User's Manual for the Teaching Machine

Revision history

- October 20, 2001. TSN Initial version. Added description of the subset supported by the classic C++ interpreter.
- October 21, 2001. TSN Added description of User Interface
- Dec 8, 2001. TSN Added documentation of built-in functions.
- Dec 9, 2001. TSN Editorial changes.
- July 28, 2002. TSN Fixed up description of phase 1 library support.
- March 2004. Deleted section on the old interpreter. Updated to reflect the current state of phase 2 support.
- April 2004. Updated the External Command Interface.
- June 2004. Updated information on starting applets and applications.
- Sept 2005. Changed the package for the main entry points.

1 User Interface

1.1 Starting the program

Applet. See section 2.

Application. If the TM is started as a standalone application, the main class is tm.TMMainFrame.

- For the Microsoft VM, the command is

  jview /vst /cp:p C:\TeachingMachine\tm.jar tm.TMMainFrame

- For the Sun JVM, the command is

java -cp C:\TeachingMachine\tm.jar;%CLASSPATH% tm.TMMainFrame

Of course the path of the .jar file may vary.

TMMainFrame takes an optional command-line argument, which is a file name. The TM will attempt to load this file.

The system property "debug" controls the level of debugging output. Usually the property is set to "high" or not set at all. It can also be set to "no", "low", or "medium". In Sun' s JVM the command line parameter is "-Ddebug=high". For Microsoft's VM, the parameter is "/d:debug=high".

1.2 Loading programs.

- **Applet.** The usual way of changing programs is to load the web page that describes the program. See section 2.
- **Application.** The menu command File/Load File may be used. Files must end with a suffix that identifies the language. See section 1.11.

1.3 Execution commands.

-  (Menu: Go/Into Expression)Execute the next expression in this subroutine.
-  (Menu: Go/Into Subroutine) Execute the next expression.
-  (Menu: Go/Over) Execute to the end of this subroutine.

-  Execute until the line the cursor is on or until the program terminates.
-  (Menu: Go/Forward) Execute until a subexpression is selected..
- (Menu: Go/Microstep) Execute one step. This is usually only useful for debugging the interpreter.
-  (Menu: Go/Backward) Undo last command. You may undo back to the start of executing the subject program.
-  Restart the interpretation of the program.

1.4 Display Options

Stack, Static, Heap, and Scratch Subwindows

-  Display one line per variable.
-  Display one line per byte.
-  Display values in binary.

1.5 Window manipulation.

All subwindows can be moved, resized, or brought to the front, though it sometimes takes a few tries to get the mouse in just the right spot.

All subwindows can be maximized, minimized, or closed. Once closed, they are gone until a file is opened.

1.6 Configuration

The layout of the subwindows and a few other aspects of the Teaching Machine's display can be saved later loaded

- Menu: File/Save Configuration File. Write the configuration file to disk.
- Menu: File/Read Configuration File. Retrieve a previous configuration.

By manually editing the configuration files, you can change font styles, colours subwindow titles, and other aspects of the display.

1.7 Exiting or hiding the teaching machine

Selecting the menu command File/Exit will exit the TM, if it is running as an application. If it is running as an applet, this will merely hide the TM's window. The window will be redisplayed if another file is loaded into it. In Windows, clicking on the x in the upper-right corner is equivalent to selecting File/Exit.

If you are running the Teaching Machine as an applet, the following will exit the teaching machine.

- **Internet Explorer.** When you leave the web page that initiated the Teaching Machine applet, the applet will be terminated.
- **Netscape Navigator and Mozilla.** When the page that initiated the Teaching Machine Applet, is no longer on Navigator's Forward-Backward stack, the applet will be terminated.

1.8 Security restrictions

If you ask the applet to do something that, because of a security restriction, it can not do, a window will pop up to tell you so. Typical restrictions involve reading or writing local files.

1.9 Trouble Shooting

Unfortunately the Java slogan of write-once, run-anywhere remains an unattained ideal. Here are some of the problems we know about.

- **Windows Go Blank.**

If windows within the Teaching Machine become blank, when they are brought to the foreground, you may need upgrade your Java system. We have only seen this problem under Windows NT with Internet Explorer 4.0.

Microsoft URLs seem to change quite often, so I won't list one here. Search Microsoft's web sites using keywords like "microsoft virtual machine", which is what they now call their Java virtual machine.

- **It's Slow As Molasses**

Under Netscape and Windows, loading a file can take up to a minute. We have no idea why. The best advice we have is to get Internet Explorer 5.0 --- or Linux.

