Java Remote Microscope for Collaborative Inspection of Integrated Circuits

by

Manuel Perez

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degrees of
Bachelor of Science in Computer Science
and Masters of Engineering in Electrical Engineering and Computer Science
at the Massachusetts Institute of Technology

May 20, 1997

© Manuel Perez, 1997. All rights reserved.
The author hereby grants M.I.T. permission to reproduce and
distribute publicly paper and electronic copies of this thesis
and to grant others the permission to do so.

Author ____________________
Department of Electrical Engineering and Computer Science
May 12, 1997

Certified by ____________________
Donald E. Troxel
Professor of Electrical Engineering
Thesis Supervisor

Certified by ____________________
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

OCT 29 1997

Java Remote Microscope for Collaborative Inspection of Integrated Circuits

by

Manuel Perez

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science in Computer Science
and Masters of Engineering in Electrical Engineering and Computer Science
at the MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Abstract

The Remote Microscope is part of a project to aid in the remote fabrication of integrated circuits. It allows one or more users to remotely view and control a microscope connected to a local or wide area network. The control takes place through a client application written entirely in Java, allowing it to run on almost any computer platform and even from an ordinary Web browser. The client boasts a user friendly interface with advanced features, such as the ability to use a VLSI layout as a navigation tool. This project makes it possible to share a microscope facility among multiple researchers and designers. It also increases the amount of collaboration, that can take place during the inspection of a wafer. These uses and features make the Remote Microscope a valuable tool in distributed design.

Thesis Supervisor: Donald E. Troxel

Title: Professor of Electrical Engineering

Acknowledgments

I would like to thank Professor Troxel for giving me the opportunity to work on this research project and continue my education at M.I.T.. In addition to opportunity, he also provided me with many great suggestion throughout my work. He is a truly great advisor, in that, he gives advice, but allows those he advises to mold these ideas into something, that they can call their own. It has been a truly great experience working for Professor Troxel.

I would also like to thank Yonald Cherry for bringing me to Professor Troxel and vouching for my abilities. He came through for me in time when I wasn't sure how in the world I was ever going to afford my fifth year at M.I.T.. And to top it all off, he even managed to find a project, which I have truly enjoyed working over the past year.

Finally, I would like to thank my family and friends, especially my mother and Carrie Morton, for their support throughout the past five years. My mother inspired me to reach for the stars, but more importantly, she showed me how to pick myself up when I fell down trying. Carrie's thoughtfulness and caring served as my center and strength, when the world seemed to face the other way.

Table of Contents

LIST OF FIGURES **7**

LIST OF TABLES **8**

INTRODUCTION **9**

BACKGROUND **11**

2.1 DEVELOPMENT 11
2.2 PREVIOUS CLIENT APPLICATION 12
2.3 JAVA 14
2.3.1 Byte Codes and the JVM *14*
2.3.2 Object Oriented Programming *15*
2.3.3 Organization *16*
2.3.4 Applications and Applets *16*

SYSTEM OVERVIEW **17**

3.1 CLIENT 17
3.2 SERVER 19
3.2.1 Hardware Setup *19*
3.2.2 Software Design *20*
3.3 SERVER/CLIENT COMMUNICATION 22
3.3.1 CHAT messaging System *22*
3.3.2 Client to Server Messages *22*
3.3.3 Server to Client Messages *25*
3.3.4 Message Interactions *27*
3.4 FOCUSING METHODS 28
3.4.1 Auto Focus *28*
3.4.2 Manual Focus *29*

CLIENT DESIGN **31**

4.1 NETWORK PACKAGE 31
4.1.1 Setup *32*
4.1.2 Receiving Messages *32*
4.1.3 Sending Messages *33*
4.1.4 Protocol Notes *33*
4.2 GRAPHIC PACKAGE 34
4.2.1 User Input *34*
4.2.1 Data Structures *35*
4.3 GIFIMAGE PACKAGE 36
4.6 LAYOUT PACKAGE 37
4.6.1 Calibration *37*
4.6.2 Navigation *38*
4.6.3 Interface Overview *38*
4.5 MAIN PROGRAM 38
4.4.1 Server to Client Messages *39*
4.5.2 User Inputs and Client to Server Messages *40*
4.5.3 Data Requests *40*
4.6 USER INTERFACE 41
4.6.1 Title and Status *43*
4.6.2 Current Settings *43*
4.6.3 New Settings *44*

4.6.4 Primary Controls *45*
4.6.5 Secondary Control *47*
4.6.6 Chat Tool *49*
4.6.7 Image Windows *50*
4.6.8 Tools Window *52*
4.6.9 Manual Focusing Windows *55*
4.7 APPLICATION OR APPLET 57

SERVER ENHANCEMENTS 59

5.1 SERVER CODE MODIFICATIONS 59
5.1.2 Server Side Image Saving *60*
5.1.3 Text Messaging *61*
5.1.4 Multithreading *61*
5.2 WEB SERVER SETUP AND CONFIGURATION 62
5.2.1 Security *62*
5.2.2 Online Documentation *64*
5.2.2 Server Side Saved Image Display *64*

MAJIK VIEWER 65

6.1 CONFIGURATION FILES 65
6.1.1 Display Styles *66*
6.1.2 Color Map *67*
6.1.3 Technology File *67*
6.1.4 Draw Map *69*
6.2 MAGIC FILES 70
6.2.1 File Description *70*
6.2.2 Internal Structures *72*
6.3 DISPLAYING A LAYOUT 72
6.4 USER INTERFACE 74
6.4.1 Layout View *74*
6.4.2 Editing Tools *76*
6.4.3 Main Controls *78*
6.4.4 Detail Controls *80*
6.4.5 Layer Controls *81*
6.4.6 Information Center *82*
6.4.7 About Controls *82*

FUTURE IMPROVEMENTS 83

7.1 LAYOUT NAVIGATION 83
7.2 LIVE VIDEO 84
7.3 ENHANCED SECURITY 84
7.4 MULTITHREADING SERVER 85
7.5 IMAGE FORMAT 85
7.6 SERVER RE-IMPLEMENTATION 85

CONCLUSION 87

REFERENCES 89

CLIENT CODE OVERVIEW 90

A.1 THE NETWORK PACKAGE (*RMNET*) 92
A.1.1 Network Package Classes *94*
A.1.2 Network Package Use *101*
A.2 THE GRAPHICS PACKAGE (*GUI*) 104

A.2.1 Graphics Package Classes *106*
A.2.2 Graphics Package Use *149*
A.3 THE GIFIMAGE PACKAGE 152
A.4 THE APPLICATION PACKAGE 154
A.4.1 Application Package Classes *156*
A.4.2 Application Package Use *166*

MAJIK CODE OVERVIEW 168

B.1 MAGIC LAYOUT CLASSES 170
B.2 MAGIC LAYOUT USE 195

USERS MANUAL 198

LIST OF FIGURES 201
INTRODUCTION 202
CLIENT APPLICATION INSTALLATION AND SETUP 203
CLIENT APPLET INSTALLATION AND SETUP 205
THE CONTROL WINDOW 207
THE IMAGE WINDOWS 210
AUTO FOCUS ALGORITHM 212
MANUAL FOCUSING 213
TOOLS WINDOW 216
LAYOUT NAVIGATION 219
TYPICAL USE 223

List of Figures

Figure 2a. Tcl/Tk Client Control Window 13
Figure 2b. Tcl/Tk Window A 13
Figure 2c. Tcl/Tk Window B 14
Figure 3a. Coordinate System 18
Figure 3b. Hardware Setup 20
Figure 3c. Server Modules 21
Figure 4a. Layout Calibration 37
Figure 4b. Server Message 39
Figure 4c. Client Message 40
Figure 4d. Data Requests 41
Figure 4e. Main Client Window 42
Figure 4g. Main Client Window 42
Figure 4h. Title and Status 43
Figure 4i. Current Settings 44
Figure 4j. Primary Controls 45
Figure 4k. Image Quality Window 46
Figure 4l. Secondary Controls 48
Figure 4m. Load Wafer Window 49
Figure 4n. Chat Tool 50
Figure 4o. Window A 50
Figure 4p. Window B 51
Figure 4q. Window B Top Information 51
Figure 4r. Window B Bottom Information 51
Figure 4s. Tools Window 52
Figure 4t. Measure Distance Tool 54
Figure 4u. Rectangle Tool 54
Figure 4v. SigF Manual Focus 55
Figure 4v. Slice Manual Focus 56
Figure 6a. Majik User Interface 75
Figure 6b. User Interface Sections 76
Figure 6c. Editing Tools 76
Figure 6d. Main Controls 78
Figure 6e. URL load dialog 78
Figure 6f. Detail Controls 81
Figure 6g. Layer Controls 81
Figure 6h. Information Center 82

List of Tables

TABLE 3A. CLIENT/SERVER MESSAGE EXCHANGE 28
TABLE 6A. DISPLAY STYLES FORMAT 66
TABLE 6B. STIPPLE PATTERN FORMAT 67
TABLE 6C. PARSED SECTIONS OF TECHNOLOGY FILE 69

Chapter 1

Introduction

The Remote Microscope was designed at MIT, as part of a project to aid in the remote fabrication of integrated circuits. It allows one or more users to remotely view and control a microscope connected to a local or wide area computer network. Such a design makes it possible to share a microscope facility among multiple researchers and designers [Kao95, Somsak96].

The client interface, to the Remote Microscope, is written in Java[1], allowing it to run on almost any computer platform and even from within a Web browser. The server runs on an IBM compatible computer, that is connected to the microscope and a video capture board. By sending requests to the server, a client can control the movement of the microscope and obtain image data.

The operation of the microscope is fairly straight forward. Typically, a user will select a desired (x,y,f) position[2] and magnification and then request an image capture. The client will forward these requests to the server, which modifies the state of the system

---

[1] Java is a programming language designed by Sun Microsystems.
[2] The (x,y,f) position refers to a point in 3d space, where the x and y have their normal meaning and the f, or focus, position is the z position.

accordingly and digitizes a new image. Usually, this involves moving the microscope to a position and magnification, performing focusing routines, capturing a new image, and, finally, returning the image to the user. Upon receiving the new image, the client will display it for the user.

The server program allows multiple connections from client applications. This enables users, at various locations on the Internet, to collaboratively inspect an integrated circuit. Though, only one user can control the microscope at a given time, any number may participate in the viewing of captured images. Clients may communicate with one another, during this process, through a built in text based messaging package.

The client application is designed to facilitate a natural interface to controlling the microscope. It contains basic controls to set image grab parameters and features to point out areas of a circuit to other clients. In addition to these basic functions, the client also supports features only available through a computer enhanced interface, such as the ability to save images, perform rough measurements, and navigate through its design layout.

This paper will examine work performed on the Remote Microscope project. Chapter 2 will cover background information, including previous work and relevant terminology. An overview of the basic components of the project and how they interact is given in Chapter 3. A detailed discussion of the client design will follow in Chapter 4. Modifications and enhancements to the server are covered in Chapter 5. Chapter 6 describes the Majik Viewer, which is used for layout navigation for the Remote Microscope. Future work and implementation ideas will be offered in Chapter 7. Chapter 8 will provide a conclusion and review work performed. Following these chapters are a set of appendices, that cover the code design and use of the Remote Microscope project.

Chapter 2

Background

This chapter will cover background information relevant to the Remote Microscope project. It will describe the development history previous to the work performed, including a look at the original client application. A brief overview of the Java programming language will also be given. Knowledge of the terminology laid out in this chapter is assumed throughout the rest of this paper.

2.1 Development

The Remote Microscope system was developed at MIT in 1995. The original work on the project was performed by James Kao. Kao's work is detailed in his thesis "Remote Microscope for Inspection of Integrated Circuits". It involved the design of both the initial client and server applications. At the completion of his work, client applications could send image capture request to the server, but an attending operator was required to respond these request and move the microscope accordingly [Kao95].

Somsak Kittipiyakul continued work on the project after Kao. He implemented the automated features of the microscope and designed a new auto focus algorithm for the

system. His work is detailed in his thesis, "Automated Remote Microscope for Inspection of Integrated Circuits" [Somsak96].

The automation of the microscope, completed the original design intent of Kao. It allowed the server side of the Remote Microscope system to operate completely independent of human intervention. This not only increased the usefulness of the Remote Microscope, but also increased the accuracy of client specified parameters, as they no longer involved possible human error.

Kittipiyakul's auto focusing algorithm offered significant improvements over the hardware defaults, included as part of the microscope. The hardware based auto focusing routines were not able to obtain quality focused images for all magnifications. The inclusion of an accurate user independent auto focusing routine was an essential component in making the Remote Microscope fully automated.

Shortly after Kittipiyakul's work was completed, the project described in this paper was begun. During this same time, Brian Lee continued work on the original set of programs. His work involved adding enhanced manual focusing options to the Remote Microscope. These options gave users the ability to accurately control the focusing of the microscope [Lee96].

2.2 Previous Client Application

The previous client application was designed by James Kao and Somsak Kittipiyakul. It is composed of two components a Tcl/Tk[3] user interface and a C daemon for handling client/server interaction. Communication between the two components is accomplished through local socket connections. The application is broken into two separate programs, because Tcl/Tk is unable to decode binary image data obtained from the Remote Microscope.

[3] Tool Command Language - a programming system for developing and using graphical user interface applications.

The visual display of the client has two main components, a control window and two windows for viewing captured images. Figure 2a shows the control window portion of the Tcl/Tk graphical user interface. It includes the manual focusing options, added by Brian Lee. The image viewing windows appears as in figure 2b and 2c.

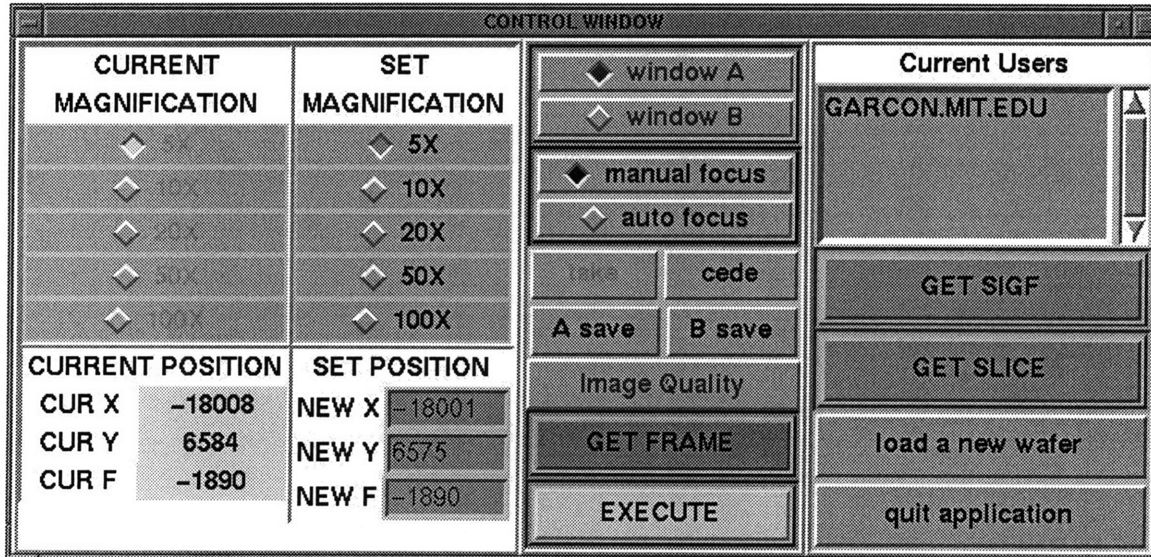


Figure 2a. Tcl/Tk Client Control Window
The graphical interface to the original client application was written in Tcl/Tk . The image above shows what the client looks like. It includes the manual focusing enhancements added by Brain Lee.

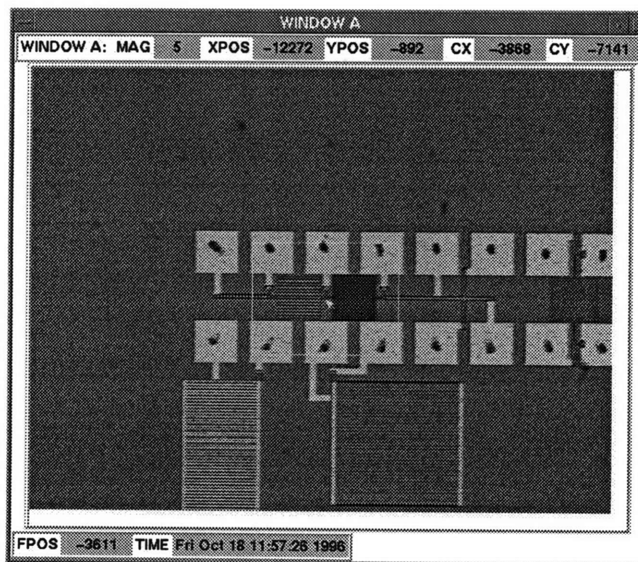


Figure 2b. Tcl/Tk Window A
The image in window displays a picture of a wafer at x5 magnification. The image of this window was obtained from the original Tcl/Tk client application.

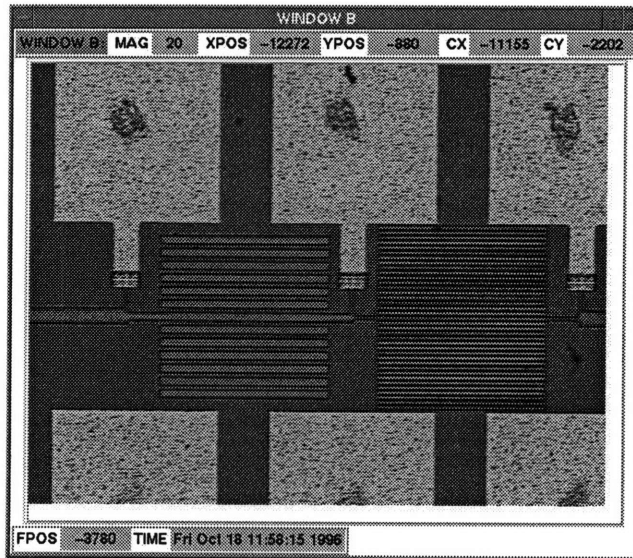


Figure 2c. Tcl/Tk Window B
The image in window displays a picture of a wafer at x20 magnification. The image of this window was obtained from the original Tcl/Tk client application.

For the view point of basic server/client interaction, the previous and new client function in exactly the same manner, despite the differences in their underlying code. Chapter 3 contains details on the general functionality of both. Great care was taken, when the new client was being designed, to maintain backwards compatibility with the previous client. The previous client does not, however, include all of the new features employed by the new design.

2.3 Java

The client interface, described in later chapters, is entirely written in the Java programming language. Java allows programmers to develop platform independent applications. It was developed an Sun Microsystems Inc. and entered into the public eye in 1995 [CORNELL96].

2.3.1 Byte Codes and the JVM

Java compilers take Java code and convert it into byte codes. Byte code can be seen as partially compiled code, as they are not actual machine instructions, as produced by a normal compiler, but are rather an architecture independent set of instructions. The

codes have been designed, such that, they translate easily into actual machine instructions for a particular computer architecture. The byte codes are interpreted and run on a Java Virtual Machine (JVM).

Java Virtual Machines are designed for specific platforms. They take in compiled Java code, in the form of byte codes, and run Java applications. The JVMs are responsible for translating the byte codes at run time into machine code for the specific CPU type.

2.3.2 Object Oriented Programming

Java is an Object Oriented Programming (OOP) language. This section will explain some of the terminology used in OOP, specifically geared towards Java. This section is not intended to cover OOP in its entirety, it does however suffice as enough background knowledge for this paper.

All code, for a particular program, is contained within objects known as classes. Each class has its own set of variables and functions, which are known as methods. Both variables and methods can be private, protected, or public. When set to private, the variables and methods are only available locally, within the class, and can not be called upon from outside classes. Protected implies that only classes inheriting from a class can gain access to them. The public setting allows all classes to access variables and methods.

A Java class can inherit from other classes. Inheritance simply means that the new class is a child of the original and acts as if it were of that particular class type. The new class will also implement new functions that expand on the functionality of the original class. Java supports two types of inheritance, "extension" and "implementation". When extending a class, the new class inherits all the functionality of the original class and adds its own set of variables and methods. When implementing a class, the new class inherits the specifications for methods within the original class. This means that the new class promises to implement all of these methods.

2.3.3 Organization

When Java classes are compiled, the byte codes for each class are stored in a separate file. Thus, for any significant programming project the number of files generated quickly becomes exceedingly large. As an organizational tool, Java supports methods for grouping classes into packages, which can each be stored in separate directories.

Packages offer a convenient method for organizing related classes. Later chapters which discuss portions of the Remote Microscope code, will make use of this organizational technique. It is important to remember that each of the packages form a distinct functionally related unit.

2.3.4 Applications and Applets

Java supports two kinds of programs, applications and applets. Applications have their compiled code stored locally and can be run from any computer which has an installed JVM. Applets can have their compiled code stored on a remote host and can be run from any Java enabled web browser.

Though both applications and applets can run on any platform, an applet implementation would be more desirable, as almost all computer users with an Internet connection make use of one or more Web browsers. Today, many operating systems are even being shipped with web browsers as part of their startup software package. Thus by using a Java applet implementation, users would not need any extra software at all. All they would need is to connect to the appropriate URL[4].

---

[4] Universal Resource Locator - used to locate documents on the World Wide Web

Chapter 3

System Overview

This chapter will detail the basic components and functionality of the Remote Microscope system. It is intended as a broad overview and is not intended to cover all areas of the design. The primary purpose of this section is familiarize the reader with how the system functions, such that the design changes, discussed in later chapters, can be fully understood.

The general function of the client and server applications will be examined. There will be a large focus on previous work on the Remote Microscope project in these sections. The underlying communications protocol and focusing methods will also be discussed.

3.1 Client

The primary function of the client is to act as a control and visualization interface for remote inspection. It allows a user to modify the state of and receive digital images from the microscope. The client is primarily composed of graphical controls, windows to view captured images, and a network layer to handle server/client communication.

Upon starting a client application, it attempts to connect to a specified server. A successful connection means that the client is now able to send and receive commands and data. Connected clients have the ability to request control of the microscope, though only one client at a time can be in this state. If the request is granted, the client is given the ability to modify the position and magnification of the microscope and request image captures

When modifying the position of the microscope, the client must specify the parameters within the coordinate system of the microscope. The coordinate system of the microscope is illustrated in figure 3a. It consists of an x, y, and f position. The three positions correspond to directions in three dimensional space, where the f, or focus position, corresponds to the z, or depth, direction.

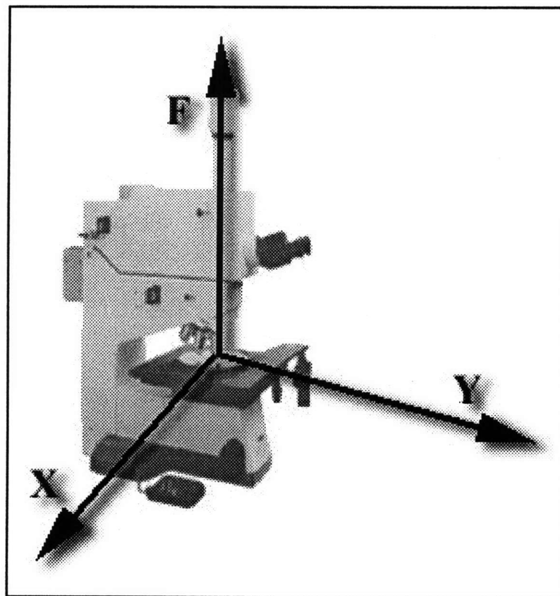


Figure 3a. Coordinate System
The image above shows the coordinate system representation used by the client and server applications. The positions are in three dimensional space. The x and y positions correspond to the horizontal and vertical direction respectively. The f (ocus) position is the depth of view.

Magnification is specified in multiples 100 and refers to the amount by which a captured image will be magnified.

3.2 Server

The servers primary purpose is to maintain the state of the system and respond to client requests. It is responsible for handling the movement of the microscope and image captures. This section examines the basic hardware setup and software applications used by the Remote Microscope server.

3.2.1 Hardware Setup

The Remote Microscope server application runs on a DELL XPS-P75 IBM compatible machine. Inside the computer is a 3Com Ethernet Adapter and a Targa+ 16/32 Image Capture board. The Ethernet adapter allows the computer to communicate with clients on a local or wide area network. The capture board is used to digitize images from a Sony DXC151A Camera w/ Zoom Lens.

Connected to the computer, through the serial port, is a Beltronics Automated Microscope controller. The controller is connected to a Zeiss Axiotron Microscope. These connections allow the host computer to control the movement of the microscope.

As can be seen, from the above description, the majority of the components used by the server are simple off the shelf computer products, with the obvious exception of the actual microscope. They are inexpensive, easily obtainable and replaceable. A visual representation of the hardware setup is given in Figure 3b.

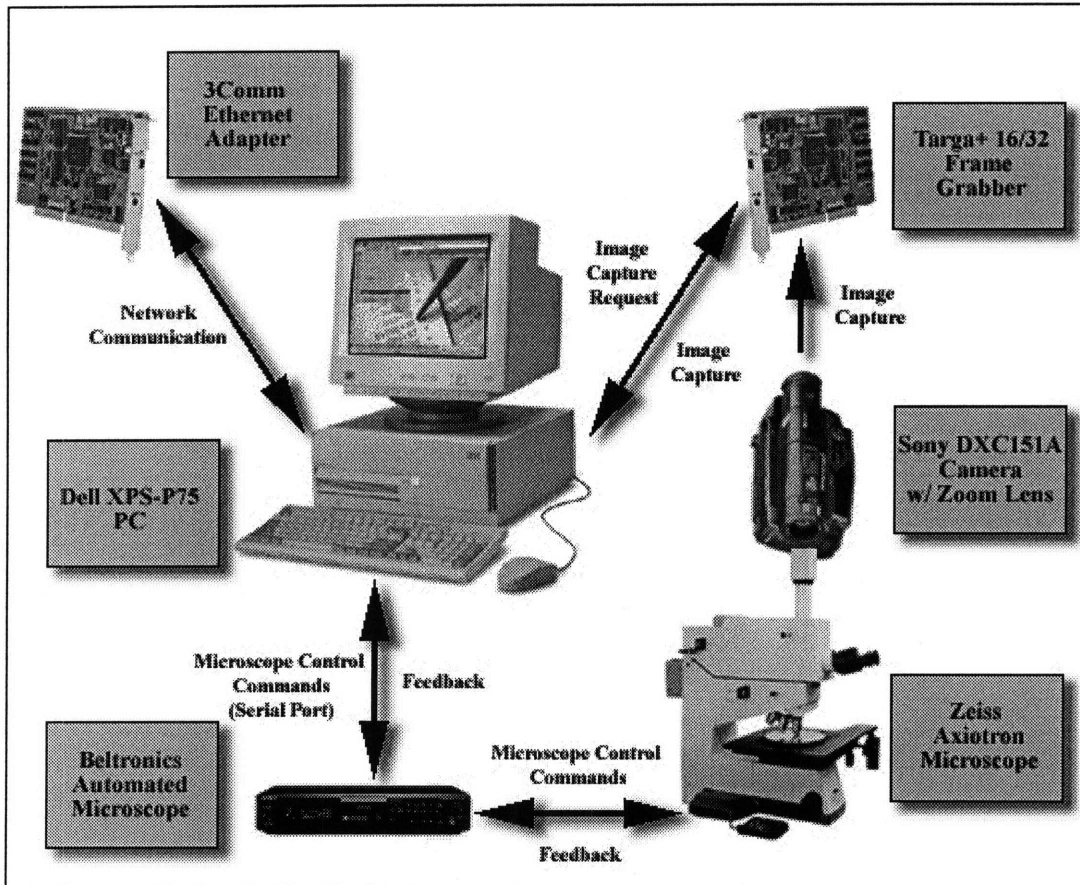


Figure 3b. Hardware Setup
This diagram illustrates the basic hardware setup for the Remote Microscope server. The images included in this diagram are not of the actual components. The arrows connection components and the text by them corresponds to common messages sent between them. The gray boxes contain the names of the elements and refer to the object nearest them.

3.2.2 Software Design

The Remote Microscope server software consists of three separate modules, the network server, microscope controller, and video subsystems. Each of these modules is designed to control a specific aspect of the microscope system. They communicate with one another through local socket connections. The three programs are written in C and have been compiled to run under the OS/2 operating system. The Connections between the modules, as well as with client applications, is illustrated in figure 3c.

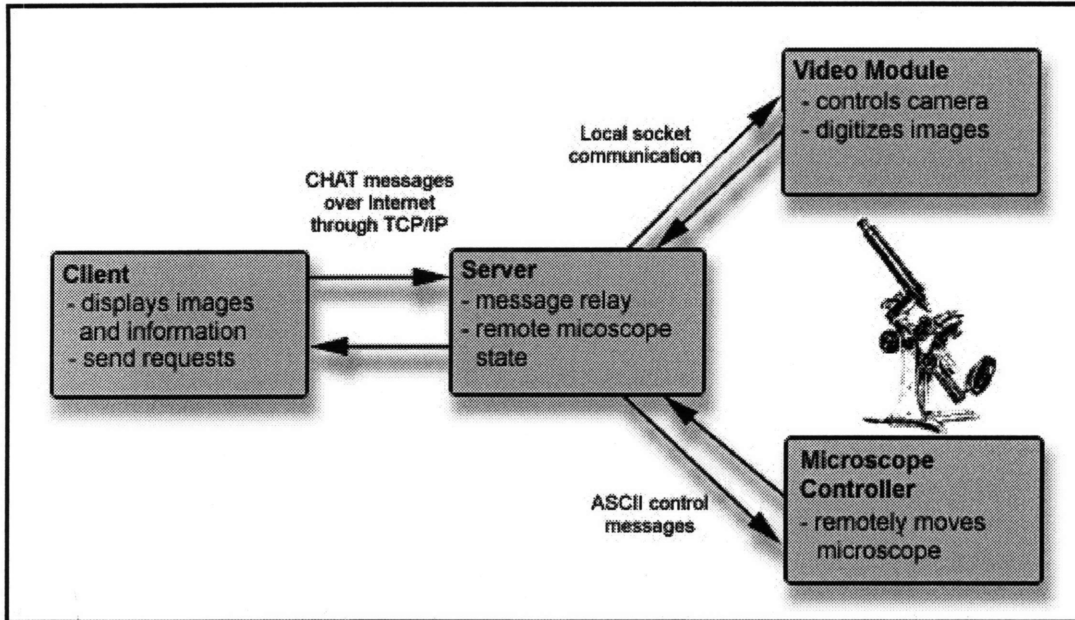


Figure 3c. Server Modules
This diagram illustrates the connections between the different server modules, as well as with the client application. The arrows indicate connections through which data is transferred. The text accompanying the arrows gives a brief description of the message type. The gray boxes contain the name of the modules in bold text and a brief listing of their functions.

The main component of the system is known as the network server module. This unit handles all interactions with client applications, as well as communication with the other two server side modules. It is responsible for updating and maintaining the state of the entire Remote Microscope system. It stores the current state of the microscope, including x position, y position, focus position, and magnification. It maintains data structures that record the parameters for the last image grabbed to each particular window. It also stores the current list of users and which, if any, of the clients currently has control of the microscope.

The microscope controller module handles all interactions between the software and the actual microscope. When a client requests a change in position or magnification, the network server will forward the request to this module. From here the microscope controller will communicate with the microscope over the serial port to make the desired modifications. It will inform the network server when it is done.

The video module is responsible for processing all image captures. When a client's message includes a request for a new image, the network server will activate this module.

The video module will digitize a new image and pass this information back to the network server.

3.3 Server/Client Communication

The clients and the server communicate over the Internet through the TCP/IP protocol[5]. This allows clients to connect to a server located anywhere on a local or wide area network. Both make use of the CHAT messaging system developed by J. Carney [Carney95].

3.3.1 CHAT messaging System

The CHAT messaging system is based on abstract objects, that can contain various data types in elements called "slots". Each "slot" is identified by a name and contains a specific type of basic data structure, such as Integer, Long, Double, Byte, Character, String, or an array of these types. Once constructed, these abstract object can be transmitted as a stream of data in either ASCII or binary form. The receiving party can then decode this information into its abstract object form and retrieve data from the "slots".

3.3.2 Client to Server Messages

The Remote Microscope client can send various request to the server application. In general these involve asking the server to modify the state of the microscope in some manner. In the original implementation of the Remote Microscope system only the client, which currently has control of the microscope, could send messages to the server. The new version has added two messages which can be executed regardless of the client's state.

This section details all of the messages which can be sent by a client to the Remote Microscope server. It includes the new "Chat Message" and "Save Image Request" commands. The format, used in this section and the next, for describing each message, is as follows:

---

[5] TCP/IP- Transmission Control Protocol/ Internet Protocol . Used to transfer data over a network.

<1: message type>

<2: description of message>
<3: parameter name>: <4: description of parameter or absolute setting>

The messages they be sent by a client to a server are:

Busy

Requests that the server inform other clients that it is about enter busy mode.
COMMAND: *"BUSY"*
MESSAGE: String containing the reason why the server is busy

Set Cursor

Requests a cursor position change.
COMMAND: *"CURPOS"*
WHICH: *(GLOBAL,ZOOM)*Target window of the change
CURX: X position
CURY: Y Position

Rectangle Position

Requests a rectangle position change.
COMMAND: *"RECTANGLEPOS"*
WHICH: *(GLOBAL,ZOOM)*Target window of the change
RX1: Top X position
RY1: Top Y Position
RX2: Bottom X position
RY2: Bottom Y Position

Magnification Change

Requests that the server change the microscope magnification.
COMMAND: *"SETMAG"*
NEWMAG: New Magnification

Take Control Request

Requests that the client be given control of the microscope..
COMMAND: *"TAKECONTROL"*

Cede Control Request

Requests that the client be cede control of the microscope..
COMMAND: *"CEDECONTROL"*

Grab Frame

Tells the server to execute a new image grab from the microscope with the specified parameters.
COMMAND: *"GRABFRAME"*
FOCUS: *(MANUAL,AUTO)* Focusing method desired
COLOR: *(c,g)* Color or Gray scale image
CTABLE: *(1,0)* Calculate a new color table if 1, else use old.

