

# **SR4 Robot Software Manual**

(Models SR4-P and SR4-N)

**Version 1.3**

**3/26/04**

## Table of Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
1.1	Conventions Used in this Manual	4
<b>2</b>	<b>Working with the SR4</b>	<b>4</b>
2.1	Medallion	4
2.1.1	Network Setup	5
2.1.2	Serial Setup	5
2.2	Medallion Login	6
2.2.1	Network Login	6
2.2.2	Serial Login	6
2.2.3	Completing Login	7
2.3	Transferring Files	7
2.3.1	Transferring Files over the Network	8
2.3.2	Transferring Files over the Serial Port	8
2.4	The Medallion Environment	9
<b>3</b>	<b>SR4 Kernel</b>	<b>11</b>
3.1	Managing Modules	11
3.1.1	Listing Modules	12
3.1.2	Full Module Information	12
3.1.3	Inserting Modules	13
3.1.4	Removing Modules	14
<b>4</b>	<b>Development Environment for the SR4</b>	<b>14</b>
4.1	Updating the SRI Runtime Environment	14
<b>5</b>	<b>Modules</b>	<b>15</b>
5.1	Writing Modules	15
5.1.1	Compiling Modules	16
5.1.2	Packaging Modules	16
5.2	Standard Modules	18
<b>6</b>	<b>SRML</b>	<b>20</b>
6.1	Writing SRML Elements	20
6.2	Writing SRML	21
6.3	Standard SRML Elements	22
<b>7</b>	<b>Clients</b>	<b>25</b>
7.1	Writing Java Clients	26
7.2	Writing Other Clients	28
7.3	Standard Clients	28
<b>8</b>	<b>SRTP</b>	<b>29</b>
	<b>Appendix A: Resources</b>	<b>32</b>

**Appendix B: /etc/sr4/sr4.conf ..... 32**

# 1 Introduction

This document describes the information needed to use and write software for the SR4 Robot from Smart Robots, Inc.

## 1.1 Conventions Used in this Manual

There are many places throughout this manual where a certain syntax or font style is used to distinguish types of information. This section lists the important conventions used in this manual.

1. All commands to be entered by the user that are listed in the flow of a paragraph, and not in a separated code listing, are shown in a bold and monospaced font.
2. Code listings or command line examples are shown in a bordered gray background in monospaced font.
  1. Command lines on the host PC use a greater than sign (>) to show the command prompt. The actual command prompt shown will vary from system to system.
  2. Command lines on the robot use a pound sign (#) to show a command prompt, and is the actual prompt used by the robot.
3. File names are distinguished from standard text by a monospaced font.
4. To show paths through menu options, “=>” is used to show a path between menu options. For instance, if the Notepad application in Windows needed to be accessed, the following would be shown: Start Menu => Programs => Applications => Notepad. This means to click on the Start Menu, hover on Programs from the provided menu, hover on Applications from the provided menu, then click on Paint from the following menu.

## 2 Working with the SR4

This section provides information on how to work with with SR4 Robot, including communicating with the robot, transferring files to the robot, and configuring the robot.

### 2.1 Medallion

The single-board computer used by the SR4 is called the Medallion, from Techsol, Inc. It runs the Linux operating system on an ARM processor, has 32 MB of on-board non-volatile memory, and 32 MB of RAM. A Micro-Media Card (MMC) can be added to expand available non-volatile memory.

The Medallion is the core computer processor of the robot, on which the SR4 software runs. It provides the processing power and communication

mechanism necessary to control and manage the many components that make up the robot.

There are two ways to communicate with the Medallion; network and serial. Network communication can use either the wireless adapter or internal wired Ethernet connection using the provided RJ45 connection on the User Panel of the robot. Serial communication is accomplished using the serial port provided on the User Panel of the robot. The preferred method for communicating with the robot is using the network interface. This is because communication with the robot software requires network connectivity. Also, when the wireless adapter is used there are no wires tethering the robot.

### 2.1.1 Network Setup

Out of the box, the robot is configured to use wireless networking with an IP address of 192.168.0.22 and gateway of 192.168.0.1. It will automatically search out a wireless access point that is broadcasting its SSID. The SSID can also be manually set in the network initialization script. WEP is not available at this time.

If these settings are not compatible with your network, then you must login to the robot using the serial interface and modify the network script located in `/etc/network.sh` on the robot. This script initializes the network settings, and includes comments on its use and modification. After changes are made to this script the robot should then be rebooted.

### 2.1.2 Serial Setup

An alternate method of communication with the robot is using the robots serial interface. This is used only when network communication with the robot cannot be established.

The connection requires a 9-pin serial cable connected from a serial port on the host PC to the serial port on the User Panel of the robot.

The type of application used to communicate using the serial port is a terminal application. In Windows, HyperTerminal is used, and in Linux, minicom is used. HyperTerminal can be found from the Start Menu => Programs => Applications => Communications => HyperTerminal. In Linux, minicom is a command line application and is used by executing the **minicom** command at a command line.

These are the settings needed to correctly configure the connection:

- Bits per second: 115200
- Data bits: 8
- Parity: none

- Stop bits: 1
- Flow control: none

Consult the documentation for the specific application used to configure these settings.

The COMM port or Serial Device depends on which serial port the cable was connected on the PC.

## 2.2 Medallion Login

To access the internals of the Medallion you must login to access a command line. Logging into the robot can be accomplished by either communication methods. Telnet is used with network communication, and a terminal application is used with serial communication. These are both described in the following sections.

