Braille2000, LLC
6801 Southfork Cir, Lincoln, NE 68516
Phone: (402) 423-4782
http://www.braille2000.com

Version 3.34

Math Guide

for

Braille2000 V3

Quick Start: Math Basics

(Math braille is unique and more complex. Braille2000 V3 is the only WYSIWYG ("what you see is what you get") braille tool in the world that prepares math-braille documents with live rendering of the math equations with typesetting abilities coupled to braille synthesis. It is suggested that any transcriber new to V3 spend a few moments to read the next four pages before beginning. The system is generally quite intuitive but some important features may not be immediately obvious.)

You need Braille2000 V3 with or without the Math Tools option. The discussion starts assuming you are doing "UEB Technical" transcriptions and your print source contains math and is either a NIMAS (xml) file or a Word (docx) file. You start by importing the print materials (but some preparatory adjusting of Word materials may be called for—see below).

You can get V3 at www.braille2000.com. You can get this document and other V3 documents there too. You can import and transcribe math materials using V3 without acquiring the "Math Tools" option (for which you pay a fee), but that option can make things easier. (You can try out the option via the Evaluate Only tab when launching V3.) The option is only for Braille2000: The Document Processing Edition–if your license is for a lesser Edition you also need to upgrade your license to that Edition. For qualifying licenses you own, there is a one-time fee of $239. For subscription licenses, the monthly fee is $7 more ($39 rather than $32). To upgrade your license, send an email to sales@braille2000.com giving the license number you want to upgrade, or phone us (phone number in the heading).

Quick Tips (for doing UEB-technical math)

1. Set the braille Code to ICEB English (unified)

2a. For NIMAS, use Open / Source File to load the text, or

2b. For Word, open the document and enter ctrl+a ctrl+c (select-all and copy-to-clipboard), then go to Braille2000 and use ctrl+v (paste). (If you have Math Tools, you could also use Open / Source File to load the document via its .docx file, but be sure Word does not also have the file open, you need to close it in Word.)

3. When working on the document in Braille2000, switch as desired between the [B] Braille view and the [S] Source view. You may activate the Interpreter (green line) for either view.

4. If you have a choice of math code to use, note that Nemeth is parsimonious and compact and easily intermixes 0-9 (digits) with letters a-j, sometimes found together in polynomial expressions and elsewhere, but that a Nemeth2022 transcription is often very complex with myriad switch indicators and semantic sensitivities: for much math, UEB-technical is easier to transcribe and is nearly as compact. V3 handles both math codes (and others), but can you?
5. If you are math-certified, the Math Tools option is less important. If you are importing print files containing math, the Math Tools option is less important. If your math-braille knowledge is limited or if your source documents do not contain accessible math (such that you have to input math expressions yourself), the Math Tools option is more important. The V3 "green line" for proofreading, and the [S] Source View are the same with or without Math Tools. With Math Tools, you have an on-board equation typesetter (similar to an external math typesetter in Word or MathType, such tools and others available from other software vendors for a fee), and the ability to use Open / Source File for docx files (as well as rtf files and xml (NIMAS) files).

Entering Math You Compose

If you do not have a print document to import, or if the document you have does not contain math data (could be just pictures of equations), you will have to input the math yourself. There are several ways to do that in V3.

1. Use the [B] Braille view. Put the cursor where the math goes and enter valid braille codes via "six-key" keyboard input. You can show the green (interpreter) line to validate your inputs. If you are doing Nemeth2022, be sure to braille valid begin Nemeth and end Nemeth indicators.
2. If you have the Math Tools option (i.e., there is a "Math" button at the top of the Braille2000 Control Panel), use the [S] Source view. Click the Math button in the Control Panel to display the Math panel. Put the cursor where the math goes and type what you can and select by menu (via the Math panel) what you can't type. Build equations from the outside in. If you need "the square root of 'x over 2' ", generate square root first. It will appear in print with a purple-square placeholder for the radicand. Purple squares are input-stops. The cursor jumps among them when you press Tab. Press the Tab key. The purple square turns blue (not always blue, it is your system highlight color, the radicand is now highlighted). Use the Math panel to select a fraction form. The highlighted spot becomes a fraction with purple squares for numerator and denominator. Press Tab. The numerator gets highlighted Type x. X is then the numerator. Press Tab. The denominator gets highlighted. Type 2. You are done. (There is a detailed discussion of the built-in typesetting tool later in this document.)
3. If you have Word, you can use its equation tool to form the desired math expression. Then highlight the entire expression and use ctrl+c to copy it to the Clipboard. In Braille2000 (in either kind of view), put the cursor where you want the equation and use ctrl+v to paste it in.
4. If you have a math typesetting tool that generates MathML, or if you know MathML and write your own MathML markup, open Notepad (or any simple text editor), enter one or more MathML expressions (each bounded by $…$ elements), use ctrl+c to copy it to the clipboard, and then use ctrl+v in Braille2000 to paste it in. You can also write MathML in a text document using Word or Wordpad (or other text editor that generates Rich Text): enter the MathML statement, as one paragraph, starting with $and ending with$, and put the directive %math in front of <math> (set off by one space). You can code multiple

MathML statements throughout the document this way. (Import the document into Braille2000 via the clipboard or via an .rtf file.)

5. If your document does not contain <math> as **other** than MathML equations, i.e., all such utterances are math equations, you may place the directive %math=on at the top of the file. With that directive, all instances of $…$ will be read as MathML markup (i.e., you don't need %math for each). (Note that MathML expressions are not normal to RTF documents… this is an extension of the "%-directives" import mechanism of Braille2000.)
6. If you have MathType, launch the tool (not Word) and build your equation. Via "Preferences", set the clipboard type to "MathML or TeX" and select "MathML 2.0 (no namespace)". Use ctrl+a ctrl+c to copy your equation to the clipboard and then use ctrl+v in Braille2000 to paste it in.

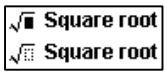
Editing Math Expressions in V3

After you have imported the print document (or generated a document yourself), there will likely be things you want to change. Editing of math materials is more complex than editing prose (unless you edit via the Braille view only) and there are some behaviors to consider.

