type),
      myError(NoErr), myErrorStr(),
      myDoClose(true), myFD(-1),
      myNonBlocking(false), mySin()
      {
175     internalInit();
        connect(host, port, type);
      }

      //Constructs the socket and opens it as a server port.
180   //@param port port number to bind the socket to
      //@param doClose automaticaly close the port if the socket is destructed
      //@param type protocol type to use
      SbSocket::SbSocket(int port, bool doClose, Type type) : myType(type),
      myError(NoErr), myErrorStr(),
185   myDoClose(doClose), myFD(-1),
      myNonBlocking(false), mySin()
      {
        internalInit();
        open(port, type);
190   }

      SbSocket::~SbSocket()
      { close(); }

195

      string SbSocket::getAddress(void)
      {
        //return string(inet_ntoa(inAddr()));
```

```
    // Convert addr into string numerical address
    char buff[100];
    inToA(inAddr(), buff);
    //create a new string
    string s(buff);
    return s;
}

bool SbSocket::hostAddr(const char * host, struct in_addr & addr)
{
  struct hostent * hp;

  if(!(hp = gethostbyname(host)))
  {
    perror("gethostbyname");
    memset(& addr, 0, sizeof(in_addr));
    return (false);
  }
  else
  {
    bcopy(hp->h_addr, & addr, hp->h_length);
    return (true);
  }
}

bool SbSocket::addrHost(struct in_addr & addr, char * host)
{
  struct hostent * hp;

  hp = gethostbyaddr((char *) & addr.s_addr, sizeof(addr.s_addr), AF_INET);
  if(hp)
    strcpy(host, hp->h_name);
  else
    strcpy(host, inet_ntoa(addr));
  return (true);
}

std::string SbSocket::getHostName()
{
  char localhost[maxHostNameLen()];
  if(gethostname(localhost, sizeof(localhost)) == 1)
    return ("");
  else
    return (localhost);
}

bool SbSocket::connect(const char * host, int port, Type type)
{
  char localhost[maxHostNameLen()];

  if(!host)
  {
    if(gethostname(localhost, sizeof(localhost)) == 1)
    {
      myError = ConBadHost;
      myErrorStr = "Failure to locate host '";
      myErrorStr += localhost;
      myErrorStr += "'";
      perror("gethostname");
      return (false);
    }
    host = localhost;
  }

  bzero(& mySin, sizeof(mySin));
  if(!hostAddr(host, mySin.sin_addr))
    return (false);
  mySin.sin_family = AF_INET;
  mySin.sin_port = hostToNetOrder(port);

  if((type == TCP) && ((myFD = socket(AF_INET, SOCK_STREAM, 0)) < 0))
  {
    myError = NetFail;
    myErrorStr = "Failure to make TCP socket";
    perror("socket");
    return (false);
  }
  else if((type == UDP) && ((myFD = socket(AF_INET, SOCK_DGRAM, 0)) < 0))
  {
    myError = NetFail;
    myErrorStr = "Failure to make UDP socket";
    perror("socket");
    return (false);
  }

  myType = type;

  if(::connect(myFD, (struct sockaddr *) & mySin,
  sizeof(struct sockaddr_in)) < 0)
  {
    myErrorStr = "Failure to connect socket";
    switch(errno)
    {
      case ECONNREFUSED:
        myError = ConRefused;
        myErrorStr += "; Connection refused";
        break;
      case ENETUNREACH:
        myError = ConNoRoute;
```

```
            myErrorStr += "; No route to host";
            break;
        default:
            myError = NetFail;
            break;
        }
        //perror("connect");
        return (false);
      }
      return (true);
    }

    bool SbSocket::open(int port, Type type)
    {
      int ret;
      char localhost[maxHostNameLen()];

      if((type == TCP) && ((myFD = socket(AF_INET, SOCK_STREAM, 0)) < 0))
      {
        myErrorStr = "Failure to make TCP socket";
        perror("socket");
        return (false);
      }
      else if((type == UDP) && ((myFD = socket(AF_INET, SOCK_DGRAM, 0)) < 0))
      {
        myErrorStr = "Failure to make UDP socket";
        perror("socket");
        return (false);
      }

      myType = type;

      if(gethostname(localhost, sizeof(localhost)) == 1)
      {
        myErrorStr = "Failure to locate localhost";
        perror("gethostname");
        return (false);
      }

      bzero(& mySin, sizeof(mySin));
      if(!hostAddr(localhost, mySin.sin_addr))
        return (false);

      mySin.sin_family = AF_INET;
      mySin.sin_addr.s_addr = htonl(INADDR_ANY);
      mySin.sin_port = hostToNetOrder(port);

      if((ret = bind(myFD, (struct sockaddr *) & mySin, sizeof(mySin))) < 0)
      {
        myErrorStr = "Failure to bind socket to port ";
        sprintf(localhost, "%d", port);
        myErrorStr += localhost;
        perror("socket");
        return (false);
      }

      if((type == TCP) && (listen(myFD, 5) < 0))
      {
        myErrorStr = "Failure to listen on socket";
        perror("listen");
        return (false);
      }

      return (true);
    }

    bool SbSocket::create(Type type)
    {
      if((type == TCP) && ((myFD = socket(AF_INET, SOCK_STREAM, 0)) < 0))
      {
        myErrorStr = "Failure to make TCP socket";
        perror("socket");
        return (false);
      }
      else if((type == UDP) && ((myFD = socket(AF_INET, SOCK_DGRAM, 0)) < 0))
      {
        myErrorStr = "Failure to make UDP socket";
        perror("socket");
        return (false);
      }

      myType = type;

      if(getSockName())
        return (true);
      else
        return (false);
    }

    bool SbSocket::findValidPort(int startPort)
    {
      char localhost[maxHostNameLen()];

      if(gethostname(localhost, sizeof(localhost)) == 1)
      {
        myErrorStr = "Failure to locate localhost";
        perror("gethostname");
        return (false);
```

