Magman Login Simulator

Gilbert Baumslag, Timothy Daly

June 15, 2007

Abstract

Finitely presented groups can be constructed so that arbitrarily complex questions can be asked which may be difficult or impossible to answer. This gives us the potential to construct an extremely strong challenge-response crypto system.

We develop a simulator program to support research efforts to use finitely presented groups in a challenge-response authentication system intended to replace passwords.

Contents

1 Magman: Test Login Encryption Protocols

1.1 Concept

Finitely presented groups can be constructed so that arbitrarily complex questions can be asked which may be difficult or impossible to answer. This gives us the potential to construct an extremely strong challenge-response [1] crypto system.

Since there are infinitely many groups and there are methods of encoding information within the group structure it is possible to give each person their own fintely presented group. Challenge questions can be drawn from the group structure which only someone knowledgable about the group structure can answer.

Magnus is a local computer algebra program capable of manipulating finitely presented groups and answering questions about group and element properties. This research is focused on the possibility of using Magnus to answer challenge-response queries and thus form the basis of a password replacement system.

1.2 Prior Work

Wagner [3] constructs a Public Key Cryptosystem based on the Word Problem. As it presents valuable background material, the section from [2], pp109-114 is quoted exactly here:

1.2.1 Wagner's Word Problem PKC Proposal

In this survey of recent PKC Proposals, the purpose has been to give the reader some sense of the type of mathematical model that can be used for the development of a PKC.

In concluding this section, we present an idea due to Wagner [WAGN 85], which uses a mathematical problem of a significantly different type than others which we have used to date.

Much of our focus to date has been on the category of NP-complete problems (See Appendix IV), the set of problems which probably do not have a polynomial-time algorithm for their solution. These are the problems we consider extremely "hard" to solve.

However, in the category of all problems, there is another class, to which we can easily award the distinction of being the "hardest". The category U of **undecidable** problems consists of those problems, for which *it can be proven that there exists no algorithm that can solve them.*

In general, it is a difficult task to prove that a problem belongs to this category U. Indeed, there are only a few known examples:

> the **Turing Halting Problem**: given an arbitrary computer program, and an arbitrary input to that program, is there an algorithm which will decide whether or not the program will eventually halt when applied to that input?

Hilbert's Tenth Problem: given an arbitrary polynomial equation $P(x_1, x_2, \ldots, x_n) = 0$, with integral coefficients, to determine whether or not it has an integral solution;

given a context-free grammer G, is there a positive integer k such that G is an LR(k) grammer?

a **tile** is a unit square colored on each of its four sides. A **tiling** is an arrangement of tiles so that any two common sides have the same color. Is there an algorithm which will decide, for any set of colors, C, and any collection of tiles T contained in C^4, whether or not the plane can be tiled with this collection?

the **word problem for finitely presented groups**.

Wagner's approach relies on the last of these undecidable problems, so we will discuss it further.

1.2.2 Finitely Presented Groups

A **Group**, defined in Appendix II, can be described in terms of **generators** and **relations**.

To say that a group G is **generated** by elements $g_1, \ldots, g_n$ implies that every element of G can be written (not necessarily uniquely) as:

$$g_{i1}^{p_i 1} g_{i2}^{p_i 2} \cdots g_{in}^{p_i n}$$

where each g_{ij} is one of the generators, and p_{ij} is an integer representing a power of g_{ij}. This string is referred to in group-theoretic terms as a **word**.

A **relation** r in G is an equation relating a word in G to the identity, e:

$$r : g_{i1}^{p_i 1} g_{i2}^{p_i 2} \cdots g_{in}^{p_i n} = e$$

Sometimes an abuse of language will occur, and the expression on the left-hand side will also be referred to as the relation.

To say that a group is **finitely presented** means that it is completely described by a finite number of generators and relations.

EXAMPLE 1:

$$G = \{\text{generator } g; \text{relation } r : g^5 = e\}$$

.

It is not difficult to see that the elements of G are $\{e, g, g^2, g^3, g^4\}$ and that it is in fact isomorphic to $<\mathcal{Z}_5, +>$.

EXAMPLE 2:

$$G = \{\text{generator } g_1, g_2; \text{no relations }\}$$

.

This group is an infinite group known as the **free group on two generators** . Some of the elements are: e, g_1, g_2, g_1g_2, $g_1g_2g_1$, $g_1g_2g_1g_2$, $g_1^2g_2$, ...

EXAMPLE 3:

$$G = \{\text{generator } g_1, g_2; \text{relations } r_1 : g_1^2 = e; r_2 : g_2^3 = e; r_3 : g_1g_2g_1g_2^{-2} = e\}$$

.

This group consists of six elements and is isomorphic to S_3, the symmetric group on three elements. The proof is left as an exercise.

Two words v, w in G are said to be **equivalent** if v can be transformed to w by a finite sequence of replacements (known as **Tietze transformations**). These are:

1. Replacing $x_jx_j^{-1}$ or $x_j^{-1}x_j$ by e, the identity.
2. Introducing $x_jx_j^{-1}$ or $x_j^{-1}x_j$ at any point in the word
3. Changing relations r_j or r_j^{-1} to e for any relation r_j
4. Introducing r_j or r_j^{-1} at any point

The **word problem** for a group G is the problem that asks, for each word w in G, if that word is equivalent to the identity.

It is one of the most remarkable results of modern mathematics, due to Boone [BOON 59]and Novikov [NOVI 55]that there exist specific groups in which the word problem is undecidable:

THEOREM (BOONE-NOVIKOV): There does not exist an algorithm which can solve, for all finitely presented groups, G, the word problem for G, that is whether or not each word $w_1w_2 \ldots w_n$ in G is equivalent to the identity element.

Furthermore, the Boone-Novikov theory constructs such groups for which the word problem is undecidable.

1.2.3 Proposal for a Word Problem PKC

PROPOSED CRYPTOSYSTEM:

SYSTEM GENERATION:

Find G, a finitely presented group, generators x_i, relators r_j, for which the word problem is undecidable.

Find as well, w_1, w_2, two inequivalent words in G.

Find a secret homomorphism $H : G \to G'$ such that H and G' will remain secret, and G' has an efficiently computable word problem.

The public key consists of G, w_1, w_2. The secret key is H and G'.

ENCRYPTION:

To encrypt a bit, use the bit to choose one of w_1, w_2 and randomly apply a sequence of Tietze transformations. Obtain v, equivalent to either w_1 or w_2. Transmit v.

DECRYPTION:

Solve the word problem for $H(v)$ in G' to decide whether it is equivalent to $H(w_1)$ or $H(w_2)$. Consequently, determine the value of the bit that was encrypted.

The weakness that prevents this cryptosystem from being practical is the following. To determine a "random sequence of Tietze transformations", we must be prepared to accept that the transformed word may grow to an arbitrary length. This, of course, renders the cryptosystem less than practical, but any limitation of the space from which the Tietze transformations may be chosen would in fact violate the conditions of the Boone-Novikov theory, and the resulting sub-problem would not be undecidable.

Nevertheless, although this method is not useful, it does indicate a new direction for further PKC research.

1.2.4 Constructing a Finitely Presented Group in Which No Solution to the Word Problem Exists

First consider subsets, S, of the integer lattice $\mathcal{Z}^n$:

S is **Diophantine** if there exists a polynomial $P(X_1, X_2, \ldots, X_n; Y_1, Y_2, \ldots, Y_m)$ such that

$(s_1, \ldots, s_n)$ is in $S \Leftrightarrow P(s_1, s_2, \ldots, s_n; Y_1, Y_2, \ldots, Y_m)$ has an integer root.

