

Math Description Engine (MDE) Software Component Library

Programmer's Guide

September 30, 2005

NASA Learning Technologies
Information Accessibility Lab
Johnson Space Center

<http://prime.jsc.nasa.gov>
info@prime.jsc.nasa.gov

Math Description Engine Programmer's Guide

TABLE OF CONTENTS

Math Description Engine (MDE) Software Component Library	1
Programmer's Guide.....	1
Math Description Engine (MDE) Software Component Library Programmer's Guide.....	6
Section 1 Introduction	6
1.1 Purpose	6
1.2 Benefits and Use.....	6
1.2.1 Product Accessibility	6
1.2.2 End-Users	6
1.2.3 Product Developers.....	7
1.2.4 MDE Demonstration.....	7
1.2.5 MDE Availability.....	7
1.2.6 Research & Development / Technology Transfer	7
1.3 Audience	8
1.4 Assumptions and Limitations.....	8
1.5 Related Documents/Resources.....	8
1.6 Feedback	9
Section 2 MDE Functional Overview	10
2.1 High Level Functional Description	10
2.1.1 Text Descriptions of Graphs.....	11
2.1.1.1 Purpose	11
2.1.1.2 Modes	11
2.1.1.3 Content	12
2.1.1.4 Description Engine Approach.....	12
2.1.2 Sonification of Graphs	13
2.1.3 Drawings of Graphs	14
2.2 Inputs	15
2.2.1 Mathematical Equations.....	15
2.2.1.1 Supported Equation Types.....	15
2.2.1.2 Multiple Inputs.....	16
2.2.1.3 API Input Format	16
2.2.1.4 Equation Syntax	16
2.2.2 Time-Series Data	18
2.2.2.1 Supported Data Types	18
2.2.2.2 Multiple Inputs.....	19
2.2.2.3 Input Formats.....	19
2.2.3 MDE Properties	20
2.2.3.1 Property Values.....	20
2.2.3.2 Property Files.....	21
2.2.3.3 Property Methods.....	21
2.3 Outputs.....	21
2.3.1 Text Descriptions	22
2.3.2 Cartesian Graphs.....	22
2.3.2.1 Settings	22
2.3.2.2 Zoom and Pan	22
2.3.2.3 Multiple Curves	23

Math Description Engine Programmer's Guide

2.3.2.4	Sonification Tracer.....	23
2.3.2.5	Graph Zoom/Reset Widget.....	23
2.3.3	Graph Sonification.....	25
2.3.3.1	Sounder and SoundControl Classes.....	25
2.3.3.2	Left/Right Panning and Up/Down Pitch Renderings.....	25
2.3.3.3	Sound Sweep Speed Control.....	26
2.3.3.4	Axis Crossing Indicators.....	26
2.3.3.5	Volume Control.....	27
2.3.4	MDE Properties.....	28
Section 3 Using the MDE API.....		29
3.1	Configuring MDE for use in your software.....	29
3.1.1	Download.....	29
3.1.2	Configuration.....	29
3.1.2.1	Java JDK and JRE Requirements.....	29
3.1.2.2	Java Access Bridge Requirements.....	29
3.1.2.3	Operating Systems.....	30
3.1.3	License.....	31
3.2	MDE Classes.....	32
3.2.1	Main Packages.....	32
3.2.2	Core Classes.....	32
3.2.2.1	Core Functionality.....	32
3.2.2.2	Core GUI Classes.....	33
3.2.2.3	Keyboard Control Classes.....	34
3.3	MDE Tutorial.....	35
3.3.1	Required Classes - Solver and MdeSettings.....	35
3.3.1.1	MdeSettings: Setting and storing MDE properties.....	35
3.3.1.2	Solver: Solution generation and management.....	35
3.3.2	Core Functionality: Descriptor, Sounder, and CartesianGraph.....	36
3.3.2.1	Descriptor: Text descriptions of graphs.....	36
3.3.2.2	Sounder: Sonifications of graphs.....	38
3.3.2.3	CartesianGraph: Drawings of graphs.....	41
3.3.2.4	Use Solver to Reset Graph (Solution) Bounds.....	42
3.3.2.5	Data Input Examples.....	43
3.3.2.6	More Example Programs.....	45
3.3.3	More About The MDE Solution Engine (Solver and Solution).....	45
3.3.3.1	Solution Class.....	45
3.3.3.2	Using Components Together - Synchronization.....	45
Appendix A: Current Text Description Examples.....		47
References.....		52

Figures

Figure 1	Example uses of the MDE Library.....	10
Figure 2	Example MDE text description output for the equation "y=3x".....	11
Figure 3	Line description, with the dynamic values highlighted.....	13
Figure 4	MDE SoundControl Widget.....	13
Figure 5	Cartesian Graph Widget.....	14

Math Description Engine Programmer's Guide

Figure 6 MDE Default Properties	21
Figure 7 Four CartesianGraph output examples.	23
Figure 8 The curves $y=\cos x$ and $y = 3*\cos x$ drawn on the same grid	25

Tables

Table 1 Equation Types with Examples	15
Table 2 Equation Symbols	17
Table 3 MDE Supported Operators	17
Table 4 Column separated time-series data from a rocket simulation	18
Table 5 Table of x and y values for graph of $y=3*x$	19
Table 6 Comma-separated data table example	20
Table 7 Visual and Sound Representations of Cartesian Graph Elements	27
Table 8 MDE Main Packages	32
Table 9 Core Functionality Classes	32
Table 10 GUI Classes	33
Table 11 Keyboard Control Classes	34
Table 12 MDE Text Description Examples	47

Code Listings

Listing 1 Command Line Describer Example	37
Listing 2 Command Line Sonifier Example	39
Listing 3 CartesianGraph Example	41
Listing 4 MDE Descriptions from Data Arrays	43

Math Description Engine Programmer's Guide

Preface

The Math Description Engine (MDE) is a software component library that generates accessible text descriptions, sonifications, and drawings of 2D graphs. This guide gives an overview of the capabilities and use of the MDE API.

This guide describes the *alpha version of MDE*. MDE is written in Java, and the alpha version was developed with Java J2SE v1.4.2_04.

Math Description Engine (MDE) Software Component Library Programmer's Guide

Section 1 Introduction

1.1 Purpose

The main purpose of the Math Description Engine (MDE) software library is to generate accessible descriptions of graphs. More specifically, MDE takes data conveyed in visual graphs and makes it accessible to *visually-impaired* users by generating alternative text and sound descriptions. MDE currently provides alternative descriptions for 2D graphs given mathematical equations or data input.

1.2 Benefits and Use

1.2.1 Product Accessibility

MDE capabilities were developed to increase accessibility of NASA outreach products containing graphs of equations, data tables and data plots. MDE was developed as a *reusable, extensible library* to facilitate addition of graph accessibility in NASA web pages, applications and other suitable products.

Though the first application is to NASA outreach products, external use and further development by commercial and non-commercial organizations is strongly encouraged.. See Research & Development / Technology Transfer.

1.2.2 End-Users

MDE can enable or enhance usability of graphs by blind and visually-impaired users by providing descriptions where no descriptions or inadequate descriptions were provided before, or by complimenting other methods of graph description for the vision impaired. For example, it can reduce the need for assistance from a sighted person to describe a graph. It might be also used to compliment other graph accessibility methods, such as tactiles or haptics.

Text descriptions generated by MDE can be read by screen readers such as Jaws for Windows, or can be input to speech synthesizing software¹ to create self-voicing applications. MDE sonification² output can be played on standard computer speakers or headphones. MDE also provides traditional "drawn" graphs with user-settable colors and line thicknesses to aid users with differing vision-impairments.

¹ Such as the Java Speech API.

² "Sonification is the use of nonspeech audio to convey information.", Sonification Report [10]

Introduction

1.2.3 Product Developers

In synthesizing text descriptions of graphs, MDE attempts to mimic how a (qualified) person might describe the graph to a person who is vision-impaired by: 1.) using natural language to describe graphs, and 2.) providing intelligent, on-demand (real-time) description capabilities. These capabilities have obvious applications to reducing the need to write alternative text (alt text) for graph images. Alt text descriptions are also limited in use for captioning dynamic data displays. Since MDE can generate descriptions on the fly, dynamic displays that were previously impossible to caption can now be captioned.

MDE's description and sonification components can be "plugged in" to various applications (standalone, client-server, etc.) and user-interfaces (pure text, GUI, etc.). For convenience, MDE provides a set of *accessible* Java Graphical User Interface (GUI) widgets for end-user control and display of MDE elements. MDE's graphical components can similarly be plugged into various applications.

MDE's description, sonification and graphing components can be used independently, or in combination. MDE's architecture supports solution synchronization among components when text, sound and graphing are used in combination. This document will describe MDE *core* components and how to use their functionality, individually and in combination.

1.2.4 MDE Demonstration

An application which demonstrates many of MDE's capabilities (from the end-user perspective) is NASA MathTrax, developed by the MDE development team. MathTrax illustrates MDE capabilities with a graphing calculator interface allowing the user to graph equations, data and simulation results. The simulations included in MathTrax are an accessible rocket simulation and a roller coaster simulation game. MathTrax is available at the NASA Learning Technologies website: (<http://prime.jsc.nasa.gov>). Looking at MathTrax will give you a good idea of MDE's current capabilities and potential for its application to other products.

1.2.5 MDE Availability

Source code demonstrating how to use the MDE API is provided in demo applications in the library distribution. Some of the demo applications are also presented in the MDE Tutorial section of this document.

1.2.6 Research & Development / Technology Transfer

The MDE solution engine, visible graph, and other components were developed in house to facilitate our own research and development into accessible graphs for NASA. To our knowledge, the MDE text description capability is a novel contribution to graphing and to accessible math and graphing technologies. We provide all of our MDE library components, including the non-novel ones, as a convenience for NASA developers who would like a reusable component they can quickly and easily plug in to make their graphs more accessible.

Introduction

We will also make the library available to commercial and non-commercial entities interested in furthering our research or applying the technology to improve the accessibility of existing graphing applications. Integration of MDE with commercial and non-commercial software is encouraged. MDE will be released as NASA Open Source [7] to facilitate further development and applications of the technology.

1.3 Audience

This guide provides an overview of MDE capabilities and usage requirements that can be used by software managers/analysts/programmers to evaluate MDE for use. It provides enough detail to assist programmers in getting started with the MDE API, implementing MDE core functionality.

You do not have to be a mathematics expert to use the MDE Library capabilities. The interface provides simple, intuitive methods for requesting descriptions, graphs and sonifications from equation and data input.

1.4 Assumptions and Limitations

We assume the programmer knows Java, and/or how to call Java libraries from the language you're programming in.

We do *not* assume you are a math whiz!