```
    }

    for(int i = 0; i + startPort < 65000; ++i)
400 {
      bzero(& mySin, sizeof(mySin));
      if(!hostAddr(localhost, mySin.sin_addr))
        return (false);
      mySin.sin_family = AF_INET;
405   mySin.sin_addr.s_addr = htonl(INADDR_ANY);
      mySin.sin_port = hostToNetOrder(startPort + i);
      if(bind(myFD, (struct sockaddr *) & mySin, sizeof(mySin)) == 0)
        break;
    }
410 return (true);
  }

  bool SbSocket::connectTo(const char * host, int port)
  {
415 char localhost[maxHostNameLen()];

    if(myFD < 0)
      return (false);

420 if(!host)
    {
      if(gethostname(localhost, sizeof(localhost)) == 1)
      {
        myErrorStr = "Failure to locate host '";
425     myErrorStr += localhost;
        myErrorStr += "'";
        perror("gethostname");
        return (false);
      }
430   host = localhost;
    }

    bzero(& mySin, sizeof(mySin));
    if(!hostAddr(host, mySin.sin_addr))
435   return (false);
    mySin.sin_family = AF_INET;
    mySin.sin_port = hostToNetOrder(port);

    return (connectTo(& mySin));
440 }

  bool SbSocket::connectTo(struct sockaddr_in * sin)
  {
    if(::connect(myFD, (struct sockaddr *) sin, sizeof(struct sockaddr_in)) < 0)
445 {
      myErrorStr = "Failure to connect socket";
      perror("connect");
      return (0);
    }
450 return (1);
  }

  char * str;

455 bool SbSocket::close()
  {
    if(myDoClose &&::close(myFD))
    {
      myFD = -1;
460   return (false);
    }
    else
    {
      myFD = -1;
465   return (true);
    }
  }

  bool SbSocket::setLinger(int time)
470 {
    struct linger lin;

    if(time)
    {
475   lin.l_onoff = 1;
      lin.l_linger = time;
    }
    else
    {
480   lin.l_onoff = 0;
      lin.l_linger = time;
    }

    if(setsockopt(myFD, SOL_SOCKET, SO_LINGER, & lin, sizeof(lin)) != 0)
485 {
      myErrorStr = "Failure to setsockopt LINGER";
      perror("setsockopt");
      return (false);
    }
490 else
      return (true);
  }

  bool SbSocket::setBroadcast()
495 {
```

```
    if(setsockopt(myFD, SOL_SOCKET, SO_BROADCAST, NULL, 0) != 0)
    {
      myErrorStr = "Failure to setsockopt BROADCAST;"
      perror("setsockopt");
      return (false);
    }
    else
      return (true);
  }

  bool SbSocket::setReuseAddress()
  {
    int opt = 1;

    if(setsockopt(myFD, SOL_SOCKET, SO_REUSEADDR, (char *) & opt,
    sizeof(opt)) != 0)
    {
      myErrorStr = "Failure to setsockopt REUSEADDR;"
      perror("setsockopt");
      return (false);
    }
    else
      return (true);
  }

  bool SbSocket::setNonBlock()
  {
    if(fcntl(myFD, F_SETFL, O_NONBLOCK) != 0)
    {
      myErrorStr = "Failure to fcntl O_NONBLOCK;"
      perror("fcntl");
      return (false);
    }
    else
    {
      myNonBlocking = true;
      return (true);
    }
  }

  //Copy socket structures.
  //Copy from one Socket to another will still have
  //the first socket close the file descripter when it is destructed.
  bool SbSocket::copy(int fd, bool doclose)
  {
    unsigned int len;

    myFD = fd;
    myDoClose = doclose;
    myType = Unknown;

    len = sizeof(struct sockaddr_in);
    if(getsockname(myFD, (struct sockaddr *) & mySin, & len))
    {
      myErrorStr = "Failed to getsockname on fd ;"
      perror("setsockopt");
      return (false);
    }
    else
      return (true);
  }

  bool SbSocket::accept(SbSocket * sock)
  {
    unsigned int len;

    len = sizeof(struct sockaddr_in);
    sock->myFD =::accept(myFD, (struct sockaddr *) & (sock->mySin), & len);
    sock->myType = myType;
    if((sock->myFD < 0 && !myNonBlocking)
    || (sock->myFD < 0 && errno != EWOULDBLOCK && myNonBlocking))
    {
      myErrorStr = "Failed to accept on socket;"
      perror("accept");
      return (false);
    }

    return (true);
  }

  void SbSocket::inToA(struct in_addr * addr, char * buff)
  { strcpy(buff, inet_ntoa(* addr)); }

  bool SbSocket::getSockName()
  {
    socklen_t size;

    if(myFD < 0)
    {
      myErrorStr = "Trying to get socket name on an unopened socket;"
      printf(myErrorStr.c_str());
      return (false);
    }

    size = sizeof(mySin);
    if(getsockname(myFD, (struct sockaddr *) & mySin, & size) != 0)
    {
      myErrorStr = "Error getting socket name;"
      perror(myErrorStr.c_str());
```