Say that P **enumerates** S.

CONSTRUCTION: Assign positive integers to polynomials.

Let $\Omega : \mathcal{Z} \rightarrow \mathcal{N}$ be

$$\Omega(z) = \begin{cases} 2|z| + 1 & \text{if } z \leq 0 \\ 2|z| & \text{if } z > 0 \end{cases}$$

Let $X_0, X_1, \ldots, X_n, \ldots$ be a list of variables. Assign to each monomial term

$$T = cx_{i1}^{e1} \cdots x_{in}^{en} \quad (c \neq 0 \text{ in } \mathcal{Z}, \text{ each } e_i \geq 1, \text{ and the } i_j\text{'s an increasing seq.})$$

The number

$$\beta(T) = 2^{\Omega(C)} p_{(i1+2)^{e1}} \cdots p_{(in+2)^{en}}$$

where p_j is the j^{th} prime.

e.g.:

$$\beta(5x_0^3 x_4^2) = 2^{10} 3^3 13^2.$$

If $P = T_1 + \cdots + T_k$, then define the Gödel numbers:

$$\partial(P) = 2^{\beta(T_1)} \cdots p_k^{\beta(T_k)}$$

and

$$S = \{\partial(P(X_{i1}, \ldots, X_{in})) | P(e, X_{i2}, \ldots, X_{in}) \text{has a root when e} = \partial(\text{P})\}$$

Then

$$G = \{a, b, c, d | a^{-i} b a^i = c^{-i} d a^i, \text{for all i in S}\}$$

is a group which has no solution to the word problem.

NOTE: G is not finitely presented; however, using the Higman Embedding Theorem, it can be embedded in, i.e., is a subgroup of, a finitely presented group, and therefore that group is a finitely presented group for which no solution to the word problem can be found.

End of quote.

1.3 Motivation

In order to focus our attention on the practical difficulties we have decided to build a program to test a variety approaches using Magnus for encryption. As there are many issues to resolve we decided to prototype the ideas. We need a program that is small and flexible so we can quickly mimic the expect interactions.

In particular, we are looking at the problem of developing a replacement for the standard login password mechanism. Rather than modify a real operating system we've chosen to create virtual terminals and virtual users.

2 The Model

In our model there are 3 parties: Alice, the person who wishes to login, Bob, the computer she wishes to use and Eve, a potential attacker. Each entity has a corresponding class with expected behavior. So there is a Person class, of which Alice is an instance. There is a Computer class, of which Bob is an instance and there is an Attacker class, of which Eve is an instance. All of these are subclasses of the general class we call an Actor.

Alice and Bob communicate using Message class objects. Messages are a superclass of two other objects, the Plaintext class and the Ciphertext class. Plaintext Messages are assumed to be string over an alphabet **P** that is human readable. Ciphertext Messages are assumed to be string over a second alphabet **C** that is not required to be human readable.

There is a third alphabet, the Key alphabet **K** which has some internal representation (a finitely presented group) from which we can generate strings for keys.

Finally, there are two Transformation functions, Encrypt **E** and Decrypt **D**. Encrypt is a function that takes a Message and a Key and returns a Message: $\mathbf{E(M,K) = M}$. Usually we expect the input to be a Plaintext Message and the output to be a Ciphertext Message, thus: $\mathbf{E(P,K) = C}$ but this is not required.

Decrypt is a function that takes a Message and a Key and returns a Message: $\mathbf{D(M,K) = M}$. Usually we expect the input to be a Ciphertext Message and the output to be a Plaintext Message, thus: $\mathbf{D(C,K) = P}$ but this is not required.

So we can describe our system with the 4-tuple [2] which defines the Keys, the Plaintext, the Ciphertext and the Transformations (Encrypt and Decrypt), thus $\mathbf{S} = (\mathbf{K}, \mathbf{P}, \mathbf{C}, \mathbf{T})$.

Each of the three Actor classes has an associated Console class. The Console class allows us to either watch the messages from the respective viewpoints or to take control and interact with the messages.

While the program is executing any or all of the available Consoles exist on the screen. They show all Message traffic. Consoles accept commands so the Message traffic can be manipulated, for example, by encrypting, logging, deleting or modifying the traffic in some way.

The three Actors are linked by a Channel class that represents the path over which Messages flow. The Channel can be connected from any Actor to any other Actor. By default it connects Alice thru Eve to Bob. Thus, Eve sees all of the Message traffic in both directions.

3 Example session

3.1 Session between Alice and Bob

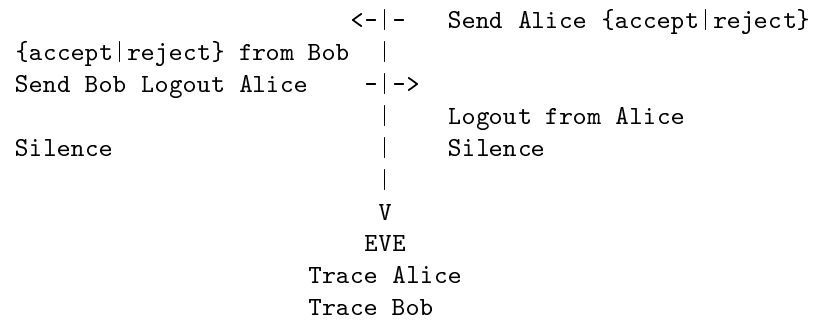
```
ALICE                          BOB

Listen                         Listen
Send Bob Login Alice     --->
                                Login from Alice
                         <---   Send Alice Question X
Question from Bob X
Send Bob Answer X.1       --->
                                Answer from Alice X.1
                         <---   Send Alice {accept|reject}
{accept|reject} from Bob
Send Bob Logout Alice     --->
                                Logout from Alice
Silence                         Silence
```

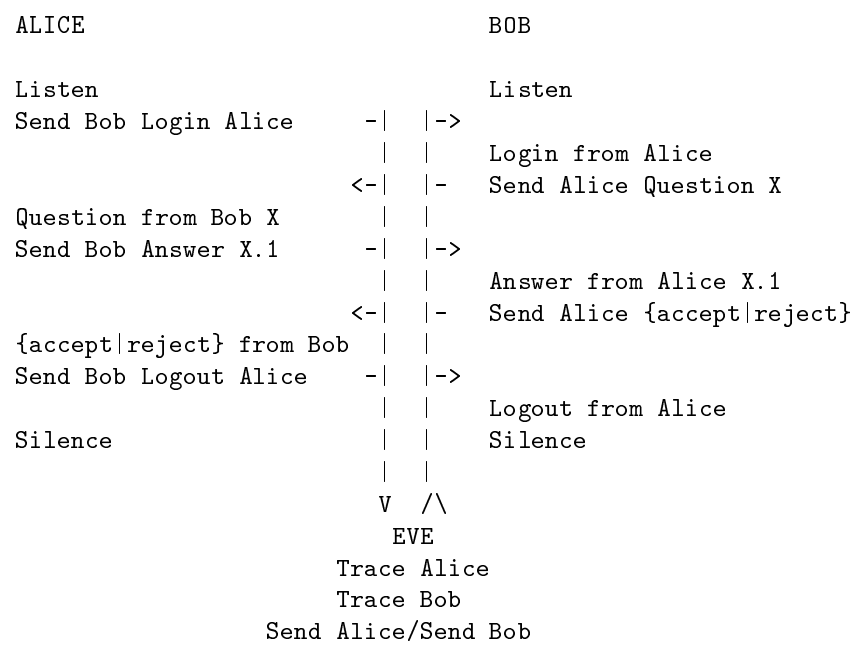
3.2 Session with Eve listening

```
ALICE                          BOB

Listen                         Listen
Send Bob Login Alice     -|->
                          |     Login from Alice
                         <-|-   Send Alice Question X
Question from Bob X       |
Send Bob Answer X.1      -|->
                          |    Answer from Alice X.1
```



3.3 Session with Eve attacking



4 Development decisions

In this section we explain the various development decisions.