1. The braille translation by Braille2000 via import may be incomplete or faulty, especially in early releases of V3. You can correct things by working in the Braille view and avoiding edits for that same paragraph in the Source view. (In what follows, the term "print view" will denote either Source view or Print view, but you should normally use Source view to view print.) In the Braille view, you can manipulate the cell sequences any way you wish, creating good math or gibberish. You can view the print equivalent text via the Interpreter (the green line). Any edit in print view sends that paragraph through the translator, and if that process is faulty, you will have to fix-up the braille yet again (after adjusting the braille, be wary of any subsequent changes in print view).
2. You can edit math in print view (assuming, of course that Braille2000 is generating good braille!). You should use the [S] Source view. With equations with text at different elevations, positioning the cursor with the mouse may be tricky. Use left/right arrow keys to move through the various parts of the equation. Note that math structures (e.g., superscript, fractions, radicals) have a thin space between the elements of the structure and you will notice that as you arrow across the expression. For example, if you arrow across e^x, you will find a cursor position after the e (on the baseline) and another position to the right of that before the x (elevated with x). Or said another way, there is a thin space between the parts. Put the cursor after the e to insert a character on the baseline. Put the cursor before the x to insert a character into the exponent. In similar fashion, there is a thin space after the x and before the next-following character (a space in this example) back on the base line. Fractions and radicals behave in an analogous way.
3. You can manipulate the characters in an equation using the keyboard as you do when editing normal prose, but such edits are constrained to the plain sub-elements of the equation: editing a superscript, or editing a numerator, or editing a radicand. You cannot edit across one of those thin spaces separating elements and you cannot insert or delete a thin space or element, i.e., working from the keyboard, the structure of the equation is fixed in place.

4. To change equation structure (e.g., change square root to cube root, or remove a fraction), you must insert new structure using the Math Toolbar (if you have the Math Tools option), or you must switch to Braille view and adjust the cells there. You can remove an entire math structure working in print view (e.g., remove a radical in the middle of an equation) if you carefully highlight the entire structure (e.g., an entire fraction) and then press Delete or Backspace. Or said differently, print view prevents you from creating structure gibberish (incomplete pieces of math structure) whereas Braille view lets you do anything, no matter how invalid from a math perspective. When highlighting a math structure, it is sometimes easier to use arrow keys while holding down Shift.

5. While editing math, it may be helpful to use "dual view", particularly when wanting to edit in print view. Click the [S] button to show print material. Then hold down ctrl and click the [B] button. The lower half of the screen will show braille. Each presentation shows the cursor, the advantage being that as you arrow through the equation in print, you can see the corresponding location in the braille… helpful for highlighting (in print) the totality of a math structure (so you can delete it or replace it).

6. If you have the Math Tools option, there is an additional compositional editing technique you might find useful. Suppose you have part of an equation that says x+1 but it is supposed to be the square root of 'x+1'. You can edit the print, removing x+1 and then using the Math Toolbar to create a radical and then press Tab and re-type x+1. That will work. But a faster way is to highlight x+1 and go to the Math Toolbar and select the radical 

with the open square. (Many forms exist with solid and open "fill-ins". The open fill-in "absorbs" any highlighted text.) The highlighted x+1 will become $\sqrt{x+1}$.

Doing Nemeth

(Note: "Nemeth" (math code braille) is no longer a single thing. It usually means Nemeth2022 (and subsequent revisions) by which Nemeth utterances are interspersed with UEB, with Nemeth utterances (even just a single number or symbol) surrounded by "begin Nemeth"/"end Nemeth" indicators (3 cells each). In V3, this is designated UEB-with-Nemeth. There are also transcriptions that are done the same way but without any indicators. In V3, this is designated UEB-and-Nemeth. And then there are some transcriptions done in pre-2012 Nemeth (a mix of 1994 Nemeth with English Braille American Edition). In V3, this is designated Nemeth (in the BANA code group).)

The preceding discussion pertains to "UEB Technical", a braille code that avoids capturing the semantics of the print. In UEB, the letter "g" standing for "grams" is transcribed exactly the same as the letter "g" standing for the variable denoting the acceleration due to gravity. Not so for transcribing in Nemeth (specifically the "Nemeth2022" code and updates). So how does Braille2000 know how to interpret "g" or "sin" or "tan" or many other utterances. There are clues in print, but they are not foolproof. Worse still, is that the Nemeth2022 code calls for ***all*** math forms to be escaped from UEB to Nemeth, including all uses of the words "sin" and "tan" in prose when they stand for a math function. Although unlikely, a print manuscript might contain such utterances used both as math and not as math. It would seemingly be impossible to manage. And it is… without added trappings.

When the imported manuscript contains utterances not necessarily "math" in nature, Braille2000 looks to the name of the font in which the utterances are expressed. Items set in the "math" font are presumed, on import, to be math items to be escaped to Nemeth code. A "math" font (a contrivance to elucidate the semantic role of utterances) is any font whose face name (fontname) contains the letters "math", such as "Cambria Math" (a font found on Windows 10/11) or "Math

from BRL2000" (a font defined by the installation of V3, just to insure that the system has at least one "math" font). Note that, except for fontname, there is nothing special about the math font.

There is another (rare) semantic situation in Nemeth transcribing. Whether < and > denote comparison operations (the norm) or whether they denote grouping symbols, as in <1,2> or in xml notation such as <math>. The same symbols are used but Nemeth2022 demands they be transcribed differently. The work-around used is to presume < and > are comparison symbols and that any look-alike grouping symbols have been a priori replaced (in print, by you, before import) with the Unicode angle-bracket symbols (something you could do, if ever needed, using Find/Replace in Word before importation into Braille2000, or via a Substitution setting in Braille2000). This is discussed further down in this paper. (Yes, in the Unicode world, there are distinct symbols for "less than" and "left angle bracket", etc., and V3 uses them.)

It would be obvious that any mixed-code transcription, to be further manipulated and edited, must not confuse which parts are in which code. When doing "UEB-with-Nemeth" (the Nemeth2022 variant of Nemeth), formal code-switch indicators set off the Nemeth portions from the UEB portions. When editing a document in Braille view, you can indeed enter the "begin Nemeth" indicator followed by Nemeth braille followed by the "end Nemeth" indicator to input Nemeth material that will be recognized as such (the Nemeth portion must be uncontracted). Note that un-paired code-switch indicators are ignored: when entering a Nemeth passage as braille, the interpretation will switch to Nemeth only when the "end Nemeth" indicator is in place (forming a pair of begin and end indicators).