```
595       return (false);
        }

        return (true);
      }
600
      unsigned int SbSocket::hostToNetOrder(int i)
      { return (htons(i)); }

      unsigned int SbSocket::netToHostOrder(int i)
605   { return (ntohs(i)); }
```

```
#ifndef SbSoundsMLT_h
#define SbSoundsMLT_h

#include "SbMsgListener.h"
#include "SbMessage.h"
#include "Aria.h"

#include <string>

//#include "ArSpeech.h"
//#include "ArSounds.h"

//this is a C file
//extern C syntax needs to span at least 3 lines.
extern "C" {
#include "BotSpeak.h"
}

using namespace std;

/// Listener for Audio playback including speech synthesis and digital
/// audio play.
/// SbSoundsT class inherits Aria's threaded ArSyncTask to take care of playing
/// wavs and synthesizing without locking the robot cycling. It also allows for
/// sending many consecutive requests.
/// First use of speak or Play will initialize  botspeak.
/// @author Marcin Balicki : Cooper Union - Mechanical Engineering
/// @date March 2004
/// @warning Be careful with locking and unlocking.
/// @warning This class is threaded and therefore should not use
/// sendMsgToHandler to fire messages back to the handler because some other
/// thread maybe using it.
/// @todo Instead of making SbMsgHandler as the singleton, the manager class
/// should be one, and the reference to it should be possessed by all
/// SbMsgListeners. However, this relation would be forced since listener
/// is far away in function from manager (listener->handler->manager).
/// @bug None


class SbSoundsMLT : public SbMsgListener, public ArASyncTask
{
public:
  //Constructor
  SbSoundsMLT(void);
  //Destructor
  ~SbSoundsMLT(void);
  /// implements the SbMsgListener pure virtual function.
  /// @param &msg SbMessage which caused the listener to fire.
  virtual void fire(const SbMessage & msg);

  ///implements the ArSyncTask main thread function
  /// @param *arg argument that does nothing.
  virtual void * runThread(void * arg);

protected:
  ///internal robot pointer
  ArRobot * robot;
  ///checks if the robot exists and is connected
  bool checkForRobot(void);
  /// internal message
  SbMessage msg;
  /// new message for speak arrived flag
  bool newSpeakMsg;
  /// new message for play arrived flag
  bool newPlayMsg;
  /// initilized botspeak flag
  bool init;

};
#endif
```

```
#include "SbSoundsMLT.h"

SbSoundsMLT::SbSoundsMLT()
{
  //start the thread
  create(true, true);
  init = false;
  newSpeakMsg = false;
  newPlayMsg = false;

  //adds message types to the msgListener (base) vector for registering with
  // msg handler.
  addMsgType("SPEAK");
  addMsgType("PLAY");
  addMsgType("AMIGOSOUNDTOG");
  addMsgType("AMIGOSOUND");

}

SbSoundsMLT::~SbSoundsMLT()
{ }

///Checks if an instance of the robot exists and if it is connected.
bool SbSoundsMLT::checkForRobot(void)
{
  //Check if the robot exists, if it does and its connected than return true;
  if((robot = Aria::findRobot("SBROBOT")) == NULL)
    return false;
  robot->lock();
  if(robot->isConnected())
  {
    robot->unlock();
    return true;
  }
  robot->unlock();
  return false;
}

//this class will not be called by the local thread so we can
//acccess it as long as we use luck/unlock within it to access local data.
void SbSoundsMLT::fire(const SbMessage & msg)
{
  cout << "Firing SOUNDS with msg : "<< msg.getAsStr() << endl;
  this->msg = msg;
  //locks this class data so we can get anything we want.
  lock();
  // SPEAK|string|| will use botspeak to synthesize speech
  // initializes if it has not done so
  // for botspeak speech variation is embadded using tags in the string.
  if(msg.getType() == "SPEAK")
  {
    if(!init)
    {
      //intialize botspeak.
      bsInit();
      bsSetMisunderstood(NULL);
      //Disables the microphone and stops recognition.
      //bsDisableMic();
      //Enables the microphone and starts recognition
      //(it is enabled and started by default).
      //void bsEnableMic()
      ArUtil::sleep(1000);
      init = true;
      cout <<endl<< "Botspeak Ready"<< endl;
    }
    newSpeakMsg = true;
  }
  // PLAY|string||  will use botspeak to play a WAV file
  // initializes if it has not done so
  // the string is the location and name of the file to be played.
  else if(msg.getType() == "PLAY")
  {
    if(!init)
    {
      //intialize botspeak.
      bsInit();
      bsSetMisunderstood(NULL);
      ArUtil::sleep(1000);
      init = true;
      cout << "Botspeak Ready"<< endl;
    }

    newPlayMsg = true;
  }
  // unlock local variables
  unlock();

  //has to be connected first.
  if(!checkForRobot())
  {
    cout << "Robot doesn't exist or is not connected:"
    <<" Will not fire command. (SbSoundsMLT)"<< endl;
    sendMsgToHandler("SEND|MSG|DISCONNECTED||");
    return;
  }

  // AMIGOSOUNDTOG|int|| turns the amig sounds off and on 0=OFF 1=ON.
  else if(msg.getType() == "AMIGOSOUNDTOG")
  {
```

