Math Speak & Write: A Programmer's Guide

Richard Fateman, Cassandra Guy, Steven Stanek
Computer Science Division, University of California at Berkeley
August 5, 2004

Introduction

Math Speak & Write is a program designed to facilitate mathematics input using both voice recognition and handwriting recognition. It is our belief that handwriting and speech can be used to produce a more user-friendly method of math input when compared to systems such as TeX or the *Microsoft Equation Editor* included in *Microsoft Word*.

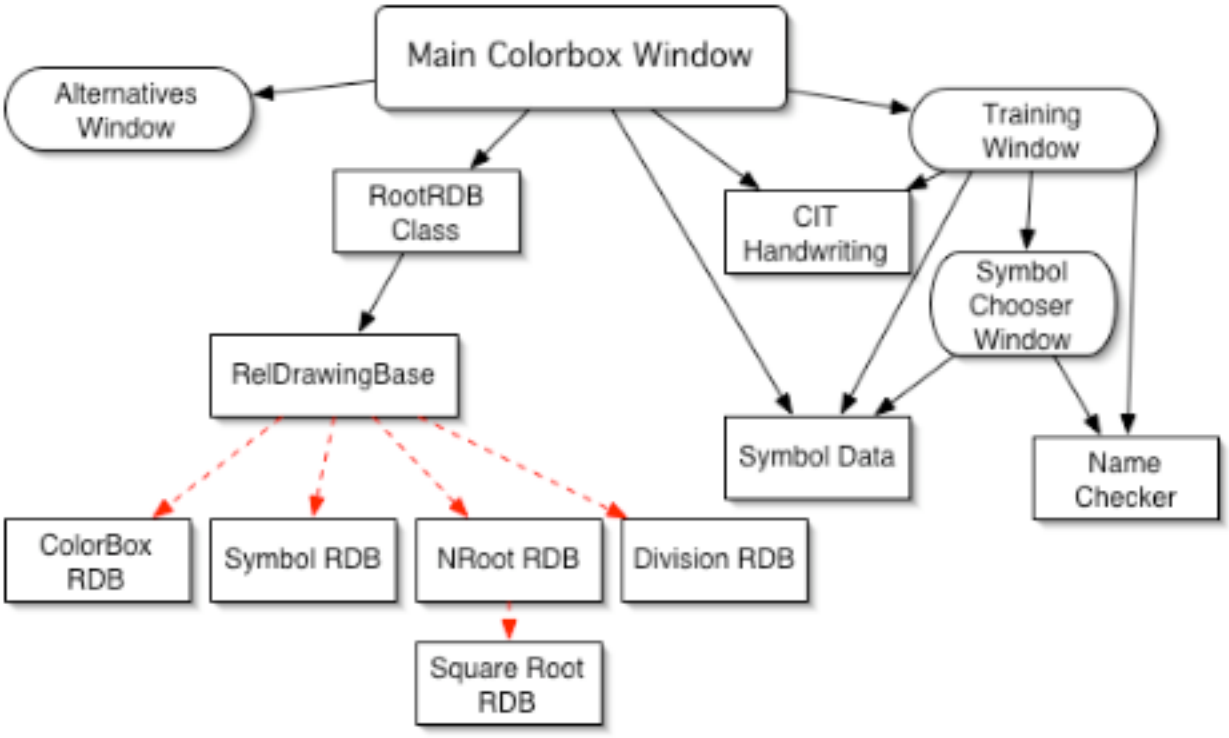
In our system, users supply math content as many one-dimensional expressions using handwriting and speech and occasionally click on "colorboxes" to instruct the program to reposition the cursor. This system allows us to eliminate ambiguity in speech input. For example the statement "sin x to the power k over y" has several different plausible interpretations: $\sin \frac{x^k}{y}$, $\sin x^{\frac{k}{y}}$ and $\frac{\sin x^k}{y}$.
(For more information regarding the use of the program, see our Help Menu. This document is intended for programmers interesting in understanding and modifying the program)

Implementation Foundations

Math Speak & Write was written for *Microsoft Windows* in the C# programming language. For handwriting recognition, it uses a combination of Microsoft's Ink system to collect ink strokes and our dll version of Jim Arvo's CIT Handwriting Recognizer[1] to recognize written characters from the strokes provided by MS Ink[2] . The speech recognition uses Microsoft's Speech SDK 5.1[3] with a custom grammar to optimize it for mathematics input.