Nemeth2022 (UEB switching to Nemeth for numerous short utterances) is darned confusing. You will be manipulating a document that is a bit of UEB and a bit of Nemeth, interspersed all over the place. You will probably enjoy color coding of the Nemeth parts. This is done via View / Selections (for the Selections Overlay dialog). Click the down-arrow for Name and choose the "Nem,Unc" item. Then put a checkmark in "Show with color" and choose a color (I like orange). The Nemeth portions will then have an orange background.

Nemeth2022 involved formal code switching, all over the place, throughout the file. Code choice is notated via the "Lang" category of annotations. Their locations can show as little yellow triangles, but only when "Show all annotations" is active (as set via Adjust / Display). There may be a lot of them and they could be distracting, so the normal setting is to show layout annotations (those affecting page layout) only. Best never to manipulate "Lang" annotations directly.

Said another way, the "Lang" annotations mark the boundaries in the document between passages of UEB and those of Nemeth. Seeing the yellow triangle may be helpful when you must insert new symbols right at the boundary… will a new symbol go in on the UEB side? Or on the Nemeth side? Remember that the cursor has TWO positions at a yellow triangle, one before it, and one after it (there is a subtle position difference of the cursor on the screen). When the cursor moves to the position of a yellow triangle, it lands left-of or right-of it based on its "momentum" — it always lands "short". To insert before the triangle, put the cursor a symbol or two before the triangle and use right-arrow to approach it from the left: a new symbol will go in on the left (in the Code for prose before the transition). To insert after the triangle, put the cursor a symbol or two

after the triangle and use left-arrow to approach it from the right: a new symbol will go in on the right (in the Code for prose after the transition).

But not all Nemeth is handled per Nemeth2022.

If you are transcribing math as embedded uncontracted Nemeth *without* code-switch indicators, you should use the Braille2000 code “UEB and Nemeth”. Minus the formal code switching, it is up to you to make things work right. With or without code switching, the embedded Nemeth passages are handled in the “embedded Nemeth” code, rather than UEB. You use the Code button in Braille2000 to assert the desired braille code for highlighted text. So, if you are doing UEB-and-Nemeth, you will need to highlight each Nemeth passage and set its Code properly, if you want the interpreter (green line) to show it properly, or if you plan to do any editing in Source view. (You can make a Speedbraille™ key to do it.) The Code annotations and any optional custom color settings work the same as above, but there are no begin/end indicators, merely “Lang” annotations to transition back and forth between UEB and Nemeth.

As just explained, Code declarations in Braille2000 are via *annotations* that assert braille code. Those annotations travel with the braille in .B2K or .BML files, and as long as the transcription is stored in such files, Nemeth passages, whether with or without braille indicators will stay intact. This means that importing print to yield a “UEB and Nemeth” transcription works the same as for “UEB-with-Nemeth”. But if you save the document as an annotation-free .BRF type of file, Nemeth passages become plain prose unless the transcription is UEB-with-Nemeth with explicit Nemeth indicators (and they are all properly paired).

Pre-2012 Nemeth transcriptions are handled like “UEB-and-Nemeth” except that the non-Nemeth portion is EBAE. As with “UEB-and-Nemeth”, math passages remain defined so long as the document remains automated (.B2K or .BML forms). The corollary of this is that existing Nemeth transcriptions (not V3-automated documents) do not have the Code annotations differentiating the math portions from the prose portions, and so such documents cannot be properly interpreted (the green line will be wacky). Such kinds of documents can be edited in Braille view (ignoring the green line). If more comprehensive work is to be done, it is up to the transcriber to read the document (in braille or in (flawed) print), identify (by highlighting) each math passage, and tell Braille2000, via the Code button, to assign the Nemeth code to the math phrase. For the pre-2012 Nemeth code setting, there is a “Rebias” menu selection in V3 that toggles the code of a highlighted passage between “EBAE” and “Nemeth” (you can define a Speedbraille™ key to do it). In this way, the missing automation can be (laboriously) recreated and then more comprehensive editing can take place, including automated conversion to UEB-Technical or Nemeth2022.

The Semantic Challenge of Nemeth

(Note: it is the view of Braille2000 that braille notation should convey print and not the underlying semantics of the expression. The UEB-technical code is superior in this regard. It is the view of Braille2000 that Nemeth is needlessly complex in such ways.)

Of fundamental importance is the issue of semantic distinction, a facet of some braille codes. UEB is free of such distinctions: each print item is converted into braille independent of use, you do not need to know whether "g" is the abbreviation for gram or is a math variable such as the acceleration due to gravity. Similarly, you do not care whether "tan" is what you get on the beach or is short for tangent. But other codes, e.g., Nemeth and UEB-with-Nemeth, are semantically-focused,

demanding different braille for the same symbols when they have different semantic roles. The issue is that any print-to-braille translator has to know the semantics, or, lacking that, have presentation cues that infer the semantics. An example is "how is Braille2000 to know when "tan" is just a word and when it is "tangent"... they are transcribed differently, especially in UEB-with-Nemeth where the latter has to be escaped into embedded Nemeth. Braille2000, lacking AI†, needs syntactic differentiators to determine the semantic role of incoming print tokens. To handle the semantic sensitivity, an entire paradigm of representation had to be invented. (It is presented below.)

†AI (artificial intelligence) uses non-deterministic weight-based search algorithms to guess at the interpretation of data: the outcome varies over time. When using AI, the user is never in total control… it's not clear braille transcribing is ready for that just yet. There is great advantage to a semantics-free braille code, such as UEB.

In some manifestations, there are already very good clues about "what is math vs. what is prose". In a NIMAS file (if it contains math), the math portions are given within the XML element <math>, and thus it is usually safe to assume that "tan" inside that element is "tangent". Similarly, a Word document that uses the built-in equation editor and has an equation involving "tan", it is likely that it refers to tangent. But in both sources, a sentence such as "Remember that tan is defined as y over x" has no clues. (It would be nice if words were just words, transcribed as prose, but the Nemeth2022 code says that "tan" and "x" and "y" must each be escaped into embedded Nemeth when they are semantically "math".) Ouch!