In order for the parts of the program to communicate with the user we need to develop the Console class first. This will give Alice, Bob and Eve a way to accept commands and show output. See the section on the Console for detailed design decisions.

5 The Implementation

5.1 The Main program

5.1.1 Magman Class

Magman is an implementation of the Mediator Pattern [4]. The various Actors in the system do not communicate with each other directly. Magman sets up and controls all of the communication.

First Magman sets up a Console object for its own use. This is useful for tracing and logging purposes. You can also run test scripts from this console for test or demonstration purposes.

The Magman class sets up the multiple threads that represent each Actor in the system. It creates a Console for each Actor, initializes it, then creates the Actor and gives it a print name and its own Console object.

Each Actor is initialized.

Finally, we listen and talk on our Console, executing commands until we receive a quit command.

⟨*Magman.java*⟩≡

```
package com.ccny.magman;

public class Magman
{

  private static void runScript()
  { Catch.log("Magman:runScript","Enter");
    Catch.log("Magman:runScript","Exit");
  }

  public static void main(String[] args)
  { Catch.log("Magman:main","Enter");
    Console Magman = new Console("Magman",null,null);
    Magman.init();
    Console EveTerm = new Console("EveTerm",null,null);
    EveTerm.init();
    Person Eve = new Person("Eve",EveTerm);
    Console BobTerm = new Console("BobTerm",null,null);
    BobTerm.init();
    Person Bob = new Person("Bob",BobTerm);
    Console AliceTerm = new Console("AliceTerm",null,null);
    AliceTerm.init();
    Person Alice = new Person("Alice",AliceTerm);
    Eve.init();
    Bob.init();
    Alice.init();
```

```
    runScript();
    Catch.log("Magman:main","Exit");
  }
}
```

5.2 The Console

5.2.1 Console Class

The Console gives Alice, Bob and Eve a way to communicate. In particular, they need the ability to accept a command set and display output. Each of these Actors has its own Console, as does Magman itself. Thus, we can expect at least 4 instances to be displayed at any given time.

The Console class will need a set of command strings that each Actor expects. It will also need an output stream for each Actor which each Actor can use to display text.

It is assumed that the creating task will provide the streams which are used to communicate from the Console to the Actor and back. Thus, a console manipulates 4 streams:

Tout – A display stream to the terminal

Tin – A display stream from the terminal

AtoC – A communication stream from the Actor to the Console

CtoA – A communication stream from the Console to the Actor

The Console has the task of parsing the input so that all of the parsing is done in one location and in a uniform way. In order to recognize correct commands all Actors have a getCommandList methods which delivers a list of valid command strings.

Console uses the Command Pattern [4] to pass commands to Actors. Commands, which are single words, are parsed from the **Tin** stream. The rest of the command string is passed to a method on the Actor called invokeCommand. (See the Actor interface).

Consoles extend JPanel. That is, they are Swing components.

Each console has an associated Shell. The Console provides the "window" and the Shell provides the command interpreter.

If the Magman window is closed we assume that the user wishes to quit the whole session so all of the windows are closed and we exit. We probably need to close all of our log files first.

⟨*Console.java*⟩≡

```
package com.ccny.magman;

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.swing.*;

public class Console extends JPanel
{
```

```
static JFrame frame;     // the root frame of the world
private Console mainMagPanel;

String myName = "none";
Shell myShell = null;
Writer AtoC = null;
Reader CtoA = null;
Writer Tout = null;
Reader Tin  = null;

public Console(String name, Writer listen, Reader talk)
{ super(true);
  Catch.log("Console:public constructor:","Enter");
  myName = name;
  AtoC = listen;
  CtoA = talk;
  setBackground(MagParms.getParmColor("Console.background"));
  setForeground(MagParms.getParmColor("Console.foreground"));
  myShell = new Shell(myName);
  add(myShell);
  setupMainWindow();
  Catch.log("Console:public constructor:","Exit");
}

private Console()
{ super(true);
  Catch.log("Console:private constructor:","Enter");
  Catch.log("Console:private constructor:","Exit");
}

private void setupMainWindow()
{ try
  { UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
  }
  catch (Exception exc)
  { Catch.log("Console:setupMainWindow","Error loading L&F",exc);
  }
  frame = new JFrame(MagParms.getParm(myName+".Console.Title"));
  WindowListener listenClose = new WindowAdapter()
  { public void windowClosing(WindowEvent e)
    { Window win = e.getWindow();
      win.setVisible(false);
      win.dispose();
      String msg = myName+" console closed";
      Catch.log("Console:setupMainWindow",msg);
      if (myName.equals("Magman"))
```

```
            System.exit(0);
          }
        };
        frame.addWindowListener(listenClose);
        JOptionPane.setRootFrame(frame);
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        Container contentPane = frame.getContentPane();
        contentPane.removeAll();
        contentPane.setLayout(new BorderLayout());
        contentPane.add(this, BorderLayout.CENTER);
        int mainWidth = MagParms.getParmInteger(myName+".Console.width");
        int mainHeight = MagParms.getParmInteger(myName+".Console.height");
        frame.setSize(mainWidth, mainHeight);
        int xlocation = MagParms.getParmInteger(myName+".Console.xlocation");
        int ylocation = MagParms.getParmInteger(myName+".Console.ylocation");
        frame.setLocation(xlocation,ylocation);
        frame.setCursor(Cursor.getPredefinedCursor(Cursor.DEFAULT_CURSOR));
        frame.validate();
        frame.show();
      }

      public void init()
      { Catch.log("Console:init","init");
      }

      public void start()
      { Catch.log("Console:start","start");
      }

      public void stop()
      { Catch.log("Console:stop","stop");
      }

    }
```

5.2.2 Shell Class

A Shell is the object that controls all of the input and output from a Console object. Each Console object has one Shell. Each Shell has one ShellDocument. Shells are similar in spirit to Unix sh or bash shells. They expect to execute in a JPanel.