```
      //amigo sounds off and on.
      switch(msg.getArgAsInt(0))
      {
        case 0:
          robot->comInt(ArCommands::SOUNDTOG, 0);
          break;
        case 1:
          robot->comInt(ArCommands::SOUNDTOG, 1);
      }
    }
    // AMIGOSOUND|int||  plays a given sound number on amigobot.
    else if(msg.getType() == "AMIGOSOUND")
      robot->comInt(ArCommands::SOUND, msg.getArgAsInt(0));
  }

  void * SbSoundsMLT::runThread(void * arg)
  {
    while(myRunning)
    {
      ArUtil::sleep(100);
      //locks this class data so we can get anything we want.
      lock();
      //Check if there is a new msg
      if(newSpeakMsg)
      {
        cout << "SPEAK: " << msg.getArgAsStr(0) << endl;
        // need to cast the string to a char pointer
        // can I just pass the address & (msg.getArgAsStr(0).c_str())
        bsSpeak((char *) (msg.getArgAsStr(0).c_str()));
        newSpeakMsg = false;
      }
      else if(newPlayMsg)
      {
        cout << "GOT PLAY: " << msg.getArgAsStr(0) << endl;
        //need to cast the string to a char pointer
        // can I just pass the address & (msg.getArgAsStr(0).c_str())
        bsPlay((char *) (msg.getArgAsStr(0).c_str()));
        newPlayMsg = false;
      }
      unlock();
    }
    return NULL;
  }
```

```
#ifndef SbStateML_h
#define SbStateML_h

#include "SbMsgListener.h"
#include "SbMessage.h"
#include "Aria.h"

#include <sstream>
#include <string>

using namespace std;

/// Manages requests for robot's state (position,velocity, sonar, motors, etc).
/// SbStateML gathers requested information and sends back to the user.
/// The information includes current motion/position, battery voltage, sonar
/// states, table sensors, motor activity.)
/// @author Marcin Balicki : Cooper Union - Mechanical Engineering
/// @date March 2004
/// @note You can use the sendMsgToHandler to fire messages back to the handler.
/// @bug None

class SbStateML : public SbMsgListener
{
public:
  ///Constructor
  SbStateML(void);
  ///Decontstructor
  ~SbStateML(void);
  ///Converts double to a string.
  string toString(double);
  ///implements the SbMsgListener pure virtual function
  ///@param &msg SbMessage that coused the SbMsgListener to be fired.
  virtual void fire(const SbMessage &msg);
private:
  ///internal robot pointer
  ArRobot * robot;
  ///checks if the robot exists and is connected
  bool checkForRobot(void);
  ///message type
  string type;
  ///internal message
  SbMessage * msgState;
  ///internal stream function used in toString()
  ostringstream oss;
};

#endif
```

```
#include "SbStateML.h"
#include <iomanip>

SbStateML::SbStateML(void)
{
  msgState = new SbMessage();

  // adds message types to the msgListener (base) vector for registering with
  // msg handler.
  addMsgType("STATE");
}

SbStateML::~SbStateML(void)
{ delete msgState; }

///Checks if an instance of the robot exists and if it is connected.
bool SbStateML::checkForRobot(void)
{
  //Check if the robot exists, if it does and its connected than return true;
  if((robot = Aria::findRobot("SBROBOT")) == NULL)
    return false;
  robot->lock();
  if(robot->isConnected())
  {
    robot->unlock();
    return true;
  }
  robot->unlock();
  return false;
}

string SbStateML::toString(double d)
{
  oss.str("");
  oss << setprecision(4) <<d; // insert double into stream
  return oss.str();
}

void SbStateML::fire(const SbMessage & msg)
{
  cout << "Firing STATE with msg : "<< msg.getAsStr() << endl;
  //has to be connected first.
  if(!checkForRobot())
  {
    cout << "Robot doesn't exist or is not connected::"
    << " Will not fire command. (SbStateML)"<< endl;
    sendMsgToHandler("SEND|MSG|DISCONNECTED||");
    return;
  }

  //prepare the message to be sent out.
  msgState->clear();

  msgState->setType("SEND");
  msgState->addArgument("MSG");
  type = msg.getType();
  //Parse the message, the Type should tell me what to do.
  //STATE|var|var|| has a number of commands under it.
  if(type == "STATE")
  {
    // STATE|| if the message has no arguments send out vital information

    // X/Y/heading/Velocity/RotVel/LeftWheelVel/RightWheelVel/Voltage//
    if(msg.getArgAsStr(0) == "")
    {
      msgState->addArgument("STATE");
      robot->lock();
      //cast the double into an int because sstream prints 0 as exp form
      msgState->addArgument(toString(robot->getX()));
      msgState->addArgument(toString(robot->getY()));
      msgState->addArgument(toString(robot->getTh()));
      msgState->addArgument(toString(robot->getVel()));
      msgState->addArgument(toString(robot->getRotVel()));
      msgState->addArgument(toString(robot->getLeftVel()));
      msgState->addArgument(toString(robot->getRightVel()));
      msgState->addArgument(toString(robot->getBatteryVoltage()));
      robot->unlock();
      sendMsgToHandler(* msgState);
      return;
    }
    // STATE|sonar|var|| if the first argument is SONARS and second is 0
    // return the values for all available sonar. ( -1 sonar did not return
    // a reading or no sonar available)
    // if the second argument is 1,2,3,4 return values for that set of sonar
    else if(msg.getArgAsStr(0) == "sonar")
    {
      msgState->addArgument("SONAR");
      robot->lock();
      //if the second argument is 0 then send ALL SONARS
      if(msg.getArgAsInt(1) == 0)
      {
        for(int i = 0; i < robot->getNumSonar(); ++i)
        {
          printf("%6d", robot->getSonarRange(i));
          msgState->addArgument(toString(robot->getSonarRange(i)));
        }
      }
      //FIRST SONAR SET var=1
      else if(msg.getArgAsInt(1) == 1)
      {
```