In some manuscripts (math materials), "tan" would never be skin-tone, but there could be some manuscript in which "tan" is used both ways (prose vs math), and who is to say which is which? Well, it is the transcriber, who, with a semantically-based code, must understand the utterances being transcribed and make that determination. That is very difficult to automate. Braille2000 manages math semantics four ways:

1. it presumes math structures (things stemming from explicit math markup) are math
2. it presumes math-like utterances next to math structures are math
3. it applies "embedding selection" rules to incoming print to denote math
4. it presumes incoming print in a "math" font is math

This approach has the following effects. (1) Math declared as such in Daisy/NIMAS and Word files are handled as math. (2) Typical math notations adjacent to math structures (e.g., an equation written informally (not all in the math formal notation, such as $1+\sqrt{x}$, where only the radical is a math structure item and the '1+' is plain text) are handled as math. (3) Via the "Properties" button in the Code dialog box, you can tune the embedding behavior of UEB-with-Nemeth so that isolated utterances you list will be handled as math (see below). (4) Print passages in the math font are handled as math.

So, if you are transcribing trigonometry and you want sin and tan to be always-embedded in Nemeth, you could get that result in a number of ways:

a. you could prep the incoming print by using Find/Replace (in Word) to set the font for all sin and tan (etc.) words to a math font, say Cambria Math
b. you could use the Properties button (in the Code dialog, for UEB-with-Nemeth) to declare that sin and tan (and a few other terms) are always math
c. you could use the Properties button to declare that all trig functions are to be handled as math

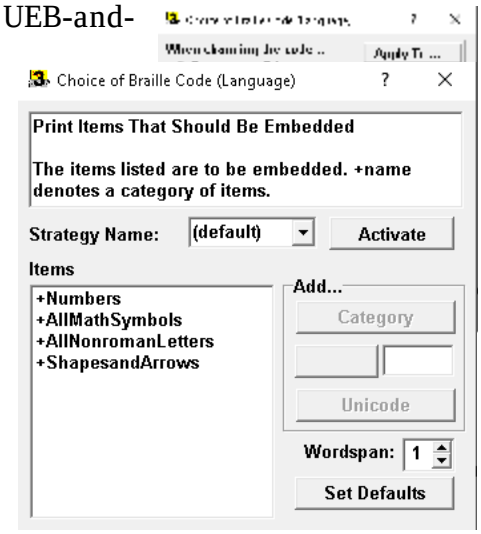
Note that (a) lets you handle each word in the print document distinctly (some “tan” could be math and some not), whereas (b) sets up a math-identifying scheme for the entire document (that is applied as you import the print and when you edit print in the Source view, whereas (c) is like (b) for the built-in list of all trig functions (those enumerated in the Nemeth2020 code, all lower case). The embedding filter (via the Properties button) performs a case-sensitive match automatically during import. The Nemeth2022 code says that “tan” is a math term when in lower case. If you want “Tan” to be embedded, you should use (a) or (b) to designate it explicitly. (Remember, this treatment is needed only when doing Nemeth2022 and only for terms and symbols occurring in prose outside formal math structures.

Via the Properties button (mentioned above), you can also control the embedding of special symbols. By default, V3 identifies all special symbols and Greek letters as math. Via Properties, you can add or subtract from what it identifies, for example, causing all single-letter words in prose (e.g., ‘x’ and ‘y’) to be embedded. (Of course, ‘a’ and ‘i’ are not included in this automated approach.) By default, V3 does not embed single-letter words unless adjacent to math material.

As to < and >, you will have to determine they are grouping symbols versus comparison symbols, so that each can be transcribed accordingly. < and > are “less than” and “greater than” and V3 handles them as math comparison symbols. When a Nemeth transcription needs angle bracket grouping symbols, < and > need to be replaced by ⟨ and ⟩, the “Left pointing angle bracket” and “Right pointing angle bracket” in Unicode, characters 0x2329 and 0x232A, respectively. Those are the distinct grouping symbols that look like “less than” and “greater than”. Unfortunately, they do not exist in common fonts (Times New Roman, Courier New, Microsoft Sans Serif) but they do exist in Unicode from BRL2000 and in Math from BRL2000. This means that if the manuscript contains, for example, XML expressions in print, such as This is a paragraph and if it is to be translated to Nemeth, the transcriber will need to use find/replace to change all < and > to ⟨ and ⟩, or else they will be treated as signs of comparison (i.e., surrounded by blanks and in a math context). Note that translating to UEB (but not UEB-with-Nemeth) has no such issue.

Managing UEB-with-Nemeth

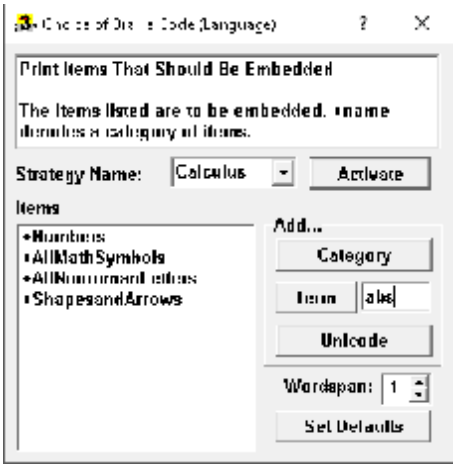
When importing print materials (from NIMAS/RTF/DOCX/Clipboard sources) and going for embedded Nemeth (i.e., using UEB-with-Nemeth or UEB-and-Nemeth), the importation mechanism may need guidance about what print utterances to embed. Braille2000 has a tunable embedding mechanism that inspects incoming print and tags certain utterances as “math” (to be embedded). In the Code dialog (right), when Braille Code selects a code that uses embedding, there is a Properties button (adjacent to Options). Using this button, you can customize the behavior of the translation as to what utterances are considered “math”. You can define and save multiple embedding strategies and activate one of them for use during import. By default, embedding includes: math structures (superscripts, subscripts, fractions, radicals), numbers,



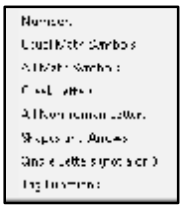
math symbols (+ - = etc. and adjacent text), non-Roman letters, shapes, arrows, plus adjacent punctuation and terms. Note that, unless found adjacent to math elements, isolated words, including “sin” and “cos” and “tan” are *not* considered math and single letters standing alone (e.g., x, y, z) are *not* considered math. These behaviors can be customized, as described below, and for RTF/DOCX/Clipboard importation, you can tag math utterances individually by assigning them a “math” font.

When you click the Properties button, the dialog shows embedding options (each set of options is a “strategy”). The default strategy is to identify incoming numbers, math symbols, non-Roman letters, shapes, and arrows (as well as formal math structures) as “math” (that gets embedded as Nemeth).