XPOS: X position
YPOS: Y position
FPOS: Focus Position
MAG: Magnification

Load Wafer

Request
Tells the server to move the microscope into position for the loading of a new wafer.
COMMAND: "*LOADNEWAFER*"

Calibration

Request
Tells the server to move the microscope into position for the loading of a new wafer.
COMMAND: "*CALIB*"
DOIT: (*YES,NO*) Do calibration?
THICKNESS: Thickness of wafer in microns
SIZEX: X dimension of wafer
SIZEY: Y dimension of wafer
TYPE: (*FLAT,ROUGH*)General Type of wafer

Get Slice

Request
Requests that the server obtain new slice focusing information.
COMMAND: "*GETSLICE*"
WINDOW: (*BIG,SMALL*) Selected image window
FOPT: (*MANUAL,AUTO*) Focusing method desired
XPOS: X position
YPOS: Y position
FPOS: Focus Position
MAG: Magnification
ROWPOS: Row position of the slice
START: Start position within the slice row
STOP: Stop position within the slice row
SIZE: Row size to grab
HYSTVAL: Hysteresis value wafer

Get SigF

Request
Requests that the server obtain new sigF focusing information.
COMMAND: "*GETSIGF*"
WINDOW: (*BIG,SMALL*) Selected image window
XPOS: X position
YPOS: Y position
FPOS: Focus Position
MAG: Magnification

Chat

Message
Requests that the server forward a message to the clients.
COMMAND: "*CHAT*"
MESSAGE: String message
USER: Name of user

Save Image

Request
Request a server side image save.
COMMAND: "*SAVEIMAGE*"

WHICH: (*GLOBAL,ZOOM*) Window image to be saved
FILENAME: Name under which to save the image
PATH: Directory under which to save the image

3.3.3 Server to Client Messages

The server messages follow the same format as described for the client in section 3.3.2. The server messages are, for the most part, designed to respond to client requests. The server will attempt to perform the desired action and then send back results. Most of the other messages are for relaying information between clients. In order to respond to the two new client messages, the server also has two new corresponding messages, "Chat Reply" and "Save Image Reply". The messages they be sent by a server to clients are:

Busy Reply

Used to inform clients that the server is currently busy executing command and can not process other inputs at this time.
COMMAND: "*BUSYREPLY*"
MESSAGE: String containing the reason why the server is busy

Magnification Reply

Used to inform the clients that the magnification of the microscope has changed. The new magnification value is passed along with the message.
COMMAND: "*MAGREPLY*"
NEWMAG: New magnification

Grab Frame Reply

Used to inform the clients it has successfully finished grabbing a new microscope image. The various parameters used for the grab are passed along with this message, as is the actual image.
COMMAND: "*GRABFRAMEREPLY*"
XPOS: X position
YPOS: Y position
FPOS: Focus Position
MAG: Magnification
FOCUS: Focus Method
TIME: Time of the image grab.
CURRENT: (*y,n*)Is this the current window?
IMAGE: Grabbed GIF Image stored as an array of bytes

Zoom Grab Reply

Used to inform the clients it has successfully finished grabbing a new microscope image. The various parameters used for the grab are passed along with this message, as is the actual image.
COMMAND: "*ZOOMGRABREPLY*"
XPOS: X position
YPOS: Y position
FPOS: Focus Position
MAG: Magnification

FOCUS: Focus Method
TIME: Time of the image grab.
CURRENT: (*y,n*)Is this the current window?
IMAGE: Grabbed GIF Image stored as an array of bytes

Cursor Position Reply

Used to inform the clients that the cursor position has changed. All client should update their views accordingly
COMMAND: "*CURPOSREPLY*"
XPOS: X position
YPOS: Y position
WHICH: Selected window

Rectangle Position Reply

Used to inform the clients that the rectangle position has changed. All client should update their views accordingly
COMMAND: "*RECTANGLEREPLY*"
RX1: top X position
RY1: top Y position
RX1: bottom X position
RY1: bottom Y position
WHICH: Selected window

Microscope State

Used to inform the clients of the current the state of the microscope.
COMMAND: "*SCOPESTATE*"
XPOS: X position
YPOS: Y position
FPOS: Focus Position
MAG: Magnification

User List Delete

Tells the clients to clear its list of users.
COMMAND: "*USERLISTDELETE*"

User List Add

Tells the clients to add a new user to its list of users.
COMMAND: "*USERLISTADD*"
USER: name of user to be added

Set State

Informs the clients that the state of the microscope has changed.
COMMAND: "SETSTATE"
STATE: (*PASSIVE,CONTROL,NONE*) the state of a specific client

Load New Wafer Reply

Used to inform the clients it has successfully loaded a new wafer and has grabbed a new microscope image. The various parameters used for the grab are passed along with this message, as is the actual image.
COMMAND: "*LOADNEWAFERREPLY*"

XPOS: X position
YPOS: Y position
FPOS: Focus Position
MAG: Magnification
FOCUS: Focus Method
TIME: Time of the image grab.
CURRENT: (*y*,*n*)Is this the current window?
IMAGE: Grabbed GIF Image stored as an array of bytes

<u>Get Slice Reply</u>

Used to inform a client that information pertaining to a get slice request has been fulfilled.
COMMAND: "*GETSLICEREPLY*"
XPOS: X position
YPOS: Y position
FPOS: Focus Position
MAG: Magnification
XDATA: Horizontal Pixel data
YDATA: Index of Pixel Data
VALSLICEF: Slice Focus Quality Value

<u>Get SigF Reply</u>

Used to inform a client that information pertaining to a get slice request has been fulfilled.
COMMAND: "*GETSLICEREPLY*"
XPOS: X position
YPOS: Y position
FPOS: Focus Position
MAG: Magnification
VALSIGF: Focus Quality Value

<u>Chat Reply</u>

Used to inform the clients of a new chat message sent by other clients.
COMMAND: "*CHATREPLY*"
MESSAGE: String message
USER: Name of user that sent the message

<u>Save Image Reply</u>

Used to inform a client of a successful server side image save.
COMMAND: "*SAVEIMAGEREPLY*"
WHICH: Selected Window
URL: Web address of the saved image

3.3.4 Message Interactions

Though, for the most part, it is obvious how the client and server messages correlate, this section will describe the basic message interactions. These easiest and most forward way to present this information is through a table containing common scenarios and the corresponding messages that are passed between the server and client applications. Table 2a, found below, is used for this purpose

Event	Description	Messages
Take Control	A client wishes to take control of the microscope and no other user currently has control	SERVER<-TAKECONTROL SETSTATE(CONTROL)->CLIENT SETSTATE(PASSIVE)->OTHERS
Cede Control	A client, which has control of the microscope, decides to release it.	SERVER<-CEDECONTROL SETSTATE(NONE)->ALL
Set Cursor	A client, which has control of the microscope, changes the position of the cursor.	SERVER-<CURPOS CURPOSREPLY->OTHERS
Rectangle Change	A client, which has control of the microscope, changes the position of the display rectangle	SERVER-<RECTANGLEPOS RECTANGLEREPLY->OTHERS
Magnification Change	A client, which has control of the microscope, requests that the magnification for next image grab change.	SERVER-<SETMAG MAGREPLY->OTHERS
Grab Image Request	A client, which has control of the microscope, requests a new image grab from the server. The messages for this section are example for "GRABFRAME" message. They are exactly the same for "ZOOMGRAB" request.	SERVER<-BUSY BUSYREPLY->OTHERS SERVER<-GRABFRAME (microscope is moved and a new image is grabbed) GRABFRAMEREPLY->ALL
Slice Manual Focus	A client, who has control of the microscope and is using manual focus, requests a new slice from the server. The messages for this section are an example of the "GETSLICE" message. They are exactly the same for "GETSIGF" requests.	SERVER<-BUSY BUSYREPLY->OTHERS SERVER<-GETSLICE (pixel information is obtained) GETSLICEREPLY->ALL
Load Wafer	A client, who has control of the microscope, wishes to view another wafer and request a wafer change.	SERVER<-BUSY BUSYREPLY->OTHERS SERVER<-LOADNEWWAFER (The server moves the microscope into docking position for a new wafer) SERVER<-CALIB LOADNEWWAFERREPLY->ALL
Chat Message	A client sends an ASCII chat message	SERVER<-CHAT CHATREPLY->OTHERS
Save Image Request	A client wishes to have an image saved by the server	SERVER<-SAVEIMAGE SAVEIMAGEREPLY->CLIENT

Table 3a. Client/Server message Exchange

This table provides a brief overlook of messages sent between client and server applications. The leftmost column contains the type of the interaction. The middle column gives a brief description of the situation. The leftmost column shows the messages which are exchanged. In the later, "*OTHERS*" refers to all other clients save the one that sent the message and "*ALL*" refers to all the client applications.

3.4 Focusing Methods

This section will describe the focusing routines included with the Remote Microscope. This includes a description of the previously mentioned auto and manual focusing methods. The focusing methods are of great importance to the project, because of the poor focusing quality obtained through the hardware defaults.

3.4.1 Auto Focus

The auto focusing algorithm was developed by Somsak Kittipiyakul and is discussed in detail in his thesis. The algorithm attempts to achieve the highest possible

value of a focus quality parameter, which he calls “SigF”. The “SigF” value is calculated through the use of a high pass filter of a scanned image. The server conducts scans, through a range of focus positions, to obtain the highest possible value [Somsak96].

3.4.2 Manual Focus

The Remote Microscope has two manual focusing options, the “SigF” and “Slice” methods. Both methods were designed by Brian Lee as part of his graduate work. This section will cover the basics of these manual focusing methods.

The “SigF” method of manual focusing makes use of the “SigF” value discussed in section 3.4.1. Basically, rather than conducting a scan of focus positions, the user can specify a specific focus position and request the “SigF” value at that point. Then these values can be manually compared, rather than relying on the server to find the best of them.

The “Slice” manual focusing method makes use of a single row or “slice” from a microscope image to conduct focusing. The client requests a horizontal scan line, at a given focus position, from the server. The server returns a set of pixel values, ranging from 0 to 32, for that particular row and focus position. These pixel values are visually displayed within the client application. By changing the focus position, new pixel values will be obtained. Since they correspond to a specific row in the actual image, they can be used as a visual check for edge transitions.

In addition to returning raw pixel values, the server will also send a focus quality value “SliceF” to the client. This value is calculated by summing the first derivatives of the pixel values. Optionally, the server can be asked to do a scan of focus positions to obtain the highest “SliceF” value in a range. This is an extra auto focusing option available only through the “Slice” focusing method.

The Slice Manual focusing method supports several parameters to customize the type of scan performed. The position of the scan line can be set and the range of pixel positions within this line may also be limited. The ability to set a hysteresis value is also supported. This value is used eliminate small transitions in pixel values [Lee96].

Chapter 4

Client Design

The entire Remote Microscope client has been re-implemented in Java. The Java version of the Remote Microscope, though containing all of the functionality of original client (and more), is implemented in an entirely different manner. Part of this change is due to the choice in using Java as a language. The original client was written in a non object oriented language, where as Java is entirely object oriented. Another factor contributing to the change is the desired expandable nature of the new client. This chapter will cover the design elements and issues of the Java Remote Microscope client, including all of the packages and user interface setup.

4.1 Network Package

The network package handles all interactions between the Remote Microscope server and client applications. It operates through a well defined protocol, to allow clients to both send and receive various message types from the server. As long as this protocol is obeyed, the higher levels of the client program can ignore the actual working of the networking methods involved. The main functionality made available to the client program is the ability to send new messages and receive information from the server.

4.1.1 Setup

The setup of the network package involves forming a relationship between the client and network package. It is established through a promise by the client to implement methods to transfer data and a promise by the network package to provide methods for sending data to the server. The first promise is kept by having the main client program implement the **RMclientShell** interface. This interface specifies functions, that allow the network package to send commands and data to the client. The handling of this data is left up to the client. The second promise is kept through the client's incorporation of a **RMclient** class instance. The **RMclient** class provides a send function for the client.

The communication between the network package and the Remote Microscope server is initialized through the **RMclient** class. The client application is required to pass a host name and port number to an instance of the **RMclient** class. An attempt will then be made, by the network package, to open a connection to the specified address. If a successful connection is established, then the messaging system is active. When this process fails an appropriate error message will be generated.

4.1.2 Receiving Messages

The main network package class, **RMclient**, will spawn a separate thread to handle messages received from the server. This thread will wait, in an infinite loop, until the server sends the client a new message. Upon receiving a message, it will be passed to a message handler class, **RMmessageHandler**. Here it will be decoded and its type will be compared to the known messages types. If the type is recognized, the appropriate command will be issued through the **RMclientShell** interface and the corresponding decoded data will be transferred. From here the client program is responsible for handling how the data is to be interpreted and what to do with. If the **RMmessageHandler** class receives an unknown type, it is simple ignored, this is to insure future compatibility with unknown messages.

4.1.3 Sending Messages

Once initialized correctly, the client will be able to send messages to the server through the **RMclient** network package class. Before any message is sent, it must be properly packaged. This packaging process simply involves placing all of the parameters of the message into an Object of type **MSGobject**, which is based on J. Carney's CHAT messaging protocols. Once in the proper form, the client passes this message to the RMclient class. Here the network class takes over and is responsible for the delivery of the message to the server.

4.1.4 Protocol Notes

It should be obvious from the above descriptions, that the methods for receiving and sending messages are very different. To receive messages, the entire operation of the network package is completely transparent to the rest of the program. All that is required is that the **RMclient** be able to execute certain commands and transfer data to the client. The sending protocol, however, is not completely transparent. It requires that the client format the message properly before passing it to the **RMclient**. Thus the client is forced to deal with the **MSGobject**, which should theoretically be invisible to the rest of the program. A brief explanation is required to justify, what at first appears to be, a violation of abstraction.

The **MSGobject** class merely provides methods for packing and unpacking data. Though it is primarily used for transferring data through network streams, its abstract implementation does not dictate this as its only possible use. Thus, in the case of the client using the **MSGobject** class, it is only using it to package data. How this data is used is defined in the network package. Thus, there is no real violation in abstraction at all. This method of transferring data, from the client to the network package, was implemented due to its ease of use. In the current client application, its use probably saves several lines of code and in future versions, that may or may not make use of the CHAT messaging system, nothing will be lost.

4.2 Graphic Package

The graphics package contains all the basic graphical user interface elements used by the Remote Microscope client. The graphical elements provided, range from specially designed buttons to self contained windows. This package does not include the main application window, though it does provide all of the elements required for its construction.

Many of the Graphical elements are used to interact with the rest of the program. This interaction takes place on two levels. The first of these involves the graphic element formatting the data and passing it to the appropriate client defined method. The second involves the main client application grabbing event signals from the graphical interface and handling them on its own.

In addition to handling user inputs, some of the graphic elements are used to store various portions of data obtained from the user and the server. Methods are provides such that data input and output from these elements is independent of the visual display. Thus from the point of view of the main program, graphic elements appear as data structures. As long as the proper method is called, each graphic element is responsible for storing and displaying data input and returning the correct output.

This section will examine the different types of interactions, that take place between the main program and the graphics package. This includes the use of the package elements as user input devices and as data structures. The actual user interface will not be discussed in this section, but will be covered in section 4.6.

4.2.1 User Input

User inputs are handled in one of two ways. Either the inputs can be handled locally, within a graphical element, and then pass information to the main program or complete control can be given to the main program and it can handle the inputs. In general, elements that do not spawn a new window or are not windows themselves, have

their inputs handled by the main program. Elements that involve a window, other than the main application window, usually handle at least some of their inputs themselves.

The reason behind having two methods, for handling user inputs, is very simple. Many interactions do not need to know about the main application to perform their given task. Thus, they should be handled locally. Other elements require heavy amounts of information from both the main program and even other packages. The graphical package should be independent of other packages and, in most cases, individual elements of this package should even be independent of one another. Thus, these elements are best handled by passing control to the main application.

For those elements that handle all or part of user inputs, the graphics package provides an interface class, **RMguiShell**, that allow graphical elements to execute commands within the scope of the main program. Through these commands, elements can access information and pass data to the main program. The **RMguiShell** can even be used to access information from complete independent graphical elements

Elements, that do not handle their own inputs, merely pass event signals to the main program. The main program is then completely responsible for making sure that the users actions are taken care of. This method is especially useful for inputs that require the client to interact with multiple elements and packages.

4.2.1 Data Structures

Information obtained from the user and from the server must be stored in some location. Though it may seem odd at first, most of the this information is stored within the graphical elements. The reasoning behind this implementation choice is as follow. In order to display a given piece of information a graphical element must have that information stored locally. The data stored within graphical elements mirror data structures that would normally hold the information. Even though graphical elements have a visually displayable component, they are still completely valid Java Objects, that

can be used to store data. Thus, by using the actual graphical elements, as data structures, repeating unneeded code. However, the function of the elements as data structure, should be independent of how data is stored and how the elements are displayed.

Graphic elements, that provide data structure functionality, disallow any direct interaction with their visual components. Thus from the main program's point of view, after creation and placement, the elements are essentially just data structures. Each individual element provides a variety of methods that must be used to set and retrieve data. When data is set the element is responsible for storing the data, as well as displaying the new information to the screen. On a request for data, the data will be converted to an appropriate for and returned.

Since the main program can only interact with the data through specified methods, a layer of abstraction has been created, that allows the individual graphic elements to have an independence between what data they store and how it is to displayed. This separation does, however, have a major limitation. The data must be stored in the same place. For minor changes to the user interface, this limitation is almost non existent and in most cases even desirable. On the other hand, a major reconstruction of the user interface would be made extremely difficult by it.

4.3 GIFImage Package

The Remote Microscope transmits all images in GIF format, as an array of bytes, to the network package. Java does not directly support the ability to convert an array of bytes into a GIF file. A Java based GIF conversion class called **GIFImage** written by Tadashi Hamano, found through the Gamelan Java repository, is used to handle this task. As, the GIFImage package was not designed specifically for this project and is merely being used as a stand alone set of methods, it will not be discussed any further. For the purposes of the Remote Microscope it suffices to say that it performs its required task [Hamano96].

4.6 Layout Package

The Remote Microscope incorporates a Java based VLSI layout tool, that can be used to navigate around an integrated circuit. The specifications for the layout navigator is well defined and has been designed to support a variety of possible candidates, including both 2D and 3D Java CAD tools. The current Remote Microscope makes use of the Majik Viewer, designed as part of this thesis project. It will be discussed in detail in chapter 6.

The Remote Microscope and the layout system communicate with one another through a Java interface class called the **RMlayoutShell**. This class specifies certain functions that are used to translate coordinates between the two programs and allow either to sync their current position. To establish this connection/translation relationship, each of the viewing tools must first be calibrated using a predefined method.

4.6.1 Calibration

The method of calibration, employed by the Remote Microscope and the Majik Viewer, is the selection of a common rectangular area on both. The calibration rectangle uses the first (x, y) pair as the center or (0,0) point for each of the tools. The difference between the first and second (x, y) pair defines a vector, which is used to calculate direction and distance.

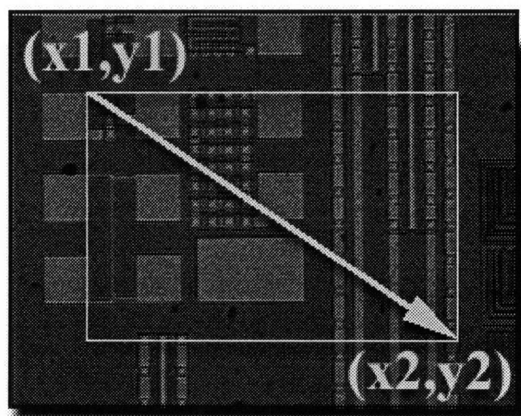


Figure 4a. Layout Calibration
The image displays a calibration rectangle, with coordinates (x1,y1) and (x2,y2). It also shows the vector, that is constructed to calculate direction and distance for point to point translations.

4.6.2 Navigation

Once both the Remote Microscope and the Majik Viewer have been calibrated, the user can select a position on either and have the other move to that location. In the case of the Remote Microscope, this simply involves telling the Magic Viewer to move the layout to the desired location. When a position is selected by the Majik Viewer, the Remote Microscope will send a new image request to the server, with the new specified position. When the server responds, an image corresponding to the requested location will be displayed.

4.6.3 Interface Overview

The methods used to accomplish the tasks described are given below:

public int[] LSgetCalibCoords()
This method is used to retrieve the calibration coordinates of a layout shell. These coordinates are used to translate between different view points. The points are returned in a single index array, such that the interface is independent of dimensions.

public void LSsetPos(RMlayoutShell *layout*, **int** *points[]*)
The current position of the layout interface is set through this method. The layout parameter specifies the class requesting a position change. The points parameter, given in the requesting classes coordinate plane, stores the desired location. Using internally stored calibration coordinates and those obtained from the requesting layout interface, through the LSgetCalibCoords() method, the position of the interface is set.

public void LSsetShell(RMlayoutShell *loShell*)
This method specifies layout shell which should be used as a receiver for position change requests.

Any navigation program, which follows this simple interface, will be able to work in conjunction with the Remote Microscope. It is even possible to have more than one layout tool.

4.5 Main Program

The main program brings together all of the packaged components and, either directly or indirectly, handles all interactions between them. It is the controller and as such, has a varied set of responsibilities. Whenever a command is issued from the network package, the main program is responsible for distributing the appropriate data to their respective data structures. If an input is detected, it performs the desired operations and,

if appropriate, returns a value. If a component requests information, it retrieves and returns the data. These are the major functions of the main program, though it handles a variety of smaller tasks as well.

4.4.1 Server to Client Messages

When the network package receives a message from the server, it issues a command to the main program. Here the parameters of the command are passed to the appropriate graphical data structures. The graphic elements store the data and change their visual components to display the new data. If the message contained a new image, this data is first decoded using the GIFImage class and then passed the appropriate graphical element. Figure 4b gives a visual representation of the relationship described above.

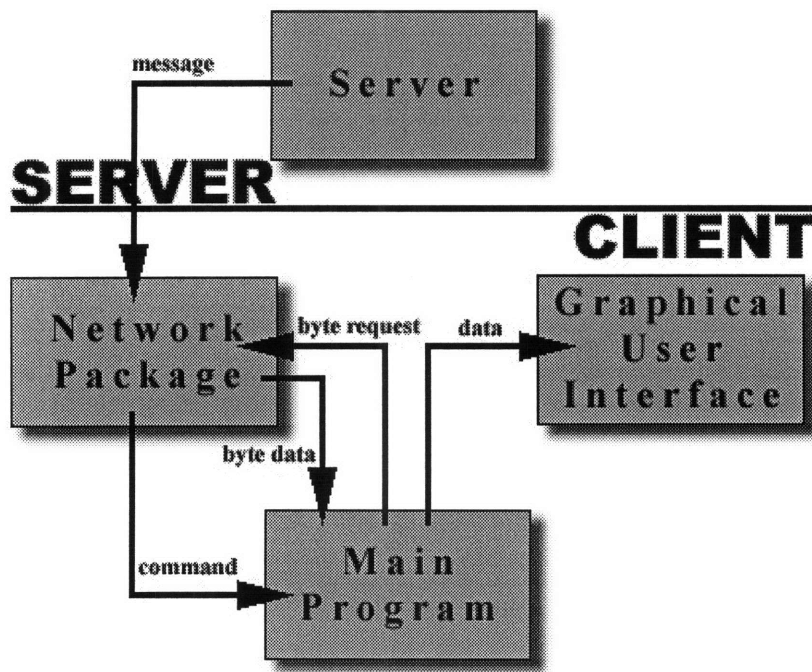


Figure 4b. Server Message
The server sends a message to the client. The network package receives this information and issues a command in the main program. The main program takes the data stored in the command's parameters and passes this to the graphical user interface. If the command was of a type that has an associated image, then a byte request is sent to the network package. The network package takes the byte information from the original server message and sends it to the main program. Here bytes are transformed into an image and sent as data to the user interface.

4.5.2 User Inputs and Client to Server Messages

User input may require that a message be sent to the Remote Microscope server. Such input can either be direct requests from the graphical interface or merely the passing of an input event, that causes the main program to issue such a command. In either case the client application behaves in much the same manner. First it requests all required data from the data structures within the graphics package. It packages this data and then sends it to the network package. From the network package it is sent to the server. Figure 4c gives a visual representation of this process.

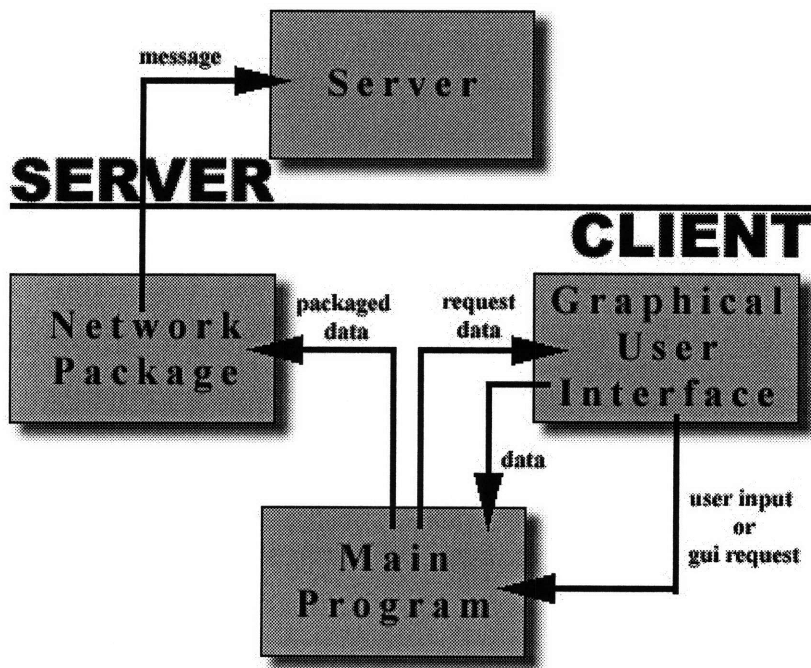


Figure 4c. Client Message
User actions, trapped from the graphical interface, may require that the a message be sent to the server. These requests are handled by having the main program retrieve necessary information from the graphical interface's data structures, packing this information, and send this information to the network package. From here the network package send the message to the server.

4.5.3 Data Requests

For some internal processes the main program or any of the other packages may require data that is not available locally. When such information is desired, data requests are sent to other portions of program. If the requesting party is the main program, then this task is accomplished by calling the proper methods within the packages that contain

the data. This is the equivalent of a normal function call. When the requesting party is one of the various other packages it must request information from the main program. The main program is then responsible for retrieving this data, from other packages if need be, and returning it. An overview of this process is detail in figure 4d.

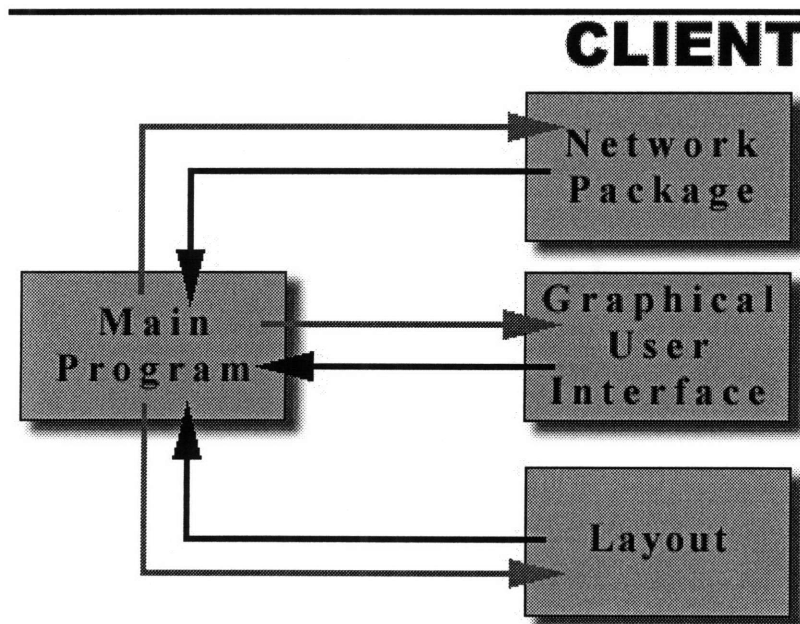


Figure 4d. Data Requests
Various packages, including the main program, may require external data that is not available locally. These requests are handled by the main program, which requests data from the packages and returns this data to those requesting it.

4.6 User Interface

This section will examine the client user interface. A complete users manual is given in appendix C. The appendix will explain how the interface is used, while this section will detail how components interact with the underlying code. Figure 4e contains an image of the main control area of the user interface.

The main control window is used both for user interaction and the display of microscope status and data variables. It is composed of six subsections, which contain (1) title and status, (2) current settings, (3) new settings, (4) primary controls, (5) secondary controls, and (6) chat tool. Figure 4g shows the breakdown of these subsections in the display.

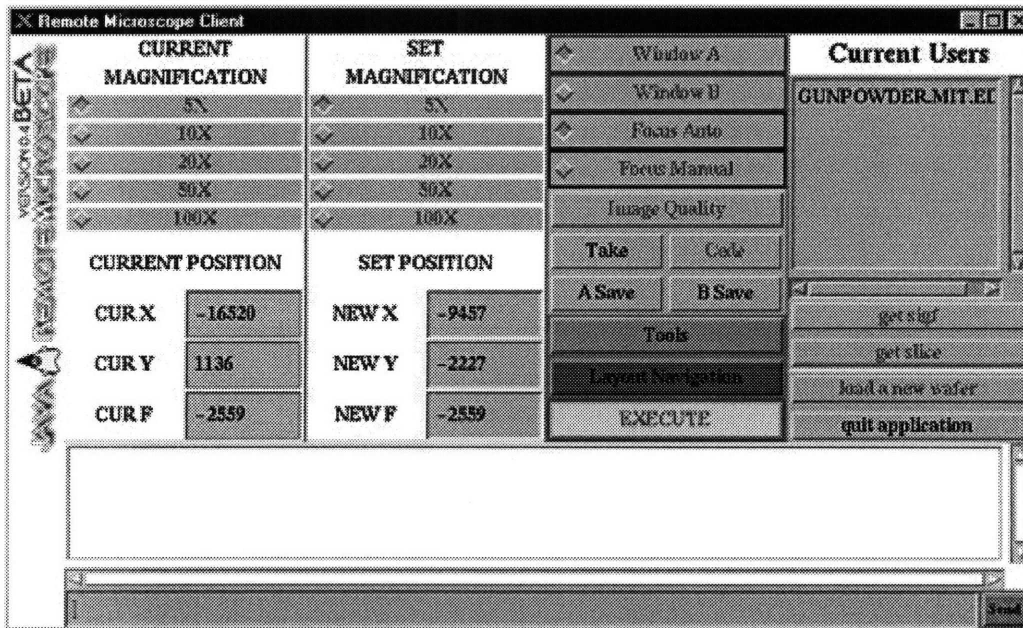


Figure 4e. Main Client Window
The main client user interface running on the standard Sun Microsystems Inc. distribution of the Java virtual machine on SunOS 5.5.1 operating system. It contains various controls to interact with Remote Microscope.

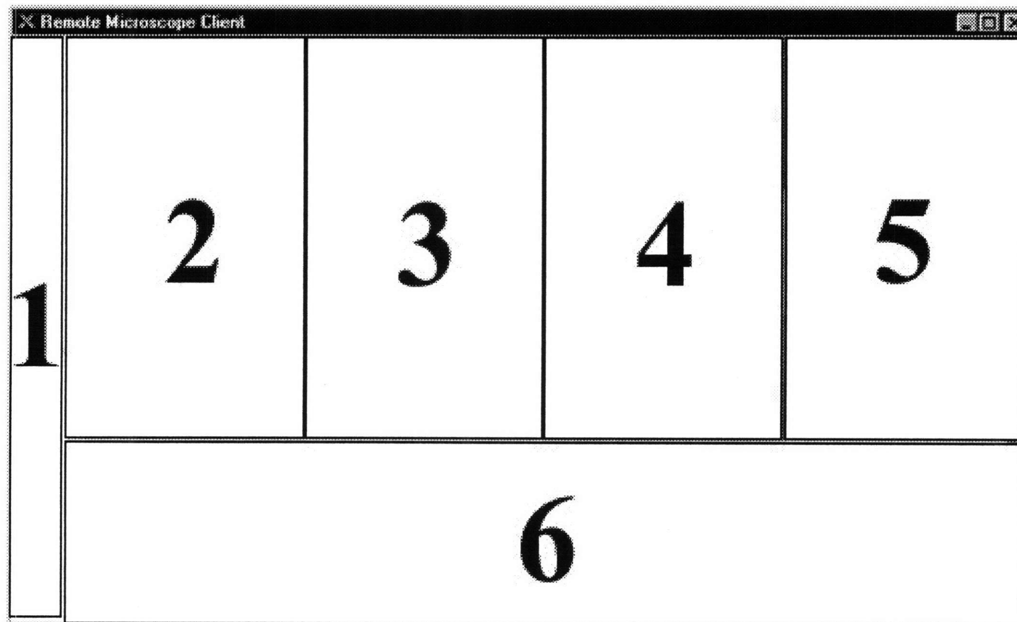


Figure 4g. Main Client Window
The breakdown of the client control window into subsections. The subsections are (1) title and status, (2) current settings, (3) new settings, (4) primary controls, (5) secondary controls, and (6) chat tool

The above mentioned sections of the main client window, as well as other user interface elements contained in secondary windows, will be discussed. These elements include the captured image display, manual focusing, and tool selection windows. The Majik Viewer will not be covered in this section, but will be fully examined in chapter 6.

4.6.1 Title and Status

The title and status panel is implemented through the **RMtitlePanel** class. This class takes in two images in its constructor. The first of these images is the title image of the application and is used to convey application name and version information. The second, is a busy image that is displayed whenever the Remote Microscope is busy executing a command on the server. The two states of the panel are shown in figure 4h.

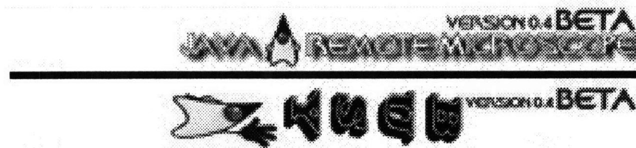


Figure 4h. Title and Status
The top image shows the panel in title mode. In this mode, it merely displays the title of the program and version information. The bottom image is the panel in busy mode. In this mode a simply bust message is displayed.

4.6.2 Current Settings

The current settings panel displays the current state of the microscope's grab parameters. It inherits from the **RMsettingsPanel** class. This class incorporates two other graphical objects, **RMmagPanel** and **RMpositionPanel**. The **RMmagPanel** is composed of five buttons that are used to select and display the magnification of the Remote Microscope. The **RMpositionPanel** contains three text areas, which are used to store x, y, and focus(f) coordinate information. Figure 4i shows a blown image of this subsection.