This document describes an *alpha* version of the MDE library:

- There are planned upgrades, which may or may not be backwards compatible with the alpha version.
- Only the compiled-source library is available for alpha. Future distributions will include source code for those wanting to extend or customize MDE capabilities.
- There are some known bugs in the code.
- The default text descriptions provided by MDE are still in the experimental phase and will possibly change.³

1.5 Related Documents/Resources

- NASA Learning Technologies, Johnson Space Center [1]
- NASA Learning Technologies Web <http://learn.arc.nasa.gov> [5]
- Math Description Engine (MDE) Compiled Source Code Distribution [2]
- Math Description Engine (MDE) Developer's Reference/API Javadoc [3]
- NASA MathTrax website and software application [4]
- NASA Learning Technologies Project Requirements
- NASA Learning Technologies Projects Description [6]

³ Future versions of MDE will allow developers to add to the default set of descriptions, i.e., develop and use their own descriptions.

Introduction

- NASA Learning Technologies Operating Plan
- NASA Open Source Software [7]
- NASA Education Enterprise Strategy [9]

1.6 Feedback

Please email suggestions, bug reports, use reports, and other feedback to info@prime.jsc.nasa.gov

Section 2 MDE Functional Overview

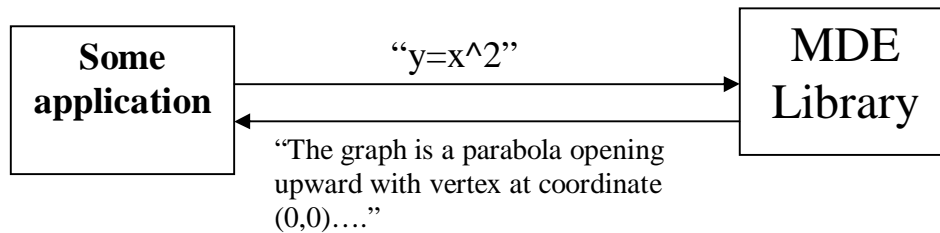
This section gives a high-level overview of MDE core functionality.

2.1 High Level Functional Description

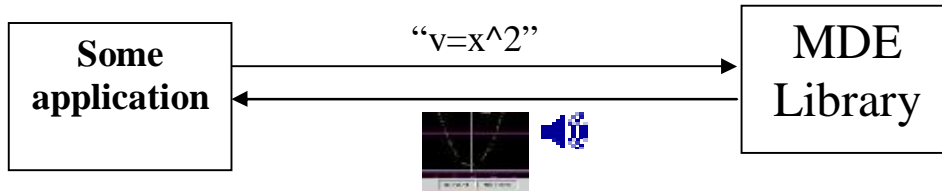
Given a mathematical equation or an ordered set of data (or both⁴), MDE can describe, sonify and/or draw the resulting graph. Figure 1 shows some typical use scenarios for the MDE Library.

Figure 1 Example uses of the MDE Library

Example 1: Request to MDE description component for a text description of the graph of $y=x^2$.



Example 2: Request to MDE graphing and sonification widgets to graph and sonify $y=x^2$.



Example 3: Request to MDE for sonification of data points.

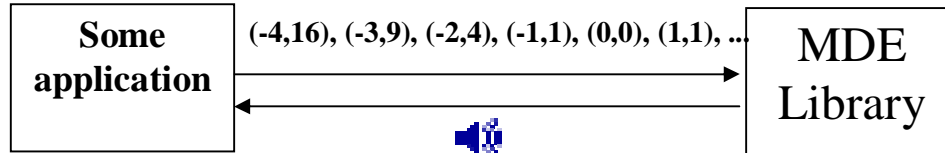


Figure 1 shows three example uses of the MDE Library. Example 1 shows an application requesting a text description from MDE, passing in the string equation $y=x^2$. MDE returns a (string) text description "The graph is a parabola opening upward with vertex at coordinate (0,0)..." Example 2 shows an application requesting MDE graph and sonification components to draw and sonify the graph of $y=x^2$. Example 3 shows an

⁴ MDE can handle multiple equations, multiple data sets, or a mixture of the two.

Functional Description

application requesting sonification of data points such as (-4,16), (-3,9), (-2,4), (-1,1), (0,0), (1,1),..., and MDE returning audio sonification of the graphed input data.

2.1.1 Text Descriptions of Graphs

2.1.1.1 Purpose

One of MDE's main purposes is to provide natural-language descriptions of computer-generated graphs so that vision-impaired users can access them. You can use the Describer class to request text descriptions.

Given a valid equation or data set (See Inputs), MDE solves for the resulting graph and describes it in terms of the curve solution and its individual features. For example, if you input the equation "y=3x", a line with slope 3, you will get one of the two following descriptions, depending on which description mode you request.

Figure 2 Example MDE text description output for the equation "y=3x"

Mode: "visual"

Your input equation is $y = 3*x$. The graph of the equation is a line. It rises steeply from left to right with a slope of 3.

Mode: "math"

Your input equation is $y = 3*x$. The graph of the equation is a line. It rises steeply from left to right with a slope of 3. The graph has an inclination of approximately 71.565 degrees or approximately 1.249 radians. The x-intercept is 0. The y-intercept is 0. The ascending region is $\{x \text{ such that } -\infty < x < \infty\}$. The equation is a linear equation. The domain of the equation is $\{x \text{ such that } -\infty < x < \infty\}$. The range of the equation is $\{y \text{ such that } -\infty < y < \infty\}$.

If you now request a description for a different line, say "y=4x", the relevant values - like the printed equation, the slope, the direction, etc. - will change. (Actually, the entire description is regenerated with the new values substituted into the appropriate locations.) See Description Engine Approach.

The MDE Tutorial will show how to use the Describer class to generate text descriptions.

More example default descriptions are in Appendix A: Current Text Description Examples.

2.1.1.2 Modes

MDE provides two description modes, "visual" or "math", for graphs of equations. For data, one type of description is currently provided (independent of mode setting).

Functional Description

The "visual" mode attempts to describe graphs of equations in qualitative terms, without too much math jargon. The "math" mode describes graphs of equations in terms of the mathematical features and their values (and may also include qualitative information).

2.1.1.3 Content

2.1.1.3.1 *Composite Descriptions*

Currently, MDE only returns composite descriptions, i.e., a description consists of several sentences describing the curve, and is returned as a single String value. We are also developing capabilities to output curve descriptions and individual feature descriptions in XML format. This enhancement will give application developers maximum flexibility for how to use and present description data, since you will have a choice of presenting a paragraph of data or selecting a single feature description to display, or only displaying descriptions for the features that have changed. The XML output option is a desirable usability enhancement, especially when the end-user is looking at the effects of changing a single equation parameter, like slope for a line. In a nutshell, XML output will enable intelligent application control and/or user-control over what portions of the description are displayed.

2.1.1.3.2 *Text or HTML option*

The MDE API can return descriptions in TEXT or HTML format. The HTML descriptions contain hyperlinks to some mathematical term definitions, but the implementation of html glossary links is minimal at this time. As stated above, we are currently developing an XML output option for the descriptions.

2.1.1.3.3 *Display Options*

Display of the text description String is up to the application developer. We may provide a Java GUI description widget in the future, as a convenience for developers. The current String output option provides for the maximum in presentation flexibility for applications and web pages.

2.1.1.4 Description Engine Approach

MDE generates descriptions by using XSLT templates. If you just want to use the default templates MDE provides, you may not care how MDE generates descriptions, but a high-level understanding of what's going will probably be useful to you. If you want to create your own templates, create additional description modes, or otherwise extend MDE capabilities, refer to the MDE open source code release [2], MDE Developer's Reference/API Javadoc [3], and MDE technical papers on the website [2] describing the method.

MDE templates contain English-language phrases with value placeholders for the dynamic values. To demonstrate, Figure 3 shows our "visual" line description example, from above. The dynamic values are: " $y=3*x$ ", "line", "rises steeply from left to right", 3. The remaining words or phrases are the template's static text: "Your input equation is....", "The graph of the equation is....", "It....with a slope of...."

Functional Description

Figure 3 Line description, with the dynamic values highlighted.

Your input equation is $y = 3*x$. The graph of the equation is a line. It rises steeply from left to right with a slope of 3

The templates contain rules for when to use a particular word or phrase (e.g., rise/fall; steeply/gradually, a/an) and how to combine individual feature descriptions to create a composite graph description. A default template rule, for example, is to state what the input equation is, then what the graph represents (e.g., line, parabola, etc), then to describe what the curve looks like and/or it's mathematical features.

As we stated in Composite Descriptions, we are implementing the capability to return descriptions in XML format so applications can override the defaults for the ordering and selection of features described. See the aforementioned references [2], [3] if you want to use different language in your templates. Send us feedback if you want to suggest modifications to the defaults.

2.1.2 Sonification of Graphs

The MDE API can generate sound alone with the Sounder class. Run the command line sonification demo for an example of MDE sonification generation with no graphical interface. The source code example is provided in the MDE Tutorial and the demo programs.

If you want a ready-made GUI widget to let the user control the sound, you can use our the SoundControl class. The NASA MathTrax Application demonstrates the various sonification controls that are available in this widget. A screen shot of the MDE SoundControl widget is shown in Figure 4.

Figure 4 MDE SoundControl Widget

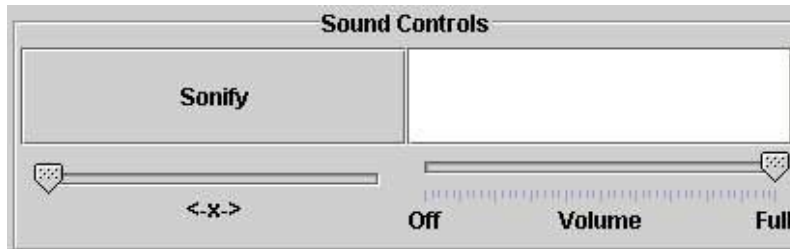


Figure 4 shows the MDE SoundControl GUI widget, which consists of four areas: a Sonify button which lets the user play/pause the sonification; a slider bar which allows manual control of the sonification back and forth across the graph domain; a read-only graph values area which displays the abscissa and ordinate (x and y) values currently being "sounded"; and a volume control. Note that the volume control adjusts the sonification volume only. It does not adjust the computer speaker volume, an important requirement for screen reader users!

Functional Description

2.1.3 Drawings of Graphs

The MDE library contains a Java GUI Widget for displaying Cartesian graphs. Many color and line setting adjustments are available. Zoom and pan capabilities exist.

The graph can be used with sonification trace turned on or off. If the sonification trace is on, a visible bar or circle animation trace will sweep the graph as the sonification plays out. The position of the trace corresponds to the points being currently sonified. For Cartesian graphs, the sweep graphic is a vertical line that moves across the graph from the left x bound to the right x bound. For polar graphs, the sweep is a circle or ball that traces the curve from theta equals zero to 360 degrees. Figure 3 shows a screenshot of the MDE CartesianGraph widget.

