Problem Authoring Reference Dec. 24, 2023

There are four types of problems that can be authored:

* **Multiple Choice** The student chooses one option from a list
* **Multiple Selection** The student can choose any number of options from a list
* **Numeric** The student enters a single numeric result
* **Embedded Input** The most flexible - the question may include several inputs

A problem is stored in an XML file with a format specified in this document. The top-level element is:

<problem type="MultipleChoice|MultipleSelection|Numeric|EmbeddedInput"  
 calculator="none|basic|scientific|graphing|full"

student-string-answer="..." (student's entered response to numeric)

num-choices="... integer ..." (# of choices to show in multiple choice/selection)

random-order="... boolean ..." (TRUE/FALSE, only for multiple choice/selection)

choice-order="1,2,3" (choice order for generated multiple choice/selection)

student-choices="1,2" (student selections in multiple choice/selection)

min-correct="... integer ..." (min correct choices to show in multiple selection)

max-correct="... integer ..."> (max correct choices to show in multiple selection)

... see below for child elements ...

</problem>

Every problem has a unique ID, (a dot-separated list of ID parts). This should match the path of the XML file under a top-level "instruction" directory. For example, a problem whose ID is "foo.bar" should be stored as "./foo/bar.xml" relative to the top-level directory.

The ID is stored in a <ref-base> element as a child of the <problem> element:

<ref-base>... ID ...</ref-base>

**Multiple Choice** and **Multiple Selection** problems may have the following additional elements as children of the top-level <problem> element.

* The number of choices to show the student. If this element is not present, all choices in the problem will be shown. If more than this number of choices exist in the problem, this number will be randomly selected and presented.

<num-choices>... Integer expr. (see Expression Reference) ...</num-choices>

* A flag indicating whether or not to randomly order the choices (note that individual choices can specify a fixed position - for example to place "None of the above" at the bottom of a randomly ordered choice list).

<random-order>... Boolean expr. (see Expression Reference) ...</random-order>

**Multiple Selection** problems may have the following elements as children of the <problem> element.

* The minimum and maximum number of correct choices to present to the student.

<min-correct>... Integer expr. (see Expression Reference) ...</min-correct>

<max-correct>... Integer expr. (see Expression Reference) ...</max-correct>

**Numeric** problems may have the following additional element as children of the top-level <problem> element.

* The minimum and maximum number of correct choices to guarantee will be presented to the student (for Multiple Choice problems, both of these values are 1).

<accept-number type="integer|real"

variance="... real constant - for formula use child element ..."/>

<variance>... Integer or Real expr. (see Expression Reference) ...</variance>

<correct-answer>... Integer or Real expr. (see Expression Reference) ...</correct-answer>

</accept-number>

If type is "integer", the variance and correct answer formulas must generate integers, and if type is "real", they must generate real numbers. The problem will accept any answer that falls within "variance" of "correct-answer" (endpoints of this interval are included).

Every type of problem may contain a <question> and <solution> child element. The <question> element is shown to the student when the problem is presented, and the <solution> child element is displayed after the item has been submitted and scored. The solution should tailor itself to the randomized variables used in the question. These are both direct children of the <problem> element.

<question>

... (See "Doc Column" in Document Reference) ...

</question>

<solution>

... (See "Doc Column" in Document Reference) ...

</solution>

**Embedded Input** problems have a "correctness" formula, used to calculate whether a particular set of user-entered input values represent a "correct" answer, and may have a "representative correct answer" that can be displayed, defined by an <answer> element (a direct child of the <problem> element). This answer is simple, without the detailed explanation provided in the <solution> element.

<correct>

... Boolean expr. (see Expression Reference) ...

</correct>

<answer>

... (See "Doc Column" in Document Reference) ...

</answer>

**Multiple Choice** and **Multiple Selection** problems have one or more choices from which to select, each defined by a <choice> element (these are direct children of the <problem> element).

<choice id="... Integer ID ..."

correct="TRUE|FALSE"

position="... Integer fixed position under random ordering ..."> (optional)

<correct>... Boolean expr. (see Expression Reference) ...</correct>

<content>... (See "Doc Column" in Document Reference) ...</content>

</choice>

Variable Reference

All types of problems may contain any number of variables or parameters, which may depend on one another (as long as there are no circular dependencies). The set of all variables in a problem will be randomly generated each time the problem is "realized" for presentation, and their values referenced in any formula within the problem.