```
          for(int i = 0; i < robot->getNumSonar() && i <= 7; ++i)
          {
            printf("%6d", robot->getSonarRange(i));
            msgState->addArgument(toString(robot->getSonarRange(i)));
          }

        }
        //SECOND SONAR SET var=2
        else if(msg.getArgAsInt(1) == 2)
        {
          for(int i = 8; i < robot->getNumSonar() && i <= 15; ++i)
          {
            printf("%6d", robot->getSonarRange(i));
            msgState->addArgument(toString(robot->getSonarRange(i)));
          }
        }
        //THIRD SONAR SET var=3
        else if(msg.getArgAsInt(1) == 3)
        {
          for(int i = 16; i < robot->getNumSonar() && i <= 23; ++i)
          {
            printf("%6d", robot->getSonarRange(i));
            msgState->addArgument(toString(robot->getSonarRange(i)));
          }
        }
        //FOURTH SONAR SET var=4
        else if(msg.getArgAsInt(1) == 4)
        {
          for(int i = 24; i < robot->getNumSonar() && i <= 31; ++i)
          {
            printf("%6d", robot->getSonarRange(i));
            msgState->addArgument(toString(robot->getSonarRange(i)));
          }
        }
        robot->unlock();
        sendMsgToHandler(* msgState);
      }

      // STATE|closestsonar|| returns the SONAR NUMBER and the distance
      // of the closest object within an angle range (start and end angles)
      else if(msg.getArgAsStr(0) == "closestsonar")
      {
        msgState->addArgument("CLOSESTSONAR");
        robot->lock();
        msgState->addArgument(
        SbMessage::convToString(robot->getClosestSonarNumber(msg.getArgAsDbl(1),
        msg.getArgAsDbl(2))));
        msgState->addArgument(
        SbMessage::convToString(robot->getClosestSonarRange(msg.getArgAsDbl(1),
        msg.getArgAsDbl(2))));
        // Returns the closest of the current sonar reading in the given range.
        // getClosestSonarNumber(double startAngle, double endAngle)
        // Returns the number of the sonar that has the closest current
        // reading in the given range.
        robot->unlock();
        sendMsgToHandler(* msgState);
      }
      //  STATE|sonaronoroff|| Checks if the SONAR are on or off
      // returns 0=OFF 1=ON.
      else if(msg.getArgAsStr(0) == "sonaron")
      {
        msgState->addArgument("SONARON");
        //turns out that the robot does not have good max and min values
        robot->lock();
        if(robot->areSonarsEnabled())
          msgState->addArgument("1");
        else
          msgState->addArgument("0");
        robot->unlock();
        sendMsgToHandler(* msgState);
      }
      // STATE|tablesensors|| Checks if the LEFT and RIGHT TABLE SENSORS
      // are triggerd
      // returns LEFT|RIGHT  0=OFF 1=ON.
      else if(msg.getArgAsStr(0) == "tablesensors")
      {
        msgState->addArgument("TABLESENSORS");
        //turns out that the robot does not have good max and min values
        robot->lock();
        if(robot->hasTableSensingIR())
        {
          if(robot->isLeftTableSensingIRTriggered())
            msgState->addArgument("1");
          else
            msgState->addArgument("0");
          if(robot->isRightTableSensingIRTriggered())
            msgState->addArgument("1");
          else
            msgState->addArgument("0");
        }
        robot->unlock();
        sendMsgToHandler(* msgState);
      }

      //STATE|motors|| send the state of the motors. 0=OFF 1=ON.
      else if(msg.getArgAsStr(0) == "motors")
      {
        msgState->addArgument("MOTORS");
        //turns out that the robot does not have good max and min values
```

```
          robot->lock();
200       if(robot->areMotorsEnabled())
            msgState->addArgument("1");
          else
            msgState->addArgument("0");
          robot->unlock();
205       sendMsgToHandler(* msgState);
        }
      }
    }
```

8.15 Test Application Source Code – Java

```
package mcontrol;

import java.awt.*;
import javax.swing.*;
//import com.borland.plaf.borland.BorlandLookAndFeel;


/**
 * <p>Title: </p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2004</p>
 * <p>Company: </p>
 * @author not attributable
 * @version 1.0
 */

public class MouseControl {
  boolean packFrame = false;

  //Construct the application
  public MouseControl() {
    MouseControlFrameframe = new MouseControlFrame();
    //Validate frames that have preset sizes
    //Pack frames that have useful preferred size info, e.g. from their layout
    if (packFrame) {
      frame.pack();
    }
    else {
      frame.validate();
    }
    //Center the window
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = frame.getSize();
    if (frameSize.height > screenSize.height) {
      frameSize.height = screenSize.height;
    }
    if (frameSize.width > screenSize.width) {
      frameSize.width = screenSize.width;
    }
    //frame.setLocation((screenSize.width - frameSize.width) / 2,
    // (screenSize.height - frameSize.height) / 2);
    frame.setVisible(true);
    //frame.move(0,0);
   // frame.resize(Toolkit.getDefaultToolkit().getScreenSize());
    //maximize:
    frame.setExtendedState(frame.MAXIMIZED_BOTH);
  }