We separate data classes from Window controller classes as is shown in the below diagram.

[1] Arvo's original handwriting recognizer is available at: http://www.cs.caltech.edu/~arvo/code/
[2] Available in the Tablet PC SDK at: http://msdn.microsoft.com/mobility/prodtechinfo/platforms/tabletpc/
[3] Available at: http://www.microsoft.com/speech/download/sdk51/



Here, Windows Form handlers are shown as rounded ovals and data management and computation classes are shown as rectangles. Solid lines represent classes which are members of others (or at least have static methods which are called by them) and dotted lines represent inheritance.

A brief description of the classes follows, more details are discussed later in the document.

- Main Colorbox Window (`CBMainWin` in CBMainWin.cs): Contains all code for drawing and managing the principal window of the program.
- Alternatives Window (`AltsWindow` in AltsWindow.cs): Used only during handwriting recognition to display alternates to recognized character and allow for user correction.
- Training Window (`TrainerMainWin` in TrainerMainWin.cs): A window which is launched from the Main Window for training new handwritten symbols or replacing old symbols.
- Symbol Chooser Window (`SymChooser` in SymChooser.cs): Called by the Training Window to prompt the user to choose which symbols to train.
- CIT Handwriting (`CITHandwriting` in Handwriting.cs): A static class which contains methods to interface with the WinCIT.dll, a dll version of Arvo's CIT recognizer.
- Symbol Data (`SymbolData` in SymbolData.cs): A class which is used to read symbol

names and their "codes" from the SymbolData files and stores them in arrays for querying.
- NameChecker (`NameChecker` in NameChecker.cs): A class which looks through the CIT data dir to see what symbols have been trained and stores them in a hash table for queries.
- RootRDB (`RootRDB` in CBMainWin.cs): A small class which contains a single RDB for drawing at a start location. All other RDBs are drawn relative to the one in the Root.
- RelDrawingBase (`RelDrawingBase` in RelDrawingBase.cs): (Henceforth called RDB) RDB is an abstract class which contains all methods and properties needed by individual drawing elements. RDBs are arranged in a tree-like structure which is discussed in more detail later.
- ColorBox RDB (`ColorBoxRBD` in RelDrawingBase.cs): An implementation of RDB which is used to draw user–clickable Colorboxes.
- Symbol RDB (`SymbolRDB` in RelDrawingBase.cs): An implementation of RDB which is used to draw Symbols, that is letters, mathematical symbols and numbers, and provide fields for exponents and subscripts.
- Division RBD (`DivisionRDB` in RelDrawingBase.cs): An implementation of RDB which is used to draw division bars and provide fields for the dividend and divisor.
- NRoot RDB (`NRootRDB` in RelDrawingBase.cs): An implementation of RDB which is used to draw the Nth Root Sign with the radicand and degree fields.
- Square Root RDB (`SqrtRDB` in RelDrawingBase.cs): A subclass of `NRootRDB` which is used to draw the square root sign, without the degree field.

Discussion of the Symbol Data Structure and Rendering System

Overview

A large part of the *Math Speak & Write* application is the symbol data structure and rendering system which is used to display mathematics. The system is built around the Relative Drawing Base abstract class and its subclasses. In addition to drawing, the RDB class provides functionality for parsing character sequences which are used to describe nonstandard symbols (for example: the hexadecimal values for Greek letters in the Symbol Font).

The rendering system is relatively straightforward and although written for Windows, it should be quite easy to port to other platforms since there is very little Windows–specific code.

Every implementation of the RDB contains at least two pointers to other RDBs, the RDB directly to its right (the "next" RDB) and the one to its left (the "prev" RDB). Almost all RDB implementations contain several other pointers. For example, a Symbol contains also contains a exponent RDB "exp" field and a subscript RDB "sub" field. All terminal RDBs (and some non-terminal ones) are `ColorBoxRDB`s. This property means that only Colorboxes are allowed to link their next fields to null.

For rendering, each RDB draws itself and then its non-previous neighbors which in turn draw all

of their non-previous neighbors. Starting at the left most node we can draw the entire mathematical expression recursively in this manner. In addition to drawing, we provide similar methods for recursively determining width and height of an expression, finding mouse click intersections with individual symbols, and converting the RDB representation to MathML or TeX strings.