If you are using Internet Explorer and it's slow, make sure that you have enabled JIT (Just in Time Compilation). Not using JIT adds significant overhead to interpreting the Java.

- **The Teaching Machine Doesn't Start At All.**

Make sure that you are using at least Internet Explorer 4.0 or Netscape Navigator (or Communicator) 4.08. Make sure that Java is enabled. Also make sure you give the Teaching Machine enough time to download.

- **Files won't load because of a security restriction.**

This sometimes happens when using the Teaching Machine as an Applet and the files are on a local disk (as opposed to being served by a web-server). We're not sure why this happens. Try a different browser or a different Java system (Internet Explorer 6 allows you to chose to use the ''Microsoft Virtual Machine'' instead of the Sun Java Plug-in. If you put the whole website onto a webserver, then it will work.

1.10 The Language Accepted by the Teaching Machine

Currently (September 2005), the TM supports C++ and Java. See sections 4 and 5.

1.11 Suffixes

File names and URLs must end with one of the following suffixes in order that the correct language be selected: .cpp, .cxx, .c++, .java, .jav (case insensitive).

1.12 Bugs and wishes

Send reports of bugs and suggested improvements to theo@engr.mun.ca.

2 Applet interface

The Teaching Machine can run as a stand-alone application or as an Applet on a web-page. As an Applet, there are two choices. tm.TMTinyApplet is an applet that creates a Window for the Teaching Machine to run on. That is, this applet draws nothing on the web page. tm.TMBigApplet displays on the web page rather than opening a separate Window.

As an Applet, the Teaching Machine works best on Internet Explorer 4.0 or later. We recommend using standard browser sniffing techniques to warn any user using another browser.

2.1 TMTinyApplet

Suppose we want to start the TM up with a particular example C++ program loaded into it.

The following listing shows how to do this.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
                                    "http://www.w3.org/TR/html4/loose.dtd">
   <html> <head>
     <script language="JavaScript" type="text/javascript">
     <!--
α        function onLoadHandler()
        {
β            document.tm_applet.loadRemoteFile('example1.cpp') ;
γ            document.tm_applet.readRemoteConfiguration('default.cfg') ;
        }
     //-->
    </script>
  </head>

δ  <body onload="onLoadHandler();">
ε     <APPLET NAME="tm_applet"
ζ           CODE="tm/TMTinyApplet.class"
η           WIDTH="1" HEIGHT="1" ALIGN="BOTTOM"
θ           ARCHIVE="tm.jar">
      </APPLET>

   </body>
</html>
```

The APPLET tag creates an instance of the tm.TMTinyApplet class (ζ), giving it a name of *tm_applet* (ε). Since the Applet draws nothing on the space it is accorded on the web-page, we give it only a 1 by 1 pixel area (η). In this case the code of the TM is contained in two .jar (Java Archive) files (θ) located in the same directory as the page. Once the page has loaded, and the Applet has started, the "onLoad" event of the page's Body element is triggered. We have established an event handler (δ) which first loads a C++ file into the instance of the applet (β) and then loads a configuration for the Teaching Machine (γ). Both the C++ file and the configuration file are stored on the same server as the web pages and the .jar files. See section **Error! Reference source not found.** for more on the arguments to the **loadRemoteFile** method.

The technique shown above is effective, but inefficient, since it requires the creation of a new Applet instance for each example. Worse, unless the user's browser caches

the .jar files, they will have to down-load again. As the files are big (about 350K at the moment), there is currently a significant delay, especially when voice-quality phone lines are used. Instead, we generally create one instance of the applet and load each example into it as the user requires. Next we look at a technique for doing that.

We generally accord the TMTinyApplet its own frame within the browser. For example, here is how one of my sites works. The main HTML file has three frames called applet_frame, toc_frame, and content_frame. Initially we load a welcome message into the content_frame and a "please wait" message into the "toc_frame". Into the applet_frame, we load the following file:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<HTML> <HEAD> </HEAD>
<BODY onLoad="top.toc_frame.location.href = 'tocframecont.html' " >
  <APPLET NAME="tm_applet"
          CODE="tm/TMTinyApplet.class"
          WIDTH="1" HEIGHT="1"
          ALIGN="BOTTOM"
          ARCHIVE="tm.jar, jsnoopy-minimal.jar">
  </APPLET>
  <A HREF="help/help.html" target="help">HELP</A>
</BODY> </HTML>
```