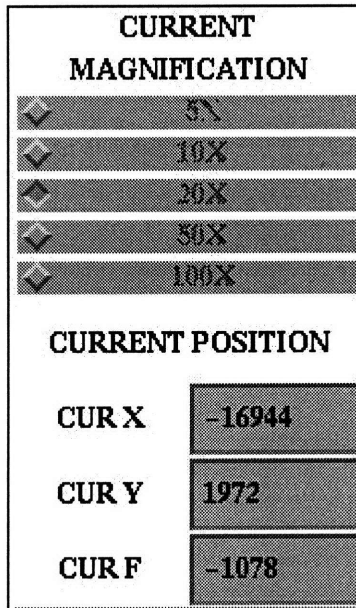


Figure 4i. Current Settings
This image shows the current settings panel. This panel contains information on current grab parameters of the Remote Microscope.

The **RMsettingsPanel** and its sub-panels, each, support the ability to disable and enable editing of the displayed information. In the case of the current settings panel, the option to do this is always turned off. This means that the values stored here can only be modified by the internals of the program and not directly by the user. All the panels also support data retrieval functions for the main program. This allows the higher levels of the program to request magnification and position information from the **RMsettingsPanel**.

4.6.3 New Settings

The section, containing the new settings, appears almost exactly like that shown for the current settings (figure 4i). This is largely due to the fact they each make use of the **RMsettingsPanel** class in their construction. Obviously, the new settings panel is also used to display similar information. In this case, however, the grab parameters are those that a user desires for the next image the microscope will be asked to grab. When a user is in control of the microscope, the information contained within this section is editable. The users may modify these values directly, though they will most likely prefer to use the various navigation enhancements.

4.6.4 Primary Controls

The primary controls contain the main execution controls for the Remote Microscope. These include buttons for window selection, focusing method, image quality, and execution request. It also contains buttons to take/cede control of the microscope, save images, and bring up secondary windows, such as those used for tools and layout navigation. Figure 4j shows a visual display of the primary controls.

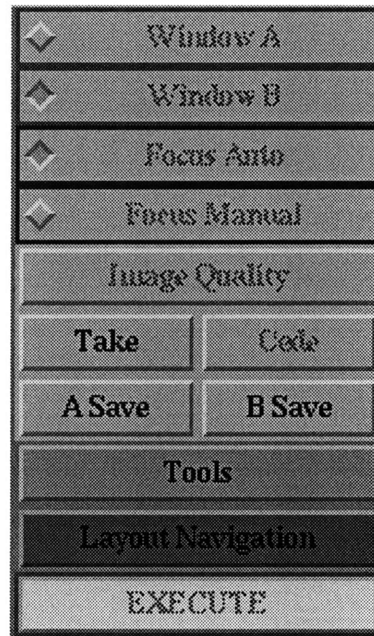


Figure 4j. Primary Controls
The primary controls allow the user to control various parameters of an image grab. It also contains buttons to perform certain secondary functions. The above image shows the controls when the current client is not in control.

The primary controls are housed in the **RMcontrolPanel** class. This class contains many sub classes that are used construct the various buttons for the interface. These include the classes **RMwindowSelectPanel**, **RMfocusSelectPanel**, **RMimageQualityPanel**, **RMsavePanel**, **RMtakeCedePanel**, and **RMexecutePanel**.

The **RMwindowSelectPanel** class forms the window selection options portion of the interface, located at the top of the controls panel. It allows the user to select the

window to which new images grabbed from the Remote Microscope server will be sent. This option is only available when the user has control of the microscope.

The **RMfocusSelectPanel** class is located just beneath the **RMwindowSelectPanel** within the display. It is also only active when the current client has control of the microscope. Options to select "manual" and "auto" focus are contained here. When in "manual" mode, certain selections within the secondary control section will be activated. With "auto" selected the microscope will execute the standard auto focus algorithm on a new grab image request.

The quality of the next image grabbed from the Remote Microscope can be specified by clicking on the **RMimageQualityPanel** object in the interface. This will bring up an image quality window, as shown in figure 4k. This contains switches to select quality parameters for the next image grabbed by the Remote Microscope. The top most button is used tell the server to re-calibrate its internal color table. The color level of the image can be selected with the bottom button items. When the done button is clicked, the user selected setting are stored locally and the window is closed. The selection of image quality is only made available when the client has control of the microscope.

Figure 4k. Image Quality Window
The above image shows the display of the image quality window. This window is used to select quality parameters for the next image grabbed by the Remote Microscope.

The **RMtakeCedePanel** lies directly below the **RMimageQualityPanel** graphical display. This class is essentially just a place holder. It just passes event signals, i.e. button presses, to the **RMmainFrame** class. The "take" button is only active when no client has

control of the microscope. The "cede" button is only available when the current client has control.

Below the **RMtakeCedePanel** visual display is the **RMsavePanel** object. This class simply contains buttons to save images from the image display windows. Based on the button selected, the **RMsavePanel** with execute the appropriate save routine within the main program. If the program was run as an application, then a file dialog window will be opened. Here the user can select a location and filename for the image to be saved. However, if the client was initialized as an applet, it will request that image be saved on the server and that the web address of that saved image be returned to the client. Upon receiving the address of the image, the program will bring up a new Web browser window containing the image. From here it can be saved to locally to disk.

Located at the bottom of the primary control section is the **RMexecutePanel** class. As with the **RMtakeCedePanel** object, this graphical element is just a place holder class. All event signals to this object are passed on to the **RMmainFrame** class. The ability to use the "Execute" button is only available when the client has control of the microscope.

Between the **RMsavePanel** and **RMexecutePanel** elements are two buttons labels "Tools" and "Layout Navigation". These buttons are used to bring up secondary control windows. The "Tools" button opens a tools window, to be used in conjunction with the image display windows. The "Layout Navigation" button brings up the Majik Viewer

4.6.5 Secondary Control

The title of this section is a bit misleading, as the area, called the secondary controls section, actually contains a variety of different items, many of which have nothing do with how the microscope is controlled. This section is composed of **RMuserPanel,**

RMmanualFocusPanel, and **RMextraPanel** objects. Figure 4l shows the visual display of the secondary controls.

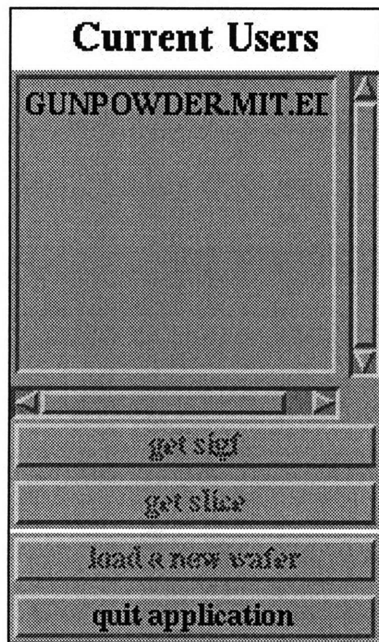


Figure 4l. Secondary Controls
The secondary controls display the current list of users, contain manual focusing options, a load wafer button, and a quit application button.

The **RMuserPanel** object appears at the top of the display. It contains a list of machine names for all the clients currently connected to the Remote Microscope server. This panel has no user input capabilities.

The **RMmanualFocusPanel** object, located beneath the user list, contains two buttons to activate different manual focusing options. These buttons are only active if the client has control of the microscope and the current focus selection is set to "manual" in the primary control section. Clicking on the "get sigF" button will bring up the "sigF focusing" window. Clicking on the "get slice" button will bring up the "slice focusing" window. The use of these windows is covered in section 4.6.9.

Below the **RMmanualFocusPanel** section, is the **RMextraPanel** display. This section contains two buttons. The first of these is labeled "load a new wafer". When this

button is clicked it will bring up a "load wafer" dialog window, which appears as in figure 4m. the **RMloadWaferPanel** is only active when the client has control of the microscope.

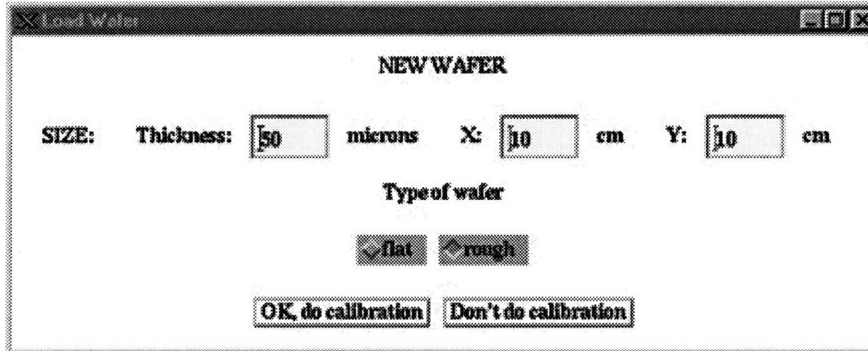


Figure 4m. Load Wafer Window
The load wafer window is used to specify the parameters for a new wafer. The parameters available are the size, dimensions, and type. After these have been determined, the bottom buttons are used specify whether or not the server should re-calibrate or not.

In addition to bringing up this window, a message will be sent to the sever to place the actual microscope in its loading position. Within the new window are parameter settings for the new wafer being loaded. These include the thickness of the wafer, its dimensions, and its general type, either "flat" or "rough". Once these parameters have been specified and on the server side a new wafer has actually been loaded, the user can click on the "OK, do calibration" or "Don't do calibration" buttons to finalize the process. The server will be sent the appropriate message and either re-calibrate itself or not accordingly.

The second item in the **RMextraPanel** is the "quit application" button. Pressing this button will closet from the Remote Microscope client program. This involves closing all network connections, closing all open windows, and terminating all internal threads.

4.6.6 Chat Tool

The chat tool section of the user interface is constructed through the **RMchatPanel** class. A blown up image of this section appears in figure 4n. This section displays text submitted by other clients and contains a section to send new messages. Messages are sent whenever the "send" button is clicked or the "ENTER" key is pressed.

A special message tag to set the user name is supported. The format of this tag is "/name <name>", where <name> specifies the user name.

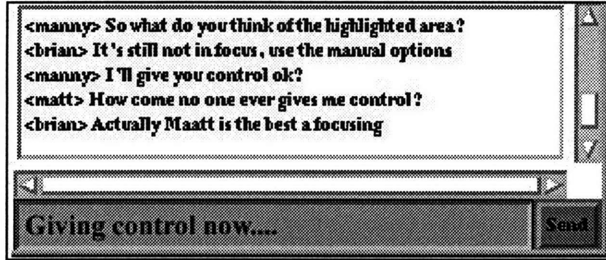


Figure 4n. Chat Tool
The top text box, with the blue text, displays the messages received from the server, that were sent by clients. The bottom text area, with black text, is an input window for a user's message. The "send" button or the "Enter" key are used to send a new message.

4.6.7 Image Windows

The client has two windows that are used to display images and image information obtained from the Remote Microscope server. In general use "Window A" is used as a global view and "Window B" is used as a zoom view, though essentially the windows can be used to display any image from the microscope. Figure 4o and 4p show image windows A and B respectively. The image windows inherit from the **RMviewFrame** class.

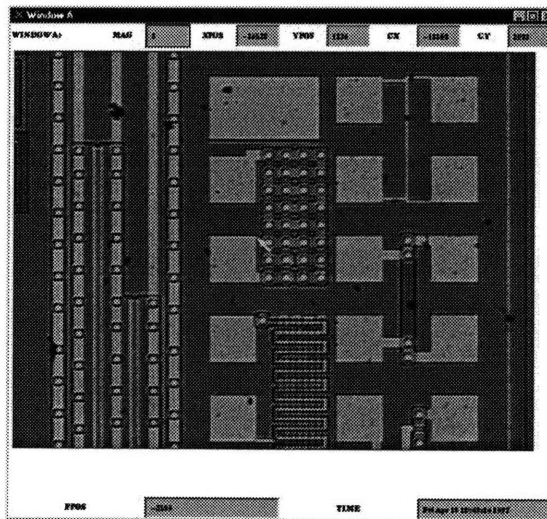


Figure 4o. Window A
The image in window displays a picture of a wafer at x5 magnification.

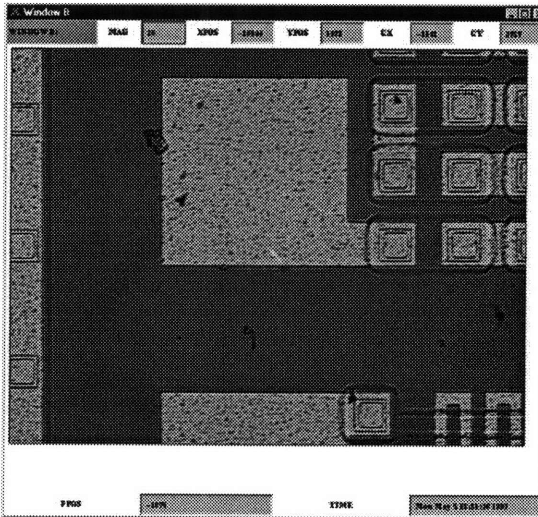


Figure 4p. Window B
The image in window displays a picture of a wafer at x20 magnification.

The data stored by the image windows is located at the top and bottom of the windows. The top display is created from the **RMtopViewPanel** class, it contains the magnification, center x position, center y position, current x position, and current y position. The center positions correspond to the position at which the microscope image was initially grabbed, while the current positions refer to the position of the mouse pointer in microscope coordinates. A blown up image of the top section of the **RMviewFrame** is shown in figure 4q.

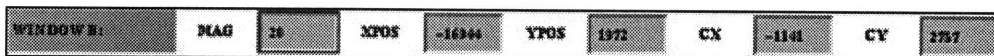


Figure 4q. Window B Top Information
The image above shows the top portion of Window B. The name of the window is green, because it was the destination for the last image grabbed from the Remote Microscope. The magnification, xpos(ition), and ypos(ition) of the grab are displayed in the corresponding fields. The cx and cy fields store the current position, in microscope coordinates of the mouse pointer.

The bottom of view window inherits from the **RMbottomViewPanel** class. It contains the focus position of the image grabbed and the time of the grab. A blown up image the bottom section appears in figure 4r.

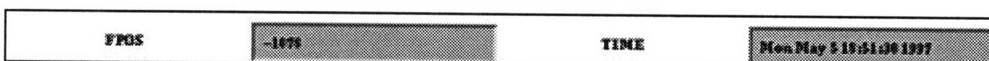


Figure 4r. Window B Bottom Information
The bottom of window B is displayed above. It stores the focus(fpos) of the image grabbed and the time of the grab.

When the mouse is within “Window A” or “Window B”, it can be used to interact with the image. The leftmost mouse button always functions in the same manner, allowing a user, in control of the microscope, to “drop” a pointer on the image. Dropping this pointer sends a message to all the other clients. Upon receiving such a message, a pointer, at the same location, appears in their windows as well. Thus the pointer can be used as a device for pointing out areas of a wafer. In addition to this, the new settings section of the user interface is updates as well. Thus the pointer can be used to choose the position of the next image grab. The rightmost mouse button’s function is determined by the tool type selected from the tools panel. The next section will cover these functions.

4.6.8 Tools Window

The tools window is based on an instance of the **RMtoolsFrame** class, whose main component is the **RMtoolsPanel** object. The **RMtoolsPanel** object contains three sections. The first of these is a set of tools that can be used in conjunction with an image window. The second is a window selection utility. The third is a quick navigation control area. An image of the tools window appears in figure 4s.

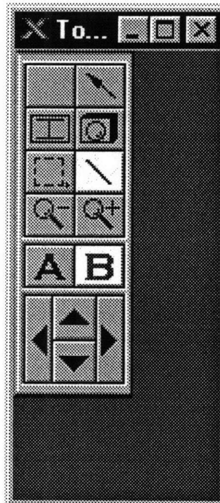


Figure 4s. Tools Window
The Tools Windows allows the user to determine the function of the right mouse button in an image window. It also allows quick and easy access to some navigation controls.

The topmost section of the RMtoolsPanel object contains a set of tools, which determine the functionality of the rightmost mouse button within an image window. The

available tools are "go to layout position", "calibrate", "grab image at", "set rectangle", "measure distance", "decrease magnification and grab", and "increase magnification and grab". The currently selected tool will appear as a depressed button.

The "go to layout position" and "calibrate" are used to interact with the layout navigation tool. The "go to layout position" tool is located at the top right corner of the tools and appears as an arrow. It is used to tell the client to move the layouts view to the location specified by a right mouse click within an image window. The "calibrate" tool is located in the second from the top row to the far left and appears as the letter "I" surrounded by a rectangle. When selected, the right mouse can be used to generate a calibration rectangle as discussed in section 4.6.1.

The "grab image at", "decrease magnification and grab", and "increase magnification and grab" tools are used to execute image grab request from directly within the image windows. When one of these tools is selected a right mouse click will send a request to the server for a new microscope image at the specified location. In the case of the "decrease magnification and grab" and "increase magnification and grab" tools, the grabbed image will have a decrease or increase in magnification. The "grab image at" tool is located in the second from the top row to the far right and appears as a camera. The "decrease magnification and grab" tool is located at the bottom of the tools section to the left and appears as a magnifying glass with a minus sign next to it. The "increase magnification and grab" tool is located at the bottom of the tools section to the right and appears as a magnifying glass with a plus sign next to it.

The "measure distance" tool is used to measure point to point distances on a microscope image. It is located in the second from the bottom row to the right and appears as a line. An example of the "measure distance" tool being used within an image window is shown in figure 4t. The starting point of the measurement is specified by a right mouse button click and the ending point is selected by the mouse button's release.

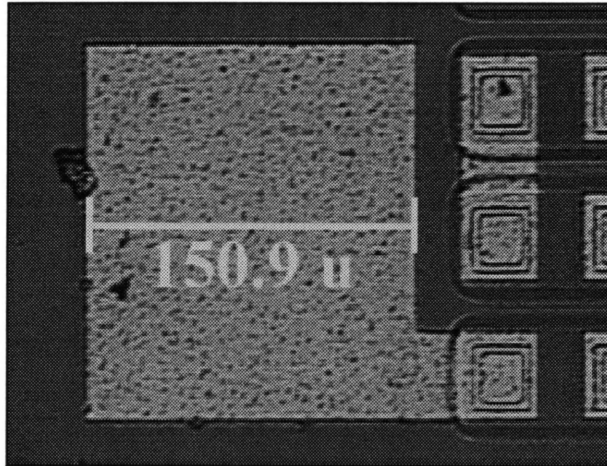


Figure 4t. Measure Distance Tool
The above image shows the use of the measure distance tool. The distance measured is specified in microns, in this case 150.9 microns.

The “rectangle” tool is the default tool for the right mouse button within an image window. If the tools window is not open it is the only usable tool. The “rectangle” tool allows the user to draw a rectangle to the image window. This rectangle will be visible on all other clients and is used primarily to point out areas on a wafer. An image of the rectangle tool in use is given in figure 4u. The rectangle tool is located in the second row from the bottom to the left and appears as a rectangle composed of dashed lines.

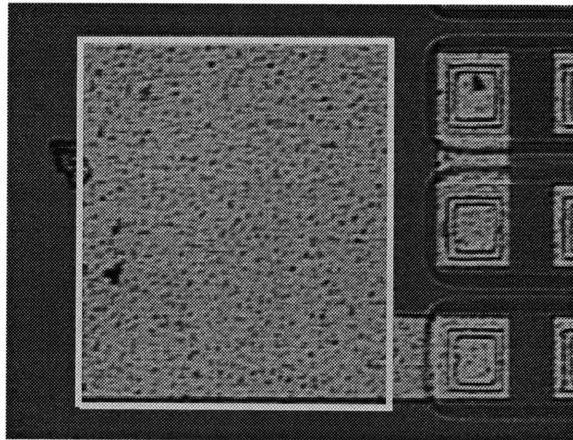


Figure 4u. Rectangle Tool
The above image shows the use of the rectangle tool. The rectangle will be displayed on all client windows.

The next section of the **RMtoolsPanel** contains two buttons, distinguished by the letters “A” and “B”. These buttons are used to specify to which windows image grabs are

to be sent. In most cases the image window that is currently being acted upon by a tool should be selected. The selected window will appear as a depressed button.

The last section contains a set of quick navigation buttons. These appear as north, south, east, and west arrows. Upon pressing any given one, an execute command will be sent to the Remote Microscope server with a position relative to the previous position plus an increment determined by the button type.

4.6.9 Manual Focusing Windows

The manual focusing windows are activated through the **RMfocusPanel** class. The buttons in the visual display of this class are only active when the client has control of the microscope and manual focus has been selected. The Remote Microscope has two manual focusing options "sigF" and "slice"

When using the "sigF" manual focus is selected the client will call up a new window through the **RMsigfManualFocus** class. This window appears as in figure 4v.

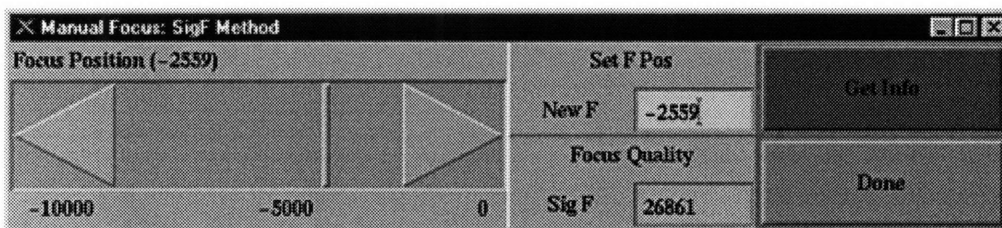


Figure 4v. SigF Manual Focus
The SigF manual focusing window allows the user to change the focus option through direct input or the slider bar. It will send a request to the server for a focus quality value at that position. This value will be displayed in the focus quality section.

The **RMsigfManualFocus** will request a focus quality value from the Remote Microscope server based on a selected focus position. The focus position can set be through direct input to the "New F" text field or by adjusting the slider bar. The main program is asked to update the focus quality value whenever the "Get Info" button is clicked or the slider bar is adjusted. The meaning of the focus quality value "sigF" is covered in Chapter 3.

The "slice" manual focus window appears as in figure4w. It is constructed through the **RMsliceManualFocus** class. This class allows the user to specify a focus position and scan parameters to obtain a graph of the corresponding pixel values.

Once activated, the **RMsliceManualFocus** class will cause a scan line to appear in the current image window. This scan line appears as a pair or horizontal red lines, with the line being scanned in-between. It is positioned at a height specified by the "Row" field in the display. The "Start" and "Stop" fields are used to set the range within this scan line to examine. When asked to retrieve new pixel values, this is the only area that need be examined.

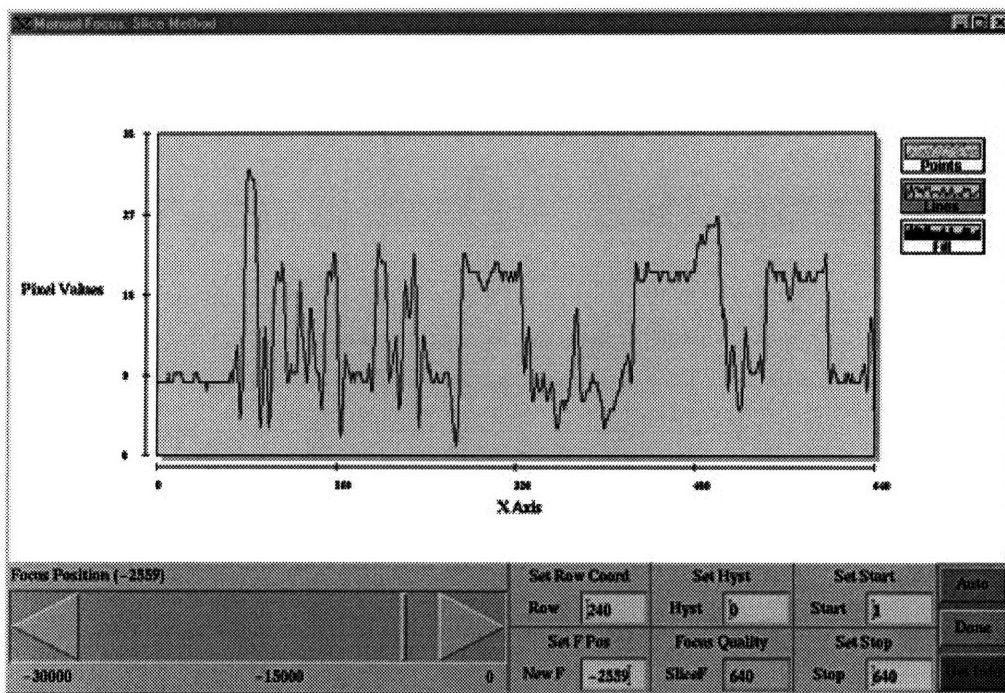


Figure 4v. Slice Manual Focus
The Slice manual focusing widow allows users to specify focus position and scan line parameters to obtain a graph of pixel values and focus quality value. User s can change the focus option through direct input or the slider bar. The scan line parameters are altered through direct input only.

By adjusting the focus position, either through direct input to the "New F" field or by using the slider bar, requests will be sent to the server for new pixel information. The request will include the scan parameters, as mentioned in the last paragraph, and a hysteresis value, which is used to eliminate small transitions.

Data obtained from the server is used to update the window. The pixel values will be displayed on the graph and a focus quality value, calculated by the server, is placed in the "SliceF" text field. The pixel values can be displayed as a series of points, lines, or as a filled graph.

The "slice" focusing method also contains an "Auto" button which is used to activate a "slice" based auto focus routine. A request is sent to the server to scan a range of focus positions. The pixel values corresponding to the focus position with the highest focus quality value.

4.7 Application or Applet

The Remote Microscope client has been designed to function as both an application and an applet. This means that the client can be run as a stand alone application on a Java Virtual Machine or through any Java enabled Web browser. This is accomplished through separate startup classes, **RMapp** and **RMapplet**, and through the use of the interface class **RMappShell**.

The startup classes are relatively small and simple. They determine the method used to parse command line arguments and transmit images to the main program. In the case of an application, **RMapp**, command line arguments are parsed by running through the parameters specified on the startup line of the program. Images are loaded through the local file system. When the program is run as an applet, **RMapplet**, the command line arguments are specified within corresponding HTML[6] startup file. The HTML code for this is,

```
<applet code="RMapplet.class" width=300 height=200>
<param name=server value="oolong.mit.edu">
<param name=port value="9997">
</applet>
```

[6] Hyper Text Markup Language: A standard format used to display information within a Web browser.

Where "name" specifies a variable name and "value" stores the corresponding data. The images within an applet are retrieved server through HTTP[7] connections.

The interface class **RMappShell** is used to allow the client to communicate with the starting application or applet. Both **RMapp** and **RMapplet** inherit from this class. The constructor, for the main program, takes in this class as one of its input parameters. The relationship between the main program and the interface is used specify how images are saved by the client, how text information is displayed, and how the program is terminated. These types of operations are very dependent on how the client was initialized, but through the use of the RMappShell it is virtually invisible to the rest of the program.

The specifications between the main program and the startup method allows the client to be used as either an application or an applet. Though, due to security issues it may be wisest to run the program as an applet. These issues are detailed in the section 5.2.1 in the next chapter.

[7] Hyper Text Transfer Protocol: Used to retrieve information and data from a Web server.

Chapter 5

Server Enhancements

This chapter describes modifications and enhancements to the Remote Microscope server. Due to the organization of the original code, most of the changes were relatively simple to make. In fact, very few changes to the server code were required at all to facilitated the new client application features. Further more, the necessary changes were implemented in a manner that did not interfere with the function of the original client.

The changes to the server fall into two main categories: changes to the Remote Microscope server code and Web server setup and configurations. The modifications to the server code include the addition of server side image saving and text based message passing. The setup and configuration for the web server involves security issues and online documentation.

5.1 Server Code Modifications

As mentioned before, only minor changes were required to the actual Remote Microscope server code. The original code is organized in a simple manner, that waits for a message from clients, determines the type of the message, and then passes relevant information to the appropriate subroutine. One draw back of the current implementation,

is that it is not multithreading. Thus it is only able to handle one message at a time. This leads to a slight problem with the modifications, which will be discussed in more detail at the end of this chapter. The server code has been altered to accept two new message types to handle server side image saving and text based communication.

5.1.2 Server Side Image Saving

The ability to save images from the Remote Microscope has always been one of its most useful functions. Up until now, these images could only be stored by saving them to disk on the client's computer. This of course could only be accomplished if the client application had access to the local file system. At the time of this writing, the security restrictions on most Java applets running inside of a Web browser made this impossible. Thus another method for saving images was required. To facilitate this, a new message was added to the server, that allowed clients to request that images be saved to the server's disk.

When a client wishes to save an image on the server, it issues a message of type "SAVEIMAGE". This message contains three parameters, "WHICH", "PATH", and "FILENAME". The "WHICH" tag contains the name of the desired window that is to be saved. The "PATH" is the directory where the image is to be saved. How this parameter is interpreted is up a predetermined relationship between the client and server. In the current server program, the "PATH" parameter stores the subdirectory in the Web server structure, where the image is to be written. The "FILENAME", as its name implies, contains the name of the file under which the image is to be saved.

Upon receiving a "SAVEIMAGE" message the server will grab the appropriate window from the "WHICH" parameter and attempt a write to the desired location specified by the "PATH" and "FILENAME". After this task is completed, the server will send a "SAVEIMAGEREPLY" message back to the client. This message contains the "WHICH" tag and a new "URL" tag. The "URL" tag will contain the Internet address of the saved image on the web server. Client's running as applets, will use the "URL"

parameter to open the saved image in a Web browser, such that it can be displayed and saved.

Though originally only intended to facilitate the saving of images, for client's that could not save to their local disk, the implementation of the "SAVEIMAGE" message has been designed in a manner to aid in future server/client enhancements. Since, the client message contains the full location information on where an image is to be saved, a small modification to the client interface could allow several images to be saved to the server's disk and even allow separate directories for each user. This could prove to be a very useful feature and would require no modifications to the current server interpretations of the "SAVEIMAGE" message.

5.1.3 Text Messaging

The Remote Microscope has a very simple text based messaging system, which allows users to send ASCII messages to one another. To send a message the client packages a message of type "CHAT" and attaches "MESSAGE" and "USER" parameters to it. The "MESSAGE" contains the ASCII message, that the client wishes to be relayed to others. The "USER" parameter stores the name of the user running the client. Once packaged the "CHAT" message is sent to the server.

The server will recognize the "CHAT" type, extract the "MESSAGE" and "USER" information, and pass the information to the appropriate subroutine. Here, the "USER" and "MESSAGE" data will be repackaged in a message of type "CHATREPLY" and sent to all the server's clients, save the one that sent the original message. It is important to note that no user name information is currently stored by the server.

5.1.4 Multithreading

It was mentioned before, that the server's current implementation does not support multithreading. In most cases, this is completely fine and reasonable, since the server

handles messages fairly quickly and only a request to grab a new image, from the microscope, causes a busy message to come into play. This busy message tells clients that the server cannot process any new requests at the present time. Before the addition of the new enhancements, there was nothing wrong with this, since only one person could send messages to the server and almost all of these messages were used to control the Remote Microscope, something that definitely can not be done during an image grab.

However, with server side image saving and a real time communications system, multiple users can send messages and it makes complete sense to be able to send messages during an image grab. So far no efforts have been made to change the nature of the current server code to facilitate this capability. This is mainly due to the fact that the new messages are the only types of messages which would require such a change and that they work completely fine, save for a minor limitation. If more real time multi-user functionality is added server multithreading is a must.

5.2 Web Server Setup and Configuration

A small subset of the new Remote Microscope features require the use of a Web server to function properly. To run the client program as a Java applet, it is required component. The type of server used is not really of great importance, since most, if not all, contain the features needed by the Remote Microscope. The Web server is used to implement security for the Remote Microscope's use, to access online documentation, and implement the displaying of server side saved images.

5.2.1 Security

With the ability to access the Remote Microscope through an Internet browser, a number of security issues come into the picture, that must be dealt with. The security issues can be resolved in one of two manners, directly in the Remote Microscope servers and clients or through a secondary element such as a Web server. The later was chosen for a variety of reasons. The first and foremost of these, concerns backward compatibility issues. While it would be possible to have the server recognize a new client and only authenticate then, this is an odd and round about method. The second reason, and

perhaps the better one, is a simple rule that one should not reinvent something that already exists. web servers have a variety of options for authentication of users, ranging from simple encoding to RSA encryption. It would be wise to use these rather than trying to re-implement them. The third and last reason is that by including security in a separate module, it can be easily configured and controlled. In fact in a secure environment they can even be ignored.

The current Web server provides security for restricting unauthorized use of the Remote Microscope on two levels. The first of these involves limiting the subnet on which the documents on the server, including the client applet itself, can be accessed. The second method authenticates a user based on a user name and password.

Limiting access based on a subnet is very appropriate for a lab setting or computers on a local area network. It allows all users on a given subnet to access the required files on the server, as long as their Internet address meets the configured settings. Currently the files can only be accessed on the 18.62.*.* subnet at MIT. Thus only a small subset of the MIT community can access the Remote Microscope HTML and Java documents found on the Web server. This is a rather heavy restrictions, but it can be easily changed to suit the needs of the desired user set.

Authentication of users, based on a user name and password pair, is a classic way to provide some security for a client/server relationship. If the correct user name and password are entered, the client is allowed to access desired files. The amount of security provided is based largely on the encryption method used to encrypt the password. The server currently has approximately telnet type security, as the user name and password are "uuencoded" This means that the they are not sent as clear text, but an interception of the correct packet could allow the message to be easily decoded.

In the future, further levels of security may be desired, but for current purposes the subnet restriction and user name/password authentication suffice. Since both of these

methods make use of the Web server, they are easily configurable based on the desired level of security. If client applications are mainly run as applets, then it is perhaps the best and most efficient way to implement security. However, if the program is distributed as a Java application, the incorporation of security parameters directly into the client should be considered.

5.2.2 Online Documentation

While not really a modification of the server, the online documentation does provide a previously unavailable and useful feature. By choosing to implement the new client in Java and use a Web browser to initiate client connections, this enhancement almost comes for free. That is to say that Web servers are primarily designed to transfer information between users on the Internet. Thus a given set of documents can easily be made available to users, through it.

All of the online documentation is written in the HTML standard and contain a few JavaScript enhancements to make them easier to navigate. The HTML, used in these files, can be used by all Web browsers. JavaScript is supported by all major browsers and for those that do not support it, the code is ignored with no loss of functionality.

5.2.2 Server Side Saved Image Display

When an image is saved on the server, through the appropriate client request and server compliance with that request, a message is sent back to the client containing an address for the image. The client can use this address to make a call to the Web server and request the desired file. Though a Web server is not necessarily required for this, it does appear to be the most usefully way to retrieve these images on the fly.