Discussion of Representation using the RDBs

As mentioned before we have 4 subclasses of the RDB, the Colorbox, Symbol, Division Bar and N-Root. Each overrides many methods found in the RDB class in order to provide the required functionality. The following summarizes all classes. For more detailed information, see the documentation comments in the code.

Description of the RelDrawingBase (RDB) Abstract Class

- *Description:* The abstract class on which all other drawing objects are based.
- *Fields:*
    - RDB prev: The RDB which links to this one using a non-prev field, its "parent".
    - RDB next: The RDB which is directly to the right of this one and is drawn on the same level. Generally, the next RDB in the mathematical expression.
    - int parents: The numbers of parents the RDB has is a crude representation of its size. Parents indicate changes of level, for example the exponent of a symbol has one more parent that the symbol, but that same symbol's next field has the same number of parents. The more parents a symbol has, the smaller it should be.
    - int ID: For debugging purposes, every RDB is assigned a unique ID.
    - float width, height: The width and height fields are used to store the width and height of an RDB, which is defined as everything it encloses. For example, the height of a division bar is the height of highest point on the dividend to the lowest point on the divisor but does not include the height of any members of its next fields. These fields, though world–readable may not be reliable, so use of the `listWidth` and `listHeight` methods for querying width and height is recommend.
    - Various static fields: Contain nextID to be issued, horizontal buffer between symbols, various fonts, brushes and TeX and MathML information.
- *Methods:* (some methods excluded)
    - `addNeighborCBs`: Looks at each of the non-previous neighbor fields for an RDB and determines if they contain colorboxes. If so, it turns the colorboxes on, otherwise it inserts colorboxes in the field, linking the new colorbox to the old contents of that field. This method is commonly used to select an RDB and display its colorboxes.
    - `removeNeighborCBs`: Used to remove "extra" colorboxes, to deselect an RDB. Colorboxes which link to other non-null RDBs are removed, and terminal

Colorboxes, colorboxes which link to null, are turned off. Effectively, this method turns off an RDB's colorboxes to deselect it.

- (static) `addWithStr`: Takes a string of text (using the special escape sequences discussed later) and parses it into a series of RDBs which it then chains to the end of a selected RBD's next field (or an otherwise specified field).
- (static) `convertTextToNumbers`: Takes a string containing a spoken representation of a number and converts it to a numerical representation. (ex: "one hundred twenty five" to 125) This solves a problem that the speech recognizer spells out numbers in some cases.
- `deleteSelf`: Causes an RDB to unlink itself from its neighbors and delete child fields, effectively deleting it from the tree.
- `draw`: Tells an RDB to draw just itself (not its children or next field) at given coords.
- `highestY`: Returns the highest Y coordinate of an RDB or any of it's non-previous (child or next) neighbors.
- `lowestY`: Returns the lowest Y coordinate of an RDB or any of it's non-previous (child or next) neighbors.
- `insertRDBAfter`: Used to insert some RDBs right after this RDB and link the last of the inserted RDBs's next field to the old contents of this RDB's next field.
- `listChangeParents`: Recursively increases the number of parents of this and all non-prev neighbors by a given value.
- `listConvertToMathML`: Recursively produces mathML expressions to represent this and all non-prev neighbors.
- `listConvertToTex`: Recursively produces TeX expressions to represent this and all non-prev neighbors.
- `listDraw`: Recursively draws this RDB and non-prev neighbors, takes the coordinates at which this RDB is to be drawn and coordinates for neighbors are mathematically generated based on those.
- `listGetParentAtLevel`: Follows the prev fields backwards to try to find an RDB which has the requested number of parents or fewer.
- `listGetRoot`: Finds the first element of a tree of RDBs, follows the prev pointers backwards until it hits a null.
- `listHeight`: Finds the height of this and all non-prev neighbors, that is the distance between the lowest and highest Y coords.
- `listHeightAboveDrawY`: Returns the maximum distance of any non-prev RDBs from the drawing Y coordinate to the highest Y coordinate.
- `listHeightBelowDrawY`: Returns the max distance of this and any non-prev RDBs from the drawing Y coordinate to the lowest Y coordinate.
- `listRDBAtCoords`: Given the drawing coordinates for this RDB, this function to finds the next non-previous RBD which was clicked on at the given click

coordinates.