  //Main method
  public static void main(String[] args) {
    try {
      UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    // UIManager.setLookAndFeel("com.borland.plaf.borland.BorlandLookAndFeel");
    }
    catch(Exception e) {
      e.printStackTrace();
    }
    new MouseControl();
  }
}
```

```
package mcontrol;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

/**
 * <p>Title: </p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2004</p>
 * <p>Company: </p>
 * @author not attributable
 * @version 1.0
 */

public class MouseControlfFrame
    extends JFrame {
  JPanel contentPane;
  MouseControlPanelmControlPanel = new MouseControlPanel();
  CardLayout cardLayout1 = new CardLayout();
  Point oldP = null;
  //Construct the frame
  public MouseControlfFrame() {
    enableEvents(AWTEvent.WINDOW_EVENT_MASK);
    try {
      jbInit();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }

  //Component initialization
  private void jbInit() throws Exception {
    contentPane = (JPanel)this.getContentPane();
    contentPane.setLayout(cardLayout1);

    this.setSize(new Dimension(620, 500));
    this.setTitle("MouseControl");
    mControlPanel.setMinimumSize(new Dimension(308, 360));
    mControlPanel.setPreferredSize(new Dimension(520, 360));
    //feedBackTxt.setMaximumSize(new Dimension(2147483647, 2147483647));
    contentPane.add(mControlPanel, "mControlPanel");
    contentPane.addMouseMotionListener(new MouseMotionAdapter() {
      public void mouseDragged(MouseEvent me) {
        Point p = me.getPoint();
          if (oldP != null) {
        //  contentPane.getGraphics().drawLine(oldP.x, oldP.y, p.x, p.y);
          // do some math
          // int diff=oldP.x-p.x;
           //send the command.

           // mControlPanel.sendCommand("RLVEL","","","");
            //rotate? command?
            //mControlPanel.sendCommand("HEADING","","","");
          }
          oldP = p;
        }
      });


    contentPane.addMouseListener(new MouseAdapter() {
      public void mouseReleased(MouseEvent me) {
        oldP = null;
      }
    });
  }

  //Overridden so we can exit when window is closed
  protected void processWindowEvent(WindowEvent e) {
    super.processWindowEvent(e);
    if (e.getID() == WindowEvent.WINDOW_CLOSING) {
      System.exit(0);
    }
  }
}
```

```
package mcontrol;

import java.awt.*;
import javax.swing.*;
import com.borland.jbcl.layout.*;
import java.awt.event.*;
import edu.cooper.me.grasp.comm.*;

/**
 * <p>Title: </p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2004</p>
 * <p>Company: </p>
 * @author not attributable
 * @version 1.0
 */

public class MouseControlPanel
    extends JPanel
    implements MessageListener {

  edu.cooper.me.grasp.comm.RobotConnection conn;
  Connector connector;

  VerticalFlowLayout verticalFlowLayout1= new VerticalFlowLayout();
  JPanel jPanel1 = new JPanel();
  JPanel jPanel2 = new JPanel();
  JButton servDisconnectButton= new JButton();
  JTextField servAddTxt = new JTextField();
  JTextField servPortTxt = new JTextField();
  JButton robotConnectButton= new JButton();
  JTextField robotConTxt = new JTextField();
  JButton servConnectButton= new JButton();
  JPanel jPanel4 = new JPanel();
  JTextField comm1Txt4 = new JTextField();
  JTextField comm1Txt3 = new JTextField();
  JTextField comm1Txt2 = new JTextField();
  JTextField comm1Txt1 = new JTextField();
  JButton comButton1 = new JButton();
  JButton comButton4 = new JButton();
  JTextField comm3Txt3 = new JTextField();
  JPanel jPanel5 = new JPanel();
  JTextField comm3Txt2 = new JTextField();
  JTextField comm3Txt1 = new JTextField();
  JTextField comm3Txt4 = new JTextField();
  JButton comButton3 = new JButton();
  JTextField comm2Txt2 = new JTextField();
  JPanel jPanel6 = new JPanel();
  JTextField comm2Txt4 = new JTextField();
  JTextField comm2Txt1 = new JTextField();
  JTextField comm2Txt3 = new JTextField();
  JButton comButton2 = new JButton();
  JPanel jPanel7 = new JPanel();
  JTextField comm4Txt2 = new JTextField();
  JTextField comm4Txt1 = new JTextField();
  JButton comButton5 = new JButton();
  JTextField comm5Txt2 = new JTextField();
  JPanel jPanel8 = new JPanel();
  JTextField comm5Txt1 = new JTextField();
  JTextField comm4Txt3 = new JTextField();
  JTextField comm4Txt4 = new JTextField();
  JButton robotDisconnectButton= new JButton();
  JButton stopButton = new JButton();
  JLabel jLabel1 = new JLabel();
  JPanel jPanel3 = new JPanel();
  JTextArea msgBox = new JTextArea();
  JButton ButtonState = new JButton();
  public MouseControlPanel() {
    try {
      jbInit();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }

  private void jbInit() throws Exception {
    this.setLayout(verticalFlowLayout1);
    servDisconnectButton.setToolTipText("");
    servDisconnectButton.setHorizontalAlignment(SwingConstants.CENTER);
    servDisconnectButton.setText("DISCONNECT-SERVER");
    servAddTxt.setPreferredSize(new Dimension(150, 20));
    servAddTxt.setSelectionStart(7);
    servAddTxt.setText("robo2.me.cooper.edu");
    servPortTxt.setPreferredSize(new Dimension(70, 20));
    servPortTxt.setText("5555");
    robotConnectButton.setText("CONNECT");
    robotConTxt.setPreferredSize(new Dimension(70, 20));
    robotConTxt.setToolTipText("");
    robotConTxt.setText("robot");
    robotConTxt.addActionListener(new
      MouseControlPanel_robotConTxt_actionAdapter(this));
    servConnectButton.setText("CONNECT-SERVER");
    servConnectButton.setHorizontalAlignment(SwingConstants.CENTER);
    jPanel1.setBackground(Color.orange);
    jPanel2.setBackground(UIManager.getColor(
        "CheckBoxMenuItem.selectionBackground"));
    comm1Txt4.setPreferredSize(new Dimension(40, 20));
    comm1Txt4.setText("0");
```

```
        comm1Txt3.setPreferredSize(new Dimension(40, 20));
        comm1Txt3.setText("0");
        comm1Txt2.setPreferredSize(new Dimension(40, 20));
        comm1Txt2.setText("0");
        comm1Txt1.setPreferredSize(new Dimension(70, 20));
        comm1Txt1.setText("MOVE");
        comButton1.setText("Send");
        comButton4.setText("Send");
        comm3Txt3.setPreferredSize(new Dimension(40, 20));
        comm3Txt3.setText("0");
        comm3Txt2.setPreferredSize(new Dimension(40, 20));
        comm3Txt2.setText("0");
        comm3Txt1.setPreferredSize(new Dimension(70, 20));
        comm3Txt1.setText("GOTO");
        comm3Txt4.setPreferredSize(new Dimension(40, 20));
        comm3Txt4.setText("0");
        comButton3.setText("Send");
        comm2Txt2.setMinimumSize(new Dimension(40, 20));
        comm2Txt2.setOpaque(true);
        comm2Txt2.setPreferredSize(new Dimension(40, 20));
        comm2Txt2.setText("0");
        comm2Txt4.setPreferredSize(new Dimension(40, 20));
        comm2Txt4.setText("0");
        comm2Txt1.setPreferredSize(new Dimension(70, 20));
        comm2Txt1.setText("ROTATE");
        comm2Txt3.setPreferredSize(new Dimension(40, 20));
        comm2Txt3.setText("0");
        comButton2.setText("Send");
        comm4Txt2.setPreferredSize(new Dimension(40, 20));
        comm4Txt2.setToolTipText("");
        comm4Txt2.setText("init");
        comm4Txt1.setPreferredSize(new Dimension(70, 20));
        comm4Txt1.setText("CAM");
        comButton5.setText("Send");
        comm5Txt2.setPreferredSize(new Dimension(400, 20));
        comm5Txt2.setText("HELLO WORLD");
        comm5Txt1.setPreferredSize(new Dimension(70, 20));
        comm5Txt1.setText("SPEAK");
        comm4Txt3.setText("0");
        comm4Txt3.setToolTipText("");
        comm4Txt3.setPreferredSize(new Dimension(40, 20));
        comm4Txt4.setText("0");
        comm4Txt4.setToolTipText("");
        comm4Txt4.setPreferredSize(new Dimension(40, 20));
        this.setMinimumSize(new Dimension(520, 307));
        this.setPreferredSize(new Dimension(520, 307));
        robotDisconnectButton.setText("DISCONNECT");
        stopButton.setBackground(Color.red);
        stopButton.setAlignmentY( (float) 0.5);
        stopButton.setMaximumSize(new Dimension(33333, 33333));
        stopButton.setMinimumSize(new Dimension(60, 20));
        stopButton.setPreferredSize(new Dimension(100, 40));
        stopButton.setText("STOP");
        jLabel1.setText("robot/simulator");
        msgBox.setEnabled(true);
        msgBox.setMinimumSize(new Dimension(500, 120));
        msgBox.setPreferredSize(new Dimension(500, 120));
        msgBox.setEditable(false);
        ButtonState.setActionCommand("STATE");
        ButtonState.setText("STATE");
        jPanel2.add(jLabel1, null);
        jPanel4.add(comm1Txt1, null);
        jPanel4.add(comm1Txt2, null);
        jPanel4.add(comm1Txt3, null);
        jPanel2.add(robotConTxt, null);
        this.add(jPanel1, null);
        jPanel1.add(servAddTxt, null);
        jPanel1.add(servPortTxt, null);
        jPanel1.add(servConnectButton, null);
        jPanel1.add(servDisconnectButton, null);
        this.add(jPanel2, null);
        jPanel2.add(robotConnectButton, null);
        jPanel2.add(robotDisconnectButton, null);
        jPanel2.add(stopButton, null);
        this.add(jPanel4, null);
        jPanel4.add(comm1Txt4, null);
        jPanel4.add(comButton1, null);
        jPanel5.add(comm4Txt1, null);
        jPanel5.add(comm4Txt2, null);
        jPanel5.add(comm4Txt3, null);
        jPanel5.add(comm4Txt4, null);
        jPanel5.add(comButton4, null);
        this.add(jPanel7, null);
        this.add(jPanel6, null);
        this.add(jPanel5, null);
        jPanel6.add(comm3Txt1, null);
        jPanel6.add(comm3Txt2, null);
        jPanel6.add(comm3Txt3, null);
        jPanel6.add(comm3Txt4, null);
        jPanel6.add(comButton3, null);
        jPanel7.add(comm2Txt1, null);
        jPanel7.add(comm2Txt2, null);
        this.add(jPanel8, null);
        jPanel8.add(comm5Txt1, null);
        jPanel8.add(comm5Txt2, null);
        jPanel8.add(comButton5, null);
        this.add(jPanel3, null);
        jPanel3.add(msgBox, null);
        jPanel3.add(ButtonState, null);
```