Chapter 6

Majik Viewer

The Majik Viewer is a fully functional MAGIC VLSI layout display program. As the Remote Microscope client application, it is completely written in Java and can run as an application or applet. It has the ability to dynamically parse configuration files and load files through URLs. Its user interface supports simple editing features, configurable detail switches, and a variety of navigation modes. The viewer fully supports the layout navigation protocol, described in chapter 4.6, which allow it to be used a navigation tool for the Remote Microscope. This chapter is devoted to the Majik Viewer application.

6.1 Configuration Files

The Majik Viewer dynamically parses MAGIC configuration files, in order to obtain all display and technology specific information. The current implementation of the viewer makes use of the "display styles", "color map", and "technology" files. Each of these files is presented to the Majik Viewer in the form of an open stream. The streams are passed to specific Java classes, which parse them into data structures. The Magic Viewer uses the parsed data to construct a "draw map", which contains all relevant data for mapping technology elements into their respective screen displayable forms. All of the

information contained in this section was obtained from the “Magic Maintainer’s Manual #2: The Technology File” [Scott90] and the “Magic Maintainer’s Manual #3: Display Styles, Color Maps, and Glyphs” [Mayo90].

6.1.1 Display Styles

MAGIC uses the “display styles” file to form a correlation between basic technology elements and the manner in which they are to be displayed. The class used by the Majik Viewer to parse this information is the **MAGICstyles** class. The “display styles” file has a simple format, which consists of two major subsections. The first of these contains the actual display styles, while the second contains stipple patterns that can be used to fill elements.

The display styles section of the file is organized, such that, each line contains either the complete drawing information, a comment, or a blank line. Blank lines and comments are ignored by the parser, while others are passed on to obtain drawing information. The organization of the display information for style entries is given on table 6a.

Number	Mask	Color	Outline	Fill	Stipple Number	Short Name	Long Name
Specifies the index for referencing the style	Bit plane information for layering	Color map index number	Specifies an outline for the style	Fill style, can be solid, stipple, cross, ...	Specifies a type of stipple pattern	short version of technology name	long version of technology name

Table 6a. Display Styles Format
The information for each style is given on a separate line in the file. Each line contains all of the information given above. Information is separated by spaces and irrelevant information is represented by a “-“. All of the above information is parsed by the Majik Viewer, though the Mask information is not currently being used.

The second subsection, containing the stipple information, is organized in a similar manner to that of the display styles, consisting of comments, blank spaces, and drawing information. The drawing information, in this case, contains a bit pattern that should be used to fill styles that make use of the corresponding stipple number. The display information for stipples is given on table 6b.

Number	Bit Pattern	Description
Specifies the index for referencing the stipple	The fill pattern for the specific stipple represented as a set of eight bit sets	Text description of what the fill pattern should look like.

Table 6b. Stipple Pattern Format
The information for each stipple is given on a separate line in the file. Each line contains all of the information given above. Information is separated by spaces and irrelevant information is represented by a "-".

The current implementation of the Majik Viewer does not parse the stipple information contained in the "display styles" file, as no convenient method of converting from bit patterns to the corresponding Java draw routines could be found. Thus, the Majik Viewer must make use of hard coded stipple patterns that attempt to mimic those described in this file. While, this is obviously not the correct way to do things, it does suffice to a certain degree. The underlying code has been designed in an abstract manner, such that once a method has been determined to convert bit patterns to Java draw routines, it can be easily incorporated.

6.1.2 Color Map

The "color map" file contains an index of all colors that are used by MAGIC to draw various objects to the screen. The file consists of RGB (Red, Green, and Blue) values, ranging from 0 to 255, followed by a corresponding index number, also ranging from 0 to 255, that is used to refer to the specific color. The parser reads in the various color specification lines and creates an indexed array of Java colors, that correspond to the given RGB values. The class used by the Majik Viewer to parse this information is the **MAGICcmap** class.

6.1.3 Technology File

The "technology" file forms the heart of the configuration files. It contains design rules and constraints for a given technology. This information includes the types and parameters for design elements, dimension characteristics for elements, plane layout, plane-to-plane contact specifications, modified/new display styles, and various other pieces of relevant technology data. Since the Majik Viewer is only concerned with the viewing of MAGIC VLSI layouts, it does not need to parse all of this information. Specifically, the Majik Viewer only retrieves the types of elements, the plane layout,

plane-to-plane contact information, and the modified/new display styles. The class used to parse this information is the **MAGICtech** class

The "technology" file is divided into several sections. Each section is denoted with a header tag, which contains the section's name, and an end tag, which marks the end of a section. In between these two markers, data is stored in a manner specific to the section. The Majik Viewer does several passes through this "technology" file, searching for specific header tags. Once these tags are found the corresponding data is parsed and stored. The types of data obtained from the "technology" file will be discussed briefly below and are summarized on table 6c.

Element types are used to represent various technology elements used in the construction of a VLSI layout. They are retrieved from the "technology" file by searching for the "types" header tag. Each line, after the header tag, contains a *plane* and a set of *names*. The *names* correspond to element types and the *plane* corresponds to the name of plane on which the elements are to be located.

The plane layout is used to place technology tiles at the appropriate depth in a design. This information is obtained by doing a search for the "planes" header tag. Individual planes follow the header in order of depth, with the lowest plane coming first. Each plane entry contains a full *name* for the plane, as well as possible *aliases*.

Plane-to-plane contact information tells the Majik Viewer, which element types form connections between planes. The contacts are stored after the header tag "contacts". Each contact has a *base* and a *list of component types*. The *base* is used to identify the tile type of the contact and the *list of component types* tells the viewer which components on different planes to make connections between.

Modified or new styles inform the Majik Viewer how technology elements are to be drawn to the screen. The display styles section is marked with the header tag "styles".

Each line contains a *style name* and a *style number*. The *style name* identifies a new style or a display style that is to be replaced. The *style number* is an index, that corresponds to the display styles that were obtained from the "display styles" file. *Style names* may me repeated in this section, as long as they are grouped together. *Style names* that repeat are used to form complex styles, that are constructed by combining all the *style numbers* listed.

	Header Tag	Format	Notes
element types	*types*	*plane name_1,name_2,...*	Tile types used to represent technology elements in the viewer and their corresponding plane layer.
Plane layout	*planes*	*name_1, name_2,...*	Order in which the planes are layered.
Plane-to-plane contacts	*contacts*	*base component_1 component_2 ...*	Connections made by elements between planes.
Modified/new display styles	*styles*	*name style_number*	Draw styles that should be used to draw specified elements to the screen.

Table 6c. Parsed Sections of Technology File
Summary of the technology file sections, that are parsed and used by the Majik Viewer. The header tag marks the beginning of a section, the format specifies how that data is stored in the technology file, and the notes give a brief description of the sections relevance.

6.1.4 Draw Map

The parsed data, in the form of the Java classes **MAGICstyles**, **MAGICcmap**, and **MAGICtech**, is passed to the higher level **MAGICdrawmap** class. In this class, the information, obtained from the various configuration files, is combined in a concise manner, such that technology elements can be quickly mapped to their corresponding display routines. This creates a "draw map" for all the tiles used by the Majik Viewer.

The "draw map" is indexed by element names and returns a **MAGICdrawref** class. The **MAGICdrawref** class is a drawing reference for the specified element and contains the all the display information for a tile, including its colors, stipple patterns, and plane location. Based on this information and a given position the elements can be correctly draw to the screen. In addition to containing drawing information, the "draw map" also contains data on the layering of planes and contacts between these planes.

6.2 MAGIC Files

The Majik Viewer is able to parse standard MAGIC layout files and convert them into a form that can be easily displayed and navigated. The current viewer is able to retrieve MAGIC files from any location on the Internet. This includes files that are located on FTP servers, HTTP servers, as well as the local file system. The files are presented to the viewer as a URL address and are passed to the **MAGICparser** class as an open stream that retrieves data from this address. The **MAGICparser** class, as its name implies, parses the MAGIC file information.

6.2.1 File Description

A MAGIC file contains an ASCII representation of a cell. A cell is a building block for designing a VLSI layout and describes the placement of various elements. The files contain three types of line groups, describing mask rectangles, subcell uses, and labels. Each group can appear anywhere in the file, but must be continuous.

The mask rectangles begins with the header tag "<< *layer* >>", where the *layer* is the tile name of the rectangle. Following the header tag is a list of elements of the form "*rect* *xpos_1* *ypos_1* *xpos_2* *ypos_2*". The *rect* tag simply marks the beginning of a new element, while the rest of the line describe the position of the rectangle. The position pairs (*xpos_1*, *ypos_1*), (*xpos_1*, *ypos_2*), (*xpos_2*, *ypos_1*), and (*xpos_2*, *ypos_2*) define the area covered by tile.

The subcell uses group is defined by a set of line entries. Each begins with a line of the form "*use* *filename* *name*". This line tells the viewer to *use* the subcell described in the file specified by *filename* and identify it with the given *name*. Following this line are lines describing how the cell is to be displayed within the main cell, as well as some other information. The additional lines that are parsed by the viewer are the *transform* and *box* lines.

The *transform* line describes a geometric transform from the subcell's coordinate system to that of the main cell and is formatted as "*transform* a b c d e f". The a, b, c, d, e, and f elements are used to form a transformation matrix of the following form:

a	**d**	0
b	**e**	0
c	**f**	1

The *box* line defines a bounding box for the subcell, which covers all elements within that cell. The line is formatted as "*box* *x_pos1 ypos_1 xpos_2 ypos_2*". The position pairs (*xpos_1, ypos_1*), (*xpos_1, ypos_2*), (*xpos_2, ypos_1*), and (*xpos_2, ypos_2*) define the area of the bounding box..

The end of a subcell is marked by the beginning of a new subcell or the beginning of a new group in the file.

The labels group is marked with the header tag "<< *labels* >>" and is followed by a list of labels of the form "*rlabel layer xbot ybot xtop ytop position text*". The *layer* entry corresponds to the layer to which the label is to attached. The (*xbot, ybot*) *and* (*xtop, ytop*) pairs form the lower-left and top-right corners of a placement rectangle for the label. The display position of the label is determined by placing the label at a location within this rectangle, decided by the *position* entry. The position values are corresponding locations specified by the *position* entry are as follows,

0	center
1	north
2	northeast
3	east
4	southeast
5	south
6	southwest
7	west
8	northwest

6.2.2 Internal Structures

The MAGIC file is decoded using the information given in the previous section. Each entry, regardless of its group, becomes an instance of the **MAGICelement** class. The **MAGICelement** class stores the group type, specific type within that group, label, position, transform, sub-elements, and state for a given element. The group type corresponds to one of groups given in section 6.3.1. If the element is a member of the mask rectangle group, then it also contains a sub-type. The sub-type is the specific tile type or technology element represented by the rectangle. Elements that are of the type subcell uses or label groups store a string in the label field, which is used when displaying the element on the screen. In the case of the subcell uses group, the label field is used to identify the element, while in the case of the label group, it is used to store descriptive text information. The position parameter is used by all group types. It stores the position and dimensions of the element. The transform of an element is a geometric transformation matrix, as described in section 6.3.1, under the subcell uses section. Only members of the subcell uses group make use of the transform parameters. Members of other groups simply place a null transform in this location, that has no effect on position. The subcell uses group is also the only type of element to use the sub-elements area. This section stores elements that are obtained from other MAGIC files through *use* calls. Finally, the state parameter is used to tell the Magic Viewer whether or not to display the contents of a subcell uses group.

All the **MAGICelement** classes, obtained from the parsed file, are stored in an instance of the **MAGICparser** class. Elements obtained from subsequent MAGIC files are stored in their own instance of the **MAGICparser** class. These parsed files are cached in memory, to both speed up the loading/parsing of these files and to dramatically reduce the memory require to store them.

6.3 Displaying a Layout

Once the configuration and the MAGIC layout files have been parsed, the Majik Viewer can display the layout elements. This is accomplished by running through the

elements in order of layout depth, starting with those elements located on the lowest plane. Then, within each plane layer each element is draw in an unspecified order. Labels and cells are drawn above all other layers.

To draw the elements, the Majik Viewer first retrieves a new **MAGICelement** from the **MAGICparser** class. The type of the element is extracted and compared to the viewer's list of element types. The types correspond to the three basic groups discussed earlier, rectangle, labels, and use cell elements. Each type is passed to the appropriate draw subroutine. With the current implementation, this process is actually not needed, since elements were initially separated by layer and each layer only contains elements of the same type. This check is a residue of previous implements, but has not been removed, because its structure leaves room for the incorporation of future non layer specific elements.

Rectangle elements are decomposed to retrieve their name and position. The name is used to look up the element specific draw routines in the "draw map". Each element is drawn in the manner described in the "draw map" at the location and in the dimensions specified by the position.

In the case of label elements, position and corresponding display text information are retrieved. The text is drawn at the specified position at a predetermined font style and size. The labels are not "attached" to layers, as specified in the MAGIC file, but rather exist on their own layer.

Cell elements have two display modes, depending on whether the element is "off" or "on". This information is stored within each element. Cells that are "off" are drawn by first retrieving the position and display text data from the element. The position is interpreted as the location and dimensions of the cell's bounding box. The text is its identifying name. The bounding box is drawn as a black rectangle and the text is drawn as

black text in its center. In the case of an "on" cell, the elements stored within it are recursively retrieved and drawn.

All layout elements are drawn to a secondary buffered image, such that they can be drawn quickly and displayed all at once. Various methods, such as the clipping of elements, are also employed to reduce the time required to display a layout. Once the secondary buffer image is completed, it is swapped with the old image and the elements appear on screen.

All draw routines, described above, as well as other special methods are contained within the **MAGIClayoutPanel** class. This class, also contains methods used by the Majik Viewer to change the magnification of elements, navigate through the layout, as well as many other routines to change how the layout is viewed. These methods are made available through the Majik Viewers user interface.

6.4 User Interface

The Majik Viewer's user interface appears as in figure 6a. Its exact appearance is of course dependent on the Java Virtual Machine, under which it is run. The Remote Microscope Users Manual, found in appendix C, contains details on how the Majik Viewer is used.. In this section, we will focus on what each of the elements do, rather than how they are used.

The user interface consists of seven main sections. The breakdown of these sections is given in figure 6b and have a direct correlation with figure 6a. The main sections are (1) the layout view, (2) editing tools, (3) main controls, (4) detail controls, (5) layer controls, (6) information area, and (7) about application information.

6.4.1 Layout View

The layout view is the area in which a loaded MAGIC VLSI layout will be displayed. The various surrounding sections are used to determine the manner in which

the user views and interacts with this area. Based on options and modes selected, the user will be able to navigate, magnify, and perform a variety of other functions with simple mouse clicks and movements. The basic functions available to the user will be discussed in the following sections and are summarized on table 6d. The layout view inherits from the **MAGIClayoutPanel** class, described in section 6.3.

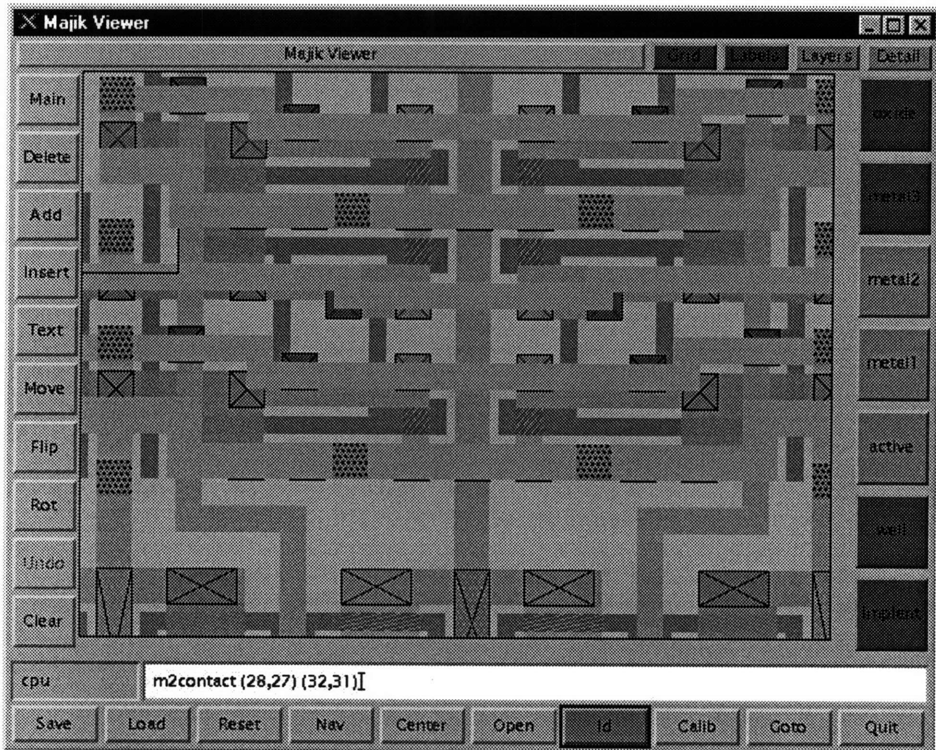


Figure 6a. Majik User Interface
This image was taken from a Majik Viewer running the standard Sun Microsystems Inc. distribution of the Java virtual machine on SunOS 5.5.1 operating system. It depicts the view of a small portion of a CPU layout obtained from a MAGIC file. Surrounding the layout are many buttons that allow the user to change display parameters, navigate layouts, load files, and even edit them.

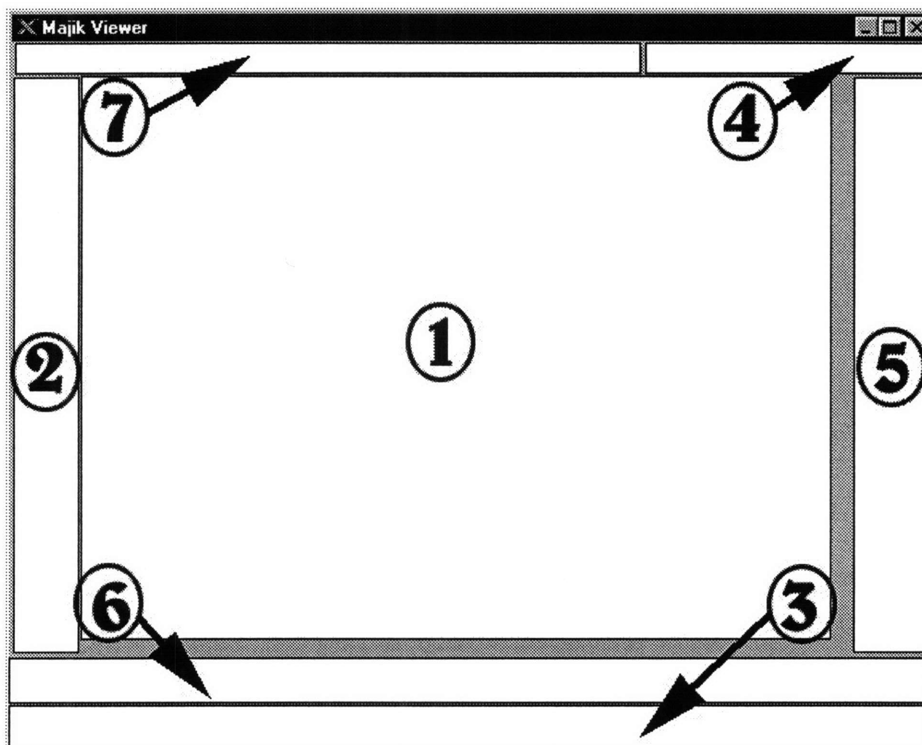


Figure 6b. User Interface Sections
This image divides up the user interface, shown in figure 6a, into functional subsections. The subsections are (1) layout view, (2) editing tools, (3) main controls, (4) detail controls, (5) layer controls, (6) information area, and (6) about program button.

6.4.2 Editing Tools

The editing tools are located to the far left of the display area. They are used to add and delete elements from a layout. The editing tools, though fully functional, do not perform any verification or rules checking for the layout. Thus, as an editor, the Majik Viewer is not very useful. These functions are chiefly included for (1) future expansion projects that are not specifically related to the Remote Microscope and (2) quick layout modification/creation for navigation purposes. Figure 6c shows a blowup image of the editing tools area of the Majik Viewer.

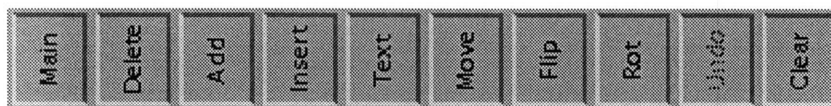


Figure 6c. Editing Tools
This image is a rotated image of the editing tools section of the Majik Viewer. Buttons here are mainly used to select various editing modes for the Viewer.

The first button is either labeled "Main" or "Cell" and specifies whether the layout view corresponded to the main or subcell view. The main layout is used for editing and

viewing the entire layout, while the subcell layout is used to specify a cell that is to be inserted into the main layout. Both layout displays can make use of all of the editing and viewing features.

The "Delete" and "Add" buttons are used to add and delete elements from the layout. When the delete button is selected the user can press any of the mouse buttons to delete the element currently under the mouse pointer. With the selection of the add button, pressing the rightmost mouse button will display a list of rectangle elements that can be added. Once a type of rectangle tile has been selected, the leftmost mouse button can be used to place and resize the element.

The "Insert" and "Text" buttons are special purpose "Add" buttons, that are used to add cells and labels to a layout. When the "insert" button is selected the user can place new cells at any position by clicking any of the mouse buttons. The type of subcell inserted is determined by the cell stored in the "Cell" layout view, discussed earlier. The "Text" button allows the user to place new labels in the layout. The string stored in the text display area (see section 6.5.6) will be inserted, as a label, at position of the mouse pointer when any of the mouse buttons are clicked.

The "Move", "Flip", and "Rot" buttons are used respectively to move/resize, flip, and rotate elements. The "Move" button allows the user to move elements with the leftmost mouse button and resize them with the rightmost. The "Flip" button will flip a cell element along an axis specified by which mouse button is used. A click of the leftmost mouse button corresponds to a horizontal flip, while the rightmost is used for vertical flips. When the "Rot" button is selected the user can rotate elements by 90 degrees clockwise with each mouse click.

The "Undo" button is activated whenever a layout is edited. By clicking it, the user will undo the last action performed. The Majik Viewer stores the last 15 to 30 editing actions performed by the user. All actions are stored as additions and deletions,

thus moving, flipping, and rotating an element take up two slots. This is the reason for the15 to 30 action range.

Lastly, at the bottom of the editing tools section, is a button labeled "Clear". Pressing this button will remove all the elements from the currently selected layout.

6.4.3 Main Controls

The main control section contains the most important set of options available to the user. There are buttons to save/load files, set navigation modes, open/close subcells, identify layout elements, interact with the Remote Microscope, and quit the application. A blowup image of this area is given in figure 6d.

Figure 6d. Main Controls
This is an image of the main control section of the Majik Viewer. Buttons here are used to save/load files, set navigation modes, open/close subcells, identify layout elements, interact with the Remote Microscope, and quit the application.

The "Load" and "Save" buttons are used to interact with the MAGIC file input/output system of the Majik Viewer. When the "Load" button is clicked it will bring up a URL dialog box, as shown in figure 6e.

Figure 6e. URL load dialog
The URL dialog is used to load new MAGIC files for display. The address of the file is placed in the top text field, in this case it is "http://dreaming.mit.edu/dm/cpu.mag". The buttons at the bottom are used to continue/cancel a load. The center text area displays error messages and status information.

Within the new window is a text field, where the user can specify the URL of the MAGIC file, that is to be loaded. The viewer attempts to open a socket connection to retrieve the file from the address. If the program is unable to make a connection to the desired file,

then the contents of the layout are left unchanged. If a connection is established, then the corresponding open stream is passed to the **MAGICparser** class. If parsed correctly the new layout will replace the old. The "Save" button only has limited functionality and only works when the program is running as an application.

The "Reset", "Nav", and "Center" buttons are used to navigate around the layout. "Reset" is used to center the view of the layout to it's base starting position and magnification. The "Nav" and "Center" buttons are used to set the navigation mode of the viewer. Both allow the user to change magnification with the rightmost mouse button and move about the layout with the leftmost. When in "Nav" mode the user can "pick up" the layout and move it around. In "Center" mode the center of the view can be redefined to the position of a mouse click.

The "open" button is used to activate the "open cell" mode of the viewer. In this mode the user can open subcells in the layout, to reveal the elements that they contain, with a left mouse click, and conceal the these elements with a right mouse click. This mode has no effect on other types of elements.

Identification mode is activated with the "Id" button. When in identification mode, the user can click on various layout elements to retrieve the tile type and exact position of the element.

The "Calib" and "Goto" buttons are used to interact with the main Remote Microscope client. The "Calib" button allows the user to specify the calibration coordinates of the Majik Viewer's layout. This is done by clicking the mouse button and dragging it, to define a rectangle. How this rectangle is used is detailed in chapter 5. Once, the viewer and the main program have been calibrated, the "Goto" button can be used to send location request coordinates to the main program. The Remote Microscope will then grab a new image, at the desired location, and send it to all the clients.

The "Quit" button's function is determined by the manner in which the Majik Viewer was started. If the program was run independent of the Remote Microscope, then the "Quit" button will close the entire application. If it was started from within the Remote Microscope, then the button merely hides the viewers window.

6.4.4 Detail Controls

The Majik Viewer's detail controls are used to modify viewing parameters of the layout. The state of each parameter is represented by a color. The range of colors is dependent of the button type. The available controls include options to turn the grid display on and off, turn the label elements on and off, enable and disable layered planes, and set the detail level of the display. An example image of the detail controls panel is given in figure 6f.

The "Grid" button is used to turn the grid display on and off. The grid display overlays an alignment grid on the layout, where each tile in the grid corresponds to the unit length 1, depending on the size of the layout. When the this button is in the on state it appears green, while in the off state it is red.

The "Labels" button simple turns the label elements on and off. In the off state, the labels do not appear on the layout at all. When they are turned on the labels will appear in their respective locations at the topmost level of the layout. The color of the states is red and green, which represent off and on respectively.

The "Layers" button also has an on and off mode, represented by the colors green and red, just as in the previous descriptions. In the off mode, all of a layouts planes are drawn to the screen, regardless of other settings. However in the on mode, the planes displayed in the layout are dependent of the layer controls.

The "Detail" button can be used to set the drawing detail of the layout tiles. It has three settings: low, medium, and high. These options are represented by the colors red,

yellow, and green respectively. In the low mode, all stipple patterns are completely ignored, since these elements take the longest to draw. The medium mode replaces stipple patterns with line based fill patterns. These line based versions can be created with standard Java draw routines and are thus have a high redraw speed with little loss of detail. While in high mode, the tile elements are drawn in complete detail, with several subroutine calls that reconstruct the stipple patterns based on some mathematical equations. This is obviously the slowest mode, but is much more visually pleasing and allows tile elements to be more easily distinguished.

Figure 6f. Detail Controls
The detail controls of the Majik Viewer appear above. In this particular example the grid display mode is turned off(red), the labels are also turned off(red), plane layering has been turned on(green), and the highest mode of display is being used(green).

6.4.5 Layer Controls

The layer controls are only active when the appropriate detail control switch has been set to on. If active, the user can use the buttons, on the layer control section, to turn certain planes on and off. When a plane is on, those elements found on that plane are drawn to the in the layout display. In the off setting, the elements on the plane are complete invisible. The on and off modes are characterized by the colors green and red, where green is on and red is off.

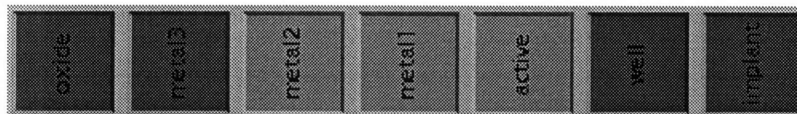


Figure 6g. Layer Controls
The image above is a rotated version of the Majik Viewers layer controls section. Each plane is represented by a button. When a plane is on the button is green and when it is off the button is red. The plane information is obtained dynamically from the technology file, in case above a SCMOS technology file was used.

Unlike the other controls on the viewer, the layer controls are generated dynamically, through information obtained from the configuration files. For each plane present in the technology file a new button is created. The label on the button corresponds

to the plane name found. Figure 6g shows a display of the controls for the SCMOS technology file.

6.4.6 Information Center

The Majik Viewer's information center has two main sections. The first of these, located to the far left, stores the name of the current file being viewed. In most cases the user will be unable to edit this name. However, when a user is trying to insert a new subcell into a layout and the viewer is set to the cell mode (see section 6.4.2), the name section is used to hold the desired cell label and can be edited. To the right of the name is a large text field. This area servers two main purposes. The first of these is relay text based information back to the user. The exact meaning of the information is dependent on the current mode and user interaction. The second use of this area is to store label information, when such an element is being added to layout. Figure 6h shows a blowup image of the information area.

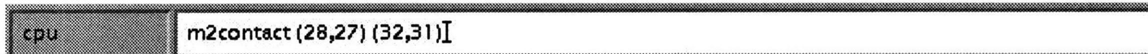


Figure 6h. Information Center
This area is used to replay, as well as, input text based information. The right section stores the name of the layout being viewed, in this case a "CPU" file. The left section is used to input labels or display text from the program, in this case it is returning the type of element selected in the viewers identification mode.

6.4.7 About Controls

The about controls are rather simple in the current implementation of the Majik Viewer. The controls consists of but one button displaying the name of the program. BY clicking on this button the viewer brings up a text window detailing the version of the program, programmer, and creating/modification date.

Chapter 7

Future Improvements

The Remote Microscope project has been a continually evolving system for the past two years. Many features and enhancements have been added, that have the project better as a whole. Future work could increase its usefulness even more. This section will examine some of the possible changes and additions to the Remote Microscope system.

7.1 Layout Navigation

The Majik Viewer is only one example of a layout tool for navigation. It is limited by the fact that it can only be used to view MAGIC files. The interface for using a layout tool, in conjunction with the Remote Microscope, is well defined and simple to implement. It would be beneficial to have multiple viewers for various layout types.

Since the layout navigation interface is coordinate system independent, it is even possible to implement a three dimensional viewer. Though default packages for displaying three dimensional structures are not included in Java, they are supported in VRML. Almost all of the latest VRML browsers support the ability to interface VRML with Java programs.

7.2 Live Video

The current Remote Microscope setup does not support a real time capture mode. This is due to the large amount of time required to move the microscope, digitizing a new image, send the image to the clients, and for clients to decoding and display the image. Overcoming some these problems could make the implementation of a live video mode possible. Since no analysis has been performed on where the actual bottleneck in this process lies, the solution to this problem can not be given here. However some general advice can be offered. By reducing the size and quality of captured images the time required to digitize, send, decode, and display an image is greatly reduced. Perhaps with such change, a real time capture mode could be supported.

7.3 Enhanced Security

Though some security has been provide through the current Web server setup, it is not enough to guarantee the integrity of the Remote Microscope system. The current setup relies on the fact that an adversary will not try to intercept packets containing a users name and password, much like a typical Telnet session. This means that an adversary with, basic knowledge of network protocols, will eventually be able to obtain a valid user name and password. If the Remote Microscope is ever to be distributed on a wide scale this is obviously not good enough.

In future versions of the Remote Microscope setup, it would be wise to include further levels of security. One way of accomplishing this goal would be to the SSL when validating user names and password. Another method would involve incorporating a password encryption scheme, such as RSA, within the actual client and server applications themselves. While the later is less versatile and more difficult to implement, it would provide security for both application and applet versions of the client, while the other can only provide security for an applet based version.

7.4 Multithreading server

The current server application only supports one thread of execute. This means that it can handle no more than a single task at any given time. Such a design was sufficient for the initial implementation of the Remote Microscope system. However, some of the new features, that were added, are limited by it. For any real time activity, the server must be able handle more than one process at a given time. By reworking the server to handle multiple threads, its functionality would be greatly increased.

7.5 Image format

The Remote Microscope currently uses GIF format to store images which are displayed by the client. GIF images have poor compression and only support 8-bits per pixel. The poor compression leads to an increased transfer time between client and server. The 8-bits per pixel limitation forces the server to quantify colors, which leads to a reduction in image quality. By using another image format, such as JPEG, it may be possible to increase rates of transfer and image quality [Lane96].

7.6 Server Re-Implementation

Though the server application modules are completely functional, they are a bit cumbersome to setup and use. It would be a good idea to rework the code and perhaps even combine the separate modules into a single package. It would also be a good idea to develop a graphical interface to allow the easy setup and startup of the server application. Support should also be added for configuration files that can used for specifying various parameters specific for a given host computer.

Perhaps the server could even be implemented in Java, thus making it platform independent as well. The only reason why such a design choice would not be a good idea is that Java does not support communication with hardware components of a system. However this lack of functionality could be compensated for through Java's ability to execute system commands. Thus hardware interaction could be supported through plug-in modules, whose operation is determined by command line arguments. Such a design

could be made to easily incorporate support for additional video capture boards and automated microscope controllers.

Chapter 8

Conclusion

The Remote Microscope is a truly useful tool for distributed design. It allows multiple people to collaboratively use a fully automated remote microscope, from any location on the Internet. Researchers and designers can examine integrated circuits as a group, even when members are located in separate countries, scattered throughout the world.

Over the past three years, the Remote Microscope has been a continually evolving and fast moving project. Many people have contributed to it and have made it a great system as a whole. In the beginning it was only idea, then Kao turned it into reality. Only two years ago, an attendant was required to move and focus the microscope. Kittipiyakul changed all that, by making it fully automated and adding auto focus capabilities.

With this ground work set, the project began to focus on how users could interact with the system. Lee's work added advanced manual focusing options and work performed for this thesis added enhanced user interface capabilities. The Remote Microscope has become a truly versatile application.

The client program, for controlling the microscope, is now written in Java, allowing it to run on almost any computer platform and even from within an ordinary Web browser. It boasts advanced features, only available through a computer enhanced interface. These features include the ability to capture/save images from the microscope, perform rough measurements, and even use a MAGIC VLSI layout as a means of navigation.

Even though the Remote Microscope is fully functional and very useful, future work will surely add to and improve upon the existing structure. As for now, inspection, communication, and collaboration become woven into one, through a single tool. The inspection of an integrated circuit has become just a mouse click away, communication between designers can be performed through a Web browser, and collaboration has been taken to a whole new level.

References

[CARNEY95] Carney, J. "*Message Passing Tools for Software Integration,*" S.M. Thesis, Department of Electrical Engineering and Computer Science Massachusetts Institute of Technology. Cambridge, Massachusetts USA. September 1995.

[CORNELL96] Cornell, Gary and Cay S. Horstmann. **Core Java**. The Sunsoft Press. Mountain View, California. 1996.

[Hamano96] Hamano, Tadashi. "*UnZip & GifImage*". *http://phantom.ghost.linc.or.jp/~hamano/java/GifImage_UnZip/GifImage_UnZip.html.* March 16, 1996.