- `readMathMLArrays, readTexArrays`: Used to load data for the MathML and TeX conversions.
- `replaceRefFor`: Takes an old RDB and searches this RDB's non-prev fields for that RDB. If the old RDB is found, it is replaced with the new RDB.
- `replaceSelfWith`: Removes this RDB from a chain (disconnects it from its prev and next fields) and inserts a set of new RDBs in its place. Commonly used for editing.
- (static) `RecAddStr`: A helper function for `addStrWith`
- (static) `tokStrings`: A helper function for `addStrWith`
- `toogleNeighborCBDraw`: A simple function which turns the drawing off for neighboring CBs.
- `wasClickedOn`: Using the height and width fields in the bounding box and the draw coordinates for this RDB, determines if it was clicked on by given click coordinates.

Discussion of ColorBoxRDB

- *Description:* `ColorBoxRDB` represents the Colorbox, a square box which can be drawn to represent a possible choice of location at which a user may input mathematics. A ColorBox is used to depict a user–clickable location where additional mathematics may be added. Colorboxes are the only RDBs which are allowed to terminate the expression by linking to null elements (these are places where additional terms could be added to the expression). This unique property requires that Colorboxes also have the ability to be invisible, that is, Colorboxes may exist without being drawn. Colorbox drawing can be turned on and off locally or indirectly using `ToogleNeighborCBDraw` on a non-Colorbox object. Lastly, in some cases, the colorbox may be drawn as "unfilled", an outline of a square instead of the usual solid. The unfilled square is drawn to demonstrate something vital is missing, for instance the divisor of a division bar.
- *Fields:*
  - bool drawSelf: Used to toggle the drawing of the colorbox on and off
  - bool drawFilled: Used to toggle drawing between filled and solid
- *Methods:* No methods of note, other than those which override those in the RDB

Discussion of the SymbolRDB Class

- *Description:* The `SymbolRDB` class is used to represent "regular" symbols, those which do not have special components. All Symbols have an exponent (exp) and subscript (sub) field, even though these fields may not directly correspond to the symbol (for example the limits of an integral are still called exp and sub). When the Symbol is converted to MathML or TeX these problems are resolved. The symbol stores its contents in the form of a text string, which uses escape sequences (more later) to define nonstandard characters,

symbols and formatting.

- *Note on Symbols:* There is some ambiguity regarding the difference between one symbol and several separate symbols though from the presentation point of view, it makes no difference. For instance, the string of symbols '1', '2', '3' or the symbol '123' should be equivalent and will produce identical TeX and MathML renderings. There, is however a difference in selection of symbols, in the former case, the user can select the '1' individually, in the later he must select the entire '123' string.
- *Fields:*
    - RDB exp: The exponent RDB, rendered to right and mostly above this RDB.
    - RDB sub: The subscript RDB, rendered to the right and mostly below this RDB.
    - string text: A string of text, including specially designed multi-character escape sequences to represent the characters in this symbol. SHOULD NOT BE SET FROM OUTSIDE, USE `SETSTRING` INSTEAD.
    - Font greekF: The font in which to render all nonstandard symbols (greek letters and math symbols). These characters must be provided through escape sequences in the code. For this program, the Greek Font is Windows Symbol Font.
    - Font normalF: The font in which all standard text is rendered. For this program, it is the Times New Roman Font.
- *Methods:*
    - `textToTex`: Used to convert the text for this symbol to TeX. Helper for `listConvertToTex`.
    - `textToMathML`: Used to convert the text for this symbol to MathML. Helper for `listConvertToMathML`.
    - `lookupTexSymbol`/`lookupMathMLSymbol`: Used in support of the text conversion to these formats. Looks up a hex symbol in an array and returns the appropriate MathML or TeX for that symbol.
    - `containsLetters`: Looks to see if the Symbol's text contains any alphabetic letters.
    - `isBold, isItalic, isUnderline, isHex`: Sees whether any part of the string is one {Bold, Italic, Underlined or Hex}.
    - `getHex`: Finds the two digit hex code for hex symbols in the text
    - `getText`: Removes all bold, italic,underline and hex from the string and return it.
    - Various calculate Functions: used to internally calculate coordinates for the sub, exp and next fields
    - `insertRDBExp, insertRBDSub`: Inserts the given RDBs chain in the sub or exp field respectively, placing any existing RDBs in those fields at the end of the chain.
    - `makeBold, makeItalic, makeUnderline`: Used to turn Italic, Bold and Underline on and off for this symbol
    - `recalculateDimensions`: if the text changes, this function can be used to