Shown below is this same dialog after typing “Calculus” into the Name field (my chosen name) and then clicking Set Defaults to load Items with the default things. The term “abs” (a non-trig function name) was then typed in the Term box. By clicking the Term button, “abs” will then appear as an Item. By clicking the Category button, additional categorical Items can be added (see list below). By clicking the Unicode button, special symbols can be added to Items. When Items lists the categories/terms/symbols you want to be universally embedded, click Activate. That will define a custom Code (a variant of UEB-with-Nemeth) that will apply the embedding to incoming print and print edits done in Source view. (Custom code definitions are persistent and can be used repeatedly.)



The categories of embeddable utterances are these: (a pop-up menu of choices):



When you click the Unicode button, the Unicode selection tool appears, letting you explore the thousands of Unicode characters to select the special symbol you want to be embedded.

A “term” is a word composed of Roman letters, such as “sin” or “ln”. Terms in prose are matched in a case-sensitive manner, i.e., “sin” will not cause “Sin” to be embedded, unless you put “Sin” into the list of items. The “Trig Functions” category includes only lower-case function abbreviations, as listed in the Nemeth2022 code definition.

Background

In today’s notions on braille literature, nearly all literary (i.e., non-math) transcription is done in the UEB (Unified English Braille) code. In all English-speaking countries except the United States, math is transcribed in UEB (namely in UEB-technical, a subset of UEB). In the United States, math is also transcribed in Nemeth (*The Nemeth Braille Code for Mathematics and Science Notation*), or in the Nemeth code embedded in UEB (“UEB-with-Nemeth”, also known as

"Nemeth2022"), or in the Nemeth code embedded in UEB without code-switch indicators ("UEB-and-Nemeth"), and in NUBS ("Nemeth Uniform Braille System", largely unknown). Braille2000 V3 supports all of these braille codes, however not every element of these codes has complete automation at this time.

Because Braille2000 is a comprehensive WYSIWYG ("what you see is what you get") tool whose document format is a sequence of braille pages, document content is maintained in both braille and print notations simultaneously. If you wish, you can view the document both ways at the same time. And even when viewing print-equivalent notations, the page geometry is that of resulting braille pages (Braille2000 is the only tool in the world that presents braille-format pages in print notation). This duality occurs also for math—Braille2000 includes a powerful math typesetting tool that can show print math structures—superscripts, subscripts, fractions, radicals, etc., nested to arbitrary complexity. But therein lies a problem, a sequence of cells in a single line of braille may represent a math structure that is tall (perhaps very tall) such as a vertical fraction. The "Print View" of a braille is designed to show print notation in the same layout as the braille, on the entire braille page… obviously tall math violates that scheme (although simple subscripts and superscripts can still be shown). And for that reason, the Print View shows complex (tall) math via the braille indicators of the structure(s) involved, such as "begin fraction" … "end fraction". Anyone fluent in math-braille can read it. To achieve full-blown math-print, the "Source View" exists, a view that is not obligated to preserve the vertical layout of the underlying braille page. In the Source View, math forms that are tall are presented in normal math typography and where necessary, fewer lines show than on the underlying braille page. Thus, when doing math, the transcriber is likely to use the Braille and Source views, rather than the Braille and Print views.

In Braille2000, both braille and print are malleable—editing one changes the other dynamically. Page layout also adjusts dynamically. When editing in print mode, the keyboard has no ability to input special symbols or math forms (there is no "subscript" on the keyboard). Such things can be entered as braille indicator cells, assuming the user knows them. With the Math Tools option, menu selections provide for print-mode input of special symbols and math forms. This is what a WYSIWYG tool is supposed to do. With the Math Tools option, you can compose math in Braille2000 in Source View with the power of an equation typesetter.

It is obvious that handling math is not like handling prose—there are all kinds of symbols and forms that can't be typed on the keyboard. Some kind of typesetting aid is required. Note that braille itself is very expressive: via six-key input, you can compose all kinds of math expressions—if you know a braille math code, you don't really need a typesetter aid, but it could still be handy.

The math strategy for Braille2000 has math originating from seven sources:

1. Six-key braille entry (direct entry to Braille2000)
2. Importation from Microsoft Word (typesetting done in Word, as in .docx files)
3. Importation from Daisy/NIMAS (xml notation) publisher files
4. Importation from RTF files created by Word
5. Importation from RTF files where MathML expressions have been added
6. Importation from the clipboard (RTF data as per Copy from (2), (4), or (5))
7. Print editing via Braille2000's own built-in typesetter (part of Math Tools)

The Built-in Math Typesetter

The left side of the Braille2000 screen is the home of the "Control Panel" where many handy processing functions are available as clickable buttons. The Control Panel shows one of three sub-panels: Tools, Math, and dtbook. ("dtbook" is the XML doctype for Daisy/NIMAS documents and its sub-panel presents the NIMAS markup tags depicting document structure.) The normally-used sub-panel is the Tools sub-panel.

All panel configurations have top buttons for Menu and Help, and Tools (on the second button row). When a NIMAS document is showing, there is a "dtbook" button next to Tools. All button functions in the Tools sub-panel are available via the menu by pressing Alt+p (keyboard users can navigate the menu starting by pressing the Alt key). In V3, the "Math" button shows only for licenses that include the "Math Tools" option.

The upper part of the Math panel comprises a number of expression-category icon buttons—categories of special symbols and forms. When you click a button, the lower part lists the available symbols or forms in that category. The snapshot shows the choices for the fraction and radical category. Forms are a powerful compositional tool and to that end, several variations are offered, differing in highlight-update behavior.

In the listed math-form graphical depictions, solid and hollow rectangles denote "fill-in" locations. If the math panel is used when nothing is highlighted, both solid and hollow slots do the same thing: they show a colored input location within the new form that you access via the TAB key. For example, if you click either square root form (shown above), you will see, inserted at the cursor, a square root expression like the first one below.

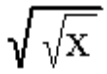
The maroon space is the input placeholder for the content of the square root. To fill in the square root, press TAB. The input placeholder will become highlighted (it changes color to blue, assuming that is your system's highlight color) and it is ready for input (the second image above). If you enter "x", the highlighted space becomes "x" and you have the square root of x (the third image above). If you wanted instead the square root of pi, you would use the math panel to select the category of lower-case Greek letters and would click on π and it would go in rather than x.

If you wanted the square root of a fraction, you would click the fraction form and it would appear inside the radical with two maroon input placeholders (one for the numerator and one for the denominator) to fill in. You would use the TAB key twice, to complete that expression, each one handled as just explained. In this way you can build expressions of arbitrary complexity.