[Kao95] Kao, James. "*Remote Microscope for Inspection of Integrated Circuits,*" S.M. Thesis, Department of Electrical Engineering and Computer Science Massachusetts Institute of Technology. Cambridge, Massachusetts USA. September 1995.

[Somsak96] Kittipiyakul, Somsak. "*Automated Remote Microscope for Inspection of Integrated Circuits,*" S.M. Thesis, Department of Electrical Engineering and Computer Science Massachusetts Institute of Technology. Cambridge, Massachusetts USA. September 6, 1996.

[Lane96] Lane, Tom ed. "*JPEG image compression FAQ*". *http://www.cis.ohio-state.edu/hypertext/faq/usenet/jpeg-faq/top.html.* November 1996.

[Lee96] Lee, Brian. "*Remote Microscope User's Manual*". Cambridge, Massachusetts USA. November 1996.

[Mayo90] Robert N. Mayo and John Ousterhout. "*Magic Maintainer's Manual #3: Display Styles, Color Maps, and Glyphs*". Berkeley, California USA. September 19, 1990.

[Scott90] Scott, Walter and John Ousterhout. "*Magic Maintainer's Manual #2: The Technology File*". Livermore, California USA. September 19, 1990.

Appendix A

Client Code Overview

This appendix contains a complete overview of the Java client code for the Remote Microscope. The code is broken into discrete packages that handle various tasks for the client. Each subsection of this appendix will detail the methods for each class in a package and give a brief discussion on how the package is used. Classes not developed specifically for the Remote Microscope will be mentioned, but not full discussed.

This appendix does not cover the classes used in the Majik Viewer. The classes for this package will be covered in appendix B. The packages covered in this appendix are given on the following page.

Remote Microscope Client Packages

A.1 THE NETWORK PACKAGE (*RMNET*).. 92

A.1.1 NETWORK PACKAGE CLASSES... 94
A.1.2 NETWORK PACKAGE USE.. 101

A.2 THE GRAPHICS PACKAGE (*GUI*) 104

A.2.1 GRAPHICS PACKAGE CLASSES... 106
A.2.2 GRAPHICS PACKAGE USE.. 149

A.3 THE GIFIMAGE PACKAGE.. 152

A.4 THE APPLICATION PACKAGE 154

A.4.1 APPLICATION PACKAGE CLASSES.. 156
A.4.2 APPLICATION PACKAGE USE... 166

A.1 The Network Package (*rmnet*)

The network package handles all interactions between the Remote Microscope server and client applications. It operates through a well defined protocol, to allow clients to both send and receive various message types from the server. As long as this protocol is obeyed, the higher levels of the client program can ignore the actual working of the networking methods involved. The main functionality made available to the client program is the ability to send new messages and receive information from the server.

A.1.1 Network Package Classes

Remote Microscope Client *(RMclient.java)* **version** 0.5

This is the primary functional layer of the network client package. It handles interactions between received messages and the main application or applet window. It also handles the sending of messages when told to do so by the application or applet window.

public **RMclient**(String *host*, int *port*, RMclientShell *shell*);
This method is used to construct a new client network layer. The *host* parameter specified the DNS name of the server and the *port* number specifies the desired connection port. The *shell* parameter is used to inform the **RMclient** of which part of the main program to send received messages to the server to.

public void **begin**() ;
Starts the network client. This enables its ability to receive messages from the server and send though requested.

public boolean **initialized**();
This function returns true if and only if the a successful connection was made to the specified server. If the connection failed, false is returned and an error message can retrieved using **initError**.

public Exception **initError**();
Returns a Java Exception based on a failed connection initialization.

public void **asciiSend**(MsgObject *m*);
Requests that an ASCII message, packaged as the MsgObject *m*, be sent to the server.

public void **killClient**();
Terminates the network connection with the server and halts all internal threads.

Remote Microscope Client Shell *(RMclientShell.java)* **version** 0.8

This is an interface shell to allow the network layer to communicate with other portions of the Remote Microscope client.

public void **command**(int *com*, String *params*[]);

Executes a command within the interface. The command, *com*, be any on of the following:

clearUserList	appendUserList	makeControlPassive
makeControlActive	makeControlNone	setScopeState
zoomGrabReply	grabFrameReply	curPosReply
rectangleReply	magReply	busyReply
saveImageReply	chatReply	loadNewWaferReply
getSliceReply	getSigfReply	

The interpretation of the *params* array is based on the specific type of message. The meaning of these parameters types is given in the main program section.

public void **bytesTransfer**(int *com*, byte *b*[]);

Sends an array of bytes through the interface. The command, *com*, specifies the type of command that is requesting a bytes transfer and the *bytes* parameter is the actual data.

public boolean **isBusy**();

Returns true if the interface is currently executing a command or waiting for s server reply. If this is not the case this method returns false.

Abstract Message Object *(MsgObject.java)* **version NA**

This class created by Eugene Hong. It is a port John Carney's object routines to Java. It specifies routines encoding and decoding abstract objects, that can contain any variety of data types. The original version of this abstract object type were used in the first Remote Microscope. It is used here for consistency. A few slight changes were required to Eugene's code to work well with the Remote Microscope project. Specifically, the manner in which byte arrays are decoded needed to be changed. Originally they were decoded byte by byte, which was very slow for the image data structures. The new code appears below.

public void **decBinary**(InputStream *bis*) throws MsgObjectException

Decodes object from binary buffer. The bis parameter specifies the input stream containing the abstract object.

```
...
case BYTE:
    //Original Code
    //************************************
    //value = new short[s.length()];
    //for (i = 0; i < s.length(); i++)
    //  ((short[]) value)[i] = (short) is.readUnsignedByte ();
    //New Code by Manuel Perez
    value = new byte[s.length()];
    byte value2[] = new byte[s.length()];
    is.readFully(value2,0,s.length());
    s.type(SBYTE);
    value = (Object) value2;
    break;
...
```

Remote Microscope Message Handler *(RMmessageHandler.java)* **version** 1.0

This class has the responsibility of decoding the CHAT messages received from the server into command and parameters that can be execute within a **RMclientShell**.

public **RMmessageHandler**(RMclientShell *client*);

Constructs a new message handler object. The client parameter specifies on where decoded commands should be executed on.

public void **handle**(MsgObject *m*);

Decodes the message *m*, repackages the appropriate data into a **RMclientShell** command and parameters array, and then attempts to execute that command on the **RMclientShell,** specified during the construction of this class.

Remote Microscope Send *(RMsend.java)* **version 0.2**

Handles messages sent from the client to the server. The main client class, **RMclient**, will forward send requests from the main program to this class.

public **RMsend**(Socket *host_socket*) throws IOException;

Constructs a new RMsend instance. The *host_socket* parameter specifies a pre-opened socket connection to the server. An exception is thrown if the a socket based error occurs.

public void **asciiSend**(MsgObject *m*);

Sends a new message, *m*, to the server.

public void **close**();

Closes the sending socket connection.

Remote Microscope Receive *(RMreceive.java)* **version 0.5**

This class inherits from the Thread Java class. It is a separate process that waits for inputs from the server. Upon receiving a message it pass it a message handler object for decoding and dispatch.

public **RMreceive**(Socket h_*soc*, RMmessageHandler *mh*) throws IOException {
Constructs a new object to receive server messages. The *soc* parameter specifies a previously opened socket connection. The message handler used for decode and dispatching messages is given in parameter *mh*.

public void **close**();
Closes the Receiving socket connection.

A.1.2 Network Package Use

The main application class, **RMmainFrame**, inherits from **RMclientShell**. When the Remote Microscope client is initialized, **RMmainFrame** constructs a new **RMclient**, with a specified host, port, and itself as input parameters (1). Internally the **RMclient** is stored for sending purposes (2). The client will then attempt to connect to the specifies server and store status information. If all is successful (3), the **RMmainFrame** will then start the **RMclient**'s send and receive protocols (4). Below is a code snippet of this process, where a has already been defined as an instance of **RMmainFrame**

```
...
/*start the client*/
RMclient c = new RMclient(server, port, a);                          (1)
a.setClient(c);                                                      (2)
if(c.initialized())                                                  (3)
{
        c.begin();                                                   (4)
        shell.outputMessage("Client connected to server.");
} else shell.outputMessage("Client failed to connect to server.");
...
```

Internal to **RMclient**, **RMsend**, **RMreceive**, and **RMmessageHandler** classes will be created and initialized. When messages are received by **RMreceive**, the **RMmessageHandler** will identify the type (1), decode parameters (2), and execute commands (3) on the **RMmainFrame** through the **RMclientShell** interface. If applicable byte information is transferred as well (4). An example of this is given below:

```
...
else if(command.equals("GRABFRAMEREPLY"))                            (1)
{
        params = new String[7];
        params[0] = m.slotValAscii("XPOS");                          (2)
        params[1] = m.slotValAscii("YPOS");                          ...
        params[2] = m.slotValAscii("FPOS");                          ...
        params[3] = m.slotValAscii("MAG");                           ...
        params[4] = m.slotValAscii("FOCUS");                         ...
        params[5] = m.slotValAscii("TIME");                          ...
        params[6] = m.slotValAscii("CURRENT");                       (2)
        myclient.command(RMclientShell.grabFrameReply, params);
        (3)
        myclient.bytesTransfer(RMclientShell.grabFrameReply,         (4)
                        m.slotValSByte("IMAGE"));
}
```

When the **RMmainFrame** classes wishes to send a message to the server, it must create an instance of an **MSGobject** (1), places parameters within it (2), and then pass this object to the send routine (3) of the **RMclient**.

```
...
MsgObject sendMsg = new MsgObject("basic message");                    (1)
sendMsg.newSlot("COMMAND", MsgObject.ASCII, "SAVEIMAGE");
        (2)
...
sendMsg.newSlot("FILENAME", MsgObject.ASCII, "windowA.gif");           (2)
...
client.asciiSend(sendMsg);                                             (3)
...
```

The **RMclient** will then handle the actually sending of the message to the server.

A.2 The Graphics Package (*gui*)

The graphics package contains all the basic graphical user interface elements used by the Remote Microscope client. The graphical elements provided, range from specially designed buttons to self contained windows. This package does not include the main application window, though it does provide all of the elements required for its construction.

Many of the Graphical elements are used to interact with the rest of the program. This interaction takes place on two levels. The first of these involves the graphic element formatting the data and passing it to the appropriate client defined method. The second involves the main client application grabbing event signals from the graphical interface and handling them on its own.

In addition to handling user inputs, some of the graphic elements are used to store various portions of data obtained from the user and the server. Methods are provides such that data input and output from these elements is independent of the visual display. Thus from the point of view of the main program, graphic elements appear as data structures. As long as the proper method is called, each graphic element is responsible for storing and displaying data input and returning the correct output.

A.2.1 Graphics Package Classes

Remote Microscope GUI Shell *(RMguiShell.java)* **version 1.4**

This class is an interface to allow graphical elements to access information from the main program, as well as execute specified commands.

public void **setCursor**(int *win*, int *x*, int *y*);
Informs the interface of a cursor position change within a graphical window. The *win* parameter specifies the window in which the cursor has changed position. The position is specified with the *x* and *y* parameters.

public void **startRectangle**(int *win*);
Signals the beginning of a rectangle within the window specified by the *win* parameter.

public void **endRectangle**(int *win*, int *x1*, int *y1*, int *x2*, int *y2*);
Signals the end of a rectangle within the window specified by the *win* parameter. The (*x1*,*y1*) pair is the top right corner of the rectangle and the (*x2*,*y2*) pair is the bottom left corner.

public boolean **hasControl**();
Returns true if the interface currently has control of the Remote Microscope.

public void **loadWafer**();
Tells the interface to execute a load wafer command.

public void **calibrate**();
Tells the interface to execute calibration command for a new wafer.

public void **nocalibrate**();
Tells the interface not to execute a calibration command for a new wafer.

public void **saveImage**(int *window*, String *pathname*, String *filename*);
Executes a save image command within the interface. The *window* parameter specifies which window to save from. The *pathname* and *filename* give the location where the image should be saved.

public void **activateManualFocus**();
Tells the interface to activate the manual focusing options.

public void **deactivateManualFocus**();
Tells the interface to deactivate the manual focusing options.

public void **getSlice**(String *size*, String *fpos*, String *rowpos*,
String *hystval*, String *start*, String *stop*, String *fopt*);
Executes a get slice information request within the interface. The *size* parameter species the horizontal size of the image. The *fpos* parameter species the focus position of the grab. *Start* and *stop* determine the range within the *size* to scan. Hystval is determines

noise reduction. The *fopt* can be either "MANUAL" or "AUTO" and determines the type of focusing performed by the server.

public void **activateScanline**(int *y*);
Tell the interface to activate a scan line within the current grab window at the species y position.

public void **deactivateScanline**();
tell the interface to turn off the scan line within the image windows.

public void **getSigf**(String *fpos*);
Executes a get sigF data request within interface. The desired focus position is specified in the *fpos* parameter.

public void **sendChatMessage**(String *msg*);
Requests that the message *msg* be sent to the server through the interface.

public void **setWindow**(int *win*);
Tells the interface to change its current grab window that specified by the *win* parameter.

public int **getWindow**();
Returns the current grab window for the interface.

public void **winChange**();
Informs the interface that the selection window has changed.

public void **paramExecute**(boolean *chgX*, boolean *chgY*, boolean *chgF*,
boolean *chgMag*, int *x*, int *y*, int f, int *mag*);
Executes a grab command, within the interface, with parameters different form a normal image grab execution. The parameter *chgX* tells the interface if the x position is different from normal. If it is the value of *x* is used as the x position. The parameter *chgY* tells the interface if the y position is different from normal. If it is the value of *y* is used as the y position. The parameter *chgF* tells the interface if the f position is different from normal. If it is the value of *f* is used as the f position. The parameter *chgMag* tells the interface if the magnification is different from normal. If it is the value of *mag* is used as the magnification value.

public int **getTool**();
Returns the current tool selected by the interface. The list of tools will be detailed in the tools panel section.

public RMviewFrame **getViewWindow**();
Returns the current viewing window selected by the interface.

public boolean **isBusy**();
Return true if the interface is currently busy, i.e. waiting for a response from the Remote Microscope server.

public void **layoutGO**(int *pos*[]);
Executes a request to the layout, through the interface, to move to the position specified by the *pos* parameter.

public void **layoutNAVset**(int *pos*[]);
Tells the interface of the calibration coordinates obtained from a graphical object.

Remote Microscope Title Panel *(RMtitlePanel.java)* **version** 0.3

This class inherits from the Canvas Java AWT class. It contains the title image for the Remote Microscope, which displays the name of the application and version information. It is also used to display a busy image when the server is busy executing a command

public **RMtitlePanel**(Image *title_img*, Image *busy_img*);
This method constructs a new title panel. The inputs specify the corresponding images that are to be displayed, when this class is in a certain state. The *title_img* parameter is the title image and the *busy_img* parameter is the image to be displayed when the server is busy.

public void **draw**();
This method updates the buffered image stored internally to the RMtitlePanel. This method, though public, is primarily used internally to the this class.

public void **doTitle**();
Sets the image to be displayed to the title image specified during construction and repaints the panel.

public void **doBusy**();
Sets the image to be displayed to the busy image specified during construction and repaints the panel.

Remote Microscope Chat Panel *(RMchatPanel.java)* **version** 0.2

This class inherits from the Panel Java AWT class. It contains a large text area for display received messages, small text field for messages that are to be sent, and a "send" button". The "send" button or hitting the "ENTER" key will cause a message to be sent. The special tag "\name *<name>*", where *<name>,* specifies a users name, is supported. The default name is "rmuser", Remote Microscope User.

public **RMchatPanel(**RMguiShell *guishell*);
This method constructs a new chat panel. The input parameter, *guishell*, specifies who "sent" messages will be passed to.

public void **deleteAll**();
Clears the received test area.

public void **appendMsg(**String *msg*);
Appends a new message to those displayed in the received text area.

public String **getName**();
Returns a locally stored name of the user.

Remote Microscope Tools Panel *(RMtoolsPanel.java)* **version** 0.3

This class inherits from the Panel Java AWT class. It displays a set of user interface tools. The panel is used to select tools, that determine the functions of other classes, and execute commands through a RMguiShell. The **RMtoolsPanel** class uses an image to represent the tools and shades selected buttons within through Java's XOR paint function.

public **RMtoolsPanel(**RMguiShell *guishell*, Image *tools*);
Constructs a new tools panel. The *guishell* parameter is used to specify the graphical interface in which the certain commands are executed. The *tools* parameter determines the appearance of the tools in this panel.

public void *draw*();
Draw the tools image to an internal buffer. Selected tools are represented as depressed buttons. This effect is accomplished through the use of Java's XOR paint function to shade them white.

public int **getTool**();
Returns the currently selected tool. The available tools are:

grab	grab a new image
rect	rectangle
measure	measure distance
zoomin	increase magnification and then grab image
zoomout	decrease magnification and then grab image
calib	calibrate wafer for layout navigation
go	move layout navigation to position

public void **setTool(**int *t*);
Sets the current tool to the tool specified by the parameter *t*. The available tools are detailed in the **getTool** method.

private void **drawDownButton**(Graphics *g*, int *x*, int *y*, *int x2*, *int y2*)
Private method for drawing the depressed or down version of a button specified by the position parameters (*x,y*) and (*x2,y2*). The *g* parameter determines the graphic context to where the buttons are drawn.

Remote Microscope Tools Frame *(RMtoolsFrame.java)* **version** 0.2

This class inherits from the Frame Java AWT class. It is a window element, which is used to display a **RMtoolsPanel** instance.

public **RMtoolsFrame**(RMguiShell *shell*, Image *back*) {
Constructs a new tools frame. The *guishell* parameter is used to specify the graphical interface in which the certain commands are executed. The *tools* parameter determines the appearance of the tools panel.

public void *draw*();
Calls the draw function within the **RMtoolsPanel** class.

public int **getTool**();
Returns the currently selected tool of the **RMtoolsPanel**. The available tools are detailed in the **getTool** method of the **RMtoolsPanel** class.

public void **setTool**(int *t*);
Sets the current tool to the tool the RMtoolsPanel specified by the parameter *t*. The available tools are detailed in the **getTool** method of the **RMtoolsPanel** class.

Remote Microscope Control Panel (*RMcontrolPanel.java*) version 0.5

This class inherits from the Panel Java AWT class. It contains a set of options to control the Remote Microscope and portions other portions of the user interface. The control panel class contains instances of the **RMwindowSelectPanel, RMfocusSelectPanel, RMimageQualityPanel, RMsavePanel, RMtakeCedePanel**, and **RMexecutePanel** classes. The functions of these classes are defined separately. The control panel supports the ability to return the individual instances for information retrieval. In addition to these element two buttons, "Tools" and "Layout Navigation", are also contained in this class. These buttons execute commands through the **RMguiShell** provide during construction.

public **RMcontrolPanel(**String *windows*[], String *focTypes*[],
RMguiShell *guishell*);

Constructs a new control panel. The *windows* parameter is an array of window names that can be selected from within the control panel. The *focTypes* is an array of focusing methods that cab be selected. The *guishell* is the interface through which the control panel may execute commands and retrieve data.

private void **add(**Component *c*, GridBagLayout *gbl*, int *x*, int *y*, int *w*, int *h*);

This is a private method used to add components to the graphical layout of the control panel class.

public void **disableAll**();

Disables all of the user input options within this panel.

public void **enableAll**();

Enables all the user input options within this panel.

public RMwindowSelectPanel **getWinPanel**();

Returns the window selection portion of the control panel.

public RMfocusSelectPanel **getFocPanel**();

Returns the focus selection portion of the control panel.

public RMimageQualityPanel **getQualityPanel**();

Returns the image quality portion of the control panel.

public RMexecutePanel **getExecPanel**();

Returns the execute button portion of the control panel.

public RMtakeCedePanel **getTakeCedePanel**();

Returns the take/cede portion of the control panel.

public RMsavePanel **getSavePanel**();

Returns the image saving portion of the control panel.

Remote Microscope Take/Cede Panel *(RMtakeCedePanel.java)* **version 0.3**

This class inherits from the Panel Java AWT class. It contains two buttons, labeled "Take" and "Cede". These buttons may be enabled and disable through this class. The event signals generated by these buttons are not handled locally.

public **RMtakeCedePanel**();
Constructs a new Take/Cede Panel.

public void **disableAll**();
Disables the "Take" and "Cede" buttons. No events are generated by either.

public void **enableAll**();
Enables the "Take" and "Cede" buttons. Allows events to be generated by them.

public void **enableTake**();
Enabled only the "Take" button, such that it can generate button press events.

public void **enableCede**();
Enabled only the "Cede" button, such that it can generate button press events.

public void **disableTake**();
Disabled only the "Take" button, such that it can not generate button press events.

public void **disableCede**();
Disabled only the "Cede" button, such that it can not generate button press events.

Remote Microscope Slice Manual Focus *(RMsliceManualFocus.java)* **version** 0.7

This class inherits from the Frame Java AWT class. It is composed of the **RMplotPanel** and **RMbottomSlicePanel** classes.

public **RMsliceManualFocus(**Image *fUp*, Image *fDown*, Image *lUp*, Image *lDown*, Image *pUp*, Image *pDown*, RMguiShell *guishell*);

Constructs a new slice manual focus window. The *fUp* and *fDown* parameters contain the fill button images, in its up and down state respectively. The *lUp* and *lDown* parameters contain the line button images, in its up and down state respectively. The *pUp* and *pDown* parameters contain the points button images, in its up and down state respectively. The *guishell* parameter provides an interface through which commands can be executed.

public int **getValue(**String *val*);

Returns the value of an internal variable specified by the tag *val*. The *val* tag can be any of the following: "*rowcoord*", *hyst*", "*start*", "*stop*", "*focusquality*", "*fpos*".

public String **getValueString(**String *val*);

Returns the value of an internal variable specified by the tag *val*. The *val* tag can be any of the following: "*rowcoord*", *hyst*", "*start*", "*stop*", "*focusquality*", "*fpos*".

public void **setValue(**String *val*, int *value*);

Sets the value of an internal variable, specified by the tag *val*, to *value*. The *val* tag can be any of the following: "*rowcoord*", *hyst*", "*start*", "*stop*", "*focusquality*", "*fpos*".

public void **setValue(**String *val*, String *value*);

Sets the value of an internal variable, specified by the tag *val*, to *value*. The *val* tag can be any of the following: "*rowcoord*", *hyst*", "*start*", "*stop*", "*focusquality*", "*fpos*".

public void **setPixels(**int *pixels*[]);

Sets the values of the pixel value for the visual graph and redraws in accordingly.

public void **sendRequest(**String *fpos*);

Uses the **RMguiShell** to execute a **getSlice** command, with the focus position specified by the *fpos* parameter and local variables specifying other parameters.

Remote Microscope Magnification Panel *(RMmagPanel.java)* **version** 0.3

This class inherits from the Panel Java AWT class. It contains a set of checkbox elements that are used to display and set the magnification for the Remote Microscope

public **RMmagPanel**(String *tag*);

Constructs a new magnification panel. The *tag* parameter specifies the name of the label that accompanies the panel. The label will be of the for "<*tag*> magnification".

public void **disableAll**();

Disables the checkbox buttons. No events are generated by them.

public void **enableAll**();

Enables the checkbox buttons. Allows events to be generated by them.

public int **getCurrent**();

Returns the current magnification specified by the checkboxes. The possible magnification values are 5, 10, 20, and 50.

public String **getCurrentString**();

Returns the current magnification specified by the checkboxes. The possible magnification values are "5", "10", "20", and "50".

public void **setCurrent**(int *x*) {

Sets the current magnification and updates the checkboxes accordingly. The possible magnification settings, specified by the *x* parameter, are "5", "10", "20", and "50".

public void **setCurrent**(String *x*) {

Sets the current magnification and updates the checkboxes accordingly. The possible magnification settings, specified by the *x* parameter, are "5", "10", "20", and "50".

public boolean **magChange**();

Returns true if the magnification has changed since the last image capture request.

private Checkbox **makeCheckBox**(String *label*, CheckboxGroup *group*,
boolean *setting*);

Private method used to aid in the creation of the checkbox elements. The *label* parameter determined the attached label for the checkbox. The *group* parameter determined the group to which the check box belongs. The *setting* parameter specifies the state of the checkbox, true is equivalent to checked and false signifies not checked.

Remote Microscope Load Wafer Focus *(RMloadWaferFrame.java)* **version 0.3**
This class inherits from the Frame Java AWT class. It creates a window, which is used for specifying the parameters for a new wafer.

public **RMloadWaferFrame(**RMguiShell *guishell*);
Constructs a new load wafer class. The *guishell* parameter provides an interface through which commands can be executed.

public int **getXsize**();
Returns the X size of a wafer, in cm, specified through interface

public int **getYsize**();
Returns the Y size of a wafer, in cm, specified through interface

public int **getThickness**();
Returns the thickness of a wafer, in microns, specified through interface

public String **getXsizeString**();
Returns the X size of a wafer, in cm, specified through interface

public String **getYsizeString**();
Returns the Y size of a wafer, in cm, specified through interface

public String **getThicknessString**();
Returns the thickness of a wafer, in microns, specified through interface

public String **getFocusType**();
Returns the type of the wafer specified through interface

private Checkbox **makeCheckBox(**String *label*, CheckboxGroup *group*,
boolean *setting*);
Private method used to aid in the creation of the checkbox elements. The *label* parameter determined the attached label for the checkbox. The *group* parameter determined the group to which the check box belongs. The *setting* parameter specifies the state of the checkbox, true is equivalent to checked and false signifies not checked.

private Panel **makeValuePanel(**String *label*, String *units*, TextField *field*);
Private method used to aid in the creation of labeled value fields. The *label* parameter determined the attached label for the panel. The *units* parameter specifies the corresponding units of the value field. The *field* parameters the area where user inputs are taken.

Remote Microscope Plot Panel *(RMplotPanel.java)* **version 0.2**

This class inherits from the Panel Java AWT class. The panel contains a plot pixel values vs. Horizontal, x, distance. The graph can be displayed as a set of points, a set of lines, or as a filled polygon. Buttons to choose between these three options are draw next to the plot. The panel handles the changing of these viewing options.

public **RMplotPanel**(int *imageX*, int *imageY*, int *pixelUp*, int *xAxisUp*,
int *pixelSpace*, int *xAxisSpace*, Image *fUp*, Image *fDown*,
Image *lUp*, Image *lDown*, Image *pUp*, Image *pDown*);

Constructs a new plot panel. The *imageX* and *imageY* parameters specify the offset of the image within the panel. The *pixelUp* and *xAxisUp* parameters determine the offset of the corresponding axis for each. The *pixelSpace* and *xAxisSpace* parameters are used to determine the spacing between points on the graph. The *fUp* and *fDown* parameters contain the fill button images, in its up and down state respectively. The *lUp* and *lDown* parameters contain the line button images, in its up and down state respectively. The *pUp* and *pDown* parameters contain the points button images, in its up and down state respectively. The *guishell* parameter provides an interface through which commands can be executed.

public void **draw**();
Draw the graphical plot to a buffered image.

public void **setPixels**(int *pixels*[]);
Sets the values of the pixel value for the visual graph and redraws in accordingly.

private void **drawPixelMarker**(Graphics *g*, int *x*);
Private method, called by the **draw** method, to draw the tick markers on the pixel axis. The *g* parameter species the graphical context to draw to. The *x* parameter specifies the distance along the axis to draw the tick mark.

private void **drawXAxisMarker**(Graphics *g*, int *x*);
Private method, called by the **draw** method, to draw the tick markers on the X axis. The *g* parameter species the graphical context to draw to. The *x* parameter specifies the distance along the axis to draw the tick mark.

private void **plotPoints**(Graphics *g*);
Private method, called by the **draw** method, to draw the graph as a set of points to the graphical context specified by the *g* parameter.

private void **plotLines**(Graphics *g*);
Private method, called by the **draw** method, to draw the graph as a set of lines to the graphical context specified by the *g* parameter.

private void **plotFill**(Graphics *g*);
Private method, called by the **draw** method, to draw the graph as a filled polygon to the graphical context specified by the *g* parameter.

private void **drawButtons**(Graphics *g*);
Private method Private method, called by the **draw** method, to draw the graph display option buttons to the graphical context specified by the *g* parameter.

Remote Microscope Focus Slide Bar *(RMfocusSlideBar.java)* **version** 0.1

This class inherits from the Panel Java AWT class. It contains a scrollbar, which is used to change a focus parameter. When the scrollbar is moved, the panel updates its internal state and passes the event information to container to which the panel belongs

public **RMfocusSlideBar**(String *title*, int *upper*, int *lower*);
Constructs a new focus slide bar instance. The *title* parameter is used to create a title label for the panel. The *upper* and *lower* parameters are used to define the upper and lower focus position values.

public int **getValue**();
Returns the current focus position selected.

public String **getValueString**();
Returns the current focus position selected.

public void **setValue**(int *f*);
Sets the current focus value for the panel. This cause an update of both the scrollbar and local display of the focus value.

public void **setValue**(String *f*);
Sets the current focus value for the panel. This cause an update of both the scrollbar and local display of the focus value.

Remote Microscope Settings Panel *(RMsettingsPanel.java)* **version 0.3**

This class inherits from the Panel Java AWT class. It contains the primary settings for the Remote Microscope. It is composed of the **RMmagPanel** and **RMpositionPanel** classes.

public **RMsettingsPanel**(String *labelA*, String *labelB*, Color *c*);
Constructs a new settings panel. The *labelA* and *labelB* parameters determine the labels for the magnification and position panels respectively. The *c* parameter specifies the color of the text fields in the position panel,

public void **disableAll**();
Disables the edit ability for entries in this panel.

public void **enableAll**();
Enables the edit ability for entries in this panel.

public RMmagPanel **getMag**();
Return the magnification panel contained by this panel.

public RMpositionPanel **getPos**();
Returns the position panel contained by this panel.

public void **setAll**(int *x*, int *y*, int *f*, int *m*);
Sets the position and magnification values for the sub panels within this panel. The *x*, *y*, and *f* parameters specify the corresponding positions. The *m* parameter determines the magnification.

public void **setAll**(String *x*, String *y*, String *f*, String *m*);
Sets the position and magnification values for the sub panels within this panel. The *x*, *y*, and *f* parameters specify the corresponding positions. The *m* parameter determines the magnification.

public boolean **magChange**();
Returns true if the value stored by the magnification panel has changed since the last image capture.

Remote Microscope Save Panel (*RMsavePanel.java*) **version** 0.2

This class inherits from the Panel Java AWT class. This panel contains buttons to selected images to be saved.

public **RMsavePanel**(String *windows*[], RMguiShell *guishell*);
Constructs a new save panel. The *windows* parameter contains a list of the view windows. The *guishell* parameter determines the interface through which save commands can be executed.

public void **disableAll**();
Disables the buttons. No events are generated by them.

public void **enableAll**();
Enables the buttons. Allows events to be generated by them.

Remote Microscope SigF Values Panel *(RMsigFValuesPanel.java)* **version 0.1**

This class inherits from the Panel Java AWT class. It contains text fields for focus position and focus quality values.

public **RMsigfValuesPanel**();
Construct a new sigF values panel,

public void **setFpos**(int *fpos*);
Sets the focus position to the value specified by the *fpos* parameter and updates the screen accordingly.

public void **setFocusQuality**(int *fq*);
Sets the focus quality to the value specified by the *fq* parameter and updates the screen accordingly.

public void **setFpos**(String *fpos*) {
Sets the focus position to the value specified by the *fpos* parameter and updates the screen accordingly.

public void **setFocusQuality**(String *fq*);
Sets the focus quality to the value specified by the *fq* parameter and updates the screen accordingly.

public String **getFposString**();
Returns the focus position.

public String **getFocusQualityString**();
Returns the focus quality.

public String **getFpos**();
Returns the focus position.

public String **getFocusQuality**();
Returns the focus quality.

private Panel **makeSubPanel**(String *title*, String *label*, TextField *field*);
Private method used to construct the value entries for the main panel. The title parameter determines the *title* of the value section. The *label* parameter determines the name of the type of value stored. The *field* parameter is the actual text field to which dada is entered.

Remote Microscope Bottom Slice Panel *(RMbottomSlicePanel.java)* **version** 0.2

This class inherits from the Panel Java AWT class. This is mainly an organizational container class, that houses RMsliceValuesPanel and RMfocusSlideBar classes.

public **RMbottomSlicePanel**();
Constructs a new bottom slice panel.

public RMsliceValuesPanel **getValues**();
Returns the slice values panel contained in this panel.

public RMfocusSlideBar **getSlideBar**();
Returns the slide bar panel contained in this panel.

Remote Microscope Bottom View Panel *(RMbottomViewPanel.java)* **version** 0.4

This class inherits from the Panel Java AWT class. It forms the bottom portion of the **RMviewFrame** class. Focus position and capture time are stored in local text fields.

public **RMbottomViewPanel**();
Construct a new bottom view panel instance.

public void **setF**(int *f*);
Sets the focus position to the value specified by the *f* parameter and updates the screen accordingly.

public void **setF**(String *f*);
Sets the focus position to the value specified by the *f* parameter and updates the screen accordingly.

public void **setTime**(String *t*);
Sets capture time to the value specified by the *t* parameter and updates the screen accordingly.

public void **setAll**(int *f*, String *t*);
Sets the focus position to the value specified by the *f* parameter and capture time to the value specified by the *t* parameter . The display is updates accordingly.

public void **setAll**(String *f*, String *t*);
Sets the focus position to the value specified by the *f* parameter and capture time to the value specified by the *t* parameter . The display is updates accordingly.

public int **getF**();
Returns the focus position.

public String **getFString**();
Returns the focus position.

public String **getTime**();
Returns the capture time.

private Panel **makeMiniPanel**(String *label*, TextField *field*);
Private method used to create the value field areas for this panel. The *label* parameter determines the name of the label next to the value. The *field* parameter specified the area to which data is written and retrieved.

Remote Microscope Position Panel *(RMpositionPanel.java)* **version 0.4**

This class inherits from the Panel Java AWT class. It contains field for x, y, and f position values.

public **RMpositionPanel(**String *tag*, String *tag2*, Color *c*);

Constructs a new position panel. The *tag* parameter specifies the label for the entire panel. The *tag2* parameter specifies a type tag attachment for each entry. The *c* parameter specifies the background color for value fields.

public void **disableAll**();

Disables the ability to edit field in this panel.

public void **enableAll**();

Enables the ability to edit field in this panel.

public int **getCurrent(**String *str*);

Returns the value of the position specified by the *str* parameter. *Str* can be "x", "y", "f", or the capital versions of these and refer the corresponding position value.

public String **getCurrentString(**String *str*);

Returns the value of the position specified by the *str* parameter. *Str* can be "x", "y", "f", or the capital versions of these and refer the corresponding position value.

public void **setCurrent(**String *str*, int *value*);

Sets the value of the position specified by the *str* parameter. *Str* can be "x", "y", "f", or the capital versions of these and refer the corresponding position value.

public void **setCurrent(**String *str*, String *value*);

Sets the value of the position specified by the *str* parameter. *Str* can be "x", "y", "f", or the capital versions of these and refer the corresponding position value.

public void **setAll(**int *x*, int *y*, int *f*);

Sets the values of all the position values. The *x*, *y*, and *f* parameters correspond to the position entries respectively.

public void **setAll(**String *x*, String *y*, String *f*);

Sets the values of all the position values. The *x*, *y*, and *f* parameters correspond to the position entries respectively.

private Panel **makePosPanel(**String *label*, TextField *field*);

Private method used to create the value field areas for this panel. The *label* parameter determines the name of the label next to the value. The *field* parameter specified the area to which data is written and retrieved.