reset the width and height fields of the object
- `setString`: Used to change the text of the symbol safely
- `setFonts`: Sets up the fonts based on the parents fields
- `strHeight`: Determines the height of string using this Symbol's fonts
- `strWidth`: Determines the width of a string using this Symbol's fonts
- `strWidthRec:` A helper function for `strWidth`
- (static) `charWidth`: Finds the width of an individual character in the given Font
- `drawAString`: Draws a string of specially formatted text characters to the screen at the given location

Discussion of the DivisionRDB

- *Description:* A `divisionRDB` is an RDB which contains a dividend (top) and divisor (bot) fields. The RDB draws a straight line to represent the division bar.
- *Fields:*
  - RDB top: The first element of the top of the RDBs
  - RDB bot: The first element of the bottom RDBs
  - float vbuf: The distance between the division bar and the bottom of top field/top of the bottom field.
- *Methods:*
  - Various Calc Methods: Used to calculate drawing coordinates
  - `setTop, setBot`: Used to replace the existing top and bot fields.

Discussion of the NRootRDB

- *Description:* The `NRootRDB` is used to represent nth root expressions. It draws the root sign using lines (a mathematical formula is used to find the locations of the lines). The NRoot consists of additional fields, the radicand (rad) and the degree (deg).
- *Fields:*
  - RDB deg: The first RDB in the degree of (the N) of the root.
  - RDB rad: The first RDB in the radicand (the thing being rooted).
  - float vbuf: Same as DivisionRDB
- *Methods:*
  - Various calculate methods: for calculating drawing coordinates.
  - `RecalculateDimensions`: If the radicand or degree change, this method fixes reset this object's sizes.

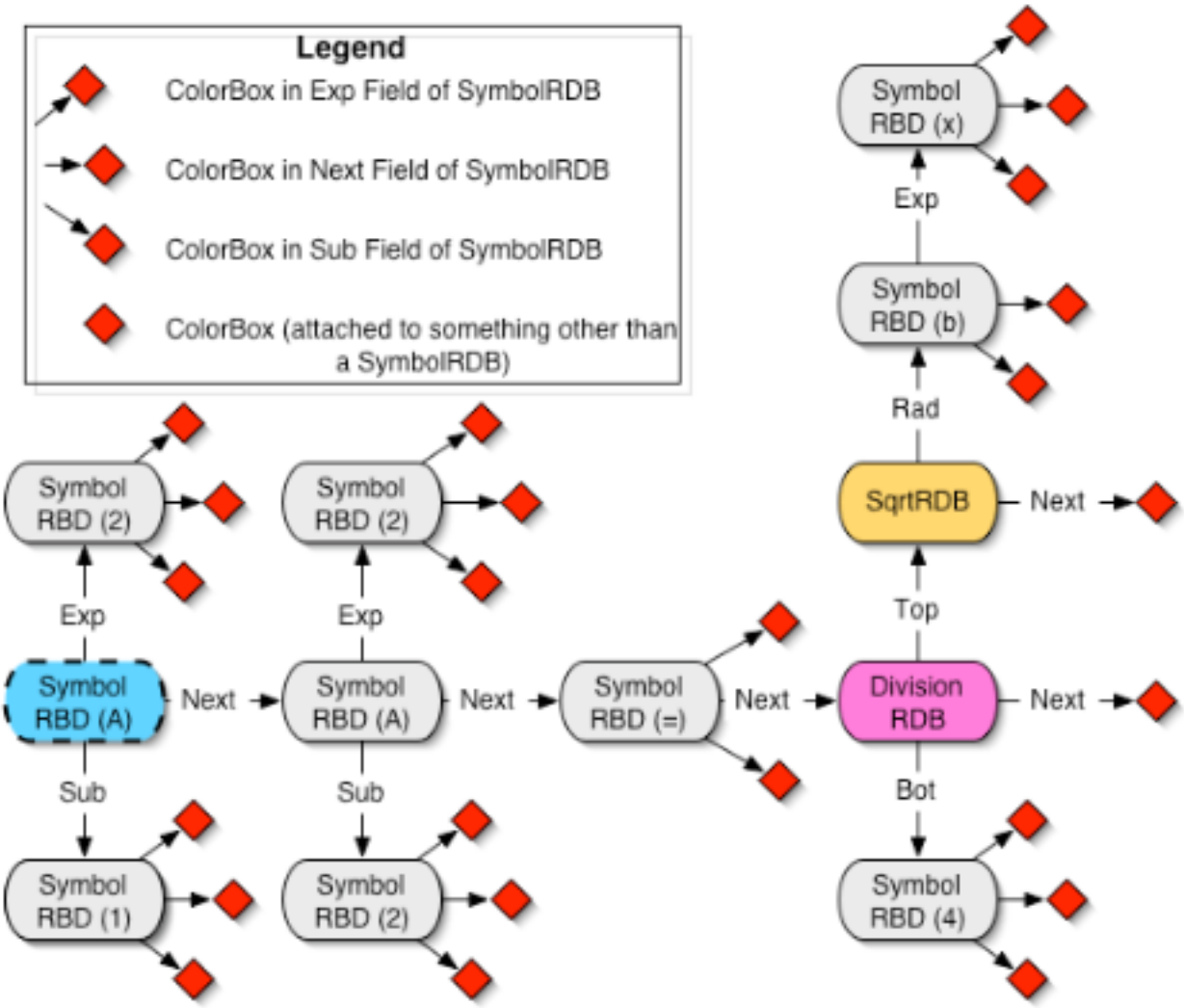
SqrtRDB (subclass of NRootRDB)