There are eleven types of variables, based on the type of value generated, and the generation method:

* **Boolean** A fixed Boolean constant, TRUE or FALSE.
* **Integer** A fixed integer constant (64-bit signed)
* **Real** A fixed real constant (double-precision floating point, including ±∞, or irrational)
* **String** A fixed character string (limited to ASCII printable characters).
* **Span**  A fixed "span" constant (see Document Reference)
* **Random Boolean** A random Boolean value (a "coin toss")
* **Random Integer** A random integer value between specified bounds, with optional excluded values
* **Random Real** A random double-precision floating point value between specified bounds
* **Random Permutation** An int-vector with a random permutation of integers in a specified range
* **Random Choice** A value selected at random from a set of options, all of the same type.
* **Derived**  A value computed by an expression, whose type becomes this variable's type.

All contexts that can accept a real number (in the format supported by Java's Double.valueOf) will also accept an "irrational" number consisting of a fraction that multiplies π, e, or the square root of an integer. The format for this sort of real value is shown below. When a graph is configured with such a value as its axis tick mark interval, it will generate labels for tick marks that are multiples of these values rather than their decimal approximations.

PI→ π PI/2→ 7PI/15→ E→ e E/2→ 7E/15→ R3→ 2R3→ 2R3/5→

In addition to variables, each input in problem is assigned a name and can be used as a variable within expressions. Things like the correctness formula in an embedded input problem can use input variable values.

The XML formats of each type of variable are shown here.

<var name="..." type="boolean" value="TRUE|FALSE"/>

<var name="..." type="int"

format='...' or decimal-format="..." (format follows DecimalFormat from Java)

value="..."/> (format follows Long.toString() from Java)

<var name="..." type="real"

format='...' or decimal-format="..." (format follows DecimalFormat from Java)

value="..."/> (see note on real values above)

<var name="..." type="span">

... (See "Simple Span" in Document Reference) ...

</var>

Note: A "span" variable may include inputs. One could define several input blocks as spans, then use a random permutation variable to present them in random order.

<var name="..." type="random-boolean"

value="TRUE|FALSE"/> (only present if value was generated)

<var name="..." type="random-int"

min="... integer ..."

max="... integer ..."

format='...' or decimal-format="..." (format follows DecimalFormat from Java)

value="... integer ..."> (only present if value was generated)

<min> ... Integer expr. (see Expression Reference) ...</min> (required, no default)

<max> ... Integer expr. (see Expression Reference) ...</max> (required, no default)

<exclude> ... Integer expr. (see Expression Reference) ...</exclude> (optional, any number)

</var>

<var name="..." type="random-real"

min="... real ..." (see note on real values above)

max="... real ..." (see note on real values above)

format='...' or decimal-format="..." (format follows DecimalFormat from Java)

value="... real ..."> (only present if value was generated)

<min> ... Real expr. (see Expression Reference) ...</min> (required, no default)

<max> ... Real expr. (see Expression Reference) ...</max> (required, no default)

</var>

<var name="..." type="random-permutation"

min="... integer ..."

max="... integer ..."

value="... int-vector ..."> (only present if value was generated)

<min> ... Integer Expr. (see Expression Reference) ...</min> (required, no default)

<max> ... Integer Expr. (see Expression Reference) ...</max> (required, no default)

</var>

<var name="..." type="random-choice"

decimal-format="..." (format follows DecimalFormat from Java)

value-type="boolean|integer|real|span|string|int-vector"

boolean="TRUE|FALSE" (only present if Boolean value was generated)

long="... integer ..." (only present if Long value was generated)

double="... real ..." (only present if Double value was generated)

irrational="... irrational ..." (only present if Irrational value was generated)

string="..." (only present if String value was generated)

int-vector="... int-vector ..."> (only present if IntVector value was generated)

<choose-from> ... Expr. (see Expression Reference) ...</choose-from> (required, one or more)

<exclude> ... Expr. (see Expression Reference) ...</exclude> (optional, any number)

<span> ... </span> (only present if Span value was generated)

</var>

<var name="..." type="random-simple-angle"

min="... integer ..."

max="... integer ..."

max-denom="... integer ..."

value="... integer ..."> (only present if value was generated)

<min> ... Integer expr. (see Expression Reference) ...</min> (default is 0)

<max> ... Integer expr. (see Expression Reference) ...</max> (default is 359)

<max-denom> ... Integer expr. (see Expression Reference) ...</max-denom> (default is 30)

<exclude> ... Expr. (see Expression Reference) ...</exclude> (optional, any integer)