⟨*Shell.java*⟩≡

```
package com.ccny.magman;

import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.*;
import javax.swing.text.*;

public class Shell extends JPanel
{
  private ShellDocument history = null;
  Hashtable commands = new Hashtable();
  EditorKit editor = null;
  JEditorPane editorPane = null;
  public final String shellname;
  public String sofar = "";
  public Magnus magnus;

  public Shell(String prompt)
  { super(true);
    Catch.log("Shell:constructor","Enter");
    shellname=prompt;
    history= new ShellDocument(prompt);
    setLayout(new BorderLayout());
    setBorder(BorderFactory.createEtchedBorder());
    editorPane = new JEditorPane();
    editorPane.setPreferredSize(
      new Dimension(
        MagParms.getParmInteger(shellname+".Console.width")-30,
        MagParms.getParmInteger(shellname+".Console.height")));
    editorPane.setFont(MagParms.getParmFont("Shell.font"));
    editorPane.setText(history.getHistory());
    editorPane.setCaretPosition(shellname.length()+1);
    KeyListener listenKey = new KeyAdapter()
    {  public void keyTyped(KeyEvent e)
       { char k = e.getKeyChar();
         int keyCode = e.getKeyCode();
         int modifiers = e.getModifiers();
```

```
      String modname = KeyEvent.getKeyModifiersText(modifiers);
      Catch.log("Shell:constructor",
                shellname+" "+modname+" "+k+" ("+keyCode+")");
      dispatch(k,keyCode,modifiers);
      if (k == 'x')
        System.exit(0);
    }
  };
  editorPane.addKeyListener(listenKey);
  JScrollPane scrollPane = new JScrollPane(editorPane);
  scrollPane.setVerticalScrollBarPolicy(
    scrollPane.VERTICAL_SCROLLBAR_ALWAYS);
  add(scrollPane,BorderLayout.CENTER);
  magnus = new Magnus();
  Catch.log("Shell:constructor","Exit");
}

public void dispatch(char k, int keyCode, int modifiers)
{ if (k == '\n')
    cmdparse();
  else
  { sofar=sofar+k;
    Catch.log("Shell:dispatch",shellname+" "+sofar);}
}

public void cmdparse()
{ if (sofar.equals("quit"))
    System.exit(0);
  if (sofar.startsWith("send"))
  {  String line = "send to "+sofar.substring(5);
     Catch.log("Shell:cmdparse:send",line);
     editorPane.setText(history.newprompt(sofar,line));
  }
  if (sofar.startsWith("listen"))
  { System.out.println("listening");
     editorPane.setText("listening"+"\n"+shellname+">");
  }
  if (sofar.startsWith("silence"))
  { System.out.println("not listening");
     editorPane.setText("not listening"+"\n"+shellname+">");
  }
  if (sofar.equals("question"))
  { String question = magnus.getNextQuestion();
    Catch.log("Shell:cmdparse:question",question);
    editorPane.setText(history.newprompt(sofar,question));
  }
```

```
    sofar="";
  }

}
```

5.2.3 ShellDocument Class

This is the Document for a Shell. It extends DefaultStyledDocument and is a direct implementation of the features we need in a shell. ShellDocuments maintain the data we print to the screen.

⟨*ShellDocument.java*⟩≡

```
package com.ccny.magman;

import java.awt.*;
import javax.swing.*;
import javax.swing.text.*;

public class ShellDocument extends DefaultStyledDocument
{
  public static String history = "";
  public static String shellprompt = "";

  public ShellDocument(String prompt)
  { Catch.log("ShellDocument:constructor","Enter");
    shellprompt=prompt+">";
    history=shellprompt;
    Catch.log("ShellDocument:constructor","Exit");
   }

  public String newprompt(String inline, String outline)
  { history=history+inline+"\n"+outline+"\n"+shellprompt;
    return(history);
  }

  public String remember(String line)
  { history=history+line;
    return(history);
  }

  public String getHistory()
  { return(history);
  }

  public String clear()
  { history=shellprompt;
    return(history);
  }

}
```

5.2.4 NewAction

⟨*NewAction.java*⟩≡

```
package com.ccny.magman;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.text.*;

public class NewAction extends AbstractAction
{
/*  public static final String newAction  = "new";

  NewAction()
  { super(newAction);
  }

  NewAction(String nm)
  { super(nm);
  }
*/
  public void actionPerformed(ActionEvent e)
  {
/* Document oldDoc = getEditor().getDocument();
    if(oldDoc != null)
      oldDoc.removeUndoableEditListener(undoHandler);
    getEditor().setDocument(new PlainDocument());
    getEditor().getDocument().addUndoableEditListener(undoHandler);
    resetUndoManager();
    revalidate();
*/
  }
}
```

5.2.5 The Magnus Interface

For the moment we give a trivial implementation of the interface to Magnus. This will call the C program listed below, grab the output stream of the program and read the question asked. We expect this interface to be more complex as we begin to use the real version of Magnus.

⟨*Magnus.java*⟩≡

```
package com.ccny.magman;

import java.io.*;

public class Magnus
{
  Runtime runtime;
  Process process;
  InputStream magmanSaid;
  BufferedReader questionStream;

  public Magnus()
  { try
    { runtime = Runtime.getRuntime();
      process = runtime.exec("./Magnus");
      magmanSaid = process.getInputStream();
      questionStream = new BufferedReader(new InputStreamReader(magmanSaid));
    }
    catch(Exception e)
    { e.printStackTrace();
    }
  }

  public String getNextQuestion()
  { String question = "";
    try
    { String line = "";
      Catch.log("Magnus:getNextQuestion","magman said:");
      while ((line = questionStream.readLine()) != null)
      { question = question+line;
        Catch.log("Magnus:getNextQuestion:",line);
      }
    }
    catch(Exception e)
    { e.printStackTrace();
    }
    return(question);
  }
```

```
  public void stop()
  { questionStream = null;
    magmanSaid = null;
    process.destroy();
    process = null;
    runtime = null;
  }
}
```

5.2.6 A dummy version of Magnus

This program exists for testing purposes only.

⟨*Magnus.c*⟩≡

```
#include <stdio.h>

int main()
{ printf("is w=1?\n");
}
```

5.3 The Actors

Actors implement the Template Pattern [4]. The Actors that derive from this class all must implement the methods listed here.

Actors use the Command Pattern [4] to accept commands.

Beside the obvious Actors it should be noted that Magman is also an Actor. Actors have Consoles and Consoles have commands and parsing. Magman fulfills the Actor interface contract so it can participate in using a Console.

5.3.1 Actor Class

All Actors (Alice, Bob, Eve) and Magman itself derive from this interface. This ensures that they all have an invokeCommand method so they can communicate with their Console class (See Console.class).

Since each Actor is expected to accept different commands but the validity of the command is checked by the Console class each Actor must supply a list of the valid commands it is prepared to accept. This is done with the getCommandList method.

⟨*Actor.java*⟩≡

```
package com.ccny.magman;

public interface Actor
{ public void invokeCommand(String cmd, String args);
  public String[] getCommandList();
}
```

5.3.2 Person Class

⟨*Person.java*⟩≡

```
package com.ccny.magman;

public class Person implements Actor
{
  String myname = "anonymous";
  Console myterm = null;

  public Person(String name, Console term)
  { myname = name;
    myterm = term;
    Catch.log("Person:constructor",myname+": created");
  }

  public void init()
  { Catch.log("Person:init",myname+": init");
  }

  public void start()
  { Catch.log("Person:start",myname+": start");
  }

  public void stop()
  { Catch.log("Person:stop",myname+": stop");
  }

  public void invokeCommand(String cmd, String args)
  { Catch.log("Person:invokeCommand",cmd+" "+args);
  }

  public String[] getCommandList()
  { Catch.log("Person:getCommandList","enter");
    String[] result = null;
    return(result);
  }

}
```

5.3.3 Computer Class

⟨*Computer.java*⟩≡

```
package com.ccny.magman;

public class Computer implements Actor
{

  public void invokeCommand(String cmd, String args)
  {}

  public String[] getCommandList()
  { String[] result = null;
    return(result);
  }

}
```

5.3.4 Attacker Class

The Attacker class uses the Strategy Pattern [4]. The basic idea is that the Attacker could evesdrop and record quietly, analyze the stream or interpose itself into the stream by replay or insertion of false messages. The Attacker gets the stream sent by each sending Actor before the receiving Actor sees it. The Attacker can modify the stream in any way based on the current selected strategy.