- *Description:* Virtually identical to the `NRootRDB` except the deg field is disabled.
- *Fields:* No useful fields
- *Methods:* No useful methods

Example RDB tree for an expression

Example expression: $A_1^2 A_2^2 = \frac{\sqrt{b^x}}{4}$

Representation (previous fields are not shown but can be found by following the incoming arrow to an RDB backward):



The `rootRDB` would point to the shaded `SymbolRDB`. All recursive functions could be called on this right–most symbol RDB to determine characteristics such as width, height, click intersections pertaining to the entire expression. If the programmer instead wishes to examine a sub-expression, she could instead call the recursive functions on the first RDB of that subexpression. For example, to determine the maximum height of expressions to the right of the equals sign, she could call `highestY` on the `DivisionRDB` without worrying about the first half of the expression. Likewise, she could call the `draw` function on `SqrtRDB` to draw $\sqrt{b^x}$

alone.

Other Classes & Files

The RDB data structures for representing mathematical expressions and their associated Colorboxes constitute the majority of the code in this project. The rest is spread across four windows forms for the four windows in the program and two classes[4] , `SymbolData` and `NameChecker` to represent data regarding handwriting symbols. This section touches on these classes, the format of strings for the RDB classes and a discussion of various important files which are used by the program. Many of the following sections are relatively brief as the details of these classes are well–documented in the code.

Format of Descriptive Text Strings

In *Math Speak & Write*, we use three different methods for input: keyboard, handwriting and voice. Our voice and handwriting recognizers are capable of recognizing symbols but can only output ASCII text descriptions of them. So, we have developed a string format used to describe mathematical expressions.

- Almost all regular, ASCII text is acceptable and simply creates `SymbolRDB`s. For instance "sin x = 10" is a perfectly valid string.
- Spaces delimit RDBs.
  For example, "sin x = 10" creates four `SymbolRDB`s,
  "sin" -> "x" ->"=" -> "10".
  Alternatively, "s i n x = 1 0" would create 7 RDBs,
  "s" -> "i" -> "n" -> "x" -> "=" -> "1" -> "0"
- Presently, all regular text characters are rendered using the Times New Roman font.
- Greek letters and mathematical symbols used in `SymbolRDB`s cannot be represented using ASCII. Instead we use escape sequences starting with "@x" and followed by a two digit hexadecimal number which specifies their hex code in the Windows Symbol font. (a table with many of these escape codes is attached as an appendix)
- In addition to the Symbol escape sequences, we define several others. Some of these are filtered before entering the `SymbolRDB` parser in the `textToScreen` function of `CBMainWin` as they are impractical to build directly into the RDB datastructures frame work.
- Some of the special escape sequences are not strictly one–dimensional, but they cover only special cases and are not fully descriptive of two–dimensional mathematical expressions.