### 2.2.1 Network Login

Logging into the Medallion over the network requires the use of telnet. Telnet is an application which allows remote access to a command line interface on the remote computer. A telnet application can be accessed on the host PC by using the **telnet** command at a command line terminal. A command line is accessed in Windows by selecting Start Menu => Run... and typing **cmd** in the provided window followed by the enter key or pressing the Ok button. At the command prompt type the following command:

```
> telnet 192.168.0.22
```

**NOTE:** The robot must be powered up before attempting to log in over the network.

If the IP address of the robot is changed, 192.168.0.22 must be substituted with the new IP address. See section 2.1.1 for information on configuring the network on the robot.

Section 2.2.3 provides information on how to complete the login process.

### 2.2.2 Serial Login

Logging into the Medallion over the serial line requires the use of a terminal application, described in section 2.1.2. After the terminal application has been configured, simply connect to the configured connection. The terminal application will display a status of the connection.

**NOTE:** After a serial connection has been established the robot should be powered up if it is not already. This must be done before attempting to login to the robot.

Serial login is different from network login in that serial login is a direct interface to the Medallion. If for instance a serial connection was established and the robot was rebooted, then from the terminal application you would see the entire bootup process. This is why the serial interface can also be considered a debug terminal for the Medallion.

Section 2.2.3 provides information on how to complete the login process.

### 2.2.3 Completing Login

After successfully connecting to the medallion by either a telnet session or terminal application, a login prompt will be presented. An example using a telnet session is shown here:

```
> telnet 192.168.0.22
Trying 192.168.0.22...
Connected to 192.168.0.22.
Escape character is '^]'.
Three login:
```

**NOTE:** A terminal application may not show the login prompt if the connection was established after the prompt was presented due to the nature of the serial connection. The prompt is first presented immediately after bootup. If a connection was made and only a blinking cursor is presented, assume it is at a login prompt that is simply not being displayed. Otherwise the connection may have not been successfully established.

At this prompt enter the login user name, which in this case is “root” followed by the enter key. It will then prompt for a password, which in this case is blank, so only press the enter key at the password prompt. The password may be changed from the initial blank password using the **passwd** command, however there is no need to use a different user or create a new user. If security is an issue on your network, then it is suggested to change the password.

Telnet can also be disabled if security is an issue. Telnet is a simple protocol with no encryption and is considered insecure. There are configuration changes and debugging that can only be done with command line access to the robot, so if telnet is disabled then a terminal application must be used over the serial line.

## 2.3 Transferring Files

There are two ways to transfer files to the robot, corresponding to the options available for communicating with the robot. The available options are though the network using **tcp-snd** and **tcp-rcv**, or through the serial port

using a terminal application's file transfer option.

### 2.3.1 Transferring Files over the Network

Presently, this requires the **tcp-snd** command from the Techsol Toolchain to be present on the host PC. This is a Linux application, and will not work in Windows. Shortly we will be publishing a fully Windows compatible method for transferring files.

1. Log into the robot using instructions from section 2.2.
2. On the robot, instruct it to wait for a file using the command:

```
# tcp-rcv -o > [filename]
```

Where [filename] is the name of the file you wish to save the transferred file as.

3. On the PC, send the desired file to the robot using the command:

```
> cat [filename] | tcp-snd -o -a [robot IP address]
```

Where [filename] is the desired file name, and [robot IP address] is the IP address of the robot to which you are sending the file.

### 2.3.2 Transferring Files over the Serial Port

This requires the use of a terminal application, described in section 2.1.2.

1. Log into the robot using instructions from section 2.2.
2. On the robot, run the command **lrz**. This command waits for a file, and when the transfer is complete will save the transferred file to the current directory.
3. Transfer file from PC
  - From **Minicom** (Linux):
    1. Press Ctrl-A S
    2. Select zmodem from Upload
    3. Select file to download from list
  - From **Hyperterminal** (Windows):
    1. From main menu, select Transfer => Send File...
    2. Select file to download from the browse button under Filename
    3. Under Protocol, select zmodem from the dropdown menu (crash recovery is optional)
    4. Click Send

## 2.4 The Medallion Environment

Working with the environment on the Medallion is a matter of understanding the needed Linux commands, and the layout of the operating system. Here is a brief overview of the available commands:

- **cd, cp, mv, pwd, rm**: These are common Linux commands available to work within the file system. Their meaning and usage is beyond the scope of this document. See appendix A, reference 2.
- **vi**: Vi is the text editor available to the system. For information on editing withing vi, consult the vi documentation, found in appendix A, reference 1.
- **lrz, tcp-snd, tcp-rcv**: These commands are used to transfer files to and from the robot. Information can be found in section 2.3.
- **tar**: This is used to extract archived files. Usage instructions can be found by typing **tar -help**
- **/sri/bin/cvm**: This invokes the J2ME Java Virtual Machine

There are many other commands available, and they can be found by searching the `/bin` and `/sbin` directories.

For additional information on using Linux, it is suggested to peruse an introduction to Linux book or on-line resource. See appendix A for links to on-line Linux resources.

Here is a brief outline of the directory structure on the Medallion, pointing out the important locations within the environment.

`/`  
This is the root directory of the filesystem.

`/boot/`  
This holds the Linux kernel, and other information needed to boot the system.

`/bin/`  
This is one of the two locations where general Linux binary applications are stored.

`/etc/`  
The location where system configuration files are stored.

`/etc/init.d/rcS`  
This file is the main startup script for the system. After the Linux kernel is loaded, this script is executed, and performs the tasks necessary to setup the runtime environment. This file primarily loads Linux modules and executes external scripts to set up networking and

start runtime applications. This file is a plain text file. For more information, consult the script, which contains several comments describing the procedures.

`/etc/sri/`

This is where the configuration files are stored that are needed for the SR4 software. The files contained in this folder are described in section 2.5.