Figure 5 Cartesian Graph Widget

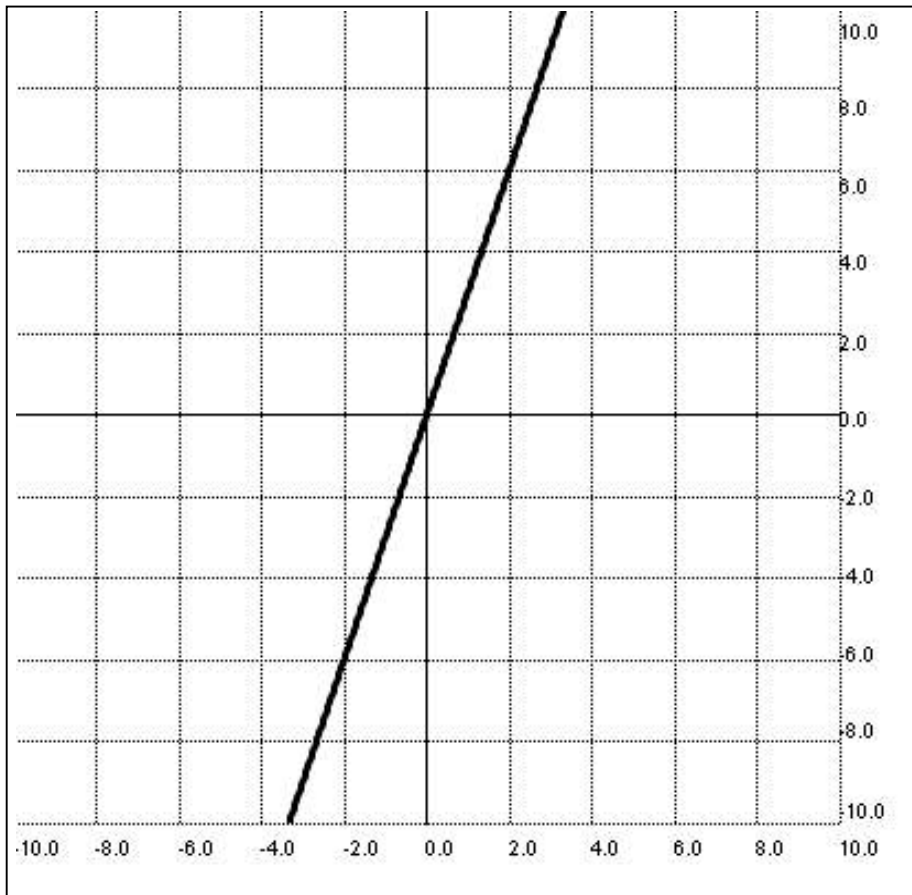


Figure 5 shows a CartesianGraph widget with a line graphed on it. The graph contains axis lines and numbered grid lines.

At this point, some readers may want to jump ahead to the MDE Tutorial for some program use context, and then return to the Inputs and Outputs sections for details on those topics.

Functional Description

2.2 Inputs

2.2.1 Mathematical Equations

2.2.1.1 Supported Equation Types

MDE has the ability to classify and generate *text descriptions* for equations in two variables such as conic sections⁵. MDE sonification and graphing components can handle some equations for which descriptions are not yet available. Table 1 shows currently supported equation types with examples. The examples are all valid MDE input formats. See Equation Syntax for more on equation format.

Table 1 Equation Types with Examples

Equation Type	Cartesian Form Example(s)	Polar Form Example(s)
NULL SET	$x-c=x$	$r-2=r$
SINGLE POINT	$x^2+y^2=0$, $x^2+(3-y)^2=0$	$r=0$
ALL POINTS	$x=x$	$r=r$
VERTICAL LINE	$x=c$	$r=1/\cos(\theta)$
HORIZONTAL LINE	$y=c$	$r=1/\sin(\theta)$
TWO PARALLEL LINES	$x^2=c$, $y^2=c$, $(x-y)^2=c$	
TWO INTERSECTING LINES	$x^2-(x-y)^2=0$	
SLOPING LINE	$y=3x+4$, $y=mx+b$	
PARABOLA	$y=x^2$, $y=(ay-x)^2$	$r=-2a/(1+\cos(\theta))$
HYPERBOLA	$x^2 - y^2 = 0$	$r=1/(2-2*\cos(\theta)+\sin(\theta))$
ELLIPSE	$x*y=1$ $x^2/a^2 - y^2/b^2 = 1$	
CIRCLE	$x^2 + a*y^2 = 25$ $x^2 + y^2 = 25$	$r=5$
POLYNOMIALS	$y=x^3$, $y=3x^5$	
RATIONAL FUNCTIONS	$y=x/(1+x^2)$	
ABSOLUTE VALUE	$y=\text{abs}(x)$	
LOGARITHM	$y=\log(x)$	
TRIG FUNCTIONS (see Note on Functions)	$y=\sin(x)$	
POLAR ROSE		$r = \sin(a*\theta)$ $r = \cos(a*\theta)$

⁵ Technically, the equation solver should handle any equation of the form $F1(y) = F2(y)$ where $F1$ and $F2$ are rational functions of the independent variable whose coefficients can be any legal expression in the independent variable.

Inputs

POLAR LEMNISCATE		$r^2 = a^2 \cos(2\theta)$, $r^2 = 2c^2 \cos(2\theta)$
POLAR TROCHOID		Cardioid: $r = a(1 - \cos(\theta))$, $r = 2b(1 - \cos(\theta))$ Other: $r = a \cos(\theta) + b \sin(\theta) + c$, vary c $r = a \cos(\theta) + b \sin(\theta) + c$, large c $r = 1 + \cos(k\theta)$ $r = a \cos(2k\theta) + b \sin(2k\theta) + c$, c small $r = a \cos((2k+1)\theta) + b \sin((2k+1)\theta) + c$

Note On Functions: If MDE cannot identify a curve as belonging to one of the supported types, MDE will check to see if the curve behaves like a function⁶ over the current graph bounds. If it does, then MDE will generate a "Function Over Interval" description. Function Over Interval descriptions describe the functional characteristics of the curve such as rise, fall, maxima, minima, etc. They are also used for data descriptions (See Appendix A: Current Text Description Examples).

2.2.1.2 Multiple Inputs

MDE can describe, graph and sonify *multiple equations at once*, e.g., it can graph two equations on the same grid, sonify both and describe both. Multiple input management will be discussed in the MDE Tutorial.

2.2.1.3 API Input Format

MDE API methods take equations stored as Java String objects, for example:

```
String equation = "y=3x+4";
solver.add(equation);
```

or

```
AnalyzedEquation solvedEquation = new AnalyzedEquation(equation);
```

These methods will be described in the tutorial.

2.2.1.4 Equation Syntax

MDE equation input allows for the equations to be entered in various forms following a basic LEFT EXPRESSION = RIGHT EXPRESSION rule. This allows for greater flexibility for the user in that equations can be entered in various standard forms, and the user doesn't have to manipulate the equation to conform to an application input constraint. All of the following example forms of the line $y = 3x + 5$ would be recognized by MDE, as would other forms conforming to LEFT EXPRESSION = RIGHT EXPRESSION:

⁶ Only one y value for each x value sampled over the interval

Inputs

$$y = 3x + 5$$

$$y - 3x = 5$$

$$3x = y - 5$$

2.2.1.4.1 Reserved Symbols

The letters *a-h*, *k* and *m* are reserved for equation **parameters** (upper or lower case). The other letters of the alphabet may be used to represent **variables**. Equations can also be entered in terms of *r* and *theta* and they will be recognized as a polar form. **Table 2** summarizes.

Table 2 Equation Symbols

MDE Equation Elements	Valid Symbols
Parameters: Cartesian or Polar Equation	a-h, k, m (upper or lower case)
Variables: Cartesian Equation	i,j,l,n-z (upper or lower case)
Variables: Polar Equation	r, theta (lower case only)

2.2.1.4.2 Assignment of Variables

Since MDE allows various symbols to be used for variables, a convention is needed to assign one variable as the dependent (y axis) and one as the independent (x axis) variable. MDE assigns this for Cartesian equations based on alphabetical order. *The lower-order letter in the equation is assigned as the independent variable.*

For example, in an equation in *x* and *y*, *x* comes before *y* alphabetically, so MDE will treat *x* as the *independent* variable and *y* as the *dependent* variable.

2.2.1.4.3 Operators

Table 3 lists the valid MDE operators.

Table 3 MDE Supported Operators

Symbol	Operation	Example
+	Add	$y=3+x$
*	Multiply	$y=3*x$
-	Subtract	$y=x-2$
/	Divide	$y=x/4$
^	Exponent	$y=x^2$
=	Equal	$y=3*x^2-1$
sqrt	Square Root	$y=\text{sqrt}(x)$
abs	Absolute Value	$y=\text{abs}(x)$
exp	Exponential Function	$y=\text{exp}(x)$

Inputs

log	Natural Logarithm	$y=\log(x)$
pi	Real Number Pi	$y=\pi*x$
sin	Sine	$y=\sin(x)$
cos	Cosine	$y=\cos(x)$
tan	Tangent	$y=\tan(x)$
()	Parentheses	$y=(x/3)^2$

2.2.1.4.4 Syntax Shortcuts

MDE correctly interprets certain omissions in syntax. For example, a multiplication sign (*) is frequently omitted in written equations, and that omission is correctly interpreted by MDE. Both

$$y=3x \text{ and}$$

$$y=3*x$$

are treated as y equals 3 times x .

2.2.2 Time-Series Data

2.2.2.1 Supported Data Types

MDE can generate output from time-series data, or more specifically, a vector-valued function of a single variable (one or more scalar functions of a common scalar variable). For example, $f(t) = [\cos(t), \sin(t), t]$ is a vector-valued function of t . If that's too mathematical for you, read on, and you'll get it!

Time series data is typically represented in column-separated tables, as Table 4 illustrates. In column 1, time is the independent variable, and the remaining six columns contain values of time-dependent quantities computed by a rocket simulation: acceleration, velocity, altitude, etc..

Table 4 Column separated time-series data from a rocket simulation

TIME (SEC)	ACCEL (M/S ²)	VEL (M/S)	ALT (M)	RNG (M)	FLTEL (DEG)	DRAG (N)
0.050	0.000	0.000	0.000	0.000	85.000	0.000
0.100	8.502	0.141	0.002	0.000	85.000	0.000
0.150	23.165	0.933	0.025	0.002	85.000	0.000
0.200	63.664	3.103	0.118	0.010	85.000	0.000
0.250	104.336	7.261	0.367	0.032	85.000	0.004
0.300	104.581	12.565	0.861	0.075	85.000	0.023
0.350	84.362	17.284	1.608	0.141	85.000	0.068
0.400	63.966	20.987	2.566	0.226	84.871	0.128
0.450	52.286	23.887	3.685	0.327	84.758	0.189
0.500	40.559	26.202	4.935	0.443	84.654	0.245

Inputs

The independent variable does not have to be time. It could be a spatial variable, such as x , for plotting x vs y . Or it could be some other scalar variable, like *temperature*, in a data set with measurements of temperature and pressure (the pressure of something measured at each temperature value). Table 5 shows a valid form of spatial data. It is a table of x and y values for the line $y = 3 * x$.