⟨*Attacker.java*⟩≡

```
package com.ccny.magman;

public class Attacker implements Actor
{
  public void invokeCommand(String cmd, String args)
  {}

  public String[] getCommandList()
  { String[] result = null;
    return(result);
  }

}
```

6 The Model Elements

6.1 Alphabets

6.1.1 Alphabet Class

⟨*Alphabet.java*⟩≡

```
package com.ccny.magman;

public interface Alphabet
{
}
```

6.1.2 Key Class

⟨*Key.java*⟩≡

```
package com.ccny.magman;

public class Key
{
}
```

6.1.3 Plaintext Class

⟨*Plaintext.java*⟩≡

```
package com.ccny.magman;

public class Plaintext
{
}
```

6.1.4 Ciphertext Class

⟨*Ciphertext.java*⟩≡

```
package com.ccny.magman;

public class Ciphertext
{
}
```

6.2 Transformations

6.2.1 Transformation Class

⟨*Transformation.java*⟩≡

```
package com.ccny.magman;

public interface Transformation
{
}
```

6.2.2 Encrypt Class

⟨*Encrypt.java*⟩≡

```
package com.ccny.magman;

public class Encrypt
{
}
```

6.2.3 Decrypt Class

⟨*Decrypt.java*⟩≡

```
package com.ccny.magman;

public class Decrypt
{
}
```

7 Communications

7.1 Inter-conole Details

7.1.1 Channel Class

⟨*Channel.java*⟩≡

```
package com.ccny.magman;

public class Channel
{
}
```

7.1.2 Message Class

⟨*Message.java*⟩≡

```
package com.ccny.magman;

public class Message
{
}
```

7.2 Network Details

7.2.1 UDPio.java

⟨*UDPio.java*⟩≡

```
// 20021101000 tpd repackage
// 20000816000 tpd repackage
// 20000815000 tpd created

package com.ccny.magman;

import java.io.*;
import java.net.*;

public class UDPio
{
  String console;
  String host;
  int port;
  byte[] buffer;
  DatagramSocket dsocket;
  int delay;

  public UDPio(String theconsole, String thehost, int theport)
  { if (theconsole == null)
      console = "none";
    else
      console = theconsole;
    if (thehost == null)
      host = MagParms.getParm(theconsole+".UDPio.host");
    else
      host = thehost;
    if (theport == -1)
      port = MagParms.getParmInteger(theconsole+".UDPio.port");
    else
      port = theport;
    int size = MagParms.getParmInteger(theconsole+".UDPio.bufferSize");
    buffer = new byte[size];
    delay = MagParms.getParmInteger(theconsole+".UDPio.delay");
    Catch.log("UDPio.constructor",": host="+host+" port="+port+
                                  " delay="+delay+" size="+size);
  }

  public String getConsole()
  { return(console);
  }
```

```
public String getHost()
{ return(host);
}

public int getPort()
{ return(port);
}

public void UDPSend(String msg)
{ try
  { buffer = msg.getBytes();
    InetAddress address = InetAddress.getByName(host);
    DatagramPacket packet = new DatagramPacket(buffer, buffer.length,
                                               address, port);
    DatagramSocket dsocket = new DatagramSocket();
    dsocket.send(packet);
    dsocket.close();
  }
  catch (Exception e)
  { Catch.log("UDPio:UDPSend","FAILED: "+msg+" to "+host+" port "+port,e);
  }
}

public String UDPReceive()
{ String result = null;
  boolean wait = true;
  try
  { if (dsocket == null) dsocket = new DatagramSocket(port);
    while(wait)
     try
     { DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
       dsocket.setSoTimeout(delay);
       dsocket.receive(packet);
       wait=false;
       result = new String(buffer, 0, packet.getLength());
       Catch.log("UDPio.UDPReceive",
         packet.getAddress().getHostName() + ": " + result);
     }
     catch(SocketException se)
     { wait=true;
       Catch.log("UDPio.UDPReceive", "UDPio: socket timer pop");
     }
     catch(InterruptedIOException ioe)
     { wait=true;
       Catch.log("UDPio.UDPReceive", "UDPio: socket timer pop 2");
```

```
      }
    }
    catch(Exception e)
    { wait=false;
      Catch.log("UDPio:UDPReceive","FAILED: port "+port,e);
    }
    return(result);
  }

  private static void usage()
  { System.out.println("UDPio send string\nUDPio receive");
  }

  public static void main(String[] args)
  { if (args.length >= 1)
    { UDPio ear = new UDPio("Magman",null,-1);
      System.out.println("using host "+ear.getHost()+" port "+ear.getPort());
      if (args[0].equals("receive"))
        System.out.println(ear.UDPReceive());
      else if (args[0].equals("send"))
        ear.UDPSend(args[1]);
      else usage();
    }
    else usage();
  }
}
```

8 Utility Classes

8.1 Prior Code Tools

8.1.1 Magman.ini

Magman.ini contains initialization parameters for Magman that can be changed without recompiling the program. This file is read when Magman is started.

Five groups are available for change. Console parameters change the location, size and title of each console. The console font is common to all of the consoles. The splash screen is not used nor is the version which will appear on the splash screen. The debugging parameters can be used to trace internal details for debugging.

⟨*Magman.ini*⟩≡

⟨*version*⟩
⟨*console parameters*⟩
⟨*console font*⟩
⟨*network parameters*⟩
⟨*splash screen*⟩
⟨*debugging parameters*⟩

The version string is available but not used at the moment. It is normally presented on the splash screen. The format of the version string is YYYYMMDDxxx where YYYY is the year, MM is the month, DD is the day and xxx is a unique integer within the day (assuming you're having a productive day).

⟨*version*⟩≡

```
#default version = Magman.ini not found
version = 20021001000
```

Magman.ini is a slightly more general form of a standard Java parameter file. Entries in this file are of the form:

```
String = String
```

This file is read by MagParms.java. The left string is used as a key to a hash table and the right string is the value returned. MagParms has special methods for returning types other than String. See the MagParms section.

Magman generates 4 consoles. Each has independent parameters which are listed below. The names should make their use obvious.

⟨*console parameters*⟩≡

```
Magman.Console.Title = Magman Login Simulator
Magman.Console.width = 400
Magman.Console.height = 300
Magman.Console.xlocation = 30
Magman.Console.ylocation = 30

AliceTerm.Console.Title = Alice Console
AliceTerm.Console.width = 400
AliceTerm.Console.height = 300
AliceTerm.Console.xlocation = 30
AliceTerm.Console.ylocation = 400

BobTerm.Console.Title = Bob Console
BobTerm.Console.width = 400
BobTerm.Console.height = 300
BobTerm.Console.xlocation = 500
BobTerm.Console.ylocation = 400

EveTerm.Console.Title = Eve Console
EveTerm.Console.width = 400
EveTerm.Console.height = 300
EveTerm.Console.xlocation = 500
EveTerm.Console.ylocation = 30

Console.background = [Color 0000FF]
Console.foreground = [Color 00FF00]
```

The consoles all use the same font. We have chosen a simple font that is available on any Java implementation. The syntax of the value string is explained in the MagParms section.

⟨*console font*⟩≡

```
# the font used by each shell running in the console
Shell.font = [Font Dialog Font.PLAIN 12]
```

The consoles send and receive messages using this host and port combination. See the section on UDPio.java for further information.