The difference between solid and hollow form elements comes into play when you first highlight part (or all) of the existing math expression. Remember that if you highlight cells or characters of prose and then enter text, it replaces what was highlighted. The same thing happens in math, but with some additional variation. If, in the square root expression above, you highlight the x and then use the math toolbar to select π, the highlighted x is replaced by π. What if you want to replace

x with a square root or other form? The toolbar graphical forms with solid rectangles do just that—if you highlight the x and select the solid square root, you get this result:

 The x is literally replaced by the new form (you then press TAB and fill in the maroon box). But it is also handy to compose expressions by adding a form "around" preexisting math, and for that you would select the hollow square root form. It would give you:

 where the highlighted material fills in the hollow element, in this case composing the square root of x. You will notice that the various math forms have at most one hollow rectangle, the element that can absorb the highlighted material. Note that this math composition mechanism works only when the highlighted material consists of a single expression, within just one paragraph, and consisting of complete math forms. You can highlight the Gettysburg Address or just a part of a fraction, but you can't use the Math Tool to compose that stuff into a math form! But you will find, for example, vertical fraction forms where the hollow element is the numerator and where it is the denominator, letting you create (for x above) the forms

 or  . This is a powerful compositional and editing tool. If you do not have the Math Tools option, you must input math forms as six-key braille by entering things like "begin radical" and "end radical" indicator cells. You can see print math taking shape by turning on the (green) interpreter line. Without the Math Tools option, special symbols can be input as braille or by using Insert / Unicode Text. (You can create a Speedbraille™ key for symbols you use frequently.)

Direct Import of .docx Files (part of the Math Tools option)

Braille2000 natively reads rich text (RTF) data, from .rtf files via Open / Source File and from the Clipboard via Paste. In V3, this includes reading math equations in RTF data as OMML markup (the way the equation tool in Word encodes math). The Math Tools option provides direct input via Open / Source File for .docx files. By that technique, Braille2000 can load math material from Word files that use Microsoft's built-in equation typesetter. Thus, it is handy to be able to import directly from .docx files. One strategy is to use the equation typesetter in Word, save the .docx file, and open it in Braille2000. Without the Math Tools option, you can import Word materials (including math) using Copy and Paste, or by first using SaveAs in Word to create the .rtf version of the document. (Math done via the built-in Microsoft typesetter does import in this process. Note that Wordpad does not support OMML and any in an .rtf file will disappear if Saved by Wordpad.)

Math Notation

Braille2000 conceptualizes math as involving three aspects:

- plain text
  This is text you can type on the keyboard. It includes numbers and the names of variables and functions and some symbols.
- special text
  This is text you cannot type on the keyboard. It includes math operators (other than + - =), special symbols, and Greek and other non-Roman letters.
- math forms
  This is math layout information (i.e., math form templates). It includes superscript, subscript,

over and under scripts, fractions, radicals, and any other layout that arranges text other than left to right along the baseline.

In Braille2000, print input of math is done in the S (Source) View (click the  button). Plain text is input using the keyboard. Special text is input either via the Math Toolbar (part of the "Math Tools" option) or via Insert / Unicode Text (the Insert button in the regular toolbar). Math forms are input via the Math Toolbar. Of course, all math can be input by entering the appropriate braille cell sequences, for which you can define Speedbraille™ keys, if you wish.

Via the Braille View, all braille can be edited. Some edits will dramatically change the math expression, perhaps making it invalid. Note that Braille2000 cannot understand a braille math form until it has all the required "parts", e.g., a fraction does not exist until "begin fraction", "slash", and "end fraction" indicator cells are present in a valid manner… until "end fraction" is input, the interpreted print will be gibberish.

In the Source View, math symbols can be edited. Print symbols can be inserted by typing or via Insert / Unicode Text or via Insert / International Characters or via the Math Toolbar. Print symbols may be removed, one letter or symbol at a time, using the backspace key or the delete key. But math forms cannot be removed that way, unless the entire math form is highlighted. While composing the highlight, the "Math" button is grayed-out when the highlight does not cover the entire form, i.e., when the Math button is bold, the material can be removed with Delete or Backspace keys. You can delete the text of the numerator of a fraction, but the fraction endures. You cannot remove a fraction one part at a time, but if you highlight a whole fraction, then you can remove it.

The Four View Types

 The Control Panel holds the view selector buttons. The view can be any one of:

A – The ASCII view. Braille cells shown using Roman characters
B – The Braille view. Braille cells shown using simbraille (visual) dots
P – The Print view. Print-equivalent text shown using cell-height lines (not all math can show)
S – The Source view. Print-equivalent text shown using variable-height lines

The views other than S show virtual braille pages (typically 40 cells by 25 lines) with vertical and horizontal scrolling. The print lines of the P view are the same height as braille lines so that the vertical layout aspect of the page remains the same as braille pages. The horizontal layout of print-equivalent text is made to mimic the layout of the braille, which sometimes causes print symbols to be widely spaced. Braille2000 is the only tool in the world that shows print-equivalent text in the geometry of the braille page.

The S view is similar to the P view, but each print line in the S view is variable height, perhaps being considerably taller than a braille line, in order to show a complex math form or to include a thumbnail image of a NIMAS graphic. Horizontal spacing in lines in S view is mostly the same as in the P view. When using the S view, the screen will not always show a full page—you may need to scroll down or up to view its enlarged vertical extent. Or you may want to reduce on-screen font size by setting Max Lines (via Adjust / Display) well above 25, say 30 or more.

Consider the expression $x^2 + \frac{1}{2} + \sqrt{y}$. When viewing math forms in the P view, items that need extra line height are not shown but their indicator cells show as simbraille instead. Here is what you see in the P view: it is stretched to the horizontal extent of the underlying braille and "tall" forms are replaced by their indicator cells (UEB is used in the example). The S view shows the expression as it was in print, perhaps stretched a bit horizontally (so it has the same horizontal extent as the braille). Note that there is just enough room in P view to show one level of superscript or subscript.

x² + 1 ⁝ 2 + ⁝ y ⁝

Split-screen Views

If you hold down the Ctrl key as you click a view selector button, the newly selected view appears only in the bottom half of the Braille2000 screen, the upper half remains in the original view mode. In this way you can see both B (braille) and S (source) presentations at the same time. The two are dynamically managed: a change to one instantly updates the other.