`/lib/SR4Kernel.jar`

This is the SR4 Kernel, which is the core of the SR4 software. Information on the SR4 Kernel is provided below.

`/log/`

Logs from SR4 software are stored in this location. Linux system logs are not stored, as there is no included logging facility (ie. syslog). Linux system information can be found in the proc file system.

`/mnt/mmc/`

This is where the MMC will be mounted. To mount the MMC, use the command **mount /mnt/mmc**. This file system is not mounted at boot time. The MMC included with the SR4-P or SR4-N has a capacity of 64 MB.

`/root/`

The home directory for user "root".

`/root/network.sh`

This script is used to start the network. It can set up either the wireless network, or wired network (via user panel RJ45 connection). The script is setup to allow quick modification of network settings. See the script for more configuration information.

`/root/run_kernel.sh`

This script starts the SR4 Kernel. It can be started at boot time, from `/etc/init.d/rcS`, or from the command line

`/sbin/`

This holds privileged Linux binary applications, accessible to user "root". It is the alternate location for system binary applications.

`/sri/`

This contains the Java Virtual Machine, using J2ME technology.

`/sri/jars/`

This is the default location of the modules used by the SR4 Kernel. They are dynamically loaded by the SR4 Kernel, similar to how Linux kernel modules are loaded. Information on how to use and write modules is provided in section 5.1.

`/wwwroot/`

The default location for the internal web server, which is part of the Smart Robots software.

### 3 SR4 Kernel

The SR4 Kernel is the core of the SR4 software. It is the software that drives the SR4 and is modeled after the Linux kernel. Among its responsibilities are to manage underlying hardware, logging, diagnostics, and configuration. It provides a means for loading and unloading modules, and allows outside processes to interact with the robot system.

The SR4 Kernel, when loaded, uses two configuration files, `/etc/sri/sr4.conf`, and `/etc/sri/modules.autoload`. The file `sr4.conf` is a properties file, containing key-value pairs. The SR4 Kernel and loaded modules can use this file to store configuration options. See appendix B for more information on `/etc/sri/sr4.conf`.

`modules.autoload` is a list of modules that will be automatically loaded by the SR4 Kernel when it is loaded. Each module must be on a separate line. Empty lines or lines beginning with a '#' character are ignored.

#### 3.1 Managing Modules

Modules are pieces of software designed for the SR4 Kernel and can be dynamically loaded and unloaded at runtime. When a module is loaded its code is incorporated into the SR4 Kernel giving it access to a wide range of functions made available by the SR4 Kernel. Modules can also use other modules, creating dependencies.

Modules can be managed at either boot time or run time. Managing modules at boot time uses `modules.autoload`, as described in section 2.4. This ensures that if the robot is rebooted, the given modules will be automatically loaded.

Managing modules at run time uses the `com.sri.clients.modutils` package, in `Modutils.jar`. This contains everything needed to manage modules within the SR4 Kernel. It can work either remotely, on a client machine, or on the robot itself. It provides a means to list modules, show full module information, load modules, and unload modules.

The Main class in Modutils.jar differentiates between the four functionalities by using the first command line argument passed to the class. The structure for using Modutils.jar from the command line is as follows:

**java -jar Modutils.jar [command] [command arguments]**

**command** This is either **lsmod** to list loaded modules, **modinfo** to show full information on a module, **insmod** to load a module, and **rmmod** to remove a module.

**command arguments** This is used to pass arguments to the specified action, and is optional, based on the supplied action

The java command refers to your Java Virtual Machine, which is **java** on J2SE platforms, and **cvm** on J2ME platforms. The **-jar** argument specifies to use the following jar file as the code base. In this case the jar file used is Modutils.jar. In the following examples, **java -jar Modutils.jar** will be expressed by the text **[modutils]**.

### 3.1.1 Listing Modules

The command **lsmod** will list available modules. The modules available are those that are in the module directory on the robot. The structure for the lsmod command is as follows:

**[modutils] lsmod** This simply lists the available modules.

An example output of this command is provided here:

```
> java -jar Modutils.jar lsmod
L Module           Ver      Needed By
N Health           0.1      N/A
N Httpd            0.1      N/A
Y Modutils         0.1      [none]
Y SRML             0.1      [none]
N UserPanel        0.1      N/A
```

The first column is a status of whether or not the module is loaded; Y for loaded and N for not loaded. The second column shows the name of the module. The third column is the version number. The fourth column is a list of modules that require that module, otherwise if no module needs this module then [none] is given, and if the module is not loaded N/A is given.

### 3.1.2 Full Module Information

The command **modinfo** will display full information on the specified module.

The structure for the modinfo command is as follows:

**[modutils] modinfo [module name]**

**module name**            This is the fully qualified class name of the module to be listed.

An example output of this command is listed here:

```
> java -jar Modutils.jar modinfo Modutils
Modinfo for Modutils
Main Class: com/sri/modules/modutils/Modutils
Title: Module Utilities
Version: 0.1
Vendor: Smart Robots, Inc
URL: http://www.smartrobots.com/
Loaded: yes
SRML: lsmod modinfo insmod rmmod
Dependencies: [none]
Needed By: [none]
```

### 3.1.3 Inserting Modules

The command **insmod** will load a specified module into the SR4 Kernel. The module loaded can either be one already on the robot, listed by the **lsmod** command (section 3.1.1) and not loaded, or one provided by the filename in the arguments. If a filename is given, then that file will be sent to the robot to be saved and loaded. The structure for using the "insmod" action is as follows:

**[modutils] insmod [module name] ([filename])**

**module name**            This is the name of the module to be loaded.

**filename**                If this argument is used, the class loaded by the robot will be the one specified by [filename]. It will be sent to the robot, saved in the robots module directory, and loaded. If this argument is not used, the class will be loaded from the robots module directory.