As can be seen, once the applet is loaded, the "please wait" message in the toc_frame is replaced by a file called tocframecont.html. This contains links that the user can click. In the head of tocframecont.html we have the line

```
<base target="content_frame">
```

so that clicking on these links causes them to go to the content_frame. A typical link in the tocframecont.html file is

```
<a HREF="data-structs-ex/03_make_list.html">
    Make and traverse an linked list.</a>
```

Each file like data-structs-ex/03_make_list.html describes a particular example and loads it into the teaching machine. Here is the start of data-structs-ex/03_make_list.html.

```
<html>
<head>
    <script LANGUAGE="JavaScript">
     function onLoad() {
      top.applet_frame.document.tm_applet.loadRemoteFile(
                          'data-structs-ex/03_make_list.cpp') ;
      top.applet_frame.document.tm_applet.readRemoteConfiguration(
                          'data-structs-ex/boxes.tmcfg') ;  }
    </script>
</head>
<body ONLOAD="onLoad()">
```

Of course this is just one way to do it. The important thing is to prevent the user from being able to send any messages to the Applet until it has fully loaded and started. In this case we ensure that the page that contains the links that will cause C++ files to be loaded into the Applet is not loaded into its frame until the page the Applet is on is fully

loaded. The technique above works in both IE and in Netscape (at least in versions 4.5 to 4.7).

2.2 The TMBigApplet

tm.TMBigApplet displays its subwindows directly within the Browser's window. It does not have menus. The user can still interact with the Teaching Machine via the buttons on the subwindows. This Applet requires about 600 (width) by 375 (height) pixels, depending on the configuration.

2.3 External Command Interface

Both the tm.TMBigApplet and the tm.TMTinyApplet implement an interface called the ExternalCommandInterface. Any method in this interface can be invoked on an applet instance using JavaScript (provided the Browser supports JavaScript to Java method calls). The exact interface is documented elsewhere. The following methods are useful for JavaScripting:

2.3.1 Loading source files

public void loadRemoteFile(String URL);

The URL is relative to the Applet's "document base"; there is no way to use an absolute URL.The URL must end with a suffix that indicates the language (C++ or Java); see section 1.11 .

public void loadRemoteFile(String rootURL, String relativeURL);

The first URL is relative to the Applet's "document base"; there is no way to use an absolute URL.It gives the directory that should be used as a "root" for resolving include files and imported files. The relativeURL must end with a suffix that indicates the language (C++ or Java); see section 1.11 .

public void loadString(String fileName, String programSource);

The fileName must end with a suffix that indicates the language (C++ or Java); see section 1.11 . This routine can be useful when the source is read off an HTML form.

public void reStart();

reStart reloads the current file, if any.

2.3.2 Setting the configuration

public void readRemoteConfiguration(String URL) ;

The URL is relative to the Applet's "document base"; there is no way to use an absolute URL. As the configuration does not revert with each load of a source file, it is recommended to call readRemoteConfiguration after each load.

2.3.3 Selection

public void setSelectionString(String ch) ;

The ch should be one of the following (case sensitive): "all", "default", or a single letter. See section 3. Each time a new file is loaded, the selection string reverts to "default".

2.3.4 Input characters

public void addInputString(String input);

addInputString is used to stuff characters into cin (or System.in). It should be called after the program is loaded (or reStarted).

2.3.5 Hiding

public void quit();

For the tm.TMTinyApplet, quit hides its Window. For the tm.TMBigApplet, nothing is done.

2.3.6 Execution

The following advance or retard the state of the machine

public void goBack();
public void goForward();
public void microStep();
public void overAll();
public void intoExp();
public void intoSub();
public void toCursor(String fileName, int cursor);

In the case of toCursor, the parameter is the line number, where the first line is line 1.

3 Pedagogical Markup

The teaching machine recognizes certain comments as "pedagogical markup". It shares this syntax with WebWriter++.

3.1 General Syntax

Comments that start with "/*#" are considered pedagogical markup and are not displayed to the user regardless of whether the Teaching Machine recognizes them or not.

3.2 Selections

The current selection can be set to one of the following settings either via the View menu or via the applet interface:

- "all" All code is displayed.
- "default" Code between /*#H*/ and /*#D*/ comments is not displayed.
- *x* where *x* is a letter (one of the 52 "English" letters). Code between /*#H*/ or /*#Hx*/ and /*#D*/ or /*#Dx*/ is not displayed.