</var>

<var name="..." type="derived"

min="... real ..." (see note on real values above)

max="... real ..." (see note on real values above)

format='...' or decimal-format="..." (format follows DecimalFormat from Java)

value-type="boolean|integer|real|span|string|int-vector"

boolean="TRUE|FALSE" (only present if Boolean value was generated)

long="... integer ..." (only present if Long value was generated)

double="... real ..." (only present if Double value was generated)

irrational="... irrational ..." (only present if Irrational value was generated)

string="..." (only present if String value was generated)

int-vector="... int-vector ..."> (only present if IntVector value was generated)

<min> ... Expr. (see Expression Reference) ...</min> (required, no default)

<max> ... Expr. (see Expression Reference) ...</max> (required, no default)

<formula> ... Expr. (see Expression Reference) ...</formula> (required, no default)

<exclude> ... Expr. (see Expression Reference) ...</exclude> (optional, any number)

<span> ... </span> (only present if Span value was generated)

</var>

Expression Reference

Expressions may generate any of the following result types.

* **Boolean** TRUE or FALSE.
* **Integer** A 64-bit signed integer
* **Real** An Integer or a double-precision floating point, including +/- Infinity
* **Span**  A "Simple Span" (see Document Reference)
* **Integer Vector** An array of Integer values
* **Real Vector** An array of Real (which includes Integer) values
* **Error**  A result type to indicate an error, such as divide by zero

The possibility of "Span"-valued expressions is powerful, but creates challenges. Consider that a span may itself contain expressions, which could contain nested spans, and so forth. This precludes using a simple text-string format for expressions, and forces the use of hierarchical XML data structures. Hopefully editors will be developed to hide this complexity.

By default, an expression is stored in a <formula> element, but the containing element may have any tag name ("formula" is simply used here as a place-holder). For example, a random integer variable within a problem has <min>, <max> and optionally any number of <exclude> child elements that each contain an expression.

<formula (or other tag name)>

... (one child element that evaluates to a value - see below) ...

</formula>

Child elements can be any of the following:

<boolean value="TRUE|FALSE"/>

<integer value="... integer ..."/> (format follows Long.toString() from Java)

<real value="... real ..."/> (see note on real formats in Problem reference section)

<string value="..."/>

<int-vector value='... int-vector ...'/>

<int-vector>

... (one or more integer-valued child elements) ...

</int-vector>

<real-vector value='... real-vector ...'/>

<real-vector>

... (one or more real-valued child elements) ...

</real-vector>

<span>

... a Simple Span (see Document Reference) ...

</span>

<error>... An error message ...</error>

<varref name="... variable name ..."

index="... index .../> (optional, used to select an entry in a vector)

<binary op="+|-|\*|/|^|%|&lt;|&gt;|\u2264|\u2265|=|~|\u2260|&|\||">

... Two child elements to which to apply the binary operation ...

</binary>

* The + operation adds one or more child values (all must be integer- or real-valued)
* The - operation subtracts the second child value from the first (both must be integer- or real-valued)
* The \* operation multiplies the two child values (both must be integer- or real-valued)
* The / operation divides the first child value by the second (the result will be a double-precision floating point value unless both child values are integers and the second evenly divides the first, in which case an integer is generated)
* The ^ operation raises the first child value to the power of the second (if both are integers and the power is positive, and the result will not overflow an integer, the result is an integer. Otherwise, the result is a double-precision real value, unless the result is not defined (see Java's Math.pow() function for cases), in which case an Error value is returned.
* The % operation computes the first child value modulo the second. The behavior follows Java's '%' operator.
* The &lt; (less-than) operation compares the two child values numerically, and returns TRUE of the first is less than the second; FALSE if not. Both child values must be numeric, or an Error is returned.
* The &gt; (greater-than) operation compares the two child values numerically, and returns TRUE of the first is greater than the second; FALSE if not. Both child values must be numeric, or an Error is returned.
* The \u2264 (less-than or equal to) operation compares the two child values numerically, and returns TRUE of the first is less than or equal to the second; FALSE if not. Both child values must be numeric, or an Error is returned.
* The \u2265 (greater-than or equal to) operation compares the two child values numerically, and returns TRUE of the first is greater than or equal to the second; FALSE if not. Both child values must be numeric, or an Error is returned.
* The = operation compares the two child values, and returns TRUE of they are equal; FALSE if not. Child vales may be Boolean, Integer, Real, or Span.
* The ~ operation compares the two child values with a tolerance (the third child value), and returns TRUE of their difference is no greater than the tolerance; FALSE if not. Child vales may be Integer or Real.
* The \u2260 (not equal) operation compares the two child values, and returns FALSE of they are equal; TRUE if not. Child vales may be Boolean, Integer, Real, or Span.
* The & (logical AND) operation performs a logical AND on one or more child values (all must be Boolean)
* The | (logical OR) operation performs a logical OR on one or more child values (all must be Boolean)

<unary op="+|-">

... One child element to which to apply the unary operation ...