⟨*network parameters*⟩≡
```
Magman.UDPio.host = 127.0.0.1
Magman.UDPio.port = 14001
Magman.UDPio.bufferSize = 4096
Magman.UDPio.delay = 2000

Alice.UDPio.host = 127.0.0.1
Alice.UDPio.port = 14001
Alice.UDPio.bufferSize = 4096
Alice.UDPio.delay = 2000

Bob.UDPio.host = 127.0.0.1
Bob.UDPio.port = 14001
Bob.UDPio.bufferSize = 4096
Bob.UDPio.delay = 2000

Eve.UDPio.host = 127.0.0.1
Eve.UDPio.port = 14001
Eve.UDPio.bufferSize = 4096
Eve.UDPio.delay = 2000
```

When the system starts we plan to pop up a "splash screen". This functionality is not currently implemented as we couldn't decide on a good image.

⟨*splash screen*⟩≡
```
#  default SplashImage = com/ccny/magman/splash.gif
SplashImage = com/ccny/magman/splash.gif
#  default SplashWait = 2000
SplashWait = 2000
```

There are two debugging tools in the system. The first is simply a general debug flag that can be turned on or off without recompiling. The second is the Catch facility (see Catch.java) which has the ability to trace calls, returns and other information to stdout and to a log file named here. During development it is probably a good idea to turn these on. During test be sure to turn them off and make sure that no debugging/catch messages appear (that is, you aren't calling println directly

⟨*debugging parameters*⟩≡

```
# general debugging flag
Debug = off
# if on, write Catch.log calls to CatchFile
CatchLog = off
CatchFile = /tmp/Magman.txt
# if on, write Catch.log calls to stdout
CatchTrace = off
```

8.1.2 Catch

Catch is a logging class that will log system failures. It outputs messages and tracebacks to the file named in Magman.ini under the CatchFile entry. Magman.ini is described elsewhere in this section. Note that Catch is a completely static class so the class does not need to be instantiated for logging to occur.

There are two useful methods, called log, one which expects three arguments:

classAndMethod – which is where the failure occurs

data – an associated error message

e – an Exception which contains a message and a stack trace

and the second which expects two arguments:

classAndMethod – which is where the failure occurs

data – an associated error message

If there is a Catchlog entry in Magman.ini thus:
CatchLog = on
CatchFile = /tmp/Magman.txt
then tracing will occur to the CatchFile file.

If there is a CatchTrace entry in Magman.ini thus:
CatchTrace = on
then tracing will occur to stdout (not Magman's console).

⟨*Catch.java*⟩≡

```
// 20030924000 tpd added log with no exception arg
// 20020917000 tpd repackage
// 20010907000 tpd repackage
// 20000816000 tpd repackage
// 20000815000 tpd created

package com.ccny.magman;

import java.io.*;

/** Catch is a logger and can be turned on/off in com/ccny/magman/Magman.ini
 */
public class Catch
{

  private static PrintWriter logFile;

  /** if CatchLog=on   in com/ccny/magman/Magman.ini then write log
   *  if CatchTrace=on in com/ccny/magman/Magman.ini then trace to stdout
   *  @param classAndMethod is a string containing both names
```

```
 *  @param data is a String of informational values
 *  @param e is the exception that was thrown
 */
public static void log(String classAndMethod, String data, Exception e)
{ if (MagParms.getParm("CatchLog").equals("on"))
  try
  { if (logFile == null)
      logFile = new PrintWriter(
                  new FileOutputStream(MagParms.getParm("CatchFile")));
    if (e != null)
    { logFile.println("Catch:"+classAndMethod+" "+data+" "+e.getMessage());
      e.printStackTrace(logFile);
    }
    else
      logFile.println("Catch:"+classAndMethod+" "+data);
    logFile.flush();
  }
  catch(Exception ioe)
  { ioe.printStackTrace(System.err);
  }
  if (MagParms.getParm("CatchTrace").equals("on"))
  { if (e != null)
    { System.out.println("Catch:"+classAndMethod+" "+data+
                         " "+e.getMessage());
    }
    else
      System.out.println("Catch:"+classAndMethod+" "+data);
  }
}

/** if CatchLog=on   in com/ccny/magman/Magman.ini then write log
 *  if CatchTrace=on in com/ccny/magman/Magman.ini then trace to stdout
 *  @param classAndMethod is a string containing both names
 *  @param data is a String of informational values
 *  @param e is the exception that was thrown
 */
public static void log(String classAndMethod, String data)
{ if (MagParms.getParm("CatchLog").equals("on"))
  try
  { if (logFile == null)
      logFile = new PrintWriter(
                  new FileOutputStream(MagParms.getParm("CatchFile")));
    logFile.println("Catch:"+classAndMethod+" "+data);
    logFile.flush();
  }
  catch(Exception ioe)
```

```
    { ioe.printStackTrace(System.err);
    }
    if (MagParms.getParm("CatchTrace").equals("on"))
      System.out.println("Catch:"+classAndMethod+" "+data);
  }

  /** Catch unit test code
   */
  public static void main(String[] args)
  { Catch.log("Catch:main",null,new Exception("main unit test1"));
    Catch.log("Catch:main","unit test 2",new Exception("main unit test2"));
  }

}
```

8.1.3 MagParms

MagParms maintains a hash table of initialization parameters that are loaded from the file Magman.ini, which is a file format that is explained elsewhere in this document. This allows us to collect all of the random constants like terminal size, initial position, color, etc. into one location. It works by creating a new standard java Properties object, initializing some default values (in case we cannot find Magman.ini) and then filling the hash table from the Magman.ini file. Programs that need constants call the getParm routine. The first call to getParm will automatically load the Magman.ini file. Constants can be changed in the running program by calling setParm.

Generally constants are stored as Strings which is the Type returned by getParm. However, this may involve parsing and casting the resulting constant to the correct final type which is tedious. Thus we add a few special getParmXXX methods where XXX is the target type. These methods do the special parsing and casting the XXX. So, for example, getParmInteger ill return an int.

This version supports parsing of Color, Font and Border objects.
getParmColor calls getParm to look up the value and parses out a color. The value should be of the form [Color rrggbb] where red=rr green=gg blue=bb in hex. So a line in Magman.ini that reads