Multiple Full-page Views

Braille2000 is a multi-document and multi-view tool, in that its main window can be linked to a choice of simultaneously available documents. At any one time, the choice of document is handled by the View Selector tabs that read sideways just to the right of the Control Panel, as shown here. As pictured, there are two documents (Unnamed1 and test.BML). Clicking the third tab will show the file test.BML via the S view. Clicking the first tab will show the file Unnamed1 via the S view and clicking the second tab will show the same file via the B view. The numbers 1 and 2 on those tabs indicate they are two views for the same file. By using View / Add View, you can create additional tabs for the same file. Each such view can be of any kind and will have its own cursor position and scrolling and highlighting. If you look at the same parts of both views of the same file, you will find them dynamically linked. Having two views into the same file is handy when completing a title page … you can edit the page in one view while scrolling through the document to check page numbers in the other view.

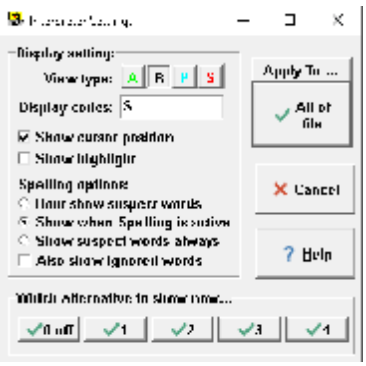
Multiple Screens

If you have a very large computer screen or multiple monitors, you can have multiple full-page views showing at the same time. First, have multiple tabs in the View Selector (see above). Then, for one tab, right-click on the tab and choose "Windowize". That view will be cast into a new, separate, Braille2000 window of its own. You can then drag that window (drag its title bar) to a desired space on your large monitor or dual-monitor setup. If the views are for the same file, they are always dynamically interconnected. You can show braille on one and print on the other.

Interpreter Overbar

The green interpreter overbar can be activated for any view using the Interpret button. When active, a green-colored one-line panel floats over regular text just below the line containing the cursor—as the cursor moves up and down the page, the green panel moves with it. The green panel is like one line of a view and can show text in any mode, A, B, P, or S, however, because it is floating, it is always variable height, and thus its P mode behaves like S mode.

Via right-click to the Interpret button, you can change the behavior of the green line. There are actually four interpreter settings, a different setting for each underlying view type. Typically, the interpreter setting in the B view is “P” and vice-a-versa, so the green line shows print when used in a braille view and shows braille when used in a print view. But you can change that. Right-click the Interpret button to get the Interpreter Settings dialog box. At the top are the four view-type buttons (the one for the current view will be down). Below it is a Codes input box used to configure the green line. In the Codes box, use the letters A B or P/S to get ASCII, braille, or print, respectively.



This snapshot was taken when the B (braille) view was active. It causes the green line to show S (print) material (via Display codes).

If you give multiple letters, the green line panel will show multiple interpretations, one line of each. Entering BA in Display codes will show a line of simbraille *and* a line of ASCII-braille. You can enter multiple configurations separated by commas to declare a sequence of behaviors. For example, B,A declares two behaviors (B and A, each one line). On each cursor line-change (when the green line moves to new material), the first behavior is active. Clicking the Interpret button advances to the next behavior. You can make yourself a Speedbraille™ key to do that too. More elaborate settings, such as B,BA,BP (three behaviors, some multi-line) are possible. And remember that each main viewing mode (A,B,P,S) has its own Interpreter settings.

The buttons at the bottom (0 1 2 3 4) are rarely used by clicking. Button 0 hides the green line while buttons 1 2 3 4 show the green line using its 1st 2nd 3rd or 4th behavior. These buttons exist to let you define Speedbraille™ keys (for one or another of these buttons), to give you Speedbraille control of the green line.

Perspective: Braille Codes for Mathematics

History recalls that the notion of tactile writing began in or before the Napoleonic era with embossed tables of azimuth data for firing cannons in the dark, i.e., cannon angles versus target distances, i.e., numbers. Charles Barbier de la Serre (1767-1841) is cited as the inventor of dot patterns for the alphabet, involving two adjacent 2x2 dot groups, each depicting a number from 1-5—those two numbers related to a 5x5 grid of letter symbols, a to z (without w… “double-v” was not used in French… w is a cell out of sequence to this day).

Louis Braille (1809-1852) was blinded as a child and educated in Catholic schools in music and literature. He was offered tactile materials in raised Roman letters and in Barbier’s system, neither of which was optimal. It was Louis who composed the 2x3 cell alphabet and the concept of “slate and stylus” as a writing system. He was an organist and also defined a code for music. Digits were based on a to i with dot-6 added (plus dots-3456 for zero, so called “French numbers”). Ultimately those cells were used for contractions, the dot-6 was dropped and the 3456 (numeric mode indicator) used to signal that a-j were to be interpreted as 1-9 and 0 (so called “upper numbers”).

Henry Martyn Taylor (1842-1927) taught math at Trinity College. He became blind as an adult and invented a notation for mathematics (that in that day was no more advanced than algebra and trigonometry). His notation used both "upper numbers" (a-j) and "lower numbers" (the same dot patterns one dot lower). His code used parenthesis as a syntactic element even when the math expression did not use them in print. Abraham Nemeth (1918-2013), blind from birth, taught math at the University of Detroit. When studying math (involving notations much more advanced than algebra and trigonometry) he found Taylor code to be insufficient. He was aided by volunteer readers and trained them on how best to read math formulas (a "math speak" language). This linearization of mathematics notation became the Nemeth Braille Code for Mathematics and Science Notation, with a vast set of symbol definitions. That code was published in 1952 and has been revised several times, lastly in 2022.

The notion of Unified English Braille (UEB) began in 1991 when Abraham Nemeth and Tim Cranmer pitched the idea of a "combined" braille code to the Braille Authority of North America (BANA). For forty years, braille educational materials in North America had been of two kinds: "literary" (using English Braille American Edition (EBAE)) and "Nemeth" (using the Nemeth math code on top of EBAE). Ordinary literature (as would be used studying liberal arts) used only EBAE but the literature for math and science used Nemeth. A braille textbook was transcribed in one or the other, and the two differed mainly in the use of upper digits in EBAE and lower digits in Nemeth. Although EBAE was the backbone of a Nemeth transcription, the two notions were viewed as separate and competing and thus Nemeth and Cranmer suggested a combining. In 1992, BANA formed a Unified English Braille Code committee to study such a merger and it issued a report. Ultimately, this became an international endeavor with the formation of the International Council on English Braille (ICEB) in 1993 (Abraham Nemeth, Tim Cramer, and Joseph Sullivan, authors of the BANA report, were on the council along with five others from other countries).