</unary>

* The + operation simply returns the child value (which must be numeric)
* The - operation negates the child value and returns the result (the child must be numeric)

<formula> ... </formula>

<function name="abs|cos|sin|tan|acos|asin|atan|exp|log|ceil|floor|round|sqrt|cbrt|  
 toDeg|toRad|radNum|radDen|not|gcd|lcm|srad2|srad3|lcase|ucase">

... One child element to which to apply the function ...

</function>

* The abs function returns the absolute value of an integer or real child value
* The cos function returns the cosine of the child value (treated as an angle in radians)
* The sin function returns the sine of the child value (treated as an angle in radians)
* The tan function returns the tangent of the child value (treated as an angle in radians)
* The acos function returns the arc-cosine of the child value (return value is in radians)
* The asin function returns the arc-sine of the child value (return value is in radians)
* The atan function returns the arc-tangent of the child value (return value is in radians)
* The exp function returns the result of raising "e" to the power of the child value
* The log function returns the natural logarithm of the child value
* The ceil function returns the least integer that is greater than or equal to the child value
* The floor function returns the greatest integer that is less than or equal to the child value
* The round function returns the nearest integer to the child value
* The sqrt function returns the square root of the child value
* The cbrt function returns the cube root of the child value
* The toDeg function returns the child value multiplied by (180/π) to convert radians to degrees
* The toRad function returns the child value multiplied by (π/180) to convert degrees to radians
* The radNum and radDen functions returns the integer numerator and denominator, respectively, of the fraction coefficient on π in the radian representation of an integer degree measure
* The not function returns the logical NOT of its Boolean child value
* The gcd function returns the greatest common divisor of an integer vector argument
* The lcm function returns the least common multiple of an integer vector argument
* The srad2 function returns the largest integer that can be factored out of a square root (for example, if the child value is 20, the result is 2, since 2 can be factored out of sqrt(20), leaving 2 sqrt(5).
* The srad3 function returns the largest integer that can be factored out of a cube root (for example, if the child value is 40, the result is 2, since 2 can be factored out of cbrt(40), leaving 2 cbrt(5).
* The lcase and ucase functions convert a string argument to lowercase or uppercase, respectively

<test>

... One Boolean-valued child element (the "IF" clause) ...

... One child element (the "THEN" clause) ...

... One child element (the "ELSE" clause) ...

</test>

* This element provides "if-then-else" logic. The first child value (which must evaluate to Boolean) is evaluated. If the result is TRUE, the second child value is evaluated and returned; otherwise, the third child value is evaluated and returned. The second and third child values must have the same type (although one could be Integer and the other Real, resulting in this element having Real value).
* Child elements may be nested <test> elements, supporting "Else-if" chains.

<switch>

<condition> ... An Integer-valued child element to select a case ... </condition>

<case value=”M”> ... the value expression when condition evaluates to M ... </case>

<case value=”N”> ... the value expression when condition evaluates to N ... </case>

...

<default> ... the value expression to return when no cases match </default>

</switch>

* This element provides case switching. The condition child is evaluated and must return an integer. That value is then compared to each “case” child’s value, and when a match is found, that case’s child expression is evaluated and returned. If no cases match, the default value is evaluated and returned (or an error is returned if there is no default value).
* Every case’s child expression and the default value’s child expression must be of the same type (although there can be a mixture of integer and real number values, in which case the switch is considered real-valued).

<is-exact>

... One real-valued child element (the value to test) ...

... One integer-valued child element (the number of decimal places, from 0 to 10) ...

</is-exact>

* This element tests whether some real value is "exact" when printed to some number of digits after the decimal. For example, "1.66" is exact to 3 decimal places, but "1.6666" is not. This is useful, for example, when choosing whether to display "=" or "" before a computed value.
* This test works by scaling the real value by 10 to the power of the number of decimal places, subtracting this value from the same value rounded to an integer, and comparing the result to a very small value (this allows for small differences caused by roundoff or floating-point arithmetic).