Table 5 Table of x and y values for graph of $y=3*x$

x	y
-4	-12
-3	-9
-2	-6
-1	-3
0	0
1	3
2	6
3	9
4	12

Specific input formats that MDE allows are discussed in the next section. But in general, the time-series data must conform to the following conventions:

- the first column must contain the values for the independent variable,
- the first column values must be numeric: integers or decimals,
- the first column must be in ascending order

MDE analyzes the first column of data to determine if values are consistently spaced, or whether there are "gaps" in the data. For example, the jump from 4 to 8 in the series 1, 2, 3, 4, 8, 9, 10 indicates a gap in the data between 4 and 8. If MDE finds a gap, it will treat the data in segments rather than interpolating between gap intervals. This analysis is only performed for file input, at the time of this writing.

2.2.2.2 Multiple Inputs

MDE can describe, graph and sonify multiple data plots at once. How this is handled through the API will be discussed in the MDE Tutorial.

2.2.2.3 Input Formats

2.2.2.3.1 Data Format Constraints

MDE input data must consist of real integer or decimal numbers (with the exception of column headers). Times must be represented as real integer or decimal numbers. MDE does not currently support mixed time formats (numeric and character).

2.2.2.3.2 API Data Input

The MDE AnalyzedData class is used to initialize two related data columns from two `double[]` arrays. The AnalyzedData constructor looks like this:

```
AnalyzedData(String xName, String yName, double[] xData, double[] yData)
```

Inputs

The xData and yData names denote abscissa (horizontal) and ordinate (vertical) values for the 2D graph, respectively..

2.2.2.3.3 File Data Input

A text file containing column-separated numeric data, with or without headings, is input using the TextDataFileParser class in the io package. Columns may be separated by tabs, commas, or spaces.

2.2.2.3.4 File Creation

Data tables can be created in programs like Excel, and saved as tab or comma-separated-values (CSV), or a text editor can be used. If column headers are not included, TextFileDataParser will insert defaults.

Table 5 showed a tab-separated data example. The same data is shown in comma-separated-value format in Table 6.

Table 6 Comma-separated data table example

```
x,y
-4,-12
-3,-9
-2,-6
-1,-3
0,0
1,3
2,6
3,9
4,12
```

There are no restrictions on filenames or filename extensions beyond what Java supports.

2.2.3 MDE Properties

2.2.3.1 Property Values

MDE uses a class named MdeSettings to store and read properties of graph, sonification and description components in a text properties file, so that user-preferences may be stored between application runs.

At the time of this writing (March 2005), all but one of the MDE component properties are graph-related. There is one property for setting an initial description mode⁷. The current properties and their default values are shown in Figure 6.

⁷ Provides consistency of major components API and allows for expansion of properties without impacting the constructors.

Inputs

Figure 6 MDE Default Properties

```
axisColor = Color.white;
backgroundColor = Color.black;
gridColor = Color.magenta;
lineColor = Color.yellow;
lineSize = 2; // medium
dataPointColor = Color.red;
dataPointsShown = true;
autoscaleGraph = true;
traceOn = true;
descriptionMode = "visual";
```

2.2.3.2 Property Files

Properties are stored/read from the user's home directory as specified by a call to Java's `System.getProperty` method (e.g., Documents and Settings on Windows O/S):

```
String folder = System.getProperty("user.home");
```

If you do not specify a filename on construction,

```
MdeSettings myMdeSettings = new MdeSettings();
```

`MdeSettings` will write the properties to a file named *MDE_Properties.properties* (if the application/user changes one of the defaults).

If you do specify a filename on construction,

```
MdeSettings myMdeSettings = new MdeSettings("myMDEProperties.txt");
```

then `MdeSettings` will attempt to initialize properties from this file. If the file doesn't exist, MDE uses the default properties (set internally).

MDE will write the properties to the specified file (or default name) once the application/user changes a property value. This occurs automatically. The application does not need to explicitly invoke a save with the `save()` method.

2.2.3.3 Property Methods

`MdeSettings` provides getters and setters for each property. See the `MdeSettings` class description in the MDE Developer's Reference/API Javadoc [3].

2.3 Outputs

This section focuses on the three core output products of the MDE API. These consist of method outputs (e.g., text descriptions), graphical user-interface outputs (e.g., drawings of graphs), or audio interface outputs (e.g., sonification output to speakers). For other API outputs, refer to the MDE Developer's Reference/API Javadoc.

Outputs

2.3.1 Text Descriptions

The Describer class returns text or html formatted descriptions as Java String, using one of two API methods, for example:

```
Describer describer;  
...  
String description = describer.getDescription("y = x^2", "math");  
  
String descriptions = describer.getDescriptions("visual");
```

The first case requests a *math* description for the equation "y = x²".

The second case gets *visual* descriptions for all AnalyzedItems that are currently "in focus" within the Solver server class. More about this option will be explained in the MDE Tutorial.

The descriptions are returned as text or html, depending on what format you previously specified via the setOutputFormat method:

```
public void setOutputFormat(String outputFormat)
```

2.3.2 Cartesian Graphs

2.3.2.1 Settings

Features of CartesianGraph include the ability to easily change the following settings: graph line thickness, line color, axis color, gridline color, background color, data point color, show/hide data points, show/hide sonification trace and auto-scale. Auto-scaling is always on for polar equation graphs, at the time of this writing.

The ability to change color settings, line thicknesses, and other visible features are important to enable users with low-vision, color-blindness, or other vision problems to set graph display characteristics that work best for them.

Figure 2.5 shows four examples of CartesianGraph's abilities to display different curve types and use different settings.

2.3.2.2 Zoom and Pan

CartesianGraph, when used with keyboard controls, provides a pan left/right/up, down capability using arrow keys. With mouse controls, you can click to re-center the graph. With zoom controls, you can zoom in or out, i.e., decrease or increase the x-bounds of the equation or data graph.

Outputs

2.3.2.3 Multiple Curves

CartesianGraph can display multiple curves on the same graph. For example, if provide Solver two equations and then request a drawing from CartesianGraph, two curves will be drawn, as in Figure 8.

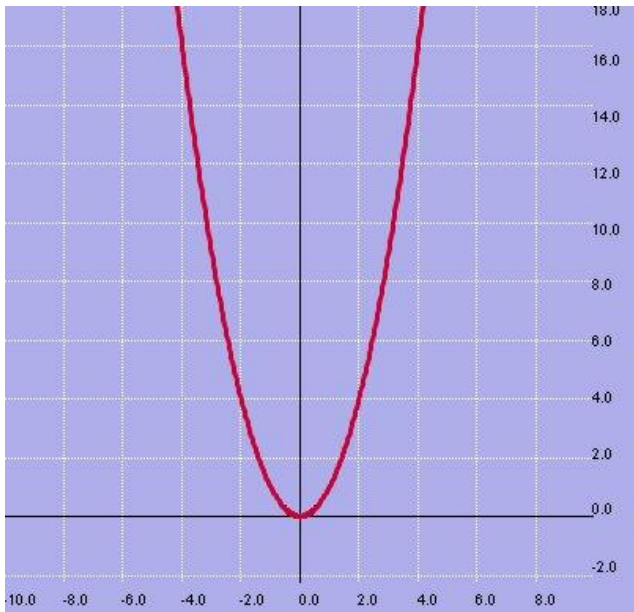
2.3.2.4 Sonification Tracer

When used with MDE SoundControl class, CartesianGraph can display a sonification tracer bar for Cartesian equation graphs and a ball tracer for polar equation graphs. Figure 7 d shows the vertical tracer bar as a vertical white line near the maximum of the curve.

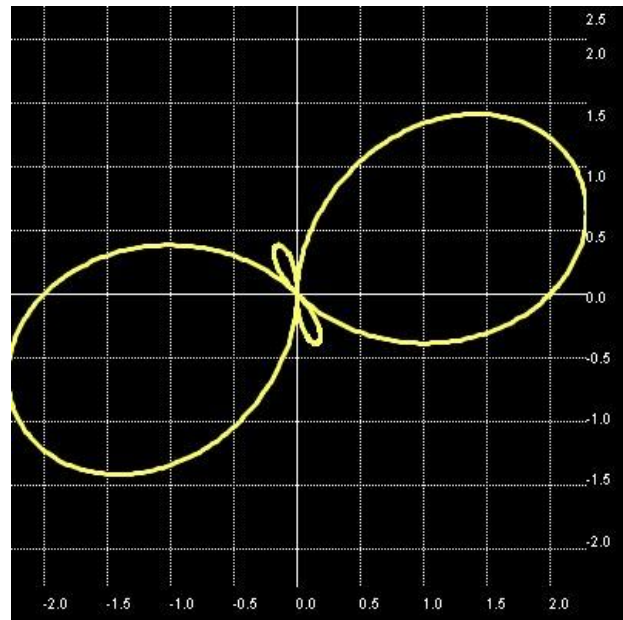
2.3.2.5 Graph Zoom/Reset Widget

MDE also provides reusable graph controls widget class, IncrementXButtons, that can be used to control zooming in/out and resetting of default bounds.

Figure 7 Four CartesianGraph output examples.

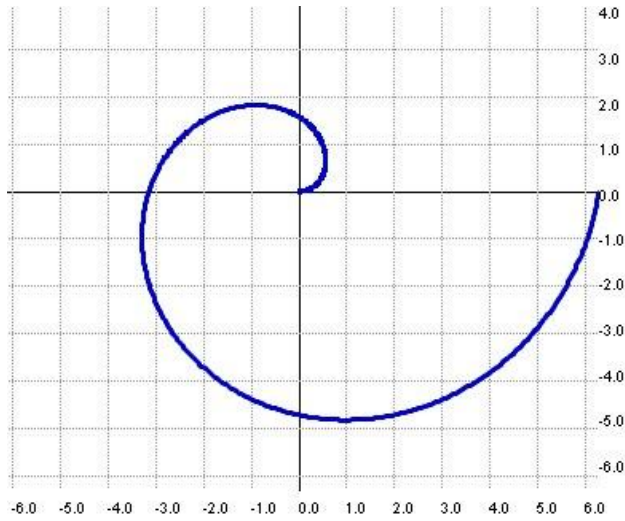


7 a. Cartesian equation plot with thick red line on light blue background, dark blue axes and white grid lines.

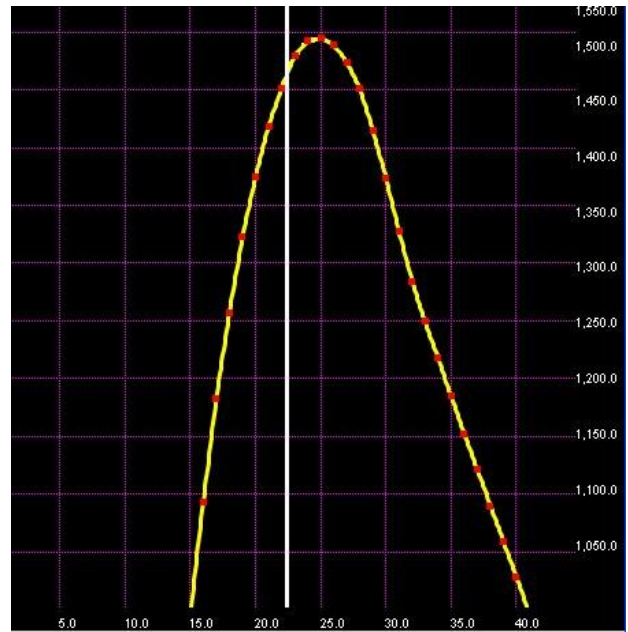


7 b. Polar equation curve with yellow graph line on black background.