foo.bar.baz = [Color 123456]
implies that we call java.awt.Color

$$r = 18, g = 52, b = 86$$

getParm

⟨*MagParms.java*⟩≡

```
// 20020917000 tpd repackage
// 20000903000 tpd break out parse routines for Color, Font, Border
// 20000903000 tpd add getParmBorder
// 20000901000 tpd bitch to System.out if getParms gets a null value
// 20000830000 tpd add getParmColor
// 20000817000 tpd add getColor, defaults
// 20000816000 tpd repackage
// 20000815000 tpd created

package com.ccny.magman;

import java.io.*;
import java.util.*;
import java.awt.*;
import javax.swing.border.*;

/** MagParms manages initialization constants.
 *  <br>
 *  This class is the interface to the file "com/ccny/magman/Magman.ini".
```

```
 *  The file contains various constants used by the CP application.
 *  This class loads the Magman.ini constants into a Properties hashtable.
 *  If a required constant is not specified then a hardcoded default value
 *  is used instead. Hardcoded defaults exist here rather than in the
 *  corresponding classes to centralize the management of constants.
 */
public class MagParms
{

 /** props is a hashtable containing known initialization constants
  */
 private static Properties props;

 /** loaded is true if we successfully found and loaded the initialization file
  */
 private static boolean loaded = false;

 /** loadParms loads "com/ccny/magman/Magman.ini" into a property table
  */
 public static void loadParms()
 { props = new Properties();
   initDefaults();
   try
   { props.load(new FileInputStream("com/ccny/magman/Magman.ini"));
     loaded = true;
   }
   catch(IOException e)
   { System.err.println("MagParms:file com/ccny/magman/Magman.ini not found");
   }
 }

 /** debugging is a simple predicate used for adding debugging tests.
  *  It can be turned on and off using the flag in Magman.ini
  */
 public static boolean debugging()
 { if (getParm("Debug").equals("on"))
     return(true);
   return(false);
 }

 /** tracing is a simple predicate used for adding trace messages
  *  It can be turned on and off using the flag in Magman.ini
  */
 public static boolean tracing()
 { if (getParm("Tracing").equals("on"))
     return(true);
```

```
  return(false);
}


/** getParmColor calls getParm to look up the value and parses out a color.
 *  The value should be of the form [Color rrggbb] where<BR>
 *    red=rr green=gg blue=bb in hex<BR>
 *    e.g. [Color 123456] == java.awt.Color[r=18,g=52,b=86]<BR>
 *  Converts the return value to an color.
 *  @param key (String) is the key we are looking up.
 *  @return (int) the value for the lookup key (possibly defaulted)
 */
public static Color getParmColor(String key)
{ String value = (String)props.get(key);
  if (debugging())
    System.err.println("MagParms:getParmColor key="+key+" value="+value);
  if (value == null)
    System.err.println(
      "MagParms:getParmColor:key="+key+" has no value in Magman.ini");
  return(parseColor(value));
}

public static Color parseColor(String value)
{ int[] nums = new int[6];
  char hexchar;
  for(int i = 0; i<6; i++)
  { hexchar = value.charAt(7+i);
    switch (hexchar)
    { case 'a':
      case 'A':
        nums[i]=10;
        break;
      case 'b':
      case 'B':
        nums[i]=11;
        break;
      case 'c':
      case 'C':
        nums[i]=12;
        break;
      case 'd':
      case 'D':
        nums[i]=13;
        break;
      case 'e':
      case 'E':
```

```
        nums[i]=14;
        break;
      case 'f':
      case 'F':
        nums[i]=15;
        break;
      default:
        nums[i]=Integer.parseInt(hexchar+"");
        break;
     }
  }
  int red   = nums[0]*16+nums[1];
  int green = nums[2]*16+nums[3];
  int blue  = nums[4]*16+nums[5];
  return(new Color(red,green,blue));
}

public static Border parseBorder(String value)
{ Border result = new EmptyBorder(1,1,1,1);
  final int EMPTY = 0;
  final int MATTE = 1;
  final int ETCHED = 2;
  int borderType = EMPTY;
  int point = 0;
  int mark = 0;
  int a = 1;
  int b = 1;
  int c = 1;
  int d = 1;
  if ((mark = value.indexOf("EMPTY")) != -1)
    borderType = EMPTY;
  if ((mark = value.indexOf("MATTE")) != -1)
    borderType = MATTE;
  if ((mark = value.indexOf("ETCHED")) != -1)
    borderType = ETCHED;
  if (mark != -1)
  { switch (borderType)
    { case EMPTY:
        point = value.indexOf(' ',mark)+1;
        mark = value.indexOf(' ',point);
        a = Integer.parseInt(value.substring(point,mark));
        point=mark+1;
        mark=value.indexOf(' ',point);
        b = Integer.parseInt(value.substring(point,mark));
        point=mark+1;
        mark=value.indexOf(' ',point);
```

```
        c = Integer.parseInt(value.substring(point,mark));
        point=mark+1;
        mark=value.indexOf(']',point);
        d = Integer.parseInt(value.substring(point,mark));
        result = new EmptyBorder(a,b,c,d);
        break;
      case MATTE:
        point = value.indexOf(' ',mark)+1;
        mark = value.indexOf(' ',point);
        a = Integer.parseInt(value.substring(point,mark));
        point=mark+1;
        mark=value.indexOf(' ',point);
        b = Integer.parseInt(value.substring(point,mark));
        point=mark+1;
        mark=value.indexOf(' ',point);
        c = Integer.parseInt(value.substring(point,mark));
        point=mark+1;
        mark=value.indexOf(' ',point);
        d = Integer.parseInt(value.substring(point,mark));
        point=mark+1;
        mark=value.indexOf('[',point);
        if (mark == -1)
        { Catch.log("MagParms:parseBorder","bad Color in Border: "+value,
            new Exception("Magman.ini:bad Color in Border: "+value));
          System.err.println(
            "MagParms:parseBorder: bad Color in Border: "+value);
        }
        else
          result = new MatteBorder(a,b,c,d,parseColor(value.substring(mark)));
        break;
      case ETCHED:
        point = value.indexOf('[',mark);
        mark = value.indexOf(']',point);
        Color highlight = parseColor(value.substring(point,mark));
        point = value.indexOf('[',mark);
        mark = value.indexOf(']',point);
        Color shadow = parseColor(value.substring(point,mark));
        result = new EtchedBorder(highlight,shadow);
        break;
    }
  }
  return(result);
}

/** getParmBorder calls getParm to look up the value and parses out a border.
 *  The value should be of the form [Border type a b c d] where<BR>
```

```
 *    a b c d are the border insets<BR>
 *    type is one of { EMPTY MATTE }<BR>
 *  Converts the return value to an border.
 *  @param key (String) is the key we are looking up.
 *  @return (int) the value for the lookup key (possibly defaulted)
 */
public static Border getParmBorder(String key)
{ String value = (String)props.get(key);
  if (debugging())
    System.err.println("MagParms:getParmBorder key="+key+" value="+value);
  if (value == null)
    System.err.println(
      "MagParms:getParmBorder:key="+key+" has no value in Magman.ini");
  return(parseBorder(value));
}

/** getParmFont calls getParm to look up the value and parses out a font.
 *  The value should be of the form [Color name type size]
 *    name=font name, type= font type, size= font pointsize
 *    e.g. [Font Dialog Font.PLAIN 12] ==
 *      java.awt.Font[family=dialog,name-Dialog,style=plain,size=12]
 *  Converts the return value to a Font.
 *  @param key (String) is the key we are looking up.
 *  @return (int) the value for the lookup key (possibly defaulted)
 */
public static Font getParmFont(String key)
{ String value = (String)props.get(key);
  if (debugging())
    System.err.println("MagParms:getParmFont key="+key+" value="+value);
  if (value == null)
    System.err.println(
      "MagParms:getParmFont:key="+key+" has no value in Magman.ini");
  return(parseFont(value));
}

public static Font parseFont(String value)
{ int point = 6;
  int mark = value.indexOf(' ',point);
  String name = value.substring(point,mark);
  point = mark+1;
  mark = value.indexOf(' ',point);
  String type = value.substring(point,mark);
  int fonttype = Font.PLAIN;
  if (type.toUpperCase().equals("BOLD"))
    fonttype = Font.BOLD;
  if (type.toUpperCase().equals("ITALIC"))
```

```
    fonttype = Font.ITALIC;
  point = mark+1;
  mark = value.indexOf(']',point);
  int size = Integer.parseInt(value.substring(point,mark));;
  return(new Font(name,fonttype,size));
}


/** getParm looks up the key in a hashtable and returns the value found.
 *  If the key is not found a hardcoded default value is returned
 *  @param key (String) is the key we are looking up.
 *  @return (String) the value for the lookup key (possibly defaulted)
 */
public static String getParm(String key)
{ if (loaded == false)
    loadParms();
  String value = (String)props.get(key);
  if ((!(key.equals("Debug"))) && ((String)props.get("Debug")).equals("on"))
    System.err.println("MagParms:getParm "+key+"="+value);
  if (value == null)
  { Catch.log("MagParms:getParm","key="+key+" has no value in Magman.ini",
         new Exception("Magman.ini is missing the key "+key));
    System.err.println("MagParms:getParm:key="+key+
                       " has no value in Magman.ini");
  }
  return(value);
}


/** setParm can be used to change existing or add new parms
 *  @param the key to be changed
 *  @param the value to be set
 */
public static void setParm(String key, String value)
{ if (loaded == false)
    loadParms();
  props.put(key,value);
}

/** getParmInteger calls getParm to look up the value and parses out an int.
 *  If the key is not found a hardcoded default value is returned
 *  Converts the return value to an integer.
 *  @param key (String) is the key we are looking up.
 *  @return (int) the value for the lookup key (possibly defaulted)
 */
public static int getParmInteger(String key)
```

```
 { return(Integer.parseInt(getParm(key)));
 }

 /** initDefault hardcodes default values for initialization constants.
  *  If the key is not found in Magman.ini we provide a default value here.
  *  @param key (String) is the key we are looking up.
  *  @param value (String) is the result we got from the Magman.ini file
  *  @return (String) a value for the lookup key
  */
 private static void initDefaults()
 { props.put("version","Magman.ini not found");
   props.put("SplashImage","com/ccny/magman/splash.gif");
   props.put("SplashWait","2000");
   props.put("Debug","on");
   props.put("CatchLog","on");
   props.put("MainPanel.defaultFont",new Font("Dialog",Font.PLAIN,12));
 }

 /** MagParms unit test code
  */
 public static void main(String[] args)
 { loadParms();
   setParm("testColor","[Color 123456]");
   System.out.println(getParmColor("testColor"));
   setParm("testColor","[Color FFFFFF]");
   System.out.println(getParmColor("testColor"));
   setParm("testBorder","[Border MATTE 5 5 5 5]");
   System.out.println(getParmBorder("testBorder"));
   System.out.println("MainPanel.defaultFont");
   System.out.println(getParmFont("MainPanel.defaultFont"));
   for(Enumeration e = props.keys(); e.hasMoreElements();)
   { String next = (String)e.nextElement();
     System.out.println((String)next+"="+props.get(next));
   }
   setParm("Debug","off");
   System.out.println("Debug="+props.get("Debug"));
   setParm("Debug","on");
   System.out.println("Debug="+props.get("Debug"));
 }

}
```