When attempting to load a module using the filename argument and a module of the same name is already on the robot, then the existing module will be unloaded if needed and replaced with the new module. It will then be loaded into the SR4 Kernel.

The status of the command will be returned specifying if the module was successfully loaded or if there was an error.

### 3.1.4 Removing Modules

The command **rmmod** will unload a specified module from the SR4 Kernel. The structure for the **rmmod** command is as follows:

```
[modutils] rmmod [module name]
```

<b>module name</b>	This is the fully qualified class name of the module to be unloaded.
--------------------	--

The status of the command will be returned specifying if the module was successfully loaded or if there was an error. A module cannot be unloaded if it is needed by another module, in which case it will return an error. In addition to removing the specified module, all SRML elements provided by the module will be unregistered.

## 4 Development Environment for the SR4

There are two types software that can work with the SR4. They are modules, and clients. Modules are loaded into the SR4 Kernel, and clients access the communication server within the SR4 Kernel over the network.

Modules must be written in Java, and are described in detail in chapter 5.

Clients can be written in any language. However, the only provided library abstracting the communication protocol used to communicate with the SR4 Kernel, SRTP, is written in Java. This is an open protocol, and is described in this chapter 8 to allow other language bindings to be written.

Setting up an environment for writing these types of applications is described below in Modules [ch 5] and Clients [ch 7].

When compiling Java code, any Java compiler may be used. The suggested compiler is from Sun Microsystems, and can be downloaded from here:

<http://java.sun.com/j2se/1.4.2/download.html>

The only change is that when compiling modules or clients, the CLASSPATH must be changed to incorporate the Smart Robots code base. Instructions on how to do this is provided with the Java distribution.

To write native-bytecode applications to run on the Medallion, you will need the Techsol toolchain, included on the user CD, which provides a version of gcc to cross compile C and C++ for the ARM processor. For this Linux is needed. The toolchain includes all needed documentation.

### 4.1 Updating the SRI Runtime Environment

The SRI runtime environment is a software package containing all the software and configuration files necessary to adapt the base Techsol runtime environment on the medallion to the SRI runtime environment.

This is needed when new SRI runtime environments or patches are released.

To update the Smart Robots runtime environment, use the following procedure:

1. Log into the robot, as described in section 2.2. and issue the following command:

```
# cd /
```

This will change directories to the root directory

2. Prepare to receive a file

- Over Network:

```
# tcp-rcv -o > [filename]
```

- Over Serial Port:

```
# lrz
```

Both commands will then wait for a file to be transferred.

3. Transfer the new runtime, which will have a `.tar` extension from the host computer using the instructions given in section 2.3.
4. Unpack the runtime using the following command:

```
# tar -xvf [filename]
```

This will unpack the runtime, listing each file as it is unpacked.

5. Sync the filesystem using the following command:

```
# sync
```

This will ensure that all new files are completely written to memory

6. Reboot the robot by turning off and on the robot.

## 5 Modules

The module loading facility allows the kernel to dynamically add and remove functionality. Modules can be divided into two categories; devices and extensions. A device is a piece of hardware, like a triangulation beacon. An extension is a piece of software that extends the ability of the robot, like a web server.

Modules can also define a set of SRML Elements, which is a structured language used to communicate with the robot. SRML is described in chapter 6.

### 5.1 Writing Modules

All modules must inherit the interface Module, show here:

```

public interface Module {
    public boolean initModule(ModuleData md);
    public void cleanupModule();
    public Vector getSRMLElements();
    public Vector getDependencies();
    public Vector getNeededBy();
    public void addNeededBy(String moduleName);
    public void configurationChange();
    public boolean runDiagnostic(int level, PrintStream
p);
}

```

The class `DefaultModule`, which provides basic functionality to the `Module` interface, is the preferred method of inheriting `Module`. It also provides utility functions for managing SRML elements and dependencies

The SRI API documentation contains the needed information for writing modules, specifically in `DefaultModule`, which is part of the package `com.sri.kernel.modules`. See appendix A, item 2, for the location of the SRI API documentation.

### 5.1.1 Compiling Modules

To compile modules, there are two Jar files needed. They are `foundation.jar` and `sri-library.jar`. The first file, `foundation.jar`, is the J2ME foundation classes. This is the class library used by the CVM on the robot. The second file, `sri-library.jar`, contains all available classes for use by the module. This includes portions of the SR4 Kernel, and other base modules provided by Smart Robots.

Those two Jar files must replace the current `CLASSPATH`, thus enabling the Java compiler to find the needed class definitions available to the module when it is loaded to the robot.

An example of a compile command can be seen here:

```

> javac -classpath foundation.jar:sri-library.jar \
        -sourcepath src \
        -d build \
        com/sri/modules/sample/Sample.java

```

### 5.1.2 Packaging Modules

Modules must be packaged in Jar files before being loaded by the robot. This helps in transporting and managing modules. There are several rules at how modules are named and packaged.

1. Module names can only contain alphanumeric characters a-z, A-Z, and 0-9.
2. Module names are case sensitive. The module "Sample" is not the same

as the module “sample”

3. Modules must be packaged in a Jar file and contain a Manifest file.
4. The Jar file must be created without compression
5. The Manifest file must conform to the module standard, described below.
6. The name of the Jar file must be the name of the module followed by the .jar extension. For example, the module “Sample” must be packaged in the Jar file “Sample.jar”.