Outputs



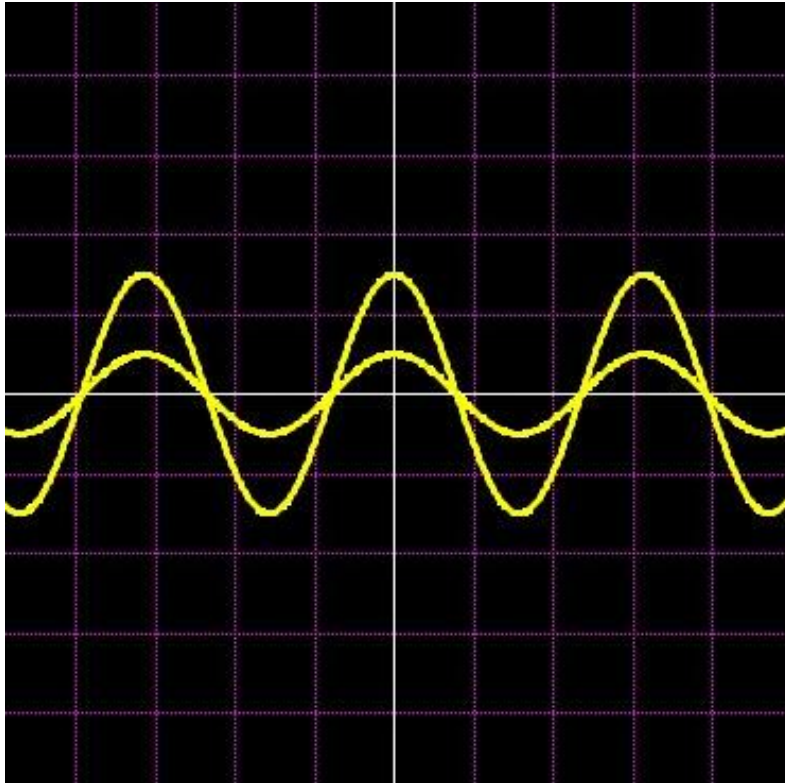
7 c. Polar equation plot with dark blue graph line on white background with gray axes and gridlines.



7 d. Plot of a section of rocket altitude data (red dots) on a yellow curve (interpolated graph), with sonifier trace (vertical white bar) paused near the maximum value.

Outputs

Figure 8 The curves $y=\cos x$ and $y = 3*\cos x$ drawn on the same grid



2.3.3 Graph Sonification

MDE sounds out graphs using audible tones. If you've never heard a sonified graph, run the MathTrax application to hear some examples. Other applications you might look at are the ViewPlus Accessible Graphing Calculator [12] and The vOICE [13]. The International Community for Auditory Display (ICAD, <http://www.icad.org>) [11] is a good source of information on sonification research and applications.

2.3.3.1 Sounder and SoundControl Classes

You can use the Sounder class or the SoundControl class/widget to output audio sonifications of graphs. As discussed above, you can also create a visual sound trace on the CartesianGraph if you use MDE graphing and sonification capabilities together. Use of MDE sonification classes is covered in the tutorial.

2.3.3.2 Left/Right Panning and Up/Down Pitch Renderings

There are many ways to represent data with sound. A typical approach, and the one MDE uses, is to scan the graph from left to right to create a visualization of the behavior of the graph, as a sighted person's eyes might do.

When we use sound instead of graphics to represent data for the visually impaired, we attempt to convey similar or complimentary mental images and indicators that the graphic would provide to a sighted person. A common approach to representing a 2D graph is to

Outputs

use tonal pitch for up and down, and left to right panning for side-to-side. Panning gives a sense of which horizontal location corresponds to the current vertical position (current pitch). You can hear it best if you wear headphones or position yourself between a left and right speaker.

For graphs of Cartesian equations or time-series data, MDE sonifications follow the graph values from the left boundary to the right boundary. For example, if we want MDE to sonify a Cartesian equation, graphed from $x = -10$ to $x = +10$, MDE will do the following:

- Generate an ordered set of sonification data points by solving the equation at some increment of x over the interval x in $[-10, 10]$. Ordering is, of course, from $x = -10$ to $x=10$.
- Convert the data point values to sound, according to the rules outlined in Table 7 and send the resulting audio to the computer audio output device.

2.3.3.3 Sound Sweep Speed Control

Another factor that enters into conveying graph information through sound is how quickly the sound information can be processed by the person listening. A sighted person can usually process a visual graph at their own rate. When sound is used, we ideally need to give the user some control over the speed of sound conveyance. One way to do this is by changing the speed of the data sweep (or pan). MDE provides this control in the `Sounder` class:

```
public void sweep (double sweepTimeInSeconds)
```

MDE also lets the user provide manual control over the sound sweep using the `Explore Values` slider bar in the `SoundControl` class/widget. The user can navigate to a point of interest on the graph using sound, and then view the x and y value(s) at that point in the `SoundControl Graph Values` window.

2.3.3.4 Axis Crossing Indicators

Sonification also employs distinctive sounds to distinguish different semantic elements of data and/or its visual representation. We referred above to the ability of a sighted person to take as much time as needed to examine and process information in a visual graph, and the need to provide similar control for the user relying on sound only.

The flip side of this is that a sighted person can process a lot of visual information at once, initially viewing the graph in its entirety and processing a lot of semantic information very quickly. For Cartesian graphs the semantic information includes coordinate system, vertical and horizontal graph bounds, and how the data plot lies on the Cartesian plane. When relying on sound to convey the same information, one has to decompose these meaningful elements and select sounds to convey them as clearly as possible.

Outputs

MDE uses different tone quality (or timbre) to convey whether graph data falls above or below the x-axis. MDE also uses a clink sound to indicate a y-axis crossing.

A summary of current MDE sonification techniques is shown in Table 7.

Table 7 Visual and Sound Representations of Cartesian Graph Elements

Graphical/Visual Representation	Sound Representation
A given point has a vertical position within the range of vertical positions (y coordinates) on the graphing plane. y-values increase from bottom to top.	Relative pitch represents the up and down "behavior" of the graph points. The tone increases in pitch as y values increase and vice versa.
A given point has a horizontal position within the range of horizontal positions (x-coordinates) on the graphing plane. x-values increase from left to right.	Left/right audio panning. When we're sonifying the left side of the graph the sound appears to be on our left side. It appears to be centered when we're in the middle of the graph's x bounds and to the right when we're sonifying the right side.
A given point is above or below the x-axis.	Timbre (tone quality) changes occur depending on whether the point being sonified is below and above the x-axis.
The graph crosses the y-axis.	y-axis crossing is indicated with a clink sound.
There is no y value for the current point.	No sound is heard. (An improvement might be to optionally provide a sound indicator that the graph is being "panned", but there are no points in the current location.)

Planned upgrades for MDE sonification include providing options for tone quality and type. Individual users process sounds differently. Methods that work well for one person do not work well for another, whether or not a hearing impairment is involved. By providing more sound options, applications implementing MDE sonification can then provide multiple user-settable options to accommodate these user differences.

2.3.3.5 Volume Control

MDE also enables graph sound volume control. The SoundControl class provides a volume control widget.

Providing for separate graph sonification volume control lets vision-impaired users turn down the sonification without turning down the audio for their computer and screen reader!

Outputs

2.3.4 MDE Properties

MDE properties is currently the only file output from the MDE library. See the MDE Properties description above.

Section 3 Using the MDE API

This section describes the core components of the MDE Library and provides a tutorial for using the API to implement the core functionality. For a complete reference on the public API, consult the MDE Developer's Reference/API Javadoc.

3.1 Configuring MDE for use in your software

3.1.1 Download

Download the MDE compiled source library from the MDE Developer's Site at <http://prime.jsc.nasa.gov/MDE>.

Download MDE demo applications from <http://prime.jsc.nasa.gov/MDE>, if desired.

Save to a directory of your choice.

Also see Java JDK and Java Access Bridge configuration requirements below.

3.1.2 Configuration

3.1.2.1 Java JDK and JRE Requirements

You need Java JDK/JRE 1.4.2_04 or higher to use the MDE Library API in your Java application. Get the Java JDK at <http://java.sun.com>.

3.1.2.2 Java Access Bridge Requirements

If you want to use or test the MDE interface to screen readers, or use the Java Accessibility API to provide self-voicing or screen reader access to your own front-end to MDE, you need to download the Java Accessibility Bridge. Version 1.2 is recommended.

You'll also need a Java-capable screen reader - unless you implement self-voicing. If you implement your own Java GUI components, the Java Accessibility API methods `setAccessibleName()` and `setAccessibleDescription()` can be used to provide component names and context help.

3.1.2.2.1 MDE Accessibility to Screen Readers

MDE *GUI* components use the Java Accessibility API to provide information to java-capable screen readers, like Jaws for Windows. Our set of GUI components are provided as a convenience to those not wanting to develop their own, but many developers will prefer to use their own custom GUI components to display and or control MDE-generated text and sonification.⁸ Select MDE components (GUI or non-GUI) based on your application and user-requirements.

⁸ MDE is slated for open-source distribution, so eventually, customization and extension of MDE components will be possible.

Configuring MDE For Use

Screen readers should be able to access MDE Describer output without use of the access bridge if the display application is already accessible. For example, if you send Describer output to a web browser, most screen readers should be able to read it as it is plain text or HTML.

MDE sonification output by Sounder will not interfere with screen reader use. The SoundControl GUI component is a fully accessible front-end to the Sounder class.

Visible graphs are typically not accessible⁹, which is why we developed MDE alternative descriptions! However, the CartesianGraph component does provide information in Java accessibleName and accessibleDescription to announce the presence of the graph and describe its fundamental characteristics (Cartesian plane, default bounds) to screen reader users.

3.1.2.2.2 Java Access Bridge Configuration

To use and/or test the java accessibility features in MDE you will need to install the Java Accessibility Bridge. You will also need a java-capable screen reader.

If MDE GUI components (and/or MathTrax) do not work with Jaws or another java-capable screen reader after you install the Java Access Bridge, try the following:

On Windows OS:

Check your system32 folder, to see if these files are there:

JavaAccessBridge.dll,
JawtAccessBridge.dll, and
WindowsAccessBridge.dll.

If any of them are not there, find them (wherever the access bridge installer put them) and copy them to the system32 folder.

Find the files accessibility.properties, access-bridge.jar and jaccess.jar and configure as shown below in each JRE directory on your system:

In jre/lib: accessibility.properties
In jre/lib/ext: access-bridge.jar and jaccess.jar

Restart your computer.

In general, it's necessary to start the screen-reader before you launch your java accessible applications.

3.1.2.3 Operating Systems

As with other java libraries, you need to include the MDE library jar location in your path. Use the appropriate approach for your operating system and development environment. An example for configuring the Windows path is provided below.