```
        jPanel7.add(comm2Txt3, null);
200     jPanel7.add(comm2Txt4, null);
        jPanel7.add(comButton2, null);
        //listeneres

        servConnectButton.addActionListener(new ActionListener() {
205       public void actionPerformed(ActionEvent e) {
            connect();
          }
        });
        servDisconnectButton.addActionListener(new ActionListener() {
210       public void actionPerformed(ActionEvent e) {

            System.out.println( (disconnect()) ? "DISONNECTED" :
                                "CANNOT DISCONNECT");
          }
215     });

        comButton1.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {

220         sendCommand(comm1Txt1.getText(),comm1Txt2.getText(),comm1Txt3.getText(),
                        comm1Txt4.getText());
          }
        });
        comButton2.addActionListener(new ActionListener() {
225       public void actionPerformed(ActionEvent e) {
            sendCommand(comm2Txt1.getText(),comm2Txt2.getText(),comm2Txt3.getText(),
                        comm2Txt4.getText());
          }
        });
230
        comButton3.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {
            sendCommand(comm3Txt1.getText(),comm3Txt2.getText(),comm3Txt3.getText(),
                        comm3Txt4.getText());
235       }
        });
        comButton4.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {
            sendCommand(comm4Txt1.getText(),comm4Txt2.getText(),comm4Txt3.getText(),
240                     comm4Txt4.getText());
          }
        });
        comButton5.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {
245         sendCommand(comm5Txt1.getText(), comm5Txt2.getText(), "", "");
          }
        });
```

```
        robotConnectButton.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {
250         sendCommand("CONNECT", robotConTxt.getText(), "", "");
          }
        });
        robotDisconnectButton.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {
255         sendCommand("DISCONNECT", "", "", "");
          }
        });

        stopButton.addActionListener(new ActionListener() {
260       public void actionPerformed(ActionEvent e) {
            sendCommand("STOP", "", "", "");
          }
        });
        ButtonState.addActionListener(new ActionListener() {
265           public void actionPerformed(ActionEvent e) {
              if (conn != null && conn.isConnected())
              {
                conn.sendActivity("STATE||");
              }
270           }
            });

      }

275   void sendCommand(String s1, String s2, String s3, String s4) {

        if (conn != null && conn.isConnected()) {

          if ( (s1 == null) || (0 == s1.compareTo(""))) {
280         System.out.println(s1);
            return;
          }
          if (s2 == null || (0 == s2.compareTo(""))) {
            s2 = "0";
285       }
          if (s3 == null || (0 == s3.compareTo(""))) {
            s3 = "0";
          }
          if (s3 == null || (0 == s4.compareTo(""))) {
290         s4 = "0";
          }

          String send = s1 + '|' + s2 + '|' + s3 + '|' + s4 + "||";
         // System.out.println(send);
295       conn.sendActivity(send);
        }
        else {
```

```
        System.out.println("CAN'T SEND: NOT CONNECTED");
      }
300   }

    boolean connect() {
      System.out.println("CONNECTING");
      if (conn == null || !conn.isConnected()) {
305     try {

        conn = new RobotConnection(servAddTxt.getText(),
                                   Integer.parseInt(servPortTxt.getText()));
        conn.addMessageListener(this);
310     conn.open();
        // connector = new Connector(conn);
      }
      catch (NumberFormatExceptionex) {

315     ex.printStackTrace();
        System.out.println("NumberFormatExeption");
        return false;
      }
      catch (Exception ex) {
320     ex.printStackTrace();
        System.out.println("Can't Connect to Server!");
        conn = null;
        return false;
      }
325   }
      else {
        System.out.println("Already connected.");
      }
      return false;
330 }

    boolean disconnect() {
      //connector = null;
      if (conn != null) {
335     sendCommand("QUIT", "", "", "");

        //conn.close();
        conn = null;
        return true;
340   }
      return false;
    }

    void robotConTxt_actionPerformed(ActionEvent e) {
345 }

    public void messageEvent(MessageEvent e) {
      //test for what type...and time..
      //print in and out.
350   if (e.getType() == e.INCOMMING) {

        if (msgBox.getLineCount() > 5) {
          msgBox.setText(e.getMessage() + '\n');
        }
355     else {
          msgBox.append(e.getMessage() + '\n');
        }
      }
      else {
360     System.out.println("SENT:  " + e.getMessage());
        //can also get type!
      }
    }

365 }

  class MouseControlPanel_robotConTxt_actionAdapter
      implements java.awt.event.ActionListener {
    MouseControlPaneladaptee;
370
    MouseControlPanel_robotConTxt_actionAdapter(MouseControlPaneladaptee) {
      this.adaptee = adaptee;
    }

375 public void actionPerformed(ActionEvent e) {
      adaptee.robotConTxt_actionPerformed(e);
    }
  }
```