[4] There is also a Handwriting Class which is simply a wrapper for the WinCITDll discussed in its own file.

@e	Erases the previous term
@2	Square the previous term
@3	Cube the previous term
@p	Raise the previous term to the next term power (if applicable)
@s	Create a square root sign
@n	Create a Nth Root sign
@d	Create a division bar
@o	"Over"- create division bar and put the previous term over the next term
@l	Make the next term a subscript of the previous (if applicable)
@i,@u,@b	Make the next term italics, underlined or bold respectively
@a#	Select Handwriting Alternative # (between 0 and 9)
@x##	Symbol Font Character using the given two digit hex code

Example Interaction:

1. User speaks "sin x squared equals two over lambda"
2. The Microsoft Speech SDK recognizes the voice and, using our math grammar specification, outputs the sequence "sin x @2 = 2 @o @x6C" [5]
3. The Reco_Event Handler in `CBMainWin` receives the above text as part of a reco event.
4. The `textToScreen` function calls `addStrWith` which parses the string and creates a RDB tree which it adds to the appropriate position within the existing expression.
5. `textToScreen` then re-draws the screen.

CBMainWin in Brief

`CBMainWin` is the main window for the program. It contains menu items for a variety of features such as enabling and disabling handwriting and speech recognition, saving files in various formats and undo and redo and training new handwriting symbols. Various notes about the design follow.

- Undo/Redo: We implement undo by using a .NET ArrayList of serialized RDB trees to represent the state of the program at each stage from the first blank screen to the present state. Clicking "Undo" simply takes the final element off the undo arrayList and places it at the beginning of the "Redo" arrayList and draws it to the screen. Clicking redo does reverse.
- Handwriting Recognition: We support only recognition of single symbols (by symbol we mean a handwriting symbol as defined by the CIT recognizer, not necessarily

[5] Actually, this isn't the exact behavior. The speech grammer would return symbols as italics, as is standard for formatting of mathematic expressions. To avoid confusion, we exclude the italics from this example.

corresponding to a single SymbolRDB). Characters may have an unlimited (?) number of strokes. We do handwriting recognition using a timer function. After a certain amount of time has elapsed since the last pen down event, the handwriting is recognized (time is set in 100ms increments using the MacTOs variable).

- Alts Window: When handwriting recognition is enabled, the Alts Window appears. When handwriting recognition is disabled, the Alts Window disappears. Likewise, the Alts Window is minimized and maximized with the `CBMainWin` (actually it uses the Timer Function for the min/max due to the lack of an appropriate event handler).

Alts Window in Brief

The alts window is initialized with a `SymbolData` array and delegate (similar to a function pointer) to a method to call back to when a selection occurs. The contents of the window are changed with the `changeAlts` function and the window can be cleared with the `clearAlts` function. Alternatives, can be selected either internally or externally using the `DoAltN` function.

Training Window in Brief

The training window consists mainly of button handlers for adding symbols to current training, clearing the screen, saving training, aborting training and closing the window. When the window is opened or after a symbol is trained, a `SymChooser` dialog is created to prompt the user for a new symbol to train. Once training of a symbol is complete, it is saved to the data directory using the CIT Handwriting system's WinCITDll. Upon closing this window, and event handler program runs the *posttrain* program to incorporate new training data into CIT Profile.

SymChooser Window in Brief

The `SymChooser` Window displays a list of various symbols to train which is a combination of either upper–case and lower–case alphabetic and Greek letters and numbers and symbols which are not case sensitive. Upper and lower case letters are explicitly labeled in the `setupLB` function and symbols for which training data already exists are marked with an asterix. Buttons allow users to toggle between upper and lower case letters, delete selected symbols, or close both this and the parent training window. After close, the `SymChooser` maintains the name and code of the chosen symbol so they can be looked up by the Training Window.

SymbolData Class in Brief

Every symbol consists of two parts, a plain text name and an escape sequence (discussed in the text formatting section). This data is stored in data files with the symbol name on left and the escape code on the right. (For example, the escape code for the multiplication sign is @xB4) The `SymbolData` class uses the `ReadData` function to read in this data into a pair of ArrayLists, one for codes and one for names. The elements of these ArrayLists correspond, that is the name for the Nth element of the codes list is the Nth element of the names list. The `SymbolData` objects can be queried for Names and Codes at various indicies using the `NameN` and `CodeN`

functions, but are more commonly asked to search to see if they have codes corresponding to certain names or names corresponding to given codes through the `NameCorrespondingToCode` and `CodeCorrespondingToName` functions respectively.