⁹ Though this is an active area of research.

Configuring MDE For Use

Windows:

Put the MDE jar file location in your CLASSPATH environment variable, typically accessed at My Computer/System Properties/Advanced/Environment Variables

For example, if you stored the MDE library jar in a directory named "MathDescriptionEngine", on drive C, include that path in your CLASSPATH:

C:\MathDescriptionEngine\MDE_lib.jar (EXAMPLE PATH ONLY)

3.1.3 License

See the MDE website and code distributions for license information.

3.2 MDE Classes

3.2.1 Main Packages

The MDE currently contains several packages, but only five are of primary concern to the application-developer wishing to implement basic MDE functionality. These are shown in Table 8.

Table 8 MDE Main Packages

MDE packages (gov.nasa.ial.mde.*)	Core Functional Description
describer	Contains classes used to obtain text descriptions of graphs
properties	Contains classes for setting state properties such as color options for graphical elements
solver	Contains classes that: <ol style="list-style-type: none"> 1. analyze input equations and data and generate data used by describer, sound, and graph classes. 2. manage synchronization of text, sound and graph components if they are used in combination. 3. maintain state on multiple input items (equations or data sets) if simultaneous graphs are desired or implied (multi-column time-series data).
sound	Contains classes used to generate and control sonification
ui	Contains classes for drawing and manipulating Cartesian graphs, enabling sonification controls, and enabling equation parameter manipulation.

See the API documentation for a complete list of MDE packages.

3.2.2 Core Classes

3.2.2.1 Core Functionality

To use the MDE library core functionality, you only need to know about a few *classes*. These are listed in Table 9 by *core functionality* (which may be supported by multiple packages):

Table 9 Core Functionality Classes

Core Functionality	Core Classes	Packages
--------------------	--------------	----------

MDE Classes

MDE Properties Access	MdeSettings	properties
Equation/Data Solution and Management	Solver Solution	solver
Text Description Generation	Describer	describer
Visual Graph Generation and Manipulation	CartesianGraph IncrementXButtons	ui.graph ui
Sonification Generation	Sounder SoundControl	sound ui
Data Handling	TextFileDataParser AnalyzedData	io solver

3.2.2.2 Core GUI Classes

MDE provides a number of prebuilt, event-driven GUI components in the *ui* package that you may want to use for input to or control of MDE components, and for display of MDE outputs. These are listed in Table 10.

Table 10 GUI Classes

Function	GUI Component Classes	Description
Display of Input or Computed Data	DataPanel	Displays data column checkbox group and data table
Equation Parameter Input and Control	EquationParameterControl NumberField	Displays editable equation parameter fields
Graphing	CartesianGraph GraphBoundsPanel IncrementXButtons ColorChooser ShowColorChooserAction	Graph display Editable graph bounds Graph zoom controls Graph color control
Sonification Controls	SoundControl	Play/pause, manual sweep, graph values viewer and volume control

MDE Classes

3.2.2.3 Keyboard Control Classes

Users of screen readers do not typically navigate using a mouse. Keyboard controls are used instead.

Navigating an accessible Java GUI, like MathTrax, with a screen reader is accomplished by tabbing from element to element. With Jaws, hitting Tab moves focus one component forward and Shift+Tab moves one component backwards. Providing navigation shortcuts becomes quite important for enabling advanced users to jump directly to the component of interest. The MDE library classes that provide some of this capability are KeyControls and GraphNavKeys. See Table 11.

Table 11 Keyboard Control Classes

Key Controls	Description
KeyControls	Sound slider key controls
GraphNavKeys	CartesianGraph panning keys

3.3 MDE Tutorial

This section provides an introductory tutorial that should help you get started using MDE in your programs. The examples demonstrate MDE's core capabilities and ease of use.

3.3.1 Required Classes - Solver and MdeSettings

There are two classes you will always need whether you want MDE to produce text descriptions, sonifications, or drawings of graphs. Those are *Solver* and *MdeSettings*.

3.3.1.1 MdeSettings: Setting and storing MDE properties

MdeSettings sets and gets property values for describer, sound, graphing components, such as line colors and thicknesses for *CartesianGraph*, and a text description mode. If you specify a filename on construction, *MdeSettings* will attempt to initialize properties from the file (if it exists). If the file doesn't exist, MDE uses a set of default properties, and will automatically write the properties file to the name specified if a property value is changed.

MdeSettings provides getters and setters to properties so your application can access and change them.

To construct *MdeSettings* from properties already stored in a file,

```
MdeSettings myMdeSettings = new MdeSettings(filename);
```

where *filename* is the name of the file you'd like for the MDE Properties file. *MdeSettings* automatically looks for (and saves) this file in the application end-user's "home" directory, e.g., My Documents on Windows machines.

3.3.1.2 Solver: Solution generation and management

Solver serves the function that its name implies - it takes the inputs to be graphed (equations or data) and derives solutions that can be described, sonified or graphed.

Solver also serves as a solution manager and synchronizer for the description, graphing and sonification components. We'll say more about this later, but for now, think of *Solver* as a service class used by describer, graph and sounder clients.

Program initialization to set up *MdeSettings* and *Solver* will look like this:

```
import gov.nasa.ial.mde.properties.MdeSettings;
import gov.nasa.ial.mde.solver.Solver;
...
//Create instances of MdeSettings and Solver
MdeSettings currentSettings = new MdeSettings("myAppsMdeProperties");
Solver solver = new Solver();
```

3.3.2 Core Functionality: Describer, Sounder, and CartesianGraph

3.3.2.1 Describer: Text descriptions of graphs

This section will show how Describer, Solver and MdeSettings are used to generate text descriptions. We'll write a command-line program that prompts the user for an equation, and uses MDE to output a text description of the equation's solution/graph.

To generate text descriptions from equations or data, you'll first ask Solver to solve the input for you, then you will request a text description from Describer.

We've shown how to initialize Solver and MdeSettings above. Describer now requires some set up. First, we create an instance of Describer, passing in the Solver and the MdeSettings object we created:

```
Describer describer = new Describer(solver, currentSettings);
```

Now, we need to tell Describer what output format we prefer. Describer has two output formats TEXT_OUTPUT or HTML_OUTPUT. The format is set using Describer's *setOutputFormat()* method, like this:

```
describer.setOutputFormat(Describer.TEXT_OUTPUT);
```

You can call *setOutputFormat* at any time to change between output formats. For this example, we'll stick with text output.

Describer is now ready to serve as a description "engine".

Let's say our application has an equation " $y=x$ " input by the user, stored as a Java String variable called *equation*. The next step is to give *equation* to Solver and to ask Solver to solve it:

```
solver.add(equation);  
solver.solve();
```

Now, we'll ask Solver whether our equation is describable.

If the equation is describable (the equation is valid), we'll ask *Describer* for the description and print it to System.out:

```
if (solver.anyDescribable()) {  
    String description = describer.getDescriptions("visual");  
    System.out.println(description);  
}
```

MDE Tutorial

And that's basically it! We just used MDE to generate a text description from an equation input.

You probably noticed that `Describer`'s `getDescriptions()` method took a `String` argument "visual". MDE provides the ability to change description "modes" between "visual" and "math"¹⁰. The visual mode is intended to provide a qualitative description of what the graph looks like. The math mode is intended to provide the description of the graph/solution in mathematical terms.

Managing Multiple Inputs

You may also have wondered why we had to *add* the equation to `Solver` before calling `solve()` and why `getDescriptions()` implies plurality. It is because MDE supports graphing/describing/sonification of multiple inputs simultaneously. If, for example, you want to graph, describe, or sonify *equation1* and *equation2* at the same time, you would first add both equations to `Solver`:

```
solver.add(equation1);  
solver.add(equation2);
```

and then you would ask `Solver` to `solve()`. `Solver` will attempt to solve *equation1* and *equation2* and store their solutions separately. (Note: We're not talking about *solving simultaneous equations* here. We're talking about generating a solution for *equation1* and a solution for *equation2* and *graphing, describing, and/or sonifying* them simultaneously.)

Assuming they were both describable, calling `getDescriptions()`, will generate one `String` containing descriptions for both equations. (Requesting a graph will draw both on the same graph, requesting sonification will sonify both - harmony may ensue.)

When you want to clear the solutions in `Solver`, you will use the `removeAll()` method:

```
solver.removeAll();
```

In the example above, if you were processing equations one after the other and only wanted to describe the current equation, you would need to call `removeAll()` before processing the next equation, as in the following listing which is the entire `CommandLineDescriber` program:

Listing 1 Command Line Describer Example

```
import gov.nasa.ial.mde.describer.Describer;  
import gov.nasa.ial.mde.properties.MdeSettings;  
import gov.nasa.ial.mde.solver.Solver;  
  
import java.io.BufferedReader;  
import java.io.InputStreamReader;  
  
public class Tutorial_CommandLineDescriber {
```

¹⁰ Future versions of MDE will allow the user to create their own descriptions and description modes.

MDE Tutorial

```
public static void main(String[] args) {
    //MDE Init:
    MdeSettings currentSettings = new MdeSettings("myAppsMdeProperties");
    Solver solver = new Solver();
    Describer describer = new Describer(solver, currentSettings);
    describer.setOutputFormat(Describer.TEXT_OUTPUT);

    //Process equations
    try {
        InputStreamReader isr = new InputStreamReader(System.in);
        BufferedReader reader = new BufferedReader(isr);

        // Prompt user for input until they enter CTRL-C.
        while (true) {
            System.out.println("\n\nEnter equation (or CTRL-C to exit): ");
            String equation = reader.readLine();

            //Give Solver equation and solve
            solver.add(equation);
            solver.solve();

            if (solver.anyDescribable()) {
                String description = describer.getDescriptions("visual");
                System.out.println("Description: " + description);
            } else {
                System.out
                    .println("MDE could not generate a description for "
                        + equation + ".");
            }

            //Clear Solver so next equation will be processed singly
            //(we only want one description at a time)
            solver.removeAll();
        }
    } catch (Exception e) {
        System.out.println(e);
    }
} // end main

} // end class Tutorial_CommandLineDescriber
```

3.3.2.2 Sounder: Sonifications of graphs

Now let's look at how to use the Sounder class to generate sonifications from equations. It's very similar to text description generation.

We'll modify the CommandLineDescriber example so that instead of generating text descriptions, we generate sonifications. Let's first look at the MDE-specific code.

We initialize MdeSettings and Solver initialization, the same as in the above example:

MDE Tutorial

```
MdeSettings settings = new MdeSettings("myAppsMdeProperties");  
Solver solver = new Solver();
```

The initialization to use Sounder is similar to that of Descriptor. We create a new instance of Sounder, passing in the Solver and MdeSettings objects.

```
Sounder sounder = new Sounder(solver,settings);
```

Sounder is now ready to serve as a sonification "engine".