Code in standard include files is prefixed by /*#H*/. Thus to see this code one must select "all".

4 C++

The TOS interpreter has been retired as of Fall 2003. This section refers to the TNG interpreter.

C++ support is being delivered in phases. Currently phase 1 items should be supported. Some phase 2 items are supported. *I will use italics to call attention to Phase 2 features that are not yet supported.*

The Teaching Machine is not really intended to be used on code with errors. Therefore, I will not go into the many kinds of errors it does not report. It will diagnose many compile and run-time errors, but not all.

The following subsections follow the ISO standard. The terminology used is that of the standard.

4.1 General

We do not claim to comply with the ISO standard. We do not diagnose a great many diagnosable errors in programs.

4.2 Lexical conventions

Phases of translation:

1. Mapping of physical to source characters. We assume ascii encoding. Trigraphs are not supported.
2. *Line splicing.* Not in Phase 1. Planned for Phase 2.
3. Decomposition into pptokens. Partially supported.
4. *Preprocessing.* Not supported in Phase 1. Include files are implemented. Full preprocessing is planned for phase 2.
5. Mapping of character literals to execution character set. The execution character set is ASCII, so this shouldn't pose a problem.
6. *Catenation of adjacent string literals.* Not in Phase 1. Planned for Phase 2.
7. Syntactic and semantic analysis. Yes, with limitations described below.
8. *Combining translation units.* Phase 1 only supports one translation unit. This will be done in Phase 2.
9. Resolution of externals. Yes.

Supported

- All keywords and most constants. All operators and punctuation.
- All escape sequences in character and string literals.

Not Supported

- Trigraphs and Digraphs are not recognized.
- Wide characters and wide strings are not supported.

Note

- Tabs stops are set at every 4 spaces.

4.3 Basic Concepts

Supported

- The following types: bool, char, signed char, unsigned char, double, float, long double, int, short, long, unsigned short, unsigned int, unsigned long, pointer, reference, array, struct, class
- const and volatile qualification
- *Wide characters (Phase 2).*
- data members (static and nonstatic)
- function members (static and nonstatic)
- *virtual functions (in Phase 2)*

- auto variables, nonlocal variables, heap variables
- *multiple translation units (in Phase 2)*
- qualified names
- *namspaces (in Phase 2, but in the mean time "using directives" are ignored, so you can use them if you want.)*

Not Supported

- union types, enum types, pointers to functions, pointers to members.

4.4 Standard Conversions

All standard conversions are supported. Except function to pointer conversions and "pointer to member " conversions.

4.5 Expressions

Supported:

- id-expressions
- parentheses
- literals *(except wide characters and wide strings in phase 1)*
- this
- Indexing of arrays and pointers with integers.
- function calls.
- Casts using int(f) notation or C(a,b,c) notation.
- postfix ++ and --
- class member selection with . or -> including selection of member functions.
- Casts using new operators static_cast, reinterpret_cast, const_cast.
- *Casts using dynamic_cast (in Phase 2)*
- sizeof
- prefix ++ --
- new (placement operands are allowed but ignored) You can not define your own new operator.
- delete and delete[]. You can not define your own delete operator.
- unary *, &, +, -, !, ~
- Casts using old notation: (int)f.
- *, /, %
- +, - (including on pointers)
- >>, <<
- <, >, <=, >=, ==, != (including pointers)
- ^, &, |
- &&, ||
- ? : *(Phase 2)*
- =, *=, /=, -=, >>=, <<=, &=, ^=, |=.

- ,

Not supported:

- anything involving the keywords "template", "typeid", or "typename"
- pointers to member selection: .* and ->*
- psuedo destructor calls

Note

- Precedence and associativity are fully respected. When the order of evaluation of operands is not specified by the standard, the TM (TNG) tends to go left-to-right at the moment.

4.6 Statements

Supported

- Expression statements
- Compound statements
- if & if-else statements
- switch, case, & default statements (but you can't use switch to jump over the declaration of any variables)
- while, do-while, for
- break, continue
- return
- declaration statements
- declarations within conditions (in "if", "while", & "for")
- *try-catch, throw (Phase 2)*

Unsupported

- Labels, and goto (*May be supported to a limited extent in Phase 2).*

4.7 Declarations

Supported

- Simple-declarations (i.e. variable declarations and function declarations)
- Declarations of structs and classes.
- typedef
- const and volatile annotations.