Remote Microscope Manual Focus Panel *(RMmanualFocusPanel.java)* **version** 0.3
This class inherits from the Panel Java AWT class. It contains button which are used to activate the **RMsliceManualFocus** and **RMsigfManualFocus** classes and bring up the appropriate windows.

public **RMmanualFocusPanel**(RMguiShell *guishell*, Image *fUp*, Image *fDown*, Image *lUp*, Image *lDown*, Image *pUp*, Image *pDown*);
The *fUp* and *fDown* parameters contain the fill button images, in its up and down state respectively. The *lUp* and *lDown* parameters contain the line button images, in its up and down state respectively. The *pUp* and *pDown* parameters contain the points button images, in its up and down state respectively. The *guishell* parameter provides an interface through which commands can be executed.

void **disableAll**();
Disables the buttons contained in this panel.

public void **enableAll**();
Enables the buttons contained in this panel.

public void **closeSliceFrame**();
Closes the slice manual focus window if it is open.

public void **closeSigfFrame**();
Closes the sigF manual focus window if it is open.

public RMsliceManualFocus **getSliceFrame**();
Returns the slice manual focus frame.

public RMsigfManualFocus **getSigfFrame**();
Returns the sigF manual focus frame.

Remote Microscope Center View Panel (*RMcenterViewPanel.java*) **version** 1.4

This class inherits from the Canvas Java AWT class. It contains a captured image, as well as routines to draw annotations to it. It also contains methods to interact with image through the Remote Microscope tools.

public **RMcenterViewPanel(**RMviewFrame *parent*, RMguiShell *shell*,
Image *load_img*, int *w*, int *h*);

Constructs a new center view canvas The **parent** parameter determines to which view frame this canvas part of. The *shell* parameter contains an interface to execute commands through. The *load_img* parameter contains an image be displayed when the Remote Microscope is busy capturing a new image. The *w* and *h* parameters specify the height and width of the canvas.

public void **setImage(**Image *i*);

Sets the current image be displayed by the canvas to image specified by the *i* parameter.. Automatically calls the proper redraw routines.

public Image **getImage**();

Returns the image currently being displayed by the canvas.

public void **setOffset(**int *x*, int *y*);

Sets the offset of the image in the canvas to values determined by the *x* and *y* parameters.

public void **setCursor(**int *x*, int *y*);

Sets the current position of the annotation cursor in this display. The position is specified through the *x* and *y* parameters. The canvas is redraw accordingly.

public void **setRectangle(**int *x1*, int *y1*, int *x2*, int *y2*);

Sets the current position and dimensions of the annotation rectangle in this display. The (*x1*,*x2*) parameters determine the upper left most corner of the rectangle and the (*x2*,*y2*) parameters determine the lower rightmost corner. The canvas is redraw accordingly.

public void **setCursor(**String *x*, String *y*);

Sets the current position of the annotation cursor in this display. The position is specified through the *x* and *y* parameters. The canvas is redraw accordingly.

public void **setRectangle(**String *x1*, String *y1*, String *x2*, String *y2*);

Sets the current position and dimensions of the annotation rectangle in this display. The (*x1*,*x2*) parameters determine the upper left most corner of the rectangle and the (*x2*,*y2*) parameters determine the lower rightmost corner. The canvas is redraw accordingly.

public void **setCursorFlag(**boolean *b*);

Sets the display flag for the annotation cursor. Where a value of true in the *b* parameter means on, and false means off.

void **setRectangleFlag**(boolean *b*) {
Sets the display flag for the annotation rectangle. Where a value of true in the *b* parameter means on, and false means off.

public int **getImageWidth**();
Returns the width of the current image being displayed.

public int **getImageHeight**();
Returns the height of the current image being displayed.

public void **setMagBox**(int *x*, int *y*);
Sets the *x* and *y* dimension of the magnification box surrounding the annotation cursor.

public boolean **getRectangleFlag**();
Returns the display flag for the annotation rectangle. Where a value of true in the *b* parameter means on, and false means off.

public boolean **getCursorFlag**();
Returns the display flag for the annotation cursor. Where a value of true in the *b* parameter means on, and false means off.

public void **doLoading**();
Tells the canvas to clear the displayed image and turn on the loading image.

public void **activateScanline**(int *y*);
Turns on the visual display of the scan line at the vertical position specified by *y*.

public void **deactivateScanline**();
Turns the visual display of the scan line off.

public void **draw**();
Draw the current displayable image and all annotations to a secondary buffer.

private void **drawPointer**(Graphics *g*, int *x*, int *y*);
Private method used by the **draw** method to construct the pointer or cursor object. The *g* parameter specified the graphic context to draw to and the (*x*,*y*) parameters determine the position.

private void **drawScanline**(Graphics *g*, int **y**);
Private method used by the **draw** method to construct the scan line object. The *g* parameter specified the graphic context to draw to and the *y* parameter determine the vertical position.

private void **drawRectangle**(Graphics *g*);
Private method used by the **draw** method to construct the rectangle object. The *g* parameter specified the graphic context to draw to.

private void **drawMeasureLine**(Graphics *g*);
Private method used by the **draw** method to construct the measurement tool object. The *g* parameter specified the graphic context to draw to.

private int[] **getParallelLines**(int *_x1*, int *_y1*, int *_x2*, int *_y2*, double *d1*);
Calculates the dimensions for two lines, that are parallel to the input line, specified by the (*_x1*,*_y1*) and (*_x2*,*_y2*), are of a size determined by the *d1* parameter, and are located at the end of the input line. The two lines are returned in an array of integers in a format such that [0],[1] ; [2],[3] ; [4],[5] ; [6],[7] are point pairs where [0],[1] ; [2],[3] and 4],[5] ; [6],[7] define two line, that are parallel to the input line and are located at its ends

Remote Microscope SigF Manual Focus *(RMsigfManualFocus.java)* **version 0.2**

This class inherits from the Frame Java AWT class. It creates a window, which is used for the sigF manual focus method. The **RMsigfValuesPanel** and **RMfocusSlideBar** classes are used in its construction. Events are handled locally and through a graphical shell.

public **RMsigfManualFocus**(RMguiShell *guishell*);
Construct a new sigF manual focus frame. The *guishell* parameter contains an interface to execute commands through

public int **getValue**(String *val*);
Returns the value of the field specified by the *val* parameter. The *val* parameter can be "focusquality", for focus quality, or "fpos", for focus position.

public String **getValueString**(String *val*);
Returns the value of the field specified by the *val* parameter. The *val* parameter can be "focusquality", for focus quality, or "fpos", for focus position.

public void **setValue**(String *val*, int *value*);
Sets the value of the field specified by the *val* parameter. The *val* parameter can be "focusquality", for focus quality, or "fpos", for focus position.

public void **setValue**(String *val*, String *value*);
Sets the value of the field specified by the *val* parameter. The *val* parameter can be "focusquality", for focus quality, or "fpos", for focus position.

public void **sendRequest**(String *fpos*);
Send a request for a new sigF focus value with the specified focus position, *fpos* parameter, through the graphical interface.

Remote Microscope User Panel *(RMuserPanel.java)* **version 0.6**

This class inherits from the Panel Java AWT class. The user panel is a bit misnamed, as it contains more information than just a list of users. It also contains instances of the **RMextraPanel** and **RMmanualFocusPanel.**

public **RMuserPanel**(RMguiShell *guishell*, Image *fUp*, Image *fDown*,
Image *lUp*, Image *lDown*, Image *pUp*, Image *pDown*);
Constructs a new user panel. The *fUp* and *fDown* parameters contain the fill button images, in its up and down state respectively. The *lUp* and *lDown* parameters contain the line button images, in its up and down state respectively. The *pUp* and *pDown* parameters contain the points button images, in its up and down state respectively. The *guishell* parameter provides an interface through which commands can be executed.

public void **deleteUsers**();
Deletes the list of users.

public void **addUser**(String *user*);
Adds a new user, specified by the *user* parameter, to the list of users.

public RMextraPanel **getExtra**();
Returns the extra panel from this panel.

public RMmanualFocusPanel **getManualFocus**();
Returns the manual focusing panel form this panel.

Remote Microscope Top View Panel *(RMtopViewPanel.java)* **version 0.4**

This class inherits from the Panel Java AWT class. It forms the top portion of an **RMviewFrame**. It contains x position, y position, and magnification text fields.

public **RMtopViewPanel**(String *window*);
Constructs a new top view panel. The *window* parameter specifies the name of the window.

public void **setX**(int *x*);
Changes the x position to the value specified by the *x* parameter.

public void **setY**(int *y*);
Changes the y position to the value specified by the *y* parameter.

public void **setMag**(int *m*);
Changes the magnification to the value specified by the *m* parameter.

public void **setCurrent**(int *x*, int *y*);
Sets the current x and y position values, based on the *x* and *y* parameters.

public void **setX**(String *x*);
Changes the x position to the value specified by the *x* parameter.

public void **setY**(String *y*);
Changes the y position to the value specified by the *y* parameter.

public void **setMag**(String *m*);
Changes the magnification to the value specified by the *m* parameter.

public void **setCurrent**(String *x*, String *y*);
Sets the current x and y position values, based on the *x* and *y* parameters

public void **titleColor**(Color *c*);
Sets the background color behind the name label of this panel to the color specified by the *c* parameter.

public void **setAll**(int *x*, int *y*, int *m*, Color *c*);
Sets the current x and y position values, based on the *x* and *y* parameters, and the background color behind the name label of this panel to the color specified by the *c* parameter.

public void **setAll**(String *x*, String *y*, String *m*, Color *c*);
Sets the current x and y position values, based on the *x* and *y* parameters, and the background color behind the name label of this panel to the color specified by the *c* parameter.

public void **setAll**(int *x*, int *y*, int *m*);
Sets the current x and y position values, based on the *x* and *y* parameters, and the magnification to the value specified by the *m* parameter.

public void **setAll**(String *x*, String *y*, String *m*);
Sets the current x and y position values, based on the *x* and *y* parameters, and the magnification to the value specified by the *m* parameter.

public int **getX**();
Returns the x position.

public int **getY**();
Returns the y position.

public int **getMag**();
Returns the magnification.

public String **getXString**();
Returns the x position.

public String **getYString**();
Returns the y position.

public String **getMagString**();
Returns the magnification.

private Panel **makeMiniPanel**(String *label*, TextField *field*);
Private method used to create the value field areas for this panel. The *label* parameter determines the name of the label next to the value. The *field* parameter specified the area to which data is written and retrieved.

Remote Microscope Extra Panel *(RMextraPanel.java)* **version 0.5**

This class inherits from the Panel Java AWT class. It contains buttons to quit the application and load a new wafer. When the wafer loading button is pressed and instance of the **RMloadWaferFrame** class is displayed.

public **RMextraPanel**(RMguiShell *guishell*);
Construct a new extra panel. The *guishell* parameter specifies an interface through which commands can be executed.

public void **disableAll**();
Disables the buttons contained in this panel.

public void enableAll();
Enables the buttons contained in this panel.

public void **disableLoadWafer**();
Disables the load wafer button.

public void **enableLoadWafer**();
Enables the load wafer button.

public void **disableQuit**();
Disables the quit application button.

public void **enableQuit**();
Enables the quit application button.

public int **getXsize**();
Returns the x size parameter for a load new wafer command.

public int **getYsize**();
Returns the y size parameter for a load new wafer command.

public String **getXsizeString**();
Returns the x size parameter for a load new wafer command.

public String **getYsizeString**();
Returns the y size parameter for a load new wafer command.

public String **getFocusType**();
Returns the type of the wafer specified for a load new wafer command. This can be "FLAT" or "ROUGH".

public int **getThickness**();
Returns the wafer thickness parameter for a load new wafer command.

public String **getThicknessString**();
Returns the wafer thickness parameter for a load new wafer command.

Remote Microscope Execute Panel *(RMexecutePanel.java)* **version** 0.2

This class inherits from the Panel Java AWT class. It simply contains a specifically formated execute button. Events from this button should be handled by this panel's container.

public **RMexecutePanel**();
Constructs a new execute panel.

public void **disableAll**();
Disables the button in this panel.

public void **enableAll**();
Enables the button in this Panel.

Remote Microscope Window Select Panel (*RMwindowSelectPanel.java*) **version** 0.3
This class inherits from the Panel Java AWT class. It contains the checkboxes to selected a destination window that will be used for the next image grabbed by the microscope.

public RMwindowSelectPanel(String *windows*[], RMguiShell *guishell*);
Constructs anew window select panel. The array of windows specified in the *windows* parameter determine the names of the windows. The *guishell* parameter specifies an interface through which commands can be executed.

public void **disableAll**();
Disables all the checkboxes in this panel.

public void **enableAll**();
Disables all the checkboxes in this panel.

public int **getCurrent**();
Returns the currently selected window.

public void **setCurrent**(int *x*);
Changes the currently selected window to the window specified by the *x* parameter.

private Checkbox **makeCheckBox**(String *label*, CheckboxGroup *group*,
boolean *setting*);
Private method used to aid in the creation of the checkbox elements. The *label* parameter determined the attached label for the checkbox. The *group* parameter determined the group to which the check box belongs. The *setting* parameter specifies the state of the checkbox, true is equivalent to checked and false signifies not checked.

Remote Microscope Slice Values Panel *(RMsliceValuesPanel.java)* **version 0.2**

This class inherits from the Panel Java AWT class. It contains text fields for row position, hysteresis value, start position, stop position, focus position, and focus quality.

public **RMsliceValuesPanel**();
Constructs a new slice values panel.

public void **setRowCoord**(int *x*);
Sets the row, or scan line, position to that specified by the *x* parameter. The new value is updated in the display.

public void **setHyst**(int *h*);
Sets the hysteresis value to that specified by the *h* parameter. The new value is updated in the display.

public void **setStartValue**(int *s*);
Sets the starting position on a scan line to that specified by the *s* parameter. The new value is updated in the display.

public void **setFpos**(int *fpos*);
Sets the focus position to be that specified by the *fpos* parameter. The new value is updated in the display.

public void **setFocusQuality**(int *fq*);
Sets the focus quality value to that specified by the *fq* parameter. The new value is updated in the display.

public void **setStopValue**(int *s*);
Sets the ending position on a scan line to that specified by the *s* parameter. The new value is updated in the display.

public void **setRowCoord**(String *x*);
Sets the row, or scan line, position to that specified by the *x* parameter. The new value is updated in the display.

public void **setHyst**(String *h*);
Sets the hysteresis value to that specified by the *h* parameter. The new value is updated in the display.

public void **setStartValue**(String *s*);
Sets the starting position on a scan line to that specified by the *s* parameter. The new value is updated in the display.

public void **setFpos**(String *fpos*);
Sets the focus position to be that specified by the *fpos* parameter. The new value is updated in the display.

public void **setFocusQuality**(String *fq*);
Sets the focus quality value to that specified by the *fq* parameter. The new value is updated in the display.

public void **setStopValue**(String *s*);
Sets the ending position on a scan line to that specified by the *s* parameter. The new value is updated in the display.

public String **getRowCoordString**();
Returns the row, or scan line, value.

public String **getHystString**();
Returns the hysteresis value.

public String **getStartValueString**();
Returns the starting position on a scan line.

public String **getFposString**();
Returns the focus position value

public String **getFocusQualityString**();
Return the focus quality value.

public String **getStopValueString**();
Returns the ending position on a scan line.

public int **getRowCoord**();
Returns the row, or scan line, value.

public int **getHyst**();
Returns the hysteresis value.

public int **getStartValue**();
Returns the starting position on a scan line.

public int **getFpos**();
Returns the focus position value

public int **getFocusQuality**();
Return the focus quality value.

public int **getStopValue**();
Returns the ending position on a scan line.

private Panel **makeSubPanel**(String *title*, String *label*, TextField *field*);
Private method used to construct the value entries for the main panel. The title parameter determines the *title* of the value section. The *label* parameter determines the name of the type of value stored. The *field* parameter is the actual text field to which dada is entered.

Remote Microscope Image Quality Panel *(RMimageQualityPanel.java)* **version** 0.4
This class inherits from the Panel Java AWT class. It contains a single button for brining up an instance of the **RMimageQualityFrame**. The class stores color and color table values.

public **RMimageQualityPanel**();
Constructs a new image quality panel.

public void **disableAll**();
Disables the image quality button.

public void **enableAll**();
Enables the image quality button

public String **getColorTable**();
Returns the value 0 or 1. 1 implies that a new color table is requested and 0 implies the opposite.

public String **getColor**();
Returns c or g, where c implies color and g implies gray scale.

public void **setValues**(String *t*, String *c*);
Sets the color table and color parameters. The *t* parameter specifies the whether or not a new color table should be grabbed, 0 or 1 respectively. The *c* parameter determines the color type, where c implies color and g implies gray scale..

public void **closeImageQualityFrame**();
Hides the image quality frame window.

Remote Microscope Image Quality Frame *(RMimageQualityFrame.java)* **version** NA
This class inherits from the Frame Java AWT class. It is a private class used by the **RMimageQualityPanel** to set its values.

public **RMimageQualityFrame**(RMimageQualityPanel *iqp*);
Constructs a new image quality frame. The *iqp* parameter specified the image quality value where values are to be sent.

private Checkbox **makeCheckBox**(String *label*, CheckboxGroup *group*,
boolean *setting*);
Private method used to aid in the creation of the checkbox elements. The *label* parameter determined the attached label for the checkbox. The *group* parameter determined the group to which the check box belongs. The *setting* parameter specifies the state of the checkbox, true is equivalent to checked and false signifies not checked.

Remote Microscope Focus Select Panel *(RMfocusSelectPanel.java)* **version 0.4**

This class inherits from the Panel Java AWT class. Contains checkboxes to select focusing options.

public **RMfocusSelectPanel**(String *types*[], RMguiShell *guishell*);

Constructs a new focus select panel. The *types* parameter contains an array of focus type names. The *guishell* parameter specifies an interface through which commands can be executed.

public void **disableAll**();

Disables the ability top select focus methods.

public void **enableAll**();

Enables the ability top select focus methods.

public int **getCurrent**();

Returns the index number of the current focus method.

public void **setCurrent**(int *x*);

Sets the current focus method to that specified by the index parameter *x*.

private Checkbox **makeCheckBox**(String *label*, CheckboxGroup *group*, boolean *setting*);

Private method used to aid in the creation of the checkbox elements. The *label* parameter determined the attached label for the checkbox. The *group* parameter determined the group to which the check box belongs. The *setting* parameter specifies the state of the checkbox, true is equivalent to checked and false signifies not checked.

Remote Microscope View Frame *(RMviewFrameI.java)* **version 1.1**

This class inherits from the Frame Java AWT class

public **RMviewFrame**(String *name*, RMguiShell *shell*, Image *load_img*) {
Constructs a new view frame. The parameter *name* specifies the name of the view frame. The *shell* parameter determines the interface through which command may be executed. The *load_img* parameter determines the image to be displayed while a new image is being loaded.

public RMtopViewPanel **getXYPositionInfo**();
Return the top view panel for this window.

public RMbottomViewPanel **getFTPositionInfo**();
Return the bottom view panel for this window.

public RMcenterViewPanel **getImageInfo**();
Return the center, or image, view panel for this window.

public void **setAll**(int *x*, int *y*, int *f*, int *m*, String *t*, Image *i*, Color *c*);
Sets the current x, y, and f position values, based on the *x*, *y*, and *f* parameters. The background color behind the name label is set to the color specified by the *c* parameter. The displayed image is set by the value of the *i* parameter. The *t* parameter sets the time.

public void **setAll**(String *x*, String *y*, String *f*, String *m*, String *t*, Image *i*, Color *c*);
Sets the current x, y, and f position values, based on the *x*, *y*, and *f* parameters. The background color behind the name label is set to the color specified by the *c* parameter. The displayed image is set by the value of the *i* parameter. The *t* parameter sets the time.

public void **setLabelColor**(Color *c*);
Sets the background color behind the name label to the color specified by the *c* parameter.

public void **setCursor**(int *x*, int *y*);
Sets the current position of the annotation cursor. The position is specified through the *x* and *y* parameters. The canvas is redraw accordingly.

public void **setRectangle**(int *x1*, int *y1*, int *x2*, int *y2*);
Sets the current position and dimensions of the annotation rectangle. The *(x1,x2)* parameters determine the upper left most corner of the rectangle and the *(x2,y2)* parameters determine the lower rightmost corner. The canvas is redraw accordingly.

public void **setCursor**(String *x*, String *y*);
Sets the current position of the annotation cursor. The position is specified through the *x* and *y* parameters. The canvas is redraw accordingly.

public void **setRectangle**(String *x1*, String *y1*, String *x2*, String *y2*);
Sets the current position and dimensions of the annotation rectangle. The (*x1*,*x2*) parameters determine the upper left most corner of the rectangle and the (*x2*,*y2*) parameters determine the lower rightmost corner. The canvas is redraw accordingly.

public void **setCursorFlag**(boolean *b*);
Sets the display flag for the annotation cursor. Where a value of true in the *b* parameter means on, and false means off.

void **setRectangleFlag**(boolean *b*) {
Sets the display flag for the annotation rectangle. Where a value of true in the *b* parameter means on, and false means off.

public void **setMagBox**(int *m*);
Sets the visually displayed magnification box based on the magnification specified by the *m* parameter.

public void **setMagBox**(String *m*);
Sets the visually displayed magnification box based on the magnification specified by the *m* parameter.

public boolean **getRectangleFlag**();
Returns the display flag for the annotation rectangle. Where a value of true in the *b* parameter means on, and false means off.

public boolean **getCursorFlag**();
Returns the display flag for the annotation cursor. Where a value of true in the *b* parameter means on, and false means off.

public void **doLoading**();
Tells the canvas to clear the displayed image and turn on the loading image.

public int **getUID**();
Returns the unique identifier for this window

public int **getYCoord**(int *y*);
Returns the *y* parameter converted from image to microscope coordinates.

public int **getXCoord**(int *x*);
Returns the *x* parameter converted from image to microscope coordinates.

public int **invYCoord**(int *y*);
Returns the *y* parameter converted from microscope to image coordinates.

public int **invXCoord**(int *x*);
Returns the *x* parameter converted from microscope to image coordinates.

public void **activateScanline**(int *y*);
Turns on the visual display of the scan line at the vertical position specified by *y*.

public void **deactivateScanline**();
Turns the visual display of the scan line off.

public Image **getImage**();
Returns the image currently being displayed.

public Dimension **getDimension**();
Returns the dimensions in which the center view image is to be drawn.

public int **adjustYCoord**(int *y*);
Adjust a base image y position, specified by *y*, to a resized one.

public int **adjustXCoord**(int *x*);
Adjust a base image x position, specified by *x*, to a resized one.

public int **unadjustYCoord**(int *y*);
Adjust a resized image y position, specified by *y*, to a base size value.

public int **unadjustXCoord**(int *x*);
Adjust a resized image x position, specified by *x*, to a base size value.

public static void **resetID**();
Resets the starting value for window UID values.

public static double **PIX** = *0.7520915*;
Conversion constant, used for microscope to screen coordinate conversion.

A.2.2 Graphics Package Use

The main application class, **RMmainFrame**, inherits from **RMguiShell**. When the Remote Microscope client is initialized, **RMmainFrame** uses the classes from the graphics package to construct all of its visual elements. The code segment below, from the **RMmainFrame** constructor shows this basic process.

```
...
currentSettingsPanel = new RMsettingsPanel("CURRENT", "CUR", Color.pink);
setNewSettingsPanel = new RMsettingsPanel("SET", "NEW", Color.orange);
String windows[] = new String[2];
windows[0] = "A";
windows[1] = "B";
String focTypes[] = new String[2];
focTypes[0] = "Auto";
focTypes[1] = "Manual";
controlPanel = new RMcontrolPanel(windows, focTypes, this);
userPanel =  new RMuserPanel(this, fUp, fDown, lUp, lDown, pUp, pDown);
chatPanel = new RMchatPanel(this);
toolsFrame = new RMtoolsFrame(this, tools);
...
```

User input are either handled through the **RMmainFrame** class or locally to the graphic classes. An example of the code used in the **RMmainFrame** class to trap graphical input events is given below. (1) shows the capture of an event from a "quit application" button press and (2) shows the capture of an "EXECUTE" button press.

```
public boolean action(Event evt, Object arg)
{
...
/*Button Press Capture*/
        if(arg.equals("quit application")) {killClient();}                (1)
        ...
        else if(arg.equals("EXECUTE")) {execute();}                       (2)
        ...
        return(true);
}
```

The commands, specified by the **RMguiShell** interface, are executed within the **RMmainFrame** class by various graphical elements. An example of the **RMsigfManualFocus** classes doing this is given below. The proper methods must be implemented for all these to function correctly.

```
public void sendRequest(String fpos)
{
        shell.getSigf(fpos);
}
```

The **RMmainFrame** class is responsible for updating the values stored in the graphical elements when appropriate. It also has the ability to retrieve information from the graphical elements. Example code for a change of magnification is given below, where (1) and (4) show an update of graphical elements and (2) and (3) show data accesses.

```
/**
 * magReply
 * -------------------------------------
 * Handles magnification reply
 * params[0]         magnification
 * ------------------|------------------
 * param params                  parameters
 *
 **/
 public void magReply(String params[])
{
        /*sets the magnification in the control panel*/
        setNewSettingsPanel.getMag().setCurrent(params[0]);              (1)

        /*selects the proper window*/
        int win;
        if(viewWindows[0].getCursorFlag()) win = 0;                      (2)
        else if(viewWindows[1].getCursorFlag()) win = 1;                 (3)
        else win = -1;

  /*resizes magnification box*/
  if(win != -1) {viewWindows[win].setMagBox(params[0]);}                 (4)
}
```

A.3 The GifImage Package

The Remote Microscope transmits all images in GIF format, as an array of bytes, to the network package. Java does not directly support the ability to convert an array of bytes into a GIF file. A Java based GIF conversion class called **GIFImage** written by Tadashi Hamano, found through the Gamelan Java repository, is used to handle this task. As, the GIFImage package was not designed specifically for this project and is merely being used as a stand alone set of methods, it will not be covered on a class by class basis.

A.4 The Application Package

The application package, unlike the other groupings, is not actually a Java package. It brings together all of the packaged components and, either directly or indirectly, handles all interactions between them. It is the controller and as such, has a varied set of responsibilities. Whenever a command is issued from the network package, the main program is responsible for distributing the appropriate data to their respective data structures. If an input is detected, it performs the desired operations and if appropriate returns a value. If a component requests information, it retrieves and returns the data. These are the major functions of the application classes, though it handles a variety of smaller tasks as well.

A.4.1 Application Package Classes

Remote Microscope Application Shell (*RMappShell.java*) version 0.2

This interface class contains methods to access application specific functions. This allows the Remote Microscope main programs to act as both applications and applets.

public void **saveImageReply**(String *params*[], Image *img*, Frame *parent*);
Issues the save image command for the application. The *params* array holds a Url in position 0 and a window in position 1. The img *parameter* is the image to be saved and the *parent* is the frame of the main application.

public void **killClient**();
Exits the program.

public void **outputMessage**(String *msg*);
Outputs a message, specified by the *msg* parameter, to a visible location.

Remote Microscope Application *(RMapp.java)* **version** 0.5

This class implements the **RMappShell** interface. It is used to startup the Remote Microscope as an application and initializes an instance of the **RMmainFrame** class. Command line parameters can be used to set host and port settings for the Remote Microscope server.

java **RMapp** <host> <port>
- starts a new application with the specified <host> and <port>
java **RMapp** <port>
- starts a new application with the specified <port> and default host
java **RMapp**
- starts a new application with the default host and port

public **RMapp()**;
Constructs a new application instance.

public void **saveImageReply(**String *params*[], Image *img*, Frame *parent*);
Issues the save image command for the application. The *params* array holds a Url in position 0 and a window in position 1. The img *parameter* is the image to be saved and the *parent* is the frame of the main application.

public void **killClient()**;
Exits the program by killing the process.

public void **outputMessage(**String *msg*);
Outputs a message, specified by the *msg* parameter, to the standard output.

Remote Microscope Applet *(RMapplet.java)* **version** 0.5

This class inherits from the Applet class and implements the **RMappShell** interface. It is used to startup the Remote Microscope as an applet and initializes an instance of the **RMmainFrame** class. Startup parameters, used to set host and port settings for the Remote Microscope server, are specified in the HTML file containing this applet. The code for this is:

```
<applet code="RMapplet.class" width=300 height=200>
<param name=server value="HOST">
<param name=port value="PORT">
</applet>
```

Where HOST and PORT specify the values of the parameters.

public InputStream **getStream**(String *file*);
Returns an input stream for loading selected input *file*.

public void **saveImageReply**(String *params*[], Image *img*, Frame *parent*);
Issues the save image command for the application. The *params* array holds a Url in position 0 and a window in position 1. The img *parameter* is the image to be saved and the *parent* is the frame of the main application.

public void **killClient**();
Exits the program by killing the process.

public void **outputMessage**(String *msg*);
Outputs a message, specified by the *msg* parameter, to the standard applet's text area.