As before, we get an equation, add it to Solver and ask Solver to solve:

```
solver.add(equation);  
solver.solve();
```

Now, we ask Solver if our equation is sonifiable. If it is, we use Sounder's *sweep()* method to sonify the graph of the equation.

```
if (solver.anySonifiable()) {  
    sounder.sweep(3.0);  
}
```

sweep() takes a parameter, double *sweeptime*, which allows you to adjust the speed of the sonification. The sweeptime corresponds (approximately) to the duration in seconds of the left to right "sweep" of a curve over the *current bounds* of the graph plane. This may include silence if the curve does not span the entire graph plane.

To close the sound stream, we call *close()*:

```
sounder.close();
```

And those are the basics of using Sounder.

Look at the complete CommandLineSonifier in Listing 2.

Listing 2 Command Line Sonifier Example

```
import gov.nasa.ial.mde.properties.MdeSettings;  
import gov.nasa.ial.mde.solver.Solver;  
import gov.nasa.ial.mde.sound.Sounder;
```

MDE Tutorial

```
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class Tutorial_CommandLineSonifier {

    public static void main(String[] args) {

        MdeSettings settings = new MdeSettings("myMDEProps.prop");
        Solver solver = new Solver();

        Sounder sounder = new Sounder(solver,settings);

        try {
            InputStreamReader isr = new InputStreamReader(System.in);
            BufferedReader reader = new BufferedReader(isr);

            while (true) {
                System.out.println("\n\nEnter equation (or CTRL-C to exit): ");
                String equation = reader.readLine();
                solver.add(equation);
                solver.solve();

                //Does user want to sonify equation?
                if (solver.anySonifiable()) {
                    boolean sonflag = true;
                    while (sonflag) {
                        System.out.println("Sonifying " + equation + " from x = " + solver.getLeft() + " to x = "
                            + solver.getRight());
                        sounder.sweep(3.0);

                        //Do they want to hear it again?
                        System.out.println("\n\nSonify again? (y/n): ");
                        String s = reader.readLine();
                        if (s.equals("n")) {
                            sonflag = false;
                        }
                    }
                } // end while sonflag

                sounder.close();
            }
            solver.removeAll();

        } // end while true
    } catch (Exception e) {
        System.out.println(e);
    }

    } // end main
} // end class Tutorial_CommandLineSonifier
```

MDE Tutorial

MDE sonifications play in a separate thread from the program that is calling them, so in our example, the user prompt asking whether to sonify again will probably be displayed before the current sonification ends.

3.3.2.3 CartesianGraph: Drawings of graphs

Our CartesianGraph example will look *very similar* to our programs for Describer and Sounder!

Our CartesianGraph constructor looks like this, with Solver and MdeSettings objects passed in:

```
CartesianGraph grapher = new CartesianGraph(solver, currentSettings);
```

And our request for a graph of our equation or data, looks like this:

```
if (solver.anyGraphable()) {  
    grapher.drawGraph();
```

Let's throw in a little variety this time, and add two equations to Solver (before we request the graph, of course). This will demonstrate MDE's ability to handle multiple inputs, and looks like this:

```
//Give Solver two equations to solve:  
String equation1 = "y=x^2-2"; // a parabola  
String equation2 = "y = x";    // a line  
solver.add(equation1);  
solver.add(equation2);  
solver.solve();
```

The procedure for handling multiple inputs (String equations, AnalyzedEquation object or AnalyzedData objects) is the same whether you're requesting descriptions, sonifications, or graphs. Here's the full program.

Listing 3 CartesianGraph Example

```
import gov.nasa.ial.mde.properties.MdeSettings;  
import gov.nasa.ial.mde.solver.Solver;  
import gov.nasa.ial.mde.ui.graph.CartesianGraph;  
  
import javax.swing.JFrame;  
  
public class Tutorial_CartesianGraph {  
  
    public static void main(String[] args) {  
        //MDE Init:  
        MdeSettings currentSettings = new MdeSettings("myAppsMdeProperties");  
        Solver solver = new Solver();  
  
        //Create a Java Swing window for our graph:  
        JFrame window = new JFrame("Tutorial_CartesianGraph");
```

MDE Tutorial

```
//Create an MDE CartesianGraph instance:
CartesianGraph grapher = new CartesianGraph(solver, currentSettings);

//Add our grapher to the Java window.
window.getContentPane().add(grapher);
window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
window.pack();
window.setVisible(true);
windowToFront();

//Give Solver two equations to solve:
String equation1 = "y=x^2-2"; // a parabola
String equation2 = "y = x"; // a line
solver.add(equation1);
solver.add(equation2);

//If our equation is graphable, draw the two graphs.
if (solver.anyGraphable()) {
    grapher.drawGraph();
} else {
    System.out.println("MDE could not generate a graph for " + equation1
        + " and " + equation2 + ".");
}
} // end main

// Since we're quitting after the first drawing, we won't bother clearing Solver, as we did in the
previous examples.

} // end class Tutorial_CartesianGraph
```

3.3.2.4 Use Solver to Reset Graph (Solution) Bounds

If you've run the above demo programs, you might have noticed the x and y bounds of the graph were [-10,10]. Those are the defaults that Solver.solve() uses.

If you want to change the graph (solution) bounds, you need to tell Solver to generate a new solution for the desired bounds.

If you use MDE graph controls, like IncrementXButtons, you don't even need to tell CartesianGraph to redraw, because a call back is built in. Otherwise, after you change the bounds, you just call CartesianGraph.drawGraph() again. CartesianGraph will pick up the new bounds and solution from Solver.

Use the Solver methods

solve(Bounds b) or
solve(double left, double right, double top, double bottom)

MDE Tutorial

to set your desired solution bounds, like this:

```
solver.solve(-5,5,5,-5);
```

And then ask MDE for a new graph, description or sonification (unless you're using a GUI component where it's a built in call-back action):

```
if (solver.anyGraphable()) {
    grapher.drawGraph();
}

if (solver.anyDescribable()) {
    String description = describer.getDescriptions("visual");
    System.out.println("Description: " + description);
}

if (solver.anySonifiable()) {
    sounder.sweep(3.0);
}
sounder.close();
```

3.3.2.5 Data Input Examples

We've shown how to request a MDE description, sonification and graph from an equation input. Now let's look at how we use data as the input to MDE.

To input data to the MDE Solver, we create an AnalyzedData item from two columns of data. The AnalyzedData constructor we use is:

```
public AnalyzedData(String xName, String yName, double[] xData, double[] yData)
```

where xName and xData are the independent variable heading and data, respectively. yName and yData are the dependent variable heading and data.

In our previous examples, we passed Solver an equation. We could also have used an AnalyzedEquation. For data, we pass Solver our AnalyzedData object. That's the only difference in using data versus an equation as input. Here's our full example program.

Listing 4 MDE Descriptions from Data Arrays

```
import gov.nasa.ial.mde.properties.MdeSettings;
import gov.nasa.ial.mde.solver.Solver;
import gov.nasa.ial.mde.describer.Describer;
import gov.nasa.ial.mde.sound.Sounder;
import gov.nasa.ial.mde.solver.symbolic.AnalyzedData;

public class Tutorial_DataArrayInput {

    public static void main(String[] args) {
        //MDE Init as always:
```

MDE Tutorial

```
MdeSettings currentSettings = new MdeSettings("myAppsMdeProperties");
Solver solver = new Solver();
Describer describer = new Describer(solver, currentSettings.getDescriptionMode());
describer.setOutputFormat(Describer.TEXT_OUTPUT);
Sounder sounder = new Sounder(solver,currentSettings);

// Let's create some data for this demonstration of MDE data array input.
// Make two columns of data with headers:

String timesHeader = "TIME";
String valuesHeader = "VALUE_AT_TIME";

double[] times = new double[10];
double[] values = new double [10];

for (int i=0; i < 10; i++){
    times[i] = i+1;
    values[i] = 10-i;
}

// Let's take our data columns and headers and create an MDE AnalyzedData object:
AnalyzedData myData = new AnalyzedData(timesHeader, valuesHeader, times, values);

//Now give Solver the AnalyzedData object and ask it to solve
solver.add(myData);
solver.solve();

//Now we ask for a description and sonification as before

if (solver.anyDescribable() ) {
    String description = describer.getDescriptions("visual");
    System.out.println("Description of data: " + description);
} else {
    System.out.println("MDE could not generate a description for your data.");
}

// Now let's sonify our data

if (solver.anySonifiable() ) {
    System.out.println("Sonifying your data now: ");
    sounder.sweep(3.0);
    sounder.close();
}

//Clear Solver so next data set will be processed singly
//(we only want one description at a time)
solver.removeAll();
} // end main

} // end class Tutorial_DataArrayInput
```

3.3.2.6 More Example Programs

As you can see, use of MDE is very simple and straightforward. The library demos contain additional MDE programming examples.

3.3.3 More About The MDE Solution Engine (Solver and Solution)

We've seen that Solver *solves* our equations and data for graphing, describing and sonification. We've also seen that Solver *serves as a manager* of multiple input equations or data sets (Descriptor: Text descriptions of graphs and in Listing 3 CartesianGraph Example).

3.3.3.1 Solution Class

Solver manages solutions with the **Solution** class. Solution stores and serves up the current solution for each AnalyzedItem (AnalyzedEquation or AnalyzedData) object being managed by the Solver. It provides public accessors for getting the solution data, checking the state of a solution, and managing state. For example, you can use `getGraphTrails()` to get plotting data, and `getPoints()` to get sonification data. If you build your own components, you'll need to know more about Solver and Solution. Refer to the MDE Developer's Reference/API Javadoc for more information.

3.3.3.2 Using Components Together - Synchronization

Our tutorial examples illustrated how to use MDE Description, Sonification and Graphing components separately. If you want to use the components together in an application, you probably want the Descriptor, CartesianGraph and Sounder or SoundControl synched up. In other words, you want them all to display information about the same solution. (Of course, applications where synchronization is not desired are possible to.)

If you looked at the section Use Solver to Reset Graph (Solution) Bounds, you might have already figured out that Solver can keep things synched up. Solver can serve as the keeper of the latest solution(s). If you want to keep components synched up, use Solver as follows.

The trick is to pass the same instance of Solver to all our components, and make calls or callbacks to Solver when the solution is modified. If our components are notified a solution change has occurred, then things are kept in synch.

Some of these callbacks and notifications are built into the MDE components. For example, let's say you're using CartesianGraph and IncrementXButtons. The user changes the bounds of the solution with IncrementXButtons. The component will make a call back to the Solver object to update the solution (Solution instance). CartesianGraph receives notice that the solution has changed and updates the drawing.

SoundControl, the GUI interface to Sounder, doesn't need to take immediate action when the solution changes, since it doesn't play a sonification until the user requests it. But it does need access to the latest solution at all times. If SoundControl is created with the same Solver object all the other components are using, it will always have access to the latest solution.

MDE Tutorial

As long as all the components are pointing at the same Solver instance, they all have access to the latest solution, and can be put on the notification list and respond appropriately.

MDE Text Description Examples

Appendix A: Current Text Description Examples

MDE generates descriptions via templates. Description templates contain phrases and sentences into which specific mathematical values and descriptive terms (adjectives, adverbs) are inserted depending on the mathematical features present in a solution/graph and their values, and the mode of description desired. Some examples are listed in Table 12.