Ignored

- using directive.
- inline, auto, register, extern, static,

Not Supported

- friend
- mutable (?)
- virtual, explicit
- union
- enum

- namespace definitions
- using declarations
- linkage specifications.
- asm

4.8 Declarators and Function Definitions

Supported

- *, &, [] (with or without an expression)
- Parameter lists
- Initializers other than aggregate initializers (ISO 8.5.1).
- ctor-initializers

Not Supported

- pointer to member
- const or volatile pointers (no excuse for this; for consistency, these qualifiers should be ignored.
- Var-args (i.e. … in parameter lists)
- Default arguments.
- function-try-block
- Aggregate initializers (ISO 8.5.1).

Deviations

4.9 Classes

Supported

- struct and class
- static data members
- nonstatic data members
- Nonvirtual nonstatic function members
- static function members
- *In-place definition of function members. (Coming in Phase 2)*

Ignored

- Access specifiers public, private, protected

Not Supported

- Unions
- Bit-fields
- Nested classes
- Local classes
- friend
- Constant initializers for const static data members

Deviation

-

4.10 Derived classes

Supported

- Base classes
- Multiple base classes
- *Virtual functions (Coming in phase 2)*
-

Ignored

- access specifiers on base classes (public, private, protected)

Not Supported

- Virtual inheritance

Deviations

-

4.11 Access Control

Access control is not supported. The friend specifier is ignored.

4.12 Special Member Functions

Supported

- Constructors
- *Destructors (Phase 2. In Phase 1 Destructors are compiled, but never called.*
- Implicit declaration and definition of the default constructor, a copy constructor, a destructor, and an assignment operator.
- Temporary objects are correctly constructed
- Constructor initializations (ctor-initializer)

Ignored

-

Not Supported

- Conversion functions
- Programmer defined allocation operators (new, delete, new[], delete[]).

Deviations

- Implicitly declared member functions will also be implicitly defined, even if they are never called.
- Trivial constructors and destructors are implicitly declared, defined, and called. This may cause the TM to appear to call trivial routines that a real compiler would not bother with.

4.13 Overloading

Overloading is supported. There are certain constructs that we do not support and so the overloading rules that apply to them are irrelevant; these include ellipsises in functions, default arguments, user defined conversion functions.

4.14 Templates

Not supported at all.

4.15 Exception Handling

Not supported at all in phase 1.

Phase 2 should support exception handling.

4.16 Preprocessing Directives

Only include is supported.

- For #include <*name*>: Only the TM's jar file is searched. You can add your own include files by adding them to the jar file in directory cpp/include/. The name of the include file should be *name*.inc, if #include <*name*> is to work.
- For #include "*name*": First *name* is searched for relative to the same "directory" as the top-level file (typically the .cpp file). For files loaded over the net, the "directory" is the "document base" for the TM applet. If the file is not found that way, then the file is searched for as if it were included with <name>.

Fairly good support for preprocessing is expected in phase 2.

4.17 Library

4.17.1 Library support in Phase 1

A few library functions and objects are supported. You need to include the appropriate header files.

Note that namespaces are not supported so new library declarations are in namespaces.

Supported or partially supported

- <iostream>
  - cin, cout
  - "cin >> x" is supported where x is an lvalue of type char, another integral type, float, or double.
  - "cout << x" is supported where x is of type char*, char, another integral type, float, or double.
  - cin.get(x) is supported where x is an lvalue of type char
  - cout.put(x) is supported where x is an integral type
  - cin will NOT convert to a bool. Use cin.fail() instead.
  - endl
- <math.h>
  - The following functions with double arguments and double results: abs acos asin atan atan2 ceil cos exp fabs floor log log10 pow sin sqrt tan
- <cmath>
  - Everything in math.h plus float and long double versions of abs acos asin atan atan2 ceil cos exp fabs floor log log10 pow sin sqrt tan

- <string.h>, <cstring>
  - strcpy, strcat, strstr, and strcmp
- <string>
  - A partial implementation. Not stable enough to document. Look at the include file for details.
- <new>
  - nothrow

4.17.2 Library support in Phase 2

In Phase 2 library support will be much improved by support for multiple translation units. Thus any library that can be coded in the supported C++ subset can be handled and users can add their own libraries. However since templates will not be supported, much of the C++ library will remain out of bounds. Other areas (e.g. file I/O) are limited by security restrictions for Java Applets.

4.18 Known Bugs and Deficiencies

- [TBD]

5 Java

Java is underdevelopment and too much in flux to document at the moment. But we're getting there.