Remote Microscope Main Frame *(RMmainFrame.java)* **version 3.5**

This class inherits from the Frame class and implements **RMclientShell, RMguiShell,** and **RMlayoutShell**. It is the main window and central control area for the Remote Microscope client.

public **RMmainFrame(**Image *titleImage*, Image *busyImage*, Image *fUp*, Image *fDown*, Image *lUp*, Image *lDown*, Image *pUp*, Image *pDown*, Image *tools*, RMappShell *shell*, MAGICframe *m*);

Constructs a new main frame. The *titleImage*, *busyImage*, and *loadImage* are used to specify the title, busy, and loading image respectively. The *fUp* and *fDown* parameters contain the fill button images, in its up and down state respectively. The *lUp* and *lDown* parameters contain the line button images, in its up and down state respectively. The *pUp* and *pDown* parameters contain the points button images, in its up and down state respectively. The *tools* parameter specifies the image to be used in the tools frame. The *shell* parameter specifies the application shell for executing commands. The *m* parameter specifies the Majik layout to be used for navigation.

private void **add(**Component *c*, GridBagLayout *gbl*, int *x*, int *y*, int *w*, int *h*, Container *addto*);

Private function used to add elements to the graphical layout.

public boolean **hasControl**();

Returns true if the client has control of the microscope. Else false.

public void **setClient(**RMclient *c*);

Sets the networking client, specified by the *c* parameter, used my this frame

public void **killClient**();

Exits the program.

public void **command(**int *com*, String *params*[]);

Executes a command within the interface. The command, *com*, be any on of the following:

clearUserList	appendUserList	makeControlPassive
makeControlActive	makeControlNone	setScopeState
zoomGrabReply	grabFrameReply	curPosReply
rectangleReply	magReply	busyReply
saveImageReply	chatReply	loadNewWaferReply
getSliceReply	getSigfReply	

The interpretation of the *params* array is based on the specific type of message. The meaning of these parameters types is given in the main program section.

public void **bytesTransfer**(int *com*, byte *b*[]);
Sends an array of bytes through the interface. The command, *com*, specifies the type of command that is requesting a bytes transfer and the *bytes* parameter is the actual data.

public boolean **isBusy**();
Returns true if the interface is currently executing a command or waiting for s server reply. If this is not the case this method returns false.

public void **busyReply**(String *params*[])
Handles a "BUSYREPLY" message from the server. The *params* array stores a message in position 0.

public void **clearUserList**();
Handles a "USERLISTADD" message from the server. Deletes the user list.

public void **makeControlActive**();
Makes the control of the client passive.

public void **makeControlNone**();
Makes the control of the client none.

public void **makeControlActive**();
Makes the control of the client active.

public void **appendUserList**(String *params*[]);
Handles a "USERLISTADD" message from the server. The *params* array stores a user name in position 0.

public void **setScopeState**(String *params*[]);
Handles a "SCOPESTATE" message from the server. The *params* array stores the x coordinate in position 0, y coordinate in position 1, f coordinate in position 2, and magnification in position 3.

public void **grabReply**(int *com*, String *params*[]);
Handles a "GRABFRAMEREPLY" and "ZOOMGRABREPLY" messages from the server. The *params* array stores the x coordinate in position 0, y coordinate in position 1, f coordinate in position 2, magnification in position 3, time in position 4, and current window flag in position 5.

public void **magReply**(String *params*[]);
Handles a "MAGREPLY" message from the server. The *params* array stores the magnification in position 0.

public void **curPosReply**(String *params*[]);
Handles a "CURPOSREPLY" message from the server. The *params* array stores the window in position 0, x coordinate in position 1, and y coordinate in position 2.

public void **rectangleReply**(String *params*[]);
Handles a "RECTANGLEREPLY" message from the server. The *params* array stores the window in position 0, top x coordinate in position 1, top y coordinate in position 2, bottom x coordinate in position 3, and bottom y coordinate in position 4.

public void **loadNewWaferReply**(String *params*[]);
Handles a "LOADWAFERREPLY" message from the server. The *params* array stores the x coordinate in position 0, y coordinate in position 1, f coordinate in position 2, magnification in position 3, and time in position 4.

public void **saveImageReply**(String *params*[]);
Handles a "GETIMAGEREPLY" message from the server. The *params* array stores the URL in position 0, and window in position 1.

public void **getSliceReply**(String *params*[]);
Handles a "GETSLICEREPLY" message from the server. The *params* array stores the x coordinate in position 0, y coordinate in position 1, f coordinate in position 2, magnification in position 3, slice focus quality in position 4, x data in position 5, and y data in position 6.

public void **getSigfReply**(String *params*[]);
Handles a "GETSIGFREPLY" message from the server. The *params* array stores the x coordinate in position 0, y coordinate in position 1, f coordinate in position 2, magnification in position 3, and sigF focus quality in position 4.

public void **chatReply**(String *params*[]);
Handles a "CHATREPLY" message from the server. The *params* array stores a message in position 0 and a user name in position 1.

public void **magChange**();
Informs the Remote Microscope server of a magnification change by the client.

public void **takeControl**();
Requests a taking of control of the microscope.

public void **cedeControl**();
Requests a cede of control of the microscope.

public void **execute**();
Requests a grab capture from the microscope.

public void **setCursor**(int *win*, int *x*, int *y*);
Informs the interface of a cursor position change within a graphical window. The *win* parameter specifies the window in which the cursor has changed position. The position is specified with the *x* and *y* parameters.

public void **startRectangle**(int *win*);
Signals the beginning of a rectangle within the window specified by the *win* parameter.

public void **endRectangle**(int *win*, int *x1*, int *y1*, int *x2*, int *y2*);
Signals the end of a rectangle within the window specified by the *win* parameter. The (*x1*,*y1*) pair is the top right corner of the rectangle and the (*x2*,*y2*) pair is the bottom left corner.

public boolean **hasControl**();
Returns true if the interface currently has control of the Remote Microscope.

public void **loadWafer**();
Tells the interface to execute a load wafer command.

public void **calibrate**();
Tells the interface to execute calibration command for a new wafer.

public void **nocalibrate**();
Tells the interface not to execute a calibration command for a new wafer.

public void **saveImage**(int *window*, String *pathname*, String *filename*);
Executes a save image command within the interface. The *window* parameter specifies which window to save from. The *pathname* and *filename* give the location where the image should be saved.

public void **activateManualFocus**();
Tells the interface to activate the manual focusing options.

public void **deactivateManualFocus**();
Tells the interface to deactivate the manual focusing options.

public void **getSlice**(String *size*, String *fpos*, String *rowpos*,
String *hystval*, String *start*, String *stop*, String *fopt*);
Executes a get slice information request within the interface. The *size* parameter species the horizontal size of the image. The *fpos* parameter species the focus position of the grab. *Start* and *stop* determine the range within the *size* to scan. Hystval is determines noise reduction. The *fopt* can be either “MANUAL” or “AUTO” and determines the type of focusing performed by the server.

public void **activateScanline**(int *y*);
Tell the interface to activate a scan line within the current grab window at the species y position.

public void **deactivateScanline**();
tell the interface to turn off the scan line within the image windows.

public void **getSigf**(String *fpos*);
Executes a get sigF data request within interface. The desired focus position is specified in the *fpos* parameter.

public void **sendChatMessage**(String *msg*);
Requests that the message *msg* be sent to the server through the interface.

public void **setWindow**(int *win*);
Tells the interface to change its current grab window that specified by the *win* parameter.

public int **getWindow**();
Returns the current grab window for the interface.

public void **winChange**();
Informs the interface that the selection window has changed.

public void **paramExecute**(boolean *chgX*, boolean *chgY*, boolean *chgF*,
boolean *chgMag*, int *x*, int *y*, int f, int *mag*);
Executes a grab command, within the interface, with parameters different form a normal image grab execution. The parameter *chgX* tells the interface if the x position is different from normal. If it is the value of *x* is used as the x position. The parameter *chgY* tells the interface if the y position is different from normal. If it is the value of *y* is used as the y position. The parameter *chgF* tells the interface if the f position is different from normal. If it is the value of *f* is used as the f position. The parameter *chgMag* tells the interface if the magnification is different from normal. If it is the value of *mag* is used as the magnification value.

public int **getTool**();
Returns the current tool selected by the interface. The list of tools will be detailed in the tools panel section.

public RMviewFrame **getViewWindow**();
Returns the current viewing window selected by the interface.

public boolean **isBusy**();
Return true if the interface is currently busy, i.e. waiting for a response from the Remote Microscope server.

public void **layoutGO**(int *pos*[]);
Executes a request to the layout, through the interface, to move to the position specified by the *pos* parameter.

public void **layoutNAVset**(int *pos*[]);
Tells the interface of the calibration coordinates obtained from a graphical object.

public static int **getNumClients**();
Returns the number of clients initialized through this class since startup

public int[] **LSgetCalibCoords**()
This method is used to retrieve the calibration coordinates of a layout shell. These coordinates are used to translate between different view points. The points are returned in a single index array, such that the interface is independent of dimensions.

public void **LSsetPos**(RMlayoutShell *layout*, int *points*[])
The current position of the layout interface is set through this method. The *layout* parameter specifies the class requesting a position change. The *points* parameter, given in the requesting classes coordinate plane, stores the desired location. Using internally stored calibration coordinates and those obtained from the requesting layout interface, through the LSgetCalibCoords() method, the position of the interface is set.

public void **LSsetShell**(RMlayoutShell *loShell*);
This method specifies layout interface, *loShell*, which should be used as a receiver for position change requests.

public static void **startClient**(String server, int port, Image titleImage, Image busyImage, Image loadImage, Image fUp, Image fDown, Image lUp, Image lDown, Image pUp, Image pDown, Image tools, RMappShell shell, MAGICframe m);
Quick startup method for initializing the Remote Microscope client. Constructs a new **RMmainFrame**, attempts to start a network connection, and returns the result. The *host* and *port* parameters specify server connection. The *titleImage*, *busyImage*, and *loadImage* are used to specify the title, busy, and loading image respectively. The *fUp* and *fDown* parameters contain the fill button images, in its up and down state respectively. The *lUp* and *lDown* parameters contain the line button images, in its up and down state respectively. The *pUp* and *pDown* parameters contain the points button images, in its up and down state respectively. The *tools* parameter specifies the image to be used in the tools frame. The *type* parameter can be "applet" or "app" and determines the type of this program. The *m* parameter specifies the Majik layout to be used for navigation.

A.4.2 Application Package Use

The application package is used to start and stop the Remote Microscope client. It can be used as an applet through the RMapplet class and an application through the RMapp class. The parameters needed to execute these classes is given the appropriate section in the class descriptions.

Appendix B

Majik Code Overview

This appendix contains a complete overview of the Majik Viewer code. The classes for the magic layout package will be discussed, as will a look at how these classes are used in higher level classes. The breakdown of this appendix is given on the following page.

MAGIC Layout Package Sections

B.1 MAGIC LAYOUT CLASSES 170

B.2 MAGIC LAYOUT USE 195

B.1 Magic Layout Classes

Remote Microscope Layout Shell *(RMlayoutShell.java)* **version** 0.1

This class defines an interface for layout navigation. It allows two programs to interchange coordinate and location information.

public int[] **LSgetCalibCoords**()

This method is used to retrieve the calibration coordinates of a layout shell. These coordinates are used to translate between different view points. The points are returned in a single index array, such that the interface is independent of dimensions.

public void **LSsetPos**(RMlayoutShell *layout*, int *points*[])

The current position of the layout interface is set through this method. The *layout* parameter specifies the class requesting a position change. The *points* parameter, given in the requesting classes coordinate plane, stores the desired location. Using internally stored calibration coordinates and those obtained from the requesting layout interface, through the LSgetCalibCoords() method, the position of the interface is set.

public void **LSsetShell**(RMlayoutShell *loShell*);

This method specifies layout interface, *loShell*, which should be used as a receiver for position change requests.

MAGIC Color Map (*MAGICcmap.java*) **version 0.1**

This class parses a magic color map file from an input stream and stores color index information for later retrieval.

public **MAGICcmap(**InputStream *in*);
Constructs a new MAGIC color map instance. The *in* parameter specifies the input stream from which the color map file should be retrieved.

public void **parseFile(**InputStream *instream*);
Parses the MAGIC color map, retrieved from the stream specifies by the *instream* parameter.

public Color **getColor(**int *i*);
Returns the Java Color specified by the index *i*.

private int **getIndex(**String *s*);
Private method is used by the **parseFile** method to retrieve index information from an input line *s*. It returns the index.

private Color **parseColor(**String *s*);
Private method used by the **parseFile** method to retrieve (r,g,b) color information from an input line *s*. It returns the color.

MAGIC Contact *(MAGICcontact.java)* **version** 0.1

This class is just a data structure to hold contacts. It simply holds the type of connection and the planes that it makes connections between

public **MAGICcontact(**String *type*, String *connections*[]);
Construct a new contact instance with the specified *type* and *connections.*

public String **type;**
Type of the contact.

public String **connections**[];
Array of plane names that the contact is connected to.

MAGIC Draw Map *(MAGICdrawmap.java)* **version** 0.4

The draw map class contains a mapping between styles/colors and technologies. At this point in time the styles parsing class (MAGICstyles.java) does not support stipple parsing and thus a hacked stipple setting is hard coded. The supported stipple types are:

dashNWtoSE	dashNEtoSW	cross
waffle	waffle2	outline
dashEtoW		

public **MAGICdrawmap(**MAGICstyles *styles*, MAGICcmap *cmap*, MAGICtech *tech*);
Constructs a new draw map from the parsed files specified by the *styles*, *cmap*, and *tech* parameters.

public MAGIClayer[] **getLayers**();
Returns an array of MAGIC layers. The index in the array corresponds to a layer number.

public MAGICdrawref **getDrawInfo(**String *s*);
Returns the draw reference for the element specified by the *s* parameter.

public Vector **getBasicTypes**();
Returns a Vector containing all of the basic element types stored.

MAGIC Draw Reference *(MAGICdrawref.java)* **version** 0.1

This class is just a data structure to hold draw references for elements.

public **MAGICdrawref(**int *styles*[], Color *colors*[]);
Constructs a new draw reference with the *styles* parameter specifying an array of visual draw styles and the *colors* parameter specifying the corresponding colors to draw them in.

public int **layer**;
Layer to draw on.

public Color **colors**[];
Array of colors. They correspond to the styles array.

public int **styles**[];
Array of styles that compose this draw reference.

public boolean **contact**;
True indicates that this draw reference refers to a contact element.

public int **contacts**[];
List of contacted planes.

MAGIC Shell *(MAGICshell.java)* **version** 0.1

This is an interface class, that allows members of this package to obtain data from higher level classes and inform these classes about element changes.

public Dimension **getDimension**();
Returns the dimensions of a drawing area.

public Image **getImage**();
Returns an image for displaying.

public void **setPos**(int *pos*[][]);
Sets the position of the interface.

public void **typeOut**(String *s*);
Used to tell the interface to display the message *s*.

public InputStream **getStream**(String *file*);
Returns an input stream to retrieve data, specified by the *file* parameter.

public void **addElement**(MAGICelement *e*);
Used inform the interface of the addition of a new element, *e*.

public void **deleteElement**(MAGICelement *e*);
Used inform the interface of the deletion of a new element, *e*.

public String **typeIn**();
Gets an input string from the interface.

MAGIC Style *(MAGICstyle.java)* **version 0.1**

This class stores information pertaining to MGAIC style elements.

MAGICstyle(int *num*, int *mask*, int *color*, int *outline*, int *fill*, int *snum*, String *sname*, String *lname*);

Constructs a new style. The parameter *num* specifies the index of the style. The *mask* a parameter determines the mask layer. The *color* parameter is an index of a color. The *outline* parameter determines the outline of the element when drawn. The *fill* parameter determines the outline of the element when drawn. The *snum* is the stipple number. The *sname* and *lname* parameters specify the short and long names of a style respectively. The valid fill pattern types are:

solid stipple cross grid
outline

public int **getNum**();
Returns the index number of the style

public int **getMask**();
Returns the mask of the style

public int **getColor**();
Returns the index color number of the style

public int **getOutline**();
Returns the outline of the style

public int **getFill**();
Returns the fill pattern of the style. Valid types are listed in the constructor

public int **getStippleNumber**();
Returns the stipple index number.

public String **getShortName**();
Returns the short name of the style.

public String **getLongName**();
Returns the long name of the style.

MAGIC Tech Type *(MAGICtechType.java)* **version** 0.1

This class is just a data structure to hold technology types, layer information, and alias mapping.

public **MAGICtechtype(**String *layer*, String *name*, String *alias*);
Constructs a new tech type instance. The *name* parameter specifies the name of the technology. The *layer* specifies which layer it should appear on. A*lias* specifies mapping of the name to the base technology type

public String **layer;**
Returns the layer of the type.

public String **alias;**
Returns the alias of the type

public String **name;**
Returns the name of the type.

MAGIC Tech Style *(MAGICtechStyle.java)* version 0.1

This class is just a data structure to hold technology styles. These are composed of a style name and list of styles that compose the new style.

public **MAGICtechstyle(**String *name*, int *styles*[]);
Constructs a new tech style. The *name* parameter determines its name and the *styles* array contains the composing styles.

public String **name;**
Returns the name of the style.

public int **styles[];**
Returns an array of composing styles.

MAGIC Layer *(MAGIClayer.java)* **version** 0.1

This class is just a data structure to hold basic layer information. Basically it just a hashtable that stores the names of technology types that can be found on a given layer. It also stores the layer's name, as well as an on/off flag

public **MAGIClayer**(String *name*);
Constructs a new layer. The *name* parameter specifies the name of the layer.

public boolean **on**;
Returns true if layer is turned on, else false.

public Hashtable **tech**;
Returns a hashtable containing the elements found on this layer.

public String **name**;
Returns the name of this layer.

MAGIC Styles *(MAGICstyles.java)* **version** 0.2

This class parses a magic styles file from an input stream and stores style information for later retrieval.

public **MAGICstyles**InputStream *in*);
Constructs a new MAGIC styles instance. The *in* parameter specifies the input stream from which the styles file should be retrieved.

public void **parseFile**(InputStream *instream*);
Parses the MAGIC styles file, retrieved from the stream specifies by the *instream* parameter.

private MAGICstyle **parseStyle**(String *s*);
Private method used by the **parseFile** method to retrieve style information from an input line *s*. It returns the MAGIC style.

public MAGICstyle[] **getElements**();
Returns an array of the parsed MAGIC styles.

MAGIC Tech *(MAGICtech.java)* **version** 0.4

This class parses a magic technology file from an input stream into types, styles, contacts, and planes.

public **MAGICtech**(InputStream *in*);
Constructs a new MAGIC tech instance. The *in* parameter specifies the input stream from which the technology file should be retrieved.

public void **parseFile**(InputStream *instream*);
Parses the MAGIC technology file, retrieved from the stream specifies by the *instream* parameter.

public Enumeration **getTypes**();
Returns an Enumeration of all the parses element types.

public Enumeration **getStyles**();
Returns an Enumeration of all the parses styles types.

public Enumeration **getContacts**();
Returns an Enumeration of all the contacts.

public String[][] **getPlanes**();
Returns a doubly indexed String array. The first indexes the planes by number and the second indexes the elements found on that plane.

private void **parsetype**(String *s*);
Private method used by the **parseFile** method to retrieve element type from an input line *s*.

private void **parseplane**(String *s*);
Private method used by the **parseFile** method to retrieve a plane from an input line *s*.

private void **parsecontact**(String *s*);
Private method used by the **parseFile** method to retrieve a contact from an input line *s*.

MAGIC Parser *(MAGICparser.java)* **version** 0.4

This class parses a magic file from an input stream into elements used by other classes.

public **MAGICparser(**InputStream *in*, MAGICdrawmap *dmap*, MAGICshell *mshell*,
int *transform*[]);
Constructs a new MAGIC parser instance. The *in* parameter specifies the input stream from which the MAGIC file should be retrieved. The *dmap* specifies the draw map for element mapping. The *mshell* specifies an interface for executing commands. The *transform* stores coordinate transformations.

public **MAGICparser(**InputStream *in*, MAGICdrawmap *dmap*, MAGICshell *mshell*);
Constructs a new MAGIC parser instance. The *in* parameter specifies the input stream from which the MAGIC file should be retrieved. The *dmap* specifies the draw map for element mapping. The *mshell* specifies an interface for executing commands.

public **MAGICparser(**Vector *in*, MAGICdrawmap *dmap*, MAGICshell *mshell*);
Constructs a new MAGIC parser instance. The *in* parameter specifies the Vector from which the MAGIC file should be retrieved. The *dmap* specifies the draw map for element mapping. The *mshell* specifies an interface for executing commands.

public void **parseFile(**InputStream *instream*);
Parses the MAGIC file, retrieved from the stream specifies by the *instream* parameter.

public Enumeration **getElements(**int *i*);
Returns all the elements stored in the layer specifies by *i*.

public Vector[] **getVector**();
Returns a Vector of all the elements.

public int[][] **transform(**int *pos*[][]);
Returns the transform that should be applied to stored elements from an input position, *pos*.

public void **delete**(MAGICelement *m*);
Deletes the element specifies by *m*.

public void **add**(MAGICelement *e*);
Adds the element specifies by *e*.

private int[][] **getRECT(**String *s*, int *offset*);
Private method, that parses the rectangle position information from an input line *s*, starting at the specified *offset*.

private int[] **getTRANSFORM(**String *s*) {
Private method, that that parses the transform from an input line *s*.

private String **getTYPE**(String *s*);
Private method, that that parses the element type from an input line *s*.

private MAGICelement **getRLABEL**(String *s*) {
Private method, that that parses an label element from an input line *s*.

private void **initialize**(MAGICdrawmap *dmap*, MAGICshell *mshell*);
Private method, used a basic initialize for all constructors. The *dmap* specifies the draw map for element mapping. The *mshell* specifies an interface for executing commands.

public MAGICelement **makeCell**(String *name*, String *tag*) {
Converts the magic parser elements into a cell. The *name* specifies the name of the cell type and the *tag* species its unique identifier.

public static int **labellayer**;
Returns the layer on which labels are found.

public static int **celllayer**;
Returns the layer on which cells are found.

public static int **length**;
Returns the length of the elements array.

MAGIC Element *(MAGICelement.java)* **version** 0.3

This class store MAGIC elements. The elements can be of type rectangle, cell, or label.

public **MAGICelement(**int *type*);
Constructs a new MAGIC element. The type parameter species the type of element that this will be. Valid types are:

rect label cell

public void **makeRECT(**String *name*, int *position*[][]);
Makes the element of type rectangle. The *name* species the name of the element. The *pos* determines the dimensions of the element.

public void **makeRLABEL(**String *name*, String label, int *mode*, int *pos*[][]);
Makes the element of type label. The *name* species the name of the element. The *label* determines the tag of the element. The *pos* determines the dimensions of the element.

public void **makeCELL(**String *name*, String *label*, int *position*[][], int *trans*[]);
Makes the element of type cell. The *name* species the name of the element. The *label* determines the tag of the element. The *position* determines the dimensions of the element. The *trans* parameter is the coordinate transformation for the cell.

public void **setElements(**MAGICparser *elem*);
Sets the recursively stored elements to that specified by *elem*.

public MAGICparser **getElements**();
Returns recursively stored elements.

public int **getType**();
Returns the type of this element.

public String **getName**();
Returns the name of this element.

public String **getLabel**();
Returns the label of this element.

public int[][] **getPos**();
Returns the dimensions of this element.

public boolean **isOn**();
Returns true if this element is on, else off

public void **setOn(**boolean *on*);
Sets the on flag to that specified by the input parameter *on*.

public String **unparse**();
Returns a string representation of this element.

public int[][] **transform**(int *pos*[][]);
Returns the transform of the input parameter *pos* for this element.

public void **setTransform**(int *pos*[]) {
Sets the transform for this element as specified by the *pos* parameter.

public int[] **getTransform**();
Returns the transform for this element.

MAGIC layout Panel(*MAGIClayoutPanel.java*) **version** 1.1

This class inherits from the Panel Java AWT class. It contains an area for drawing a MAGIC layout. This is the main class of the MAGIC java files and handles all screen updates. This class may be used without the MAGICframe class to display MAGIC layout information, though it should still have some sort of wrapped, a MAGICshell, to control different modes.

public **MAGIClayoutPanel(**MAGICparser *elements*, MAGICcmap *cmap*,
MAGICdrawmap *dmap*, int *width*, int *height*,
MAGICshell *mshell*);

Constructs a new MAGIC layout panel. The elements to be draw are given in the *elements* parameter. The *cmap* and *dmap* parameters determine the draw map and color map respectively. The *width* and *height* specify the size of this panel. The *mshell* is an interface for executing commands.

public void **delElement(**MAGICelement *m*) {
Removes an element *m* form the appropriate layers.

public void **addElement(**MAGICelement *m*);
Adds an element *m* to the appropriate layers

public void **setElements(**MAGICparser *elements*);
Sets the elements for this layout as specified by the *elements* parameter.

public MAGICparser **getElements**();
Returns the elements for this layout.

public void **reset**();
Resets the view position to its default starting values.

public void **setMode(**int *newmode*);
Sets a new mouse mode for this panel. The modes are:

viewmode	calibmode	gotomode	centermode
openmode	idmode	deletemode	addmode
movemode	flipmode	rotmode	addcellmode
addlabelmode			

public int **getMode**();
Returns this panel's current mode of operation. The modes are given in the **setMode** method.

public void **setDetail(**int *newdetail*);
Sets a new detail level for this panel. The modes are:

lowdetail	middetail	highdetail

public int **getDetail**();
Returns the detail level. The detail levels are given in the **setDetails** method.

public void **setLabels**(boolean *lables*);
Sets a flag to turn the labels on or off, as specified by the *labels* parameter.

public boolean **getLabels**();
Returns true if the labels are turned on, else false.

public boolean **getGrid**();
Returns true if the grids are turned on, else false.

public void **setGrid**(boolean *grid*);
Sets a flag to turn the grids on or off, as specified by the *grid* parameter.

private void **center**(int *x*, int *y*);
Center the layout view at the position specified by (*x*,*y*).

private void **recenter**();
Recenter the center of the layout view to the center of the layout

public MAGIClayer[] **getLayers**();
Returns the array of MAGIClayers found in the drawmap.

public void **setLayermode**(boolean *lmode*);
Sets a flag to turn the layer mode on/off, based on the input *lmode* parameter.

boolean **getLayermode**();
Returns true if layer mode is on, else returns false.

public MAGICelement **getElement**(int *x*, int *y*);
Returns the element found at the position (*x*,*y*).

private void **getElementREC**(int *x*, int *y*, MAGICparser *elem*,
Vector *transforms*, Vector *found*);
Stores the elements found at (*x*,*y*) in the *found* Vector. Does a recursive search from the *elem* parameter. Performs the proper *transforms*. Ignores hidden layers.

public void **draw**();
Draws the layout and annotations to a buffered image.
private int[][] **getPosition**(MAGICelement *m*, Vector *transforms*);
Returns the transformed position of an element m, based on the *transforms* Vector.

private boolean **clipcheck**(int *pos*[][]);
Returns true if the given position *pos* of an element should be clipped

private boolean **drawRECTelement(**String *name*, Graphics *g*,
MAGICdrawref *df*, int *pos*[][]);
Attempts to draw the selected element to the graphics context. Returns true if it was drawn to the graphics context. This method only draws RECT elements, if another type is passed to it returns false. The *name* specifies the name/type of the element. The *g* parameter specifies the graphics context to draw to. *Df* is the draw map for the element. *Pos* is position.

public void **draw(**Graphics *g*, MAGICparser *elem*, Vector *transforms*);
Draws the layout to the specified graphics context *g*. The elements to draw are specified by *elem* parameter. The *transform* Vector stores the desired coordinate transformations.

private void **drawLABEL(**Graphics *g*, String *label*, int *pos*[][]);
Draw a text string *label*, with dimensions specified by the *pos* parameter, to the graphic context *g*.

private void **fillRECT**(Graphics *g*, int *pos*[][]) {
Draw a filled rectangle, with dimensions specified by the *pos* parameter, to the graphic context *g*.

private void **xRECT**(Graphics *g*, int *pos*[][]) {
Draw an "X", with dimensions specified by the *pos* parameter, to the graphic context *g*.

private void **dashNWSRECT**(Graphics *g*, int *pos*[][]);
Draw a northwest dashes, with dimensions specified by the *pos* parameter, to the graphic context *g*.

private void **dashSESRECT**(Graphics *g*, int *pos*[][]);
Draw a southeast dashes, with dimensions specified by the *pos* parameter, to the graphic context *g*.

private void **dashNSRECT**(Graphics *g*, int *pos*[][]);
Draw a north to south dashes, with dimensions specified by the *pos* parameter, to the graphic context *g*.

private void **dashEWRECT**(Graphics *g*, int *pos*[][]);
Draw a east to west dashes, with dimensions specified by the *pos* parameter, to the graphic context *g*.
private void **drawRECT**(Graphics *g*, int *pos*[][]);
Draw a rectangle outline, with dimensions specified by the *pos* parameter, to the graphic context *g*.

private void **waffleRECT**(Graphics *g*, int *pos*[][]);
Draw a waffle rectangle, with dimensions specified by the *pos* parameter, to the graphic context *g*.

private void **waffleLINE**(Graphics *g*, int *x1*,int *y1*, int *x2*, int *factor*);
Draws a waffle line, with dimensions (*x1*,*y1*) (*x2*,*y2*), to the graphics context *g* with the specified *factor*.

private void **basicTypes**(Graphics *g*);
Draws the basic types to the graphics context *g* for selection

public void **getSelected**(int *xpos*, int *ypos*);
Selects the element to be draw to the buffer during a draw command by that specified by the *xpos* and *ypos* parameters.

public int[] **LSgetCalibCoords**()
This method is used to retrieve the calibration coordinates of a layout shell. These coordinates are used to translate between different view points. The points are returned in a single index array, such that the interface is independent of dimensions.

public void **LSsetPos**(RMlayoutShell *layout*, int *points*[])
The current position of the layout interface is set through this method. The *layout* parameter specifies the class requesting a position change. The *points* parameter, given in the requesting classes coordinate plane, stores the desired location. Using internally stored calibration coordinates and those obtained from the requesting layout interface, through the LSgetCalibCoords() method, the position of the interface is set.

public void **LSsetShell**(RMlayoutShell *loShell*);
This method specifies layout interface, *loShell*, which should be used as a receiver for position change requests.

MAGIC frame(*MAGICframe.java*) **version 0.7**

This class inherits from the Frame Java AWT class and implements the **MAGICshell** interface.. It is used for viewing MAGIC files. This was really just meant to be an example frame that used a **MAGIClayoutPanel** to display magic files, but with a lack of time and effort this one is being included in the package. to make a more organized one I'm just using this one.

public **MAGICframe(**Image *img*, InputStream *mp*, InputStream *ms*, InputStream *mc*, InputStream *mt*);

Constructs a new MAGIC frame. The *img* parameter determines the title image. The *mp*, *ms*, *mc*, and *mt* input streams point to the base magic file, magic styles file, color map file, and technology file respectively.

public void **setShell(**RMlayoutShell *loShell*);

Sets the layout interface used for navigation.

public RMlayoutShell **getShell**();

Returns the layout interface used for navigation.

public void **setElements(**MAGICparser *elem*);

Sets the elements *elem* to be displayed in the layout panel.

public MAGICelement **getElement**() ;

Returns the elements displayed in the layout panel.

public String **typeIn**();

Returns the string stored in the information area.

public MAGICdrawmap **getDrawmap**();

Returns the drawmap that was calculated based on the original stream inputs.

public void **setBase(**String *s*);

Sets the base URL address, as specified by the *s* parameter, for allocating new streams.

public void **setName(**String *s*);

Sets the name of a new elements to *s*.

public Dimension **getDimension**();

Returns the dimensions in which the center layout panel should be drawn.

public void **setPos(**int *pos*[][]);

Sets the position.

public void **typeOut(**String *s*);

Displays a string in the information area

public Image **getImage**();
Passes the title image to other classes.

public Image **getImage**();
Returns an image for displaying.

public InputStream **getStream**(String *file*);
Returns an input stream to retrieve data, specified by the *file* parameter.

public void **addElement**(MAGICelement *e*);
Used inform the interface of the addition of a new element, *e*.

public void **deleteElement**(MAGICelement *e*);
Used inform the interface of the deletion of a new element, *e*.

private void **undoAction**();
Private method used to undo previous addition or deletion commands.

public void **setKillMode**(int *i*);
Specifies the "quit" option used. The *i* parameter can be 0 or 1, where 0 means hide frame and 1 means an exit from the program.

private void **buttonColor**(Button *b*, int *mode*, boolean *set*);
Sets the button color of *b*, based on the *mode* and *set* parameters.

MAGIC **URLdialog**(*URLdialog.java*) **version** NA

This private class inherits from the Dialog Java AWT class. It is a simple dialog to select new URL based files for input. New files are parsed and based to the MAGICframe

public **URLdialog**(MAGICframe *parent*, String *title*, boolean *modal*,
MAGIClayoutPanel *lpanel*);

Constructs a new URL dialog instance. The *parent* parameter determines the MAGIC frame to which information is sent. The *title* determines the title of the dialog box. The *modal* parameter specifies the mode. The *lpanel* parameter determines the draw layout area.

MAGIC ABOUTdialog(*ABOUTdialog.java*) **version** NA

This private class inherits from the Dialog Java AWT class. It is a simple about dialog with creator and version information.

public **ABOUTdialog**(MAGICframe *parent*, String *title*, boolean *modal*,);
Constructs a new about dialog instance. The *parent* parameter determines the MAGIC frame to which information is sent. The *title* determines the title of the dialog box. The *modal* parameter specifies the mode.

B.2 MAGIC Layout Use

The MAGIC layout classes are used to display and interact with MAGIC VLSI layout files. The main class used is the **MAGIClayoutPanel**. Higher level classes make use of this class to display the actual layouts. These classes must inherit from the **MAGICshell** class. The **MAGICframe** class, included in the package, is an example of such a class. This class and how it is used with the Remote Microscope and as an individual program is given below. However, any class implementing the **MAGICshell** class and using a **MAGIClayoutPanel** will suffice. F

The following code is from the initialization of the **MAGICframe** as a separate application.

```
...
String title_img = "images/majik.gif";                          (1)
Image i = Toolkit.getDefaultToolkit().getImage(title_img);      (2)
mp = new FileInputStream("magicfiles/palette.mag");             (3)
ms = new FileInputStream("magicfiles/magic.styles");            (4)
mc = new FileInputStream("magicfiles/magic.cmap");              (5)
mt = new FileInputStream("magicfiles/scmos.tech");              (6)

MAGICframe f = new MAGICframe(i, mp,ms,mc,mt);                  (7)
f.setKillMode(1);                                               (8)
f.show();                                                       (9)
...
```

The (1) title image location is specified and (2) loaded. The (3)-(6) required input files are turned into streams. The (7) **MAGICframe** is constructed and (8) the manner in which the application is exited is set. Finally the frame is displayed (9).

The initialization routine for when Majik Viewer is used by the Remote Microscope is very similar to the above, save it has an extra step. Within the **startup** method of the **RMmainFrame** class the layout navigation parameters are set.

```
...
magic.setShell(this);                                           (1)
layout = magic.getShell();                                      (2)
...
```

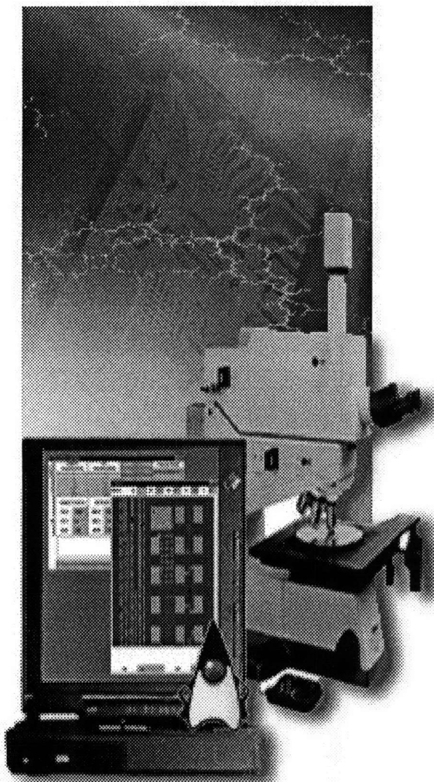
First the (1) layout used by the Majik Viewer's go to commands is set. Then the (2) layout used by the Remote Microscope's go to commands is retrieved.

Appendix C

Users Manual

This appendix contains a users manual for the Remote Microscope and the Majik Viewer. It includes information on how to start the client as an application or applet and it contains complete details on the function of each of the visual elements. Furthermore, it gives an example of the typical use of the Remote Microscope client.

REMOTE MICROSCOPE USER'S MANUAL



- Client Application and Applet Initialization
- Microscope Control
- Auto and Manual Focusing Options
- Majik Viewer Layout Navigation

Manuel Perez

Remote Microscope User's Manual

Documentation by: Manual Perez
Based on Original Documentation by: James Kao, Somsak Kittipiyakul, and Brian Lee.

Table of Contents

LIST OF FIGURES 201

INTRODUCTION 202

CLIENT APPLICATION INSTALLATION AND SETUP 203

CLIENT APPLET INSTALLATION AND SETUP 205

THE CONTROL WINDOW 207

THE IMAGE WINDOWS 210

AUTO FOCUS ALGORITHM 212

MANUAL FOCUSING 213

TOOLS WINDOW 216

LAYOUT NAVIGATION 219

TYPICAL USE 223

List of Figures

FIGURE 1 205
FIGURE 2 207
FIGURE 3 208
FIGURE 4 210
FIGURE 5 211
FIGURE 6 213
FIGURE 7 214
FIGURE 8 216
FIGURE 9 217
FIGURE 10 217
FIGURE 11 219
FIGURE 12 220
FIGURE 13 220

Chapter 1
Introduction

The Remote Microscope system allows clients to connect over the Internet to a remote server which controls an automated microscope. In this way clients are able to view wafers collaboratively for inspection and design.

Clients are presented with a control panel and two image windows which provide the client with both a panoramic view of the wafer as well as a magnified image which concentrates on a region of interest on the wafer. The client then can use the control panel to issue commands to the server (i.e., such as navigation of the wafer), and receive new microscope images.

Microscope Features:

- Manual or automatic focusing options
- Simultaneous displays of panoramic view and magnified view
- Multiple client capability
- Built in arbitration scheme insures only one controller of microscope at a time
- Visual aids such as pointers and rectangles which can be overlaid onto microscope images
- Real-time text client-to-client chat tool
- Layout based navigation

Chapter 2
Client Application Installation and Setup

This section details the installation and setup of a Java Remote Microscope Client, that is to be run as an application. If the software is already installed on your system or you wish to run the program as an applet, you may skip this section.

System Requirements

- Java Virtual Machine (JVM) version 1.0 or higher. If you do not have one, then one may be obtained from Sun Microsystem's Java Web site, found at "*http://java.sun.com*". (*Note: The Remote Microscope makes use of classes that may be removed from the JVM in future releases. It is safest to run under versions before 1.1*)
- TCP/IP Internet connection

Installation

To install the client application the user should unzip the distribution package "*rmapp09.zip*" to the local hard disk. This will create a directory called "*rmapp*" with all the required files and sub directories stored within it. Optionally the files can be installed individually.