A sample Jar command is shown here:

```
> jar -0 -cf Sample.jar \  
    -m com/sri/modules/sample/Sample.mf \  
    com/sri/modules/sample/Sample.class
```

A sample Manifest file is provided here:

```
Main-Class: com/sri/modules/sample/Sample  
Implementation-Title: Sample Module  
Implementation-Version: 0.1  
Implementation-Vendor: Smart Robots, Inc  
Implementation-URL: http://www.smartrobots.com/
```

This contains all needed Manifest entries. Here is a more detailed description of the Manifest module standard.

#### Main-Class

This describes the location of the main class file as located within the Jar file. This entry is required for the module to be loaded.

#### Implementation-Title

This is the title of the module. It is purely for information purposes and is not required.

#### Implementation-Version

This lists the version of the module. It is used in resolving dependencies and is required for the module to be loaded.

#### Implementation-Vendor

This is the vendor or author of the module. It is purely for information purposes and is not required.

#### Implementation-URL

This is meant to provide a URL where the module can be downloaded. It is purely for information purposes and is not required.

After a module has been packaged it can then be sent to the robot using the instructions from section 3.1.3.

## 5.2 Standard Modules

This is a list of Modules that come with the the Robot, and are by default loaded into the SR4 Kernel.

Module Name: Health  
sr4.conf: *none*  
SRML Elements: diagnose  
Dependencies: *none*  
Description: This module is the user portion of the diagnostic facility built into the SR4 Kernel.

Module Name: Httpd  
sr4.conf: httpd.wwwroot, httpd.port, httpd.dirlisting  
SRML Elements: *none*  
Dependencies: *none*  
Description: This is an internal web server, used to host various files and Java applets.

Module Name: Magellan  
sr4.conf: *none*  
SRML Elements: move  
Dependencies: OOPic  
Description: This is the navigation module. It provides basic navigation routines needed to instruct the robot to change position. The navigation methodology used is coordinate based.

Module Name: Mail  
sr4.conf: mail.address, mail.incoming.hostname,  
mail.incoming.protocol, mail.incoming.ssl,  
mail.incoming.username, mail.incoming.password,  
mail.incoming.delay, mail.outgoing.hostname,  
mail.outgoing.ssl, mail.outgoing.username,  
mail.outgoing.password  
SRML Elements: mail  
Dependencies: none  
Description: Provides email capability to the robot. This module checks an email account for new mail, and will process new messages as SRML, returning the results the the sender. This also provides the ability for the robot to send email.

Module Name: Modutils  
sr4.conf: none  
SRML Elements: lsmmod, modinfo, insmod, rmmmod  
Dependencies: none  
Description: Allows the module subsystem to be accessed using SRML.

Module Name: OOPic  
sr4.conf: oopic.serial, oopic.sonar, oopic.bumper,  
oopic.rightmotor, oopic.leftmotor,  
oopic.rightencoder, oopic.leftencoder, oopic.in1,  
oopic.drive, oopic.rotate  
SRML Elements: drive, rotate  
Dependencies: none  
Description: This manages the onboard OOPic, offering an object abstraction to the hardware controlled by the OOPic board and access to user defined basic navigation functions.

Module Name: Speech  
sr4.conf: speech.address, speech.volume, speech.speed,  
speech.pitch

Module Name: Speech  
SRML Elements: speak  
Dependencies: none  
Description: This module controls the text to speech board on the robot.

Module Name: SRML  
sr4.conf: none  
SRML Elements: br, echo, loop, sleep, exit  
Dependencies: none  
Description: This provides a set of basic SRML elements.

Module Name: UserPanel  
sr4.conf: userpanel.address  
SRML Elements: temp  
Dependencies: none  
Description: This controls the User Panel on the robot, which includes the temperature reading, LCD control, and battery voltage.

## 6 SRML

SRML (Smart Robots Markup Language) is an XML based language used to communicate with the robot. It is meant to be an intermediate language that is both human and robot readable. The SRML language set is dynamic. Elements can be added and removed at runtime. This allows modules to provide a set of SRML elements which are incorporated into the SRML language engine when the module is loaded.

### 6.1 Writing SRML Elements

An SRML Element is a java object which inherits SRMLElement, and can handle a given SRML element (or tag). When SRML is parsed, and the start of an element is found, it searches its list of SRMLElement's for a corresponding SRMLElement object and will use that object when parsing that element.

SRML Elements are generally defined as a subclass of a module. This allows the SRML Element to use resources provided to modules by the SR4

Kernel module loader.

SRMLElements are registered by its parent module when the module is loaded. Information on how to register SRMLElements is contained in the API documentation for DefaultModule, and is discussed in chapter 5.

The SRMLElement interface is shown here:

```
public interface SRMLElement {
    public String getName();
    public void init();
    public SRMLElement newInstance();
    public String execute();
    public void processText(String s);
    public boolean selfParses();
    public String parse(KXmlParser parser) throws
SRMLNoElementException, SRMLInvalidArgsException;
}
```

The class DefaultSRMLElement, which provides basic functionality to the SRMLElement interface, is the preferred method of inheriting SRMLElement.

Supporting attributes within your elements is a simple task. When an attribute exists within an element, a corresponding method is called within the SRMLElement. This method is named as such: SRMLset[Attribute], where [Attribute] is the attribute name with the first letter upper case, and the remainder lower case. The value of the attribute is passed to the method as a String.

So if you would like to use the attribute "message" in a "speak" element like this: **<speak message="hello world"/>**, you would define a method **"public void SRMLsetMessage(String message)"**. This is called internally by the SRML parser using Method Reflection. If an attribute is used that is not defined in the specified SRMLElement, an error is returned.