9 The Makefile

In order to build Magman from this pamphlet (we assume you have noweb installed) you need only type

```
notangle -t8 Magman.pamphlet >Makefile
make
```

The build process has 4 steps: extract, tex, test and show.

The extract process will check that the ./com/ccny/magman directory exists and is empty. This is the directory that will hold the Java package for Magman. Since Java knows this path name it is important. Each Java file is extracted from the pamphlet and created in the package directory. Additionally the initialization file (Magman.ini) is created there. For testing purposes, in the absence of a real copy of Magnus, we create the Magnus.c file in the current directory. Finally we compile the Java and C code.

The tex process builds Magman.tex and runs latex on it (twice to ensure that the citations are correct). This assumes that the noweb.sty file is in the current directory or the tex search path. If not, copy noweb.sty to the current directory and run

```
make tex
```

The test process will run the Magman code.

The show process will invoke xdvi (in expert mode with magnification 3) show this document can be read.

⟨*⟩≡

```
# 20021101000 tpd add UDPio for networking
# 20021030000 tpd merge Makefile and Magman
# 20021029000 tpd added cmdparse
# 20020924000 tpd added Shell1.java
# 20020924000 tpd added Shell.java
# 20020923000 tpd added NewAction.java
# 20020923000 tpd enhanced Console
# 20020317002 tpd added Catch.java
# 20020317001 tpd added Magman.ini
# 20020317000 tpd added MagParms.java
# 20020313000 tpd created

PKG=com/ccny/magman
PAM=Magman.pamphlet

all: extract tex test show

extract:
        @( if test ! -d com ; \
            then mkdir com ; fi ; )
```

```
        @( if test ! -d com/ccny ; \
            then mkdir com/ccny ; fi ; )
        @( if test ! -d com/ccny/magman ; \
            then mkdir com/ccny/magman ; fi ; )
        @rm -f ${PKG}/*
        @notangle -RActor.java          ${PAM} >${PKG}/Actor.java
        @notangle -RAlphabet.java       ${PAM} >${PKG}/Alphabet.java
        @notangle -RCatch.java          ${PAM} >${PKG}/Catch.java
        @notangle -RChannel.java        ${PAM} >${PKG}/Channel.java
        @notangle -RCiphertext.java     ${PAM} >${PKG}/Ciphertext.java
        @notangle -RComputer.java       ${PAM} >${PKG}/Computer.java
        @notangle -RConsole.java        ${PAM} >${PKG}/Console.java
        @notangle -RDecrypt.java        ${PAM} >${PKG}/Decrypt.java
        @notangle -REncrypt.java        ${PAM} >${PKG}/Encrypt.java
        @notangle -RKey.java            ${PAM} >${PKG}/Key.java
        @notangle -RMagman.java         ${PAM} >${PKG}/Magman.java
        @notangle -RMagman.ini          ${PAM} >${PKG}/Magman.ini
        @notangle -RMagnus.c            ${PAM} >Magnus.c
        @notangle -RMagnus.java         ${PAM} >${PKG}/Magnus.java
        @notangle -RMagParms.java       ${PAM} >${PKG}/MagParms.java
        @notangle -RMessage.java        ${PAM} >${PKG}/Message.java
        @notangle -RNewAction.java      ${PAM} >${PKG}/NewAction.java
        @notangle -RPerson.java         ${PAM} >${PKG}/Person.java
        @notangle -RPlaintext.java      ${PAM} >${PKG}/Plaintext.java
        @notangle -RShell.java          ${PAM} >${PKG}/Shell.java
        @notangle -RShellDocument.java  ${PAM} >${PKG}/ShellDocument.java
        @notangle -RTransformation.java ${PAM} >${PKG}/Transformation.java
        @notangle -RUDPio.java          ${PAM} >${PKG}/UDPio.java
        @javac ${PKG}/*.java
        @gcc -o Magnus Magnus.c

tex:
        @noweave -delay ${PAM} >Magman.tex
        @latex Magman.tex
        @latex Magman.tex
show:
        @xdvi -expert -s 3 Magman.dvi &
test:
        @java com.ccny.magman.Magman
```

References

[1] Delfs, Hans; Knebl, Helmut, *Introduction to Cryptography*, p6, Springer-Verlag, (2002), ISBN 3-540-42278-1

[2] Patterson, Wayne, *Mathematical Cryptology*, Rowman and Littlefield, Totowa, New Jersey 07512, (1987), ISBN 0-8476-7438-X

[3] Wagner, Neal R.; Magyarik, Marianne R., "A Public Key Cryptosystem Based on the Word Problem", Advances in Cryptology: Proceedings of Crypto 84, pp 19-36, Springer-Verlag, Berlin, (1985)

[4] Cooper, James; *Java Design Patterns*, Addison-Wesley, Reading MA. (2000) ISBN 0-201-48539-7

[5] Ramsey, Norman, "http://www.eecs.harvard.edu/ nr/noweb/"

[6] Ramsey, Norman,
"Literate programming simplified", IEEE Software, 11(5):97-105, September 1994, "http://www.eecs.harvard.edu/~nr/pubs/lpsimp.pdf"

[7] Sun Microsystem Source Code "j2sdk1.4.0/demo/jfc/Stylepad/src/Notepad.java"