Table 12 MDE Text Description Examples

Equation Type	Describer mode = "visual"	Describer mode = "math"
NULL SET	Your input equation is $y - 2 = y$. The graph of the equation is a null set. The equation has no solution.	Your input equation is $y - 2 = y$. The graph of the equation is a null set. The equation has no solution.
SINGLE POINT	Your input equation is $x^2 + y^2 = 0.0$. The graph of the equation is a single point. The single point solution is $(0, 0)$.	
ALL POINTS	Your input equation is $x = x$. The solution is the set of all points. The solution will not be graphed.	
VERTICAL LINE	Your input equation is $x = 0.0$. The graph of the equation is a vertical line. The slope is undefined.	Your input equation is $x = 0.0$. The graph of the equation is a vertical line. The slope is undefined. The graph has an inclination of 90 degrees or approximately 1.571 radians. The x-intercept is 0. The equation is a linear equation. The domain of the equation is $\{x \text{ such that } 0 \leq x \leq 0\}$. The range of the equation is $\{y \text{ such that } -\infty < y < \infty\}$.
HORIZONTAL LINE	Your input equation is $y = 0.0$. The graph of the equation is a horizontal line. It is flat with a slope of 0.	Your input equation is $y = 0.0$. The graph of the equation is a horizontal line. It is flat with a slope of 0. The graph has an inclination of 0 degrees or -0 radians. The y-intercept is 0. The equation is a linear equation. The domain of the equation is $\{x \text{ such that } -\infty < x < \infty\}$. The range of the equation is $\{y \text{ such that } 0 \leq y \leq 0\}$.
TWO PARALLEL	Your input equation is $x^2 =$	Your input equation is $(y -$

MDE Text Description Examples

LINES	1.0. The graph of the equation is two parallel lines. The lines are a distance of 2 units apart. They have an inclination of 90 degrees.	$1.0*x^2 = 1.0$. The graph of the equation is two parallel lines. The lines are a distance of approximately 1.414 units apart. They have an inclination of 45 degrees. The x intercepts are -1, 1. The y intercepts are -1, 1. The equation is a degenerate parabola.
TWO INTERSECTING LINES	Your input equation is $x^2 - y^2 = 0$. The graph of the equation is two intersecting lines. The lines cross at the point (0, 0) and have inclinations of -45 degrees and 45 degrees.	Your input equation is $(x - 2.0*y)^2 - y^2 = 0$. The graph of the equation is two intersecting lines. The lines cross at the point (0, 0) and have inclinations of approximately 18.435 degrees and 45 degrees. The x-intercept is 0. The y-intercept is 0.
SLOPING LINE	<p>Your input equation is $y = 3*x$. The graph of the equation is a line. It rises steeply from left to right with a slope of 3.</p> <p>Note that MDE has the ability to change qualitative words like "steeply" depending on the line's characteristics.</p>	Your input equation is $y = 3*x$. The graph of the equation is a line. It rises steeply from left to right with a slope of 3. The graph has an inclination of approximately 71.565 degrees or approximately 1.249 radians. The x-intercept is 0. The y-intercept is 0. The ascending region is $\{x \text{ such that } -\infty < x < \infty\}$. The equation is a linear equation. The domain of the equation is $\{x \text{ such that } -\infty < x < \infty\}$. The range of the equation is $\{y \text{ such that } -\infty < y < \infty\}$.
PARABOLA	Your input equation is $y = 1.0*x^2 + 0.0$. The graph of the equation is a parabola. It opens to the North. Focal length can be a measure of a parabola's width . The focal length of this parabola is 0.25. This is a good 'reference parabola' to compare other parabolas to. What happens to the focal length and parabola width when you change the coefficient of x^2 ? Enter $y=c*x^2$, with $c=1$. Then	Your input equation is $y = 2.0*x^2 + 0.0$. The graph of the equation is a parabola. The vertex is located at the point (0, 0). The curve has an axis of symmetry which is the line given by $1*x = 0$. Its axis of symmetry is oriented at an angle of 90 degrees from the positive x - axis. In other words, the curve opens to the North. The focus is located at the point (0, 0.125). The focal length is 1/8. The directrix is the line given by $8*y+1 = 0$. The angle of inclination of the directrix is 0. The x-intercept is 0. The y-

MDE Text Description Examples

	change c to see what happens to the parabola.	intercept is 0. The ascending region is $\{x \text{ such that } 0 \leq x < \infty\}$. The descending region is $\{x \text{ such that } -\infty < x \leq 0\}$. The equation is a conic section. The domain of the equation is $\{x \text{ such that } -\infty < x < \infty\}$. The range of the equation is $\{y \text{ such that } 0 \leq y < \infty\}$.
HYPERBOLA	Your input equation is $x^2/(1.0^2) - y^2/(1.0^2) = 1$. The graph of the equation is a hyperbola. The graph consists of two separate pieces called branches that approach each other as if they would cross, but then bend back away from each other. The points on each piece where the branches are closest together are called vertices. The vertices are located at the points (1, 0), (-1, 0). The midpoint of the line segment between the vertices is called the center of the hyperbola. The center is at (0, 0). Way out on each branch, a hyperbola is nearly straight and actually approaches a straight line called an asymptote. The equations of the asymptotes are: $1*x - 1*y = 0$ and $1*x + 1*y = 0$.	Your input equation is $x^2/(2.0^2) - y^2/(1.0^2) = 1$. The graph of the equation is a hyperbola. The center is at (0, 0). The vertices are located at the points (2, 0), (-2, 0). The eccentricity is approximately 1.118. The focal length is approximately 2.236. The equation of the transverse axis is $1*y = 0$. The length of the semitransverse axis is 2. The equation of the conjugate axis is $1*x = 0$. The length of the semiconjugate axis is 1. The foci are located at the points (2.236, 0), (-2.236, 0). The equations of the asymptotes are: $1*x - 2*y = 0$ and $1*x + 2*y = 0$. The x intercepts are -2, 2. The equation is a conic section.
ELLIPSE	Your input equation is $x^2/(1.0^2) + y^2/(3.0^2) = 1$. The graph of the equation is an ellipse. Ellipses are oval shaped curves. How 'flat' or how rounded the oval is depends on the length of the major axis compared to the length of the minor axis. The longer the major axis compared to the minor axis,	Your input equation is $x^2/(2.0^2) + y^2/(1.0^2) = 1$. The graph of the equation is an ellipse. The center is at (0, 0). The eccentricity is approximately 0.866. The semimajor axis is half the distance across the ellipse along the longest of its axes. The length of the semimajor axis is 2. The major axis is given by the line $1*y = 0$. The major axis inclination is 0

MDE Text Description Examples

	<p>the 'flatter' the ellipse. Another term for flatness is eccentricity. The major axis of this ellipse with length 6 is approximately 3 times the length of the minor axis with length 2. This ellipse is pretty 'flat'. It's a nice long oval.</p>	<p>degrees. The semiminor axis is half the distance across the ellipse along its shortest principal axis. The length of the semiminor axis is 1. The minor axis is given by the line $1*x = 0$. The minor axis inclination is 90 degrees. The foci are located at the points (1.732, 0), (-1.732, 0). The focal length is approximately 1.732. The x intercepts are -2, 2. The y intercepts are -1, 1. It is a closed curve. The equation is a conic section. The domain of the equation is {x such that $-2 \leq x \leq 2$}. The range of the equation is {y such that $-1 \leq y \leq 1$}.</p>
CIRCLE	<p>Your input equation is $x^2 + y^2 = 9.0$. The graph of the equation is a circle. The center is at (0, 0). The width of the circle is 6.</p>	<p>Your input equation is $x^2 + y^2 = 9.0$. The graph of the equation is a circle. The center is at (0, 0). The radius is 3. The x intercepts are -3, 3. The y intercepts are -3, 3. It is a closed curve. The equation is a conic section. The domain of the equation is {x such that $-3 \leq x \leq 3$}. The range of the equation is {y such that $-3 \leq y \leq 3$}.</p>
POLYNOMIAL	<p>Your input equation is $y = x^3$. This is the graph of a cubic polynomial . The curve rises from the far lower left to an inflection point at the point (0, 0) and rises to the far upper right</p>	
POLAR ROSE	<p>Your input equation is $r = 1.0*\sin(3.0*\theta)$. The graph of the equation is a polar rose. The graph looks like a 3-bladed propeller with its blades symmetric about the origin.</p>	
RATIONAL FUNCTION	<p>Your input equation is $y = x/(1 + x^2)$. This is the graph of a function. The curve is nearly flat from a horizontal asymptote at the line $y = 0$ at</p>	

MDE Text Description Examples

	<p>the far left to a local minimum at the point (-1, -0.5), rises to a local maximum at the point (1, 0.5) and is nearly flat to a horizontal asymptote at the line $y = 0$ at the far right .</p>	
<p>DATA DESCRIPTION EXAMPLE: ALTITUDE (ALT) VS TIME</p>	<p>The ALT(M) curve has the following characteristics. The portion of the graph in the visible window consists of a single continuous graph. The curve rises from a boundary point at the point (0.05, 0) to a local maximum at the point (7.05, 216.615) and falls to a boundary point at the point (14.2, -0.974).</p>	

References

References

1. NASA Learning Technologies, Johnson Space Center, Information Accessibility Lab
<http://prime.jsc.nasa.gov>
2. Math Description Engine (MDE) Compiled Source Code Distribution
<http://prime.jsc.nasa.gov/MDE>
3. Math Description Engine (MDE) Developer's Reference/API Javadoc
4. NASA MathTrax <http://prime.jsc.nasa.gov/MathTrax/>
5. NASA Learning Technologies <http://learn.arc.nasa.gov/>
6. NASA Learning Technologies Projects Description
<http://learn.arc.nasa.gov/app/Learning%20Technologies%20Projects.pdf>
7. NASA Open Source Software <http://www.nas.nasa.gov/Research/Software/Open-Source/index.html>
8. NASA Education Home <http://education.nasa.gov/home/index.html>
9. NASA Education Enterprise Strategy
<http://education.nasa.gov/about/strategy/index.html>
10. Kramer, G., Walker, B., Bonebright, T. Cook, P., Flowers, J., Miner, N., Neuhoff, J., et al. (1999). Sonification report: Status of the field and research agenda. Report prepared for the National Science Foundation by members of the International Community for Auditory Display
<http://www.icad.org/websiteV2.0/References/nsf.html>
11. International Community for Auditory Display (ICAD)
<http://www.icad.org/>
12. ViewPlus Accessible Graphing Calculator
<http://viewplus.com/>
13. The vOICe - Seeing with Sound
<http://www.seeingwithsound.com/>
14. Jaws for Windows, Freedom Scientific
<http://www.freedomscientific.com/>
15. Java Development Kit <http://java.sun.com/j2se/index.jsp>
16. Java Runtime Environment <http://java.sun.com/j2se/index.jsp>
17. Java Access Bridge <http://java.sun.com/products/accessbridge/>
18. Java Tutorial <http://java.sun.com/docs/books/tutorial/index.html>