The UEB code was developed by ICEB and published in 2004. Although Nemeth urged the use of lower digits, the council adopted upper digits exclusively. (Note that, historically, digits always "overlapped" other things… letters in EBAE and UEB, punctuation in Nemeth.) Abraham Nemeth was concerned that UEB, with exclusively upper digits, to be promoted as the "unifying single braille code" would be inferior for mathematics, and so he developed a competing unifying code called the Nemeth Uniform Braille System (NUBS) published in 2010. He tried to sell the concept to BANA, never realizing that the politics of the international braille community (the members of ICEB) had pre-ordained UEB as the only option. (At the time, BANA pretended to consider NUBS but never intended to allow it). Part of the problem was that the Nemeth code itself was unknown outside North America, and thus had no international constituency.

Some organizations (a subset of members of BANA) objected to UEB and in 2012 BANA adopted** UEB as the recognized braille code for North America but only for "literary" transcriptions (the old two-kinds-of-braille approach continued, now with UEB versus Nemeth replacing EBAE versus Nemeth). Canada rejected that approach and so, today, the United States is the only country still using Nemeth braille.

****BANA has no actual authority. It is not an official entity. It is an organization of blindness-related agencies and its "authority" comes from stifling opposing sentiment by having its member agencies be of one voice.**

Because the Nemeth code was built as an overlay for EBAE, it is not compatible with UEB, i.e., the graceful EBAE-with-Nemeth (used for over 40 years) does not apply to UEB-with-Nemeth, and to make it work, UEB was patched with code switch indicator symbols ("begin Nemeth" and

“end Nemeth” three-cell tokens) by which Nemeth material could be inserted into a UEB transcription. This destroys the notion of a ***combined*** braille code (meaning one overarching code without duality of symbols). The UEB code, by itself, is indeed such a combined code. It is skillfully composed and is highly expressive. Its use of exclusively upper digits makes algebraic expressions sometimes cumbersome and it is particularly cumbersome when a math student is using it to author math homework and write scholarly reports (Abraham Nemeth often cited the power of brevity for such tasks, providing quicker data entry and greater coverage per line of braille—aspects that become important in advanced study).

Outside the United States, anything involving English is transcribed in UEB. But in the United States, math transcriptions are done in various ways based on local notions, in the following manners:

- UEB (referred to as UEB-technical)
- UEB-with-Nemeth (UEB using code switch indicators to patch in Nemeth)
- UEB and Nemeth (UEB and Nemeth without code switch indicators)
- EBAE-with-Nemeth (the pre-2012 approach)
- NUBS (rarely used, politically incorrect, a ***combined*** code with the power of UEB)

Note that Nemeth per se has no contractions. Contracted narrative can occur in the UEB or EBAE portions of a transcription. There are published code definitions for all approaches, although some are much less known. Note especially that the original Nemeth code (that which predates NUBS and on which Nemeth2022 is defined) requires semantic encoding, with the pitfalls and special techniques discussed earlier. Braille2000 is designed to support all of these techniques and it is unique in this regard.

Practicality

In the United States (where many braille readers are familiar with Nemeth code), the efficiency of the Nemeth code is considered superior to that of UEB, at least for advanced math. But given that BANA says that UEB is the standard, blind children are taught UEB and today’s liberal arts literature in the United State does not use Nemeth notation (it doesn’t need to). That means that braille readers are presumed fluent with numbers, fractions, subscripts, and superscripts in UEB. Those notations are totally different in Nemeth braille, the point being that the UEB-with-Nemeth approach requires dual awareness by the reader. This is tragic, but the pure UEB approach can be so convoluted with number indicators and grade-one indicators that math becomes obscure (both notions vary in severity depending on content—there is no pure play).

The 2022 Nemeth-with-UEB specification takes the notion that embedded Nemeth sub-expressions are to be used for all non-narrative utterances, each such utterance requiring six code-switch indicator cells, switching not just symbol sets but also notions on typeform and alphabet signaling. This takes the pre-2012 dual world of EBAE versus Nemeth braille volumes to the present notion of dueling UEB versus Nemeth on every page. In general, such a transcription would have two kinds of numbers, two kinds of Greek letters, two kinds of + - = in math, two kinds of bold and italics, to say nothing about math forms such as fractions, superscripts, subscripts, and radicals. The purpose of embedding Nemeth into UEB is to better present math, and so it is reasonable to relegate math expressions to the Nemeth embedding. That manages much of the duality. But what about narrative that mentions numbers, or Greek letters, or uses

superscripts as a reference indicator? Those elements are found in general literature and any braille reader of those elements has to have been exposed to them already in general literature, as UEB.

The 2022 UEB-with-Nemeth code wants all math utterances to be in embedded Nemeth. That is every number, every subscript, every superscript, every Greek or non-Roman letter, every function name (such as sin or cos or tan), as well as math equations per se. It is reasonable not to use UEB for "unusual" notations, e.g., the reader need not know the UEB notation for pi if all uses of pi are in the Nemeth notation instead. Same for fractions and radicals and math symbols—it is reasonable to push those elements uniformly into embedded Nemeth. But it is not reasonable to push "sin" into embedded Nemeth when it is in narrative context, as in "The sin function is defined as …". Switching to Nemeth, as mandated by the 2022 code for the function name "sin" is ridiculous and it is impossible for automated translation to know whether that ordinary word is a math function or a transgression. Same for "tan". (But of course 2+sin θ would be transcribed as Nemeth.) Similarly, the 2022 code specifies that single narrative words linking one math expression to another (such as "becomes") is to be "escaped" from Nemeth, back to UEB, mid-Nemeth, via dot-6 dot-3, so it can be contracted. Doing that takes two extra cells (all indicators hinder readability) and it cannot be that a reader of math equations would be troubled by an occasional uncontracted English word—the natural approach is to just spell it out, in Nemeth code—anything else is "form over substance" nonsense. (It is one thing to suggest human transcribers slave over such issues, it is something else to try to automate such things.) The point being made is that Braille2000 supports the Nemeth 2022 standard (embedded Nemeth) in a comprehensive but sensible way, and that purists will find deviations, that in this author's opinion enhance readability and do not hinder it.