NameChecker in Brief

The purpose of the `NameChecker` class is to examine the CIT training directory and obtain a list of the files. It then parses the file names to turn them into the escape codes used in the RDB text representation strings. The results of this parse are stored in a hash table which can be queried to see if various symbols have been stored using the containsName method. Likewise, files cans can be deleted if the `NameChecker` is passed the Symbol's Code (not the actual file name) in the `deleteFile` method. Should the contents of the directory change, it can be re-read and entered into the hash table using overloaded reloadTable method.

Required Files

Our implementation uses three SymbolData files: AllLower.txt, AllUpper.txt and AllNumSym.txt. Which contain the lower case alphabetic and greek letters, upper case alphabetic and greek letters and numbers and symbols respectively. In each file, a symbol occupies the left side followed by a tab and then it's code (as defined in the text string section). One can add or remove symbols by simply editting these files.

The Data directory must contain all CIT training files generated by WinCITDll. These files are named based on (though not in equivalent to) the names of their text representation. The conversion between text and file name occurs in NameChecker.

HexSingleTokens.xml is a Microsoft Speech 5.1 command and control grammar, specially designed to allow math input using our text escape sequences for symbols.

The mathMLHexArray.txt contains a list of strings which have corresponding MathML symbols in mathMLSymbolArray.txt. These two files will be loaded into arrays on the start up of the application and are used for translating the math expression into MathML. The same files exist for the TeX translation, texHexArray.txt and texSymbolArray.txt.

Math Speak And Write Help.chm is a Windows compiled help file. This file includes all the user help files and is accessed in *Math Speak & Write* by the contents option in the help menu.

SpeakWriteIcon.ico is the application icon.

The PostTrain application and WinCITDll are needed for the MathSpeakAndWrite application to run.

Microsoft keeps generating Interop.SpeechLib.dll so I'd advise not fooling with it. Ditto for OUT.DAT and DataCIT, which are generated by PostTrain.

Appendix: Common Escape Sequences

Symbol	Case	Code	Symbol	Case	Code
Alpha	Upper	41	Alpha	LOWER	61
Beta	Upper	42	Beta	LOWER	62
Gamma	Upper	47	Gamma	LOWER	67
Delta	Upper	44	Delta	LOWER	64
Epsilon	Upper	45	Epsilon	LOWER	65
Zeta	Upper	5A	Zeta	LOWER	7A
Eta	Upper	48	Eta	LOWER	68
Theta	Upper	51	Theta	LOWER	71
Iota	Upper	49	Iota	LOWER	69
Kappa	Upper	4B	Kappa	LOWER	6B
Lambda	Upper	4C	Lambda	LOWER	6C
Mu	Upper	4D	Mu	LOWER	6D
Nu	Upper	4E	Nu	LOWER	6E
Xi	Upper	58	Xi	LOWER	78
Omicron	Upper	4F	Omicron	LOWER	6F
Pi	Upper	50	Pi	LOWER	70
Rho	Upper	52	Rho	LOWER	72
Sigma	Upper	53	Sigma	LOWER	73
Tau	Upper	54	Tau	LOWER	74
Upsilon	Upper	55	Upsilon	LOWER	75
Phi	Upper	46	Phi	LOWER	66
Chi	Upper	43	Chi	LOWER	63
Psi	Upper	59	Psi	LOWER	79
Omega	Upper	57	Omega	LOWER	77

Ampersand	26	Integral	F2	Product	D5
And	D9	Intersection	C7	Prop to	BB
CoProduct		Less Than	3C	Single Quote	A2
Degrees	B0	Less Than Eq	A3	Sqrt	D6
Divided By	B8	Not	D8	Subset of	C9
Dot	B7	Not Equal To	B9	Summation	E5
Element OF	27	O plus	C5	There Exists	24
For All	22	O slash	C6	Therefore	5C
Gradiant	D1	O times	C4	Times	B4
Greater Than	3E	Or	DA	Union	C8
Greater Than eq	B3	Partial Derv	B6		
Infinity	A5	Perpendicular	5E		