The API documentation contains the needed information for writing modules that provide SRML Elements. See also the SRMLElement interface and DefaultSRMLElement class, which is part of the package com.sri.kernel.srml.

## 6.2 Writing SRML

There are three rules for writing SRML:

1. The SRML must itself be a valid XML document.
2. An SRML document must consist of a single root element **<srml>**, which may have any number of valid SRML elements as its children. This is similar to how HTML documents must have **<html>** as its root element.

3. SRML elements used must be registered to the SRML parser using the method outlined in the previous section.

An example SRML document is this:

```
<srml>
  <loop count="5">
    The temperature is <temp/> degrees<br/>
  </loop>
</srml>
```

It will output the following text:

```
The temperature is 72.0 degrees
The temperature is 72.0 degrees
The temperature is 72.0 degrees
The temperature is 73.0 degrees
The temperature is 73.0 degrees
```

Each SRML Element returns a string value. If the element has a parent element this string will be passed to the parent element using the **processText(String text)** method. If the element has no parent element, in which case it is the root element, the returned string value is returned to wherever the SRML originated. In the case of the SrmlSend client, described in section 7.3, the returned text is displayed on the console. An example session of SrmlSend might look like this:

```
> java -jar SrmlSend.jar show_temp.xml
The temperature is 72.0 degrees
The temperature is 72.0 degrees
The temperature is 72.0 degrees
The temperature is 73.0 degrees
The temperature is 73.0 degrees
>
```

In this case show\_temp.xml is an SRML file containing the SRML shown above.

This is how the temperature is able to be embedded into the sentence “The temperature is 72.0 degrees”, and then repeated 5 times by the loop element.

### 6.3 Standard SRML Elements

This is a list of standard SRML elements that are included in the standard modules.

Element:       br  
Attributes:     none  
Provided By:   SRML

Element:	br
Usage:	<code>&lt;br /&gt;</code>
Description:	Inserts a line break "\n" into the current stream.
Element:	echo
Attributes:	text
Provided By:	SRML
Usage:	<code>&lt;echo text="hello world" /&gt; &lt;echo&gt;hello world&lt;/echo&gt;</code>
Description:	Returns the specified text from the attribute or encapsulated text.
Element:	loop
Attributes:	none
Provided By:	SRML
Usage:	<code>&lt;loop count="5" &gt;     .     .     . &lt;/loop&gt;</code>
Description:	This will execute nested elements [count] times.
Element:	sleep
Attributes:	minutes, seconds, millis
Provided By:	SRML
Usage:	<code>&lt;sleep minutes="2" /&gt;</code>
Description:	Will pause execution for (([minutes] * 60 * 1000) + ([seconds] * 1000) + [millis]) milliseconds.
Element:	exit
Attributes:	none
Provided By:	SRML
Usage:	<code>&lt;exit /&gt;</code>
Description:	This will force the SR4 Kernel to safely shutdown without powering down the robot.

Element: speak  
Attributes: message, volume, speed, pitch  
Provided By: Speech

Usage: 

```
<speaK>hello world</speaK>
<speaK volume="6" speed="3" pitch="5">
  hello world
</speaK>
<speaK message="hello world"/>
```

Description: Speaks the specified message or any nested text, including those from nested elements, at the given volume, speed, and pitch. Any attributes omitted will default to the last specified value.

Element: temp  
Attributes: none  
Provided By: UserPanel

Usage: 

```
<temp/>
```

Description: Will return the current temperature as measured by the robots temperature sensor.

Element: diagnose  
Attributes: output=[speech|log|console], level=[full|config|default|basic], all=[true|false]

Provided By: Health

Usage: 

```
<diagnose output="speech" level="full">
  Speech
  UserPanel
</diagnose>
```

Description: This will run diagnostics on the specified modules, which must be listed by their full class name as nested text and separated by whitespace. The [all] attribute can be used to diagnose all loaded modules, without listing each module individually.

Element: move  
Attributes: x, y, units=[in|ft|cm|m]  
Provided By: Magellan

Element: move  
Usage: `<move x="36" y="48" units="in"/>`  
Description: Moves the robot to the specified position, in inches, feet, centimeters, or meters. The origin is taken to be the position of the robot when the module is loaded, with the robot facing the positive y-axis. The default unit is inches.

Element: drive  
Attributes: distance, units=[in|ft|cm|m]  
Provided By: OOPic  
Usage: `<drive distance="2" units="m"/>`  
Description: Will instruct the robot to move [distance] [units]. The default unit is inches. A positive value will move the robot forward, a negative value will move the robot backwards.

Element: rotate  
Attributes: angle  
Provided By: OOPic  
Usage: `<rotate angle="90"/>`  
Description: Instructs the robot to rotate [angle] degrees. A positive value will rotate clock-wise, and a negative value will rotate counter clock-wise.

## 7 Clients

Clients are user applications used to communicate with the robot. They can be anything from Java applets to desktop applications. The communication protocol used is called SRTP, and is described in chapter 7.

The core abilities of clients is they can execute SRML through the robot and send files or modules to the robot. This gives a context to many functions of the robot. While modules and SRML elements provide functionality, clients provide usability. In the case of the Speech client, it allows a graphical application on the desktop to be used to send text to the robot which it will speak. Also, the module management application, Modutils, is a client, allowing the user to remotely manage modules. Modutils makes use of both

the SRML and module transfer ability of SRTP. It uses SRML to modify existing modules, and module transfer to send new modules to the robot.

## 7.1 Writing Java Clients

Java clients are provided with a library containing the necessary classes used to communicate with the robot. The classes used from this library are `com.sri.common.net.SRISocket`, and `com.sri.common.net.SRISocketReader`.