Setup and Startup

To run the client application, the user must have the directory "." in the Java CLASSPATH. On an IBM PC running Microsoft Windows, this is accomplished by adding the following,

```
SET CLASSPATH=.;%CLASSPATH%
```

to the "*AUTOEXEC.BAT*" file. On a system running UNIX, this is accomplished by setting the CLASSPATH environmental variable as follows,

```
setenv CLASSPATH=.:
```

Once this is done, the application can be run from a command line prompt. To do this, the user should change to the "rmapp" directory and issue one of the following commands,

java **RMapp** <host> <port>
  - starts a new application with the specified <host> and <port>
java **RMapp** <port>
  - starts a new application with the specified <port> and default host
java **RMapp**
  - starts a new application with the default host and port

The default host is "*oolong.mit.edu*" and default port is "*9997*". Within a few moments a control window and two image viewing windows should appear.

Chapter 3

Client Applet Installation and Setup

This section details how to run the Java Remote Microscope Client as an applet. If you wish to run the program as an application, consult the previous section.

System Requirements

- Netscape 2.0+ or another Java enabled Web browser
- TCP/IP Internet connection

Installation

No installation of the client is required for the applet version of the client. All required files will be obtained from the Web server.

Setup and Startup

To run the client applet, the user will need to open up Netscape and make a connection to a Remote Microscope Web server. At the time of this writing, the only Remote Microscope Web server is located at "*http://oolong.mit.edu/rm/*". The user will be prompted for a user name and password. If these are entered correctly, the main page will be displayed in the browser as in Figure 1. (*WARNING: Passwords are NOT encrypted*)

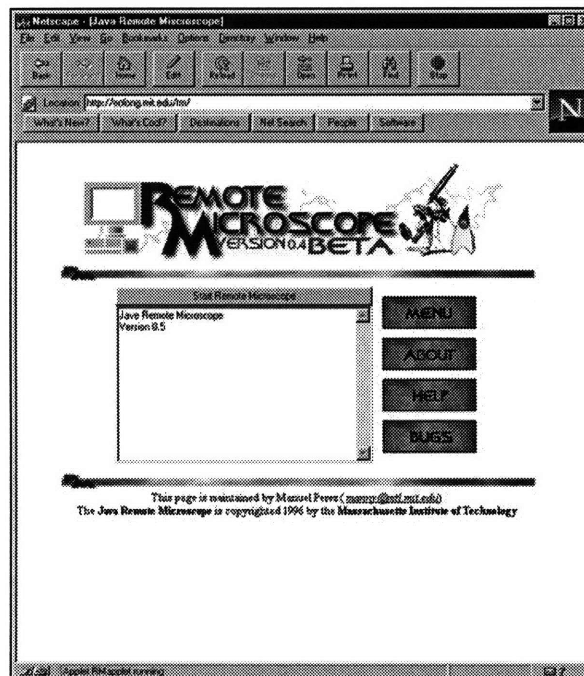


Figure 1

The main page consists of two main areas, the applet and the menu items. The applet is located to the left of the screen below the title. It is composed of a button labeled "*Start Remote Microscope*" and a text status area. Clicking on the button will attempt to start

the client. If all goes correctly, a control and two image viewing windows should appear. Whether this succeeds or not, information regarding the attempt will be placed in the text status area. The menu items are buttons that lead to various information areas about the Remote Microscope Client applet.

Chapter 4

The Control Window

The interface to the microscope is meant to be as intuitive and as user-friendly as possible. The control window appears as in Figure 2. Some options may be disabled; disabled options appear as grayed-out buttons.

The control window consists of four large columns, a title image to the far left, and a text chat area at its base.

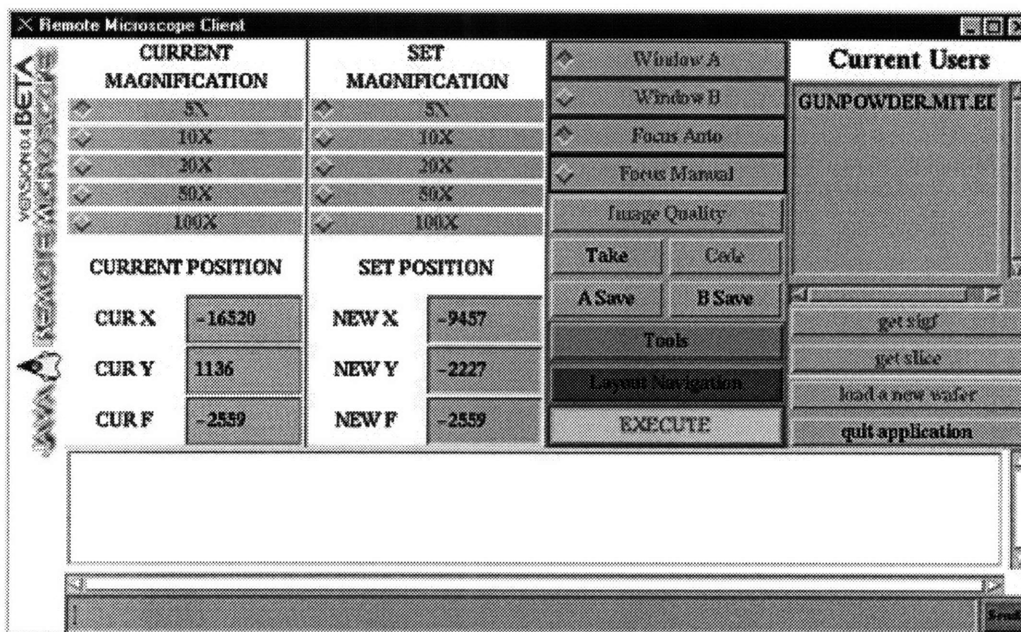


Figure 2

The first column displays the current status of the microscope: the magnification setting and the X, Y, and F positions. The X and Y coordinates correspond to the X and Y positions on the wafer, while the F position corresponds to the focus axis position of the stage.

The second column is used to set the magnification as well as the X, Y, and F positions. This can be done only if the client is in control of the microscope. The magnification is set by selecting one of the radio buttons in the "**SET MAGNIFICATION**" box. The X, Y, and F positions are set by entering the values into the text fields in the "**SET POSITION**" box. (*Note: The X and Y position coordinates are also set by clicking the left mouse on a desired point on one of the image windows, and the F position can also be set from the manual focus option windows.*) If automatic focusing is desired, it is not necessary to specify a position coordinate for the focus axis.

The third column consists of the following boxes:
Window Selection: selection (A or B) of window for next image capture
Focus Selection: selection (manual or auto) of focusing option
Take/Cede Buttons: **TAKE** is used to take control of the microscope. Note that this can occur only if no other client is currently in control of the microscope. **CEDE** is used to cede control of the microscope after one has taken control. This is typically done when the current client no longer wishes to control the microscope, and wishes to either exit the program or allow another client to control the microscope.
Image Quality: allows the client to select either gray-scale or color images. Note that there is no time savings of image acquisition for any given selection of image quality. This option also allows the client to request a new color map to be generated by the video frame grabber board on the server. The Image Quality Window appears as in Figure 3.

Figure 3

Save: if run as an application, allows the client to save the window A or B images to disk. If run as an applet, it saves the window A or B to the server and calls up a Web browser window containing the image for saving.
Tools: brings up the Remote Microscope Tools Window. The use of this window will be detailed in a later section.
Layout Navigation: brings up the Majik Viewer window. The use of this window will be detailed in a later section.
EXECUTE: moves the microscope to the new X, Y, and F coordinates, and grabs a new image. If the auto focus option is selected, the server will also attempt to auto focus the image.

The fourth column consists of the following boxes:
Current Users: displays current clients on the Remote Microscope system GET.
SIGF/GET SLICE: manual focus options. These options will be fully explained under the section "Manual Focus Options".
load new wafer: prompts server to load new wafer into the microscope.
quit: Exit the client program.

The title image normally displays the name and version of the Remote Microscope Client. However, when the server is busy capturing a new image, this panel will display a "*BUSY*" image.

The chat area can be used to send messages to other clients. Messages are sent whenever the "**send**" button is clicked or the "**ENTER**" key is pressed. A special message tag to set

the user name is supported. The format of this tag is "*/name <name>*", where <name> tag specifies the user name.

Chapter 5
The Image Windows

There are two image windows provided for the client. They appear as in Figure 4. For normal use, one window is used to display a panoramic view of the wafer, while the second window is used for magnified views of areas of interest. It does not matter which window is used for which purpose, but for the sake of description, it will be assumed that window A is the panoramic view and window B is the magnified view.

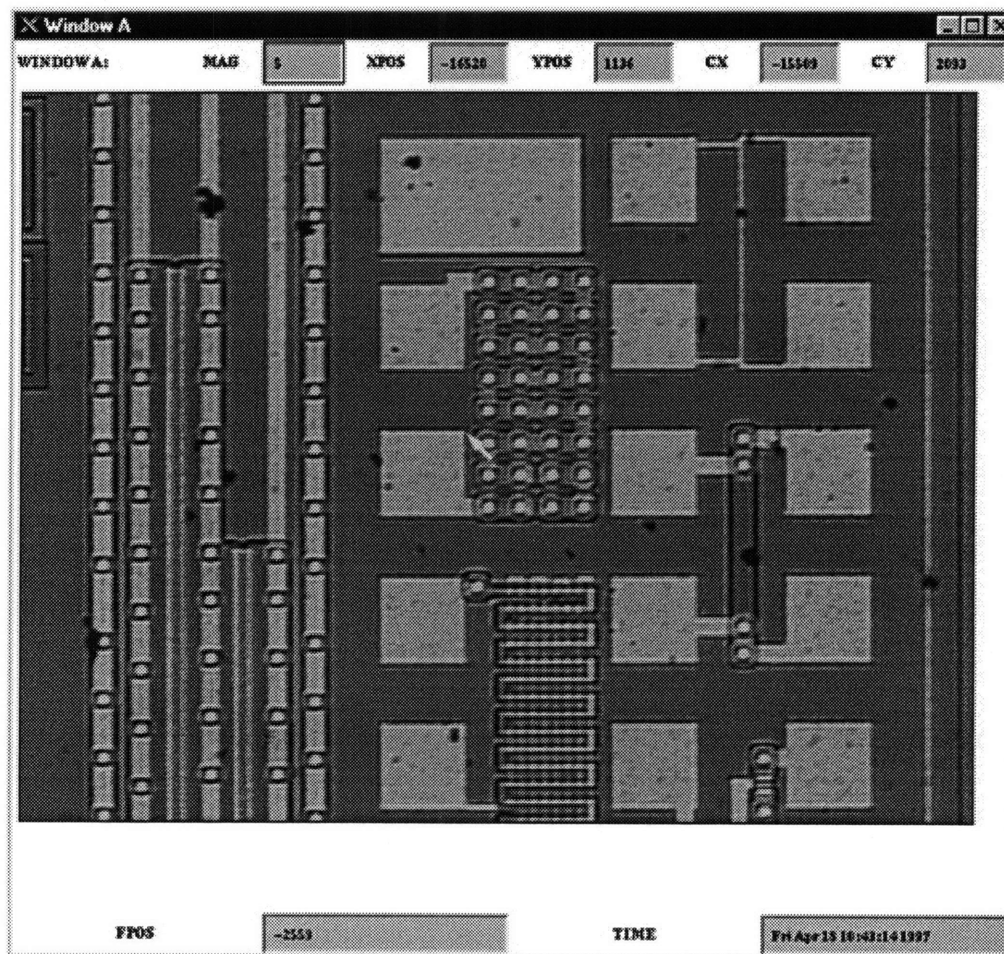


Figure 4

After the client captures a panoramic image of the wafer, the mouse may be used to designate new X and Y position coordinates by simply clicking on the point of interest on the wafer using the left mouse button.

Also, the right mouse button may be used to initialize special functions determined by the tools selected from the Tools Window. These functions will be covered in Tools Window section. By Default the "*set rectangle*" tool is selected.

When the Remote Microscope server is busy executing an image capture the destination image window will appear as in Figure 5.

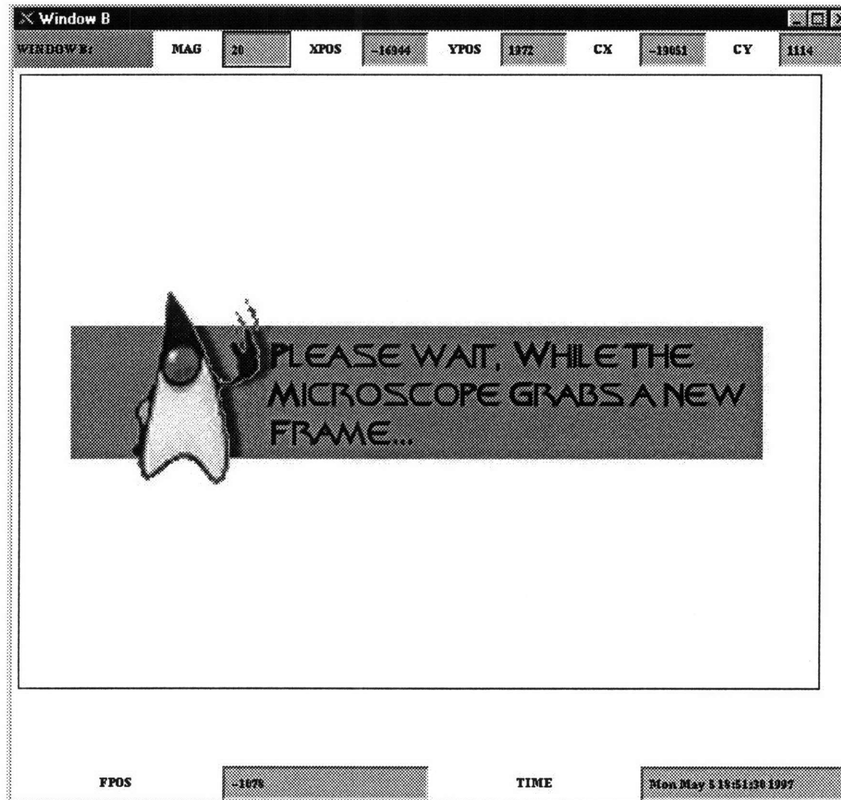


Figure 5

Chapter 6
Auto Focus Algorithm

The auto focus algorithm is a hardware-based method in which a series of scans across the focus axis is used to determine the maximum focus quality. Focus quality is measured by a quantity known as *SIGF*, which is calculated by high pass filtering the image, and integrating over 100 frames. The scan ranges and step sizes are optimized for each objective to maximize the chance of achieving a successful auto focus. For more information, see thesis "Automated Remote Microscope for Inspection of Integrated Circuits" [Somsak Kittipiyakul, 1996]

Chapter 7
Manual Focusing

The manual focus options are based on the assumption that there may be circumstances where the automatic focus algorithm will fail or produce undesirable results, such as focusing on a piece of dust on the wafer rather than the wafer itself. The object of the manual focusing option is to transmit a small quantity of image data to the client (i.e. less data than the data contained in an actual image) and let the client adjust the focus axis position, interpret the image data, and decide where the optimal focus position lies. This will allow the client several advantages over the auto focus option, such as focusing on a very specific area of the image. In addition, the manual focus option can eventually be developed into a second automatic focus option which will give the user a choice of auto focus methods.

There are two manual focusing options available to the client using the remote microscope: **SIGF** and **SLICE**. Note: The client must be in control of the microscope, and have the manual focus option selected, before manual focus options are available.

The image data in **SIGF** is the *SIGF* value mentioned in the Auto Focus Algorithm section. To use this option, the client selects the "**GET SIGF**" button and is presented with the **GET SIGF** manual focus window as shown in Figure 6.

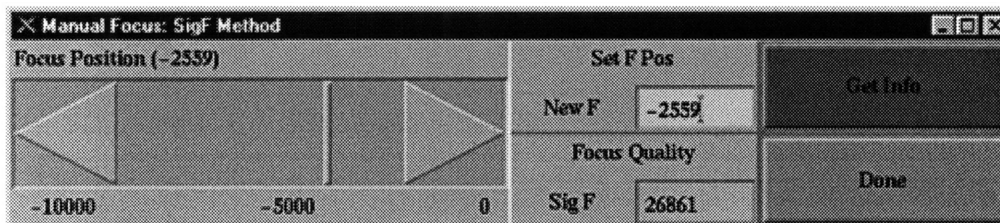


Figure 6

There are two ways the user can change the focus axis position: directly entering the new focus axis position into the text entry field and manipulating the slider bar. The slider bar may be manipulated in three ways: clicking the arrow buttons, clicking within the slider bar area, and by picking up and dragging the slider bar marker. The arrow buttons are used for fine adjustments: they move the focus position by one increment and then grab the appropriate data from the server. Clicking in the slider bar area is used for coarse adjustment; the focus position is moved by 100 increments and then grabs the appropriate data from the server. Picking up and dragging the slider bar marker allows the highest degree of control: the user can move the marker to any focus position and then click on the "**Get Info**" button to grab the appropriate data from the server.

The goal of this focus option is to manipulate the focus axis position until the value of *SIGF* is a maximum. After achieving a maximum value for *SIGF*, the client then selects the "**EXECUTE**" button on the control window to grab the focused image.

The second manual focus option is the **SLICE** option. Selecting the "**GET SLICE**" option brings up the following window as shown in Figure 7.

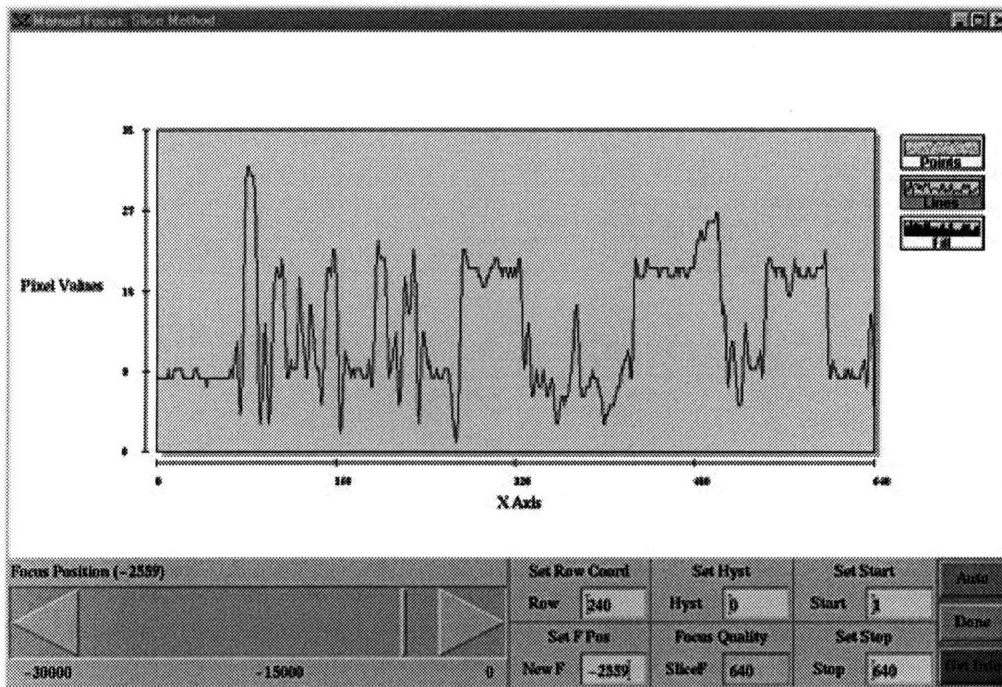


Figure 7

The image data which is sent to the client is a horizontal scan line which represents the pixel values of a particular row of the image (on a scale of 0 to 32). The data is taken from the image captured in the frame grabber board on the server, based on the command and parameters issued by the client. The basic premise here is that a focused image will have sharp contrasts in pixel values, lots of edges and transitions, while a poorly focused image will have smaller edges and less sharp transitions. By adjusting the focus axis position, and setting the focus parameters, it is possible to achieve the optimal focus position. The methods of adjusting the focus axis are the same as for the **SIGF** method.

There are several parameters which can be set by the user:
ROW COORD: the row of the image where the scan line information should be taken. As a guide, a pair of red lines will appear in the image window denoting the row which is currently valid.
HYST: the client can specify a hysteresis value which can be used to eliminate small transitions which should not be significant in the consideration of a focused image. For instance, in a focused image, it is more important when two consecutive pixel values have a difference of perhaps five or more, since this signifies sharp transitions. Therefore, by setting the hysteresis value to five, the smaller transitions can be filtered out.

SLICEF: a measure of focus quality. This value is calculated by summing the first derivatives of the scan line data. Specifically, it is the sum of the absolute values of the differences of each pixel value and the next pixel value.
START: set the starting position for the row of scan line data computation
STOP: set the end position for the row of scan line data. When used in conjunction with START, these parameters specify a certain region of data to be used in the calculation of SLICEF. In this way, it is possible to concentrate focusing on only a particular feature of the wafer.
auto: automatic focusing using this method.
GET INFO: a request for scan line data is usually sent only when the user changes the focus axis position. However, if the user has changed the start/stop values, or the hysteresis value, and wants to get pixel value data for the current focus axis position, then this button should be selected.
DONE: exits the **GETSLICE** manual focus window.

Chapter 8
Tools Window

The tools window appears as in Figure 8. The topmost section of the window contains a set of tools, which determine the functionality of the right mouse button within an image window. The available tools are "*go to layout position*", "*calibrate*", "*grab image at*", "*set rectangle*", "*measure distance*", "*decrease magnification and grab*", and "*increase magnification and grab*". The currently selected tool will appear as a depressed button.

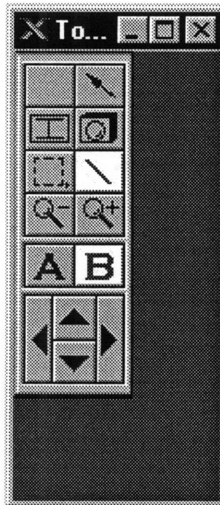


Figure 8

The "*go to layout position*" and "*calibrate*" are used to interact with the layout navigation tool. The "*go to layout position*" tool is located at the top right corner of the tools and appears as an arrow. It is used to tell the client to move the layout's view to the location specified by a right mouse click within an image window. The "*calibrate*" tool is located in the second from the top row to the far left and appears as the letter "I" surrounded by a rectangle. When this tool is selected, the right mouse button can be used to generate a calibration rectangle. The use of this rectangle will be discussed in the layout navigation section.

The "*grab image at*", "*decrease magnification and grab*", and "*increase magnification and grab*" tools are used to execute image grab request from directly within the image windows. When one of these tools is selected a right mouse click will send a request to the server for a new microscope image at the specified location. In the case of the "*decrease magnification and grab*" and "*increase magnification and grab*" tools, the grabbed image will have a decrease or increase in magnification. The "*grab image at*" tool is located in the second from the top row to the far right and appears as a camera. The "*decrease magnification and grab*" tool is located at the bottom of the tools section to the left and appears as a magnifying glass with a minus sign next to it. The "*increase magnification and grab*" tool is located at the bottom of the tools section to the right and appears as a magnifying glass with a plus sign next to it.

The "measure distance" tool is used to measure the distance between any two points on a microscope image. It is located in the second from the bottom row to the right and appears as a line. An example of the "measure distance" tool being used within an image window is shown in Figure 9. The starting point of the measurement is specified by a right mouse button click and the ending point is selected by the mouse button's release.

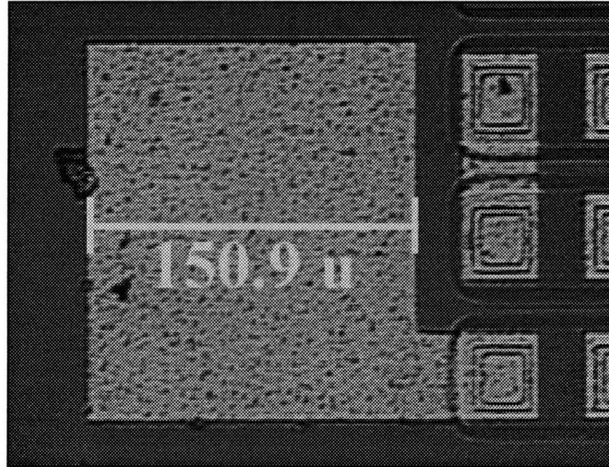


Figure 9

The "*set rectangle*" tool is the default tool for the right mouse button within an image window. If the tools window is not open it is the only usable tool. The "*set rectangle*" tool allows the user to draw a rectangle to the image window. This rectangle will be visible on all other clients and is used primarily to point out areas on a wafer. An image of the rectangle tool in use is given in figure 10. The rectangle tool is located in the second row from the bottom to the left and appears as a rectangle composed of dashed lines.

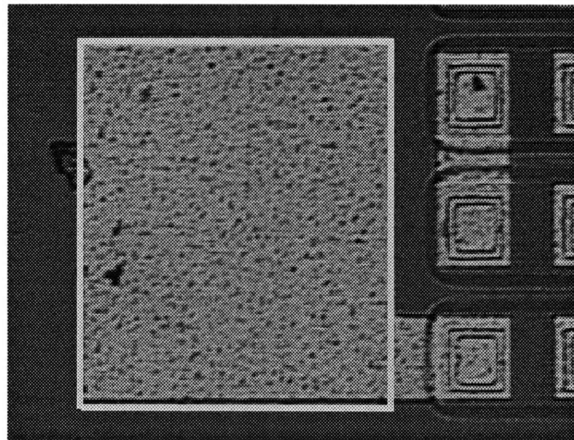


Figure 10

The next section of the window contains two buttons, distinguished by the letters "*A*" and "*B*". These buttons are used to specify to which windows image grabs are to be sent. In most cases the image window that is currently being acted upon by a tool should be selected. The selected window will appear as a depressed button.

The last section contains a set of quick navigation buttons. These appear as north, south, east, and west arrows. Upon pressing any given one, an execute command will be sent to the Remote Microscope server with a position relative to the previous position plus an increment determined by the button type.

Chapter 9
Layout Navigation

The Remote Microscope client uses the Majik Viewer for layout based navigation. The Majik Viewer is a fully functional MAGIC VLSI layout viewer. It appears as in Figure 11.

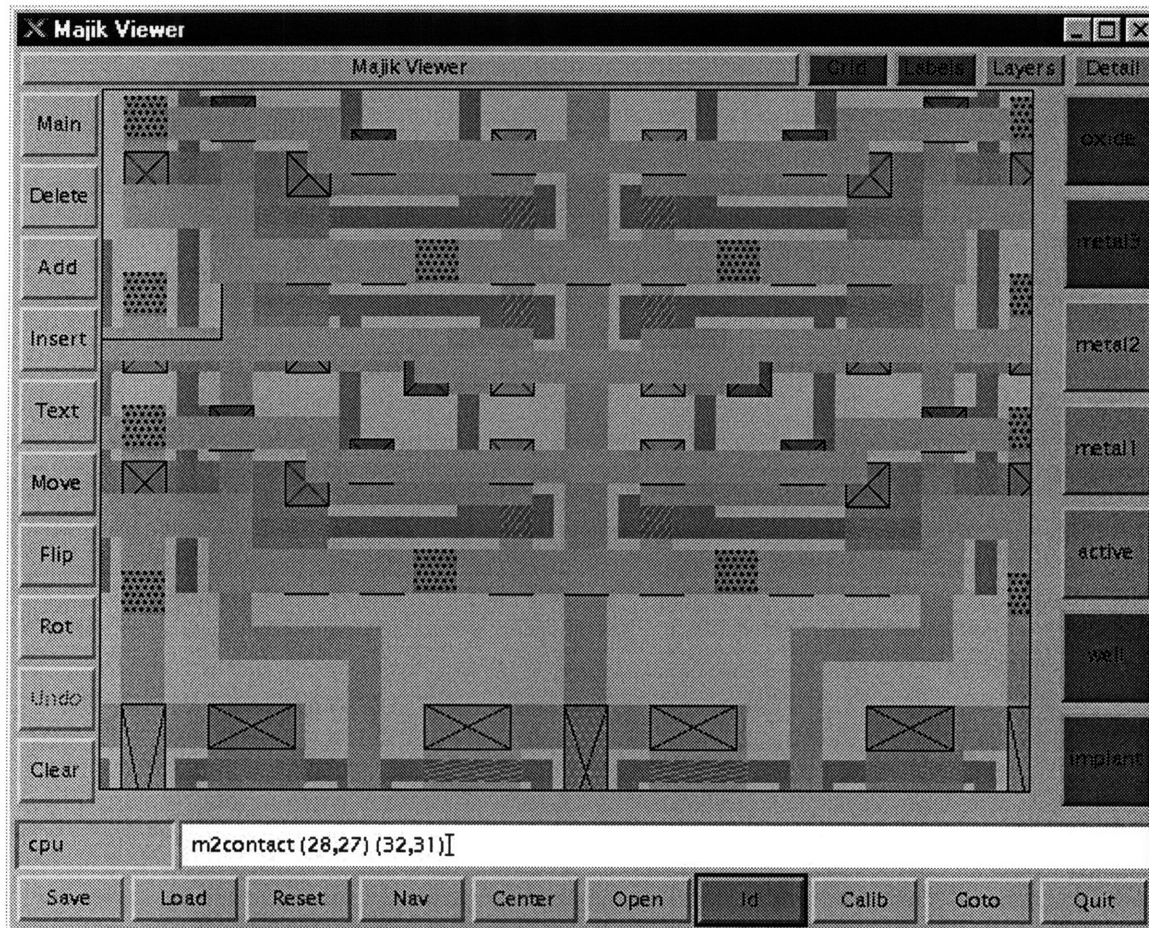


Figure 11

Loading a Layout File

The first step in preparing to use the Majik Viewer is loading a layout file. By clicking on the "**Load**" button, a "**URLdialog**" window will appear as in Figure 12. From here, the user can specify the Internet address of the layout file that corresponds to the wafer that is being viewed. Clicking the "**OK**" button will issue an attempt to grab the file; errors will be reported in the large text area. The "**CANCEL**" button can be used to cancel the request.

Figure 12

Calibration

The next step, in preparing for layout navigation, is the calibration of the Remote Microscope Client and the Majik Viewer. This is done by using the calibration tools in each of the respective programs to select a common area on both.

The calibration tool in the Remote Microscope Client is found in the Tools Window. After it is selected, the right mouse button can be used to select an area by dragging a calibration rectangle. Figure 13 shows an example of such a rectangle.

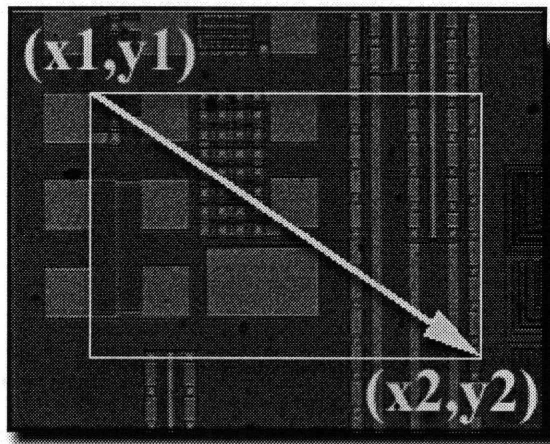


Figure 13

The calibration rectangle will remain on the screen until another tool is selected and one of the mouse buttons is clicked in one of the image windows.

The calibration tool in the Majik Viewer is selected by clicking the "**Calib**" button. After it is selected, the left mouse button can be used to select an area by dragging a calibration rectangle. The same area as in the Remote Microscope Client MUST be selected. The calibration rectangle will remain on the screen until another tool is selected.

Navigation

From this point on the user can use the Majik Viewer to navigate around the wafer or vise versa. From within the Remote Microscope Client, when the "*go to layout position*" tool is selected, a left mouse click will move the center of the Majik Viewer to the same position as the position of the mouse click. From the Majik Viewer, when the "**Goto**" button is selected, the same process can be performed in reverse. By clicking the left mouse button, a command will be sent to the server to grab a new image at the position of the mouse down. The new image will be displayed in an image window.

Using the Majik Viewer

The above information has revealed nothing about how the Majik Viewer is actually used. This section will briefly examine its main functionality.

The main control area is located at the bottom of the Majik Viewer window. It consist of a set of buttons to control basic interactions with the layout. The buttons and their functions are:

Save: is not yet implemented.
Load: brings up a dialog window to load a new layout, discussed earlier.
Reset: centers the layout view and changes it back to its base magnification.
Nav: enters basic navigation mode. In this mode the left mouse button can be used to pick up and move the layout, and the right mouse button can be used to change magnification. Magnification is increased by right clicking, dragging the mouse to the right, and then releasing. It is decreased by right clicking, dragging the mouse to the left, and then releasing.
Center: enters center navigation mode. In this mode the left mouse button can be used to define a new center point of the viewing area, and the right mouse button can be used in the same manner as described for the "**Nav**" button.
Open: enters open mode. In this mode the left mouse button can be used to open subcell elements, and the right mouse button can be used to close them.
Id: enters identification mode. In this mode the mouse buttons can be used to identify elements of the layout. Information appears in the text area just below the layout.
Calib: is used for calibration, discussed earlier.
Goto: is used for Remote Microscope navigation, discussed earlier.
Quit: quits the application.

Just above the main control buttons are two text areas. The first of these has gray background and displays the name of the file being viewed. The second has a white background and is used to relay information back to the user, when elements are being identified.

At the top of the Majik Viewer are four colored buttons. These buttons are used to select detail levels for the layout. The buttons and their functions are:

Grid: (red) grid lines are off and (green) grid lines are on.

Labels: (red) no visible labels and (green) all labels visible
Layers: (red) view all layers and (green) only view specified layers. Layers are specified using the rightmost set of buttons.
Detail: (red) lowest, (yellow) medium, and (green) highest detail level. The lower the detail level the less visually pleasing the layout becomes and the harder it gets to identify elements in the layout, but the faster the redraw speed gets.

To the right of the Majik Viewer are several colored buttons. These buttons correspond to planes in the layout. When the layers button (see above) is green, these buttons are used to determine which plane layers are to be viewed. Red indicates that this plane is not viewed and green indicates that it is viewed.

To the left of the Majik Viewer lie a collection of editing buttons. These buttons are not fully functional and will not be discussed in this manual.

Chapter 10
Typical Use

A typical use of the microscope would be as follows:

1. Start up client program (see Remote Microscope Start-up Instructions).
2. Select Take. Note several options become enabled.
3. Select "auto focus"; select "window A"
4. Select "EXECUTE"; wait for image to be returned. It should be returned in window A.
5. Click on point of interest on window A (left mouse button). Note the change in the values in the NEW X and NEW Y fields on the control panel.
6. Select "20X" on the "SET MAGNIFICATION" box. Note the yellow rectangle which appears in window A, showing the area of the image which would be seen at the selected magnification.
7. Select "window B" in the window selection box.
8. Select "EXECUTE"; wait for image to appear in window B.
9. Repeat process at different point on window A and different magnifications.