`SRISocket` is an abstraction of a `java.io.Socket`. When the object is instantiated, either by sending it an existing socket or the needed information to open a new socket, it will start a new thread to read from the socket and parse incoming requests.

The SRTP protocol is abstracted by translating SRTP requests into Java method calls, and vice-versa. Data is sent to the client via an `SRISocketReader`.

In addition to socket information, the `SRISocket` needs an `SRISocketReader` object to pass back information acquired from the socket. This can be done from the client by either creating a new class which implements `SRISocketReader`, or by implementing `SRISocketReader` itself.

In addition to the `SRISocketReader` interface, there is also a class `DefaultSRISocketReader`, which provides basic functionality. It by default implements each `SRISocketReader` method with no actions and returns false for both `sendAuth()` and `requiresAuth()`. This can be used to save code in your client.

An example client is provided here:

```
package com.sri.clients.testclient;

import com.sri.common.net.SRISocket;
import com.sri.common.net.SRISocketReader;

import java.io.*;

/**
 * A simple client to read SRML from the command line
 */
public class TestClient extends DefaultSRISocketReader {

    private static boolean connected;

    public TestClient() {

    }

    /**
     * Specified by SRISocketReader. Called when SRML
     * is read from the socket.
    }
}
```

```

    */
    public void readSRML(SRISocket socket, String srml) {
        System.out.println("SRML:\n" + srml + "\n");
    }

    /**
     * Specified by SRISocketReader. Called when text
     * is read from the socket.
     */
    public void readText(SRISocket socket, String text) {
        System.out.println("TEXT:\n" + text + "\n");
    }

    /**
     * Specified by SRISocketReader. Called when an error is
     * produced.
     */
    public void readError(SRISocket socket, String error) {
        System.out.println("Error: " + error);
    }

    /**
     * Specified by SRISocketReader. This is used by the
     * SRISocket to determine if authentication information
     * should be sent with the requests. The SR4 requires
     * authentication.
     */
    public boolean sendAuth() { return true; }

    /**
     * Specified by SRISocketReader. This is used by the
     * SRISocket to determine if the connection requires
     * authentication. If this returns true, then any
     * incoming requests will be checked for a user name
     * and password
     */
    public boolean requiresAuth() { return false; }

    /**
     * Specified by SRISocketReader. Used to close the
     * connection
     */
    public void close(SRISocket socket) {
        connected = false;
    }

    public static void main(String args[]) {
        String site = "192.168.0.22";
        int port = 42411;
        String username = "sr4";
        String password = "sr4";

        // creates a new TestClient to send to the SRISocket
        TestClient sender = new TestClient();

        connected = true;

        // instantiates a new SRISocket

```

```
SRISocket socket = new SRISocket(sender,
                                site,
                                port,
                                username,
                                password);

// sends the SRML to the socket
socket.writeSRML(args[0]);

// this waits for the socket to finish execution,
// and is required because of its threaded nature
try {
    socket.getThread().join();
} catch (InterruptedException e) { }
}
```

There are many improvements that can be made to this code, but at its core it is functional, and provides the necessary code to communicate with the robot using SRTP.

More information on how to use SRISocket and SRISocketReader can be found in the API documentation under the package com.sri.common.net.

### 7.2 Writing Other Clients

Clients can be written in languages other than Java, such as C, C++, C#, and Visual Basic to name a few. However, they require the code necessary to use the SRTP protocol as the SRTP library is written in Java. It is described in chapter 8.

### 7.3 Standard Clients

Here is a list of clients included with the robot.

- Name: Modutils
- Type: Command line application
- Usage: [see section 3.1]
- Description: Module management utilities. It uses a configuration file, `sr4.conf` in the users home directory to manage the required settings such as ip address, port, user name, and password.
  
- Name: Speech
- Type: Applet
- Usage: `http://[robot IP address]/speech/`

Name: Speech  
 Description: A graphical interface to the speech of the robot.

Name: SrmlIDE  
 Type: Applet  
 Usage: `http://[robot IP address]/srmlide/`  
 Description: This provides an interactive interface to writing SRML. It works as an SRML interpreter, where you input SRML, execute the SRML, and are presented with the results of the SRML.

Name: SrmlSend  
 Type: Command line application  
 Usage: **`java -jar SrmlSend.jar [filename]`**  
 Description: This will send an SRML file specified by [filename] to the robot which it will parse and return any results of the SRML. It uses a configuration file, `sr4.conf` in the users home directory to manage the required settings such as ip address, port, user name, and password.

## 8 SRTP

This describes SRTP (Smart Robots Transfer Protocol), which is the protocol used by the robot for communicating over a network.

The only need for using this protocol directly is when writing clients in languages that are not directly supported by Smart Robots. At this time, Java is the only language in which the SRTP API is provided.

If you are communicating with the robot using Java, you will use `com.sri.common.net.SRISocket`, which will abstract the SRTP protocol. This is described in chapter 7.1.

For clients written in languages other than Java, the SRTP protocol must be implemented by the client when communicating with the robot over a network socket.

SRTP is modeled closely after the HTTP protocol. Its structure consists of a header containing request information followed by the data.

A sample request is shown here:

```
SRTP/1.2
Authorization: c3I0OnNyNA==
```

```
Connection: close
Content-Type: text/srml
Content-Length: 52

<echo>
  The Temperature is <temp/> degrees
</echo>
```

Each line must be separated by either "\n" or "\r\n" to indicate that a line has ended.

Here is a detailed description of the protocol:

### Protocol

[protocol]

This is the first line of the protocol, and is currently "SRTP 1.1". If this string does not match the receiving sockets protocol or is not found in the first line of a request the connection will be refused.

This is a required line.

### Authorization

Authorization: <base64 encoding of "[username]:[password]">

This specifies the username and password combination in a base64 encoding. The steps for producing the authorization string is to concatenate the desired username, the ':' character, and the desired password, then to encode the resulting string in base64 encoding.

This is not a required line. If it is not given and authorization is required by the receiving socket the connection will be refused.

### Connection

Connection: [type]

Allows you to request the connection type. This type can be either "close" or "keep-alive". By default, the socket will be closed after the request has been processed. If "keep-alive" is used, then the connection will remain open until the socket is explicitly closed. An example that would use "keep-alive" is an applet that consistently sends data to/from the robot.

This is not a required line. It will default to "close".

### Content Type

Content-Type: [type]

This specifies the type of data that is being sent. Types of data that can be sent are "text/srml", which is SRML for the robot to parse, "text/plain", which is used to send generic information, "application/octet-stream", which is used to send files to the robot, "application/module", which is used to send modules to the robot which will then be loaded by the SR4 Kernel, and "text/error", which is text that represents an error that occurred.

This is not a required line. It will default to "text/plain".

### Content Length

Content-Length: [length]

This specifies the length of data that has been sent, in bytes. The receiving socket will read [length] bytes from the stream after the header has been read and then end the current request.

This is a required line.

### Blank Line

A blank (or zero length) line is used to differentiate the header from the content.

This is a required line.

### Content

This is the actual content. The content that is sent and received is broken into chunks of data to allow safe delivery of large amounts of data. Each chunk of data is 1024 bytes for protocol "SRTP/1.2".

If the Content Type is of type "application/octet-stream", then the following structure is used:

```
[destination filename]
[bytes of file]
```

The Content Length of this type is the string length of the destination filename plus the number of bytes in the class. Note that the destination filename is followed by a new line, "\n" or "\r\n", and new line character(s) is not included in the Content Length calculation.

A Content Type of type "application/module" is structured in the same

manner, with the name of the module replacing the destination filename.

## Appendix A: Resources

### 1. Linux resources:

- The Linux Documentation Project: <http://tldp.org/>
- Busybox: <http://www.busybox.net/>
- vi: <http://ex-vi.berlios.de/>

### 2. SRI API: [SRI User CDROM]/Documentation/

### 3. Techsol Website: <http://www.techsol.ca/>

### 4. Techsol Documentation: [SRI User CDROM]/techsol/docs/

### 5. Techsol Toolchain: [SRI User CDROM]/techsol/tools/

### 6. J2ME Documentation: <http://java.sun.com/products/cdc/>

## Appendix B: /etc/sr4/sr4.conf

This is a properties file containing key-value pairs. It is can be used by the SR4 Kernel and any loaded modules to store configuration options.

An excerpt of this file is given here:

```
kernel.logger.path=/log
kernel.logger.level=0
kernel.modules.auto=/etc/sri/modules.autoload
kernel.modules.path=/sri/jars
kernel.i2c=/dev/i2c-0
```

There are 3 types of values that are used, string, boolean, and vector. string value type is the default, and can represent any character string value. Boolean type is either "true" or "false", and within the software is converted to a boolean data type. The vector type is a comma separated list of string values, and is converted to a Vector in the software.

Key	Default	Description
kernel.logger.path	/log	The root path for the logger.
kernel.logger.level	0	The default logger level. 0 is all information, 5 is least information.

Key	Default	Description
kernel.modules.auto	/etc/sri/modules.autoload	The file containing a list of modules to be automatically loaded
kernel.modules.path	/sri/jars	The directory used to store modules
kernel.i2c	/dev/i2c-0	Device used for I <sup>2</sup> C communication
kernel.comm.port	42411	Network port for communication server
kernel.comm.username	sr4	User name used to log into comm server
kernel.comm.password	sr4	Password used to log into comm server
oopic.serial	/dev/ttyAM0	Serial port device connected to OOPIC
oopic.sonar	63	Address of sonar object
oopic.bumper	69	Address of bumper object
oopic.rightmotor	41	Address of right motor object
oopic.leftmotor	46	Address of left motor object
oopic.rightencoder	51	Address of right wheel encoder object
oopic.leftencoder	57	Address of left wheel encoder object
oopic.in1	46	Address of in1 object, used to pass data to user functions
oopic.drive	51	Address of drive function
oopic.rotate	57	Address of rotate function
httpd.wwwroot	/wwwroot	Base directory of internal web server

Key	Default	Description
httpd.port	80	Network port for internal web server
httpd.dirlisting	false	"true" to list directories, "false" to send error
mail.address	<i>none</i>	Email address of robot
mail.incoming.hostname	<i>none</i>	Incoming mail host name
mail.incoming.protocol	<i>none</i>	Incoming mail protocol, either "pop" or "imap"
mail.incoming.ssl	<i>none</i>	"true" to use SSL, "false" otherwise
mail.incoming.username	<i>none</i>	Incoming mail user name
mail.incoming.password	<i>none</i>	Incoming mail password
mail.incoming.delay	30	Number of seconds to wait before checking mail
mail.outgoing.hostname	<i>none</i>	Outgoing mail host name
mail.outgoing.ssl	<i>none</i>	"true" to use SSL, "false" otherwise
mail.outgoing.username	<i>none</i>	Outgoing mail user name, if required
mail.outgoing.password	<i>none</i>	Outgoing mail password, if required
speech.address	56	I <sup>2</sup> C address of speech board
speech.volume	6	Default speech volume
speech.speed	2	Default speech speed
speech.pitch	5	Default speech pitch
userpanel.address	54	I <sup>2</sup> C address of user panel