Document Reference

Document structures are used in two ways in the assessment system. Questions and solutions are represented as a **DocColumn**, which is similar to an HTML page. Given a region of specified width, it lays out its content from top to bottom, left to right. Other contexts allow a **Span** of content, which will be placed into some containing **DocColumn** eventually. A span provides content and style, but does not do layout. The span's contents are arranged only after they have been inserted into a **DocColumn**. An example is a span-valued variable.

**Element Styles**

Every element in a document may have styles attached. If an element does not define a style, it inherits the value of that style from its parent. The top-level element in a document tree provides default style values (plain Serif font, 24 pixels in size, black color).

Style properties are set using attributes on the element being styled:

<tag color="... a color name ..."

fontname="... a font name ..."

fontsize="... a font size in pixels or a font scale factor percentage, like '85%' ..."

fontstyle="... see below ..."> ... </tag>

Color names allow all X11 colors, plus a few additional CSU colors, and RGB triples of the form "RRGGBB". See the appendix for a full list of pre-defined color names.

Font names can be "SANS" or "SERIF" or "MONOSPACE".

Font styles is either "plain", or a comma-separated list of any of the following:

* "bold", "italic", "strikethrough", "underline", "overline", or "boxed"

**DocColumn**

A **DocColumn** may contain one or more inputs to allow the user to make a selection or enter data. Every input has a name, and the **DocColumn** can provide a list of its inputs and their current values. These values can be treated like variables within a problem (in fact, things like a "correctness" formula must use these values to properly score a problem).

A **DocColumn** has no default tag name - it could be "question", "solution", "choice", etc.

<tag ... style properties (see above) ...>

... zero or more <p> or <v-space> elements ...

</tag>

A **DocColumn** may contain only "paragraphs" (which generate a block, the full width of the column, and whose height is determined by their content), or "vertical space". This is similar to a "block formatting context" in HTML, but without the possibility of a "float".

Vertical space is created by a "v-space" element, which defines a height (in "ems", where 1.0 represents the height of a single line of text at the current font size).

<v-space height='...a numeric constant...'>

<height>... a Real expr. (see Expression Reference) ...</height>

</v-space>

Either a constant height or a formula-based height should be provided (not both). If neither is provided, a value of 1.0 is assumed. This object is emitted in HTML as a <div> element with the specified height (in "ems").

**Paragraph**

A paragraph, defined by a <p> element, lays out its content in top-down, left to right order. By default, a paragraph is "left justified", but a "justification" attribute allows it to be right-justified, full-justified, or centered.

<p ... style properties (see above) ...  
 justification="left|right|center|full">

... child elements - see below ...

</p>

A paragraph may contain any of the following elements (see below for definitions)

* ordinary text Text will be laid out as the paragraph is flowed into container's width.
* symbol reference Pre-defined symbol references. For example, {\pi} to enter a π symbol.
* variable reference Variable references. to insert variable values. For example, {x}.
* <span> A styled run of content.
* <nonwrap> A span that will not break across lines.
* <math> A math environment.
* <fraction> A fraction construction, with numerator, denominator, and fraction bar.
* <radical> A radical construction with optional root.
* <rel-offset> A relative offset construction with base, superscript, subscript, over, or under.
* <fence> Wraps its content in auto-sized "fence" characters like parentheses or brackets.
* <h-space> A horizontal space.
* <image> A raster image from a URL.
* <table> A table with rows and cells.
* <drawing> A drawing.
* <graphxy> An x-y graph that can show axes, function curves, or drawing primitives.
* <input> An input with which the user can interact.
* <symbol-palette> A palette that acts like a virtual keypad to enter symbols like "π" (experimental).

When emitted as HTML, a paragraph is emitted as a <div> with 0.5em top and bottom margin, and 'text-align' attribute set based on the paragraph justification.

**Text, symbol references, and variable references**

Text is entered normally, although strings of whitespace will be treated as a single whitespace, meaning that you can format your XML file with carriage returns, tabs, and spaces, but the generated will reflow itself with a single whitespace between words.

The "escape" symbols from XML can be used to enter characters that would otherwise have meaning in XML:

&gt; for > &lt; for < &quot; for " &apos; for ' &amp; for &

Unicode characters can be included using a \u followed by four hex digits. For example, \u221e is the "infinity" symbol. However, symbol references should be used where possible.

To add formatting (color, font changes, etc.) to text, enclose it in a <span> or <nonwrap> tag.

A variable reference is entered by surrounding the variable name with curly brackets. For example:

<p>The value of x is {x}.</p>

This will embed the value of the "x" variable in the text. If the "x" variable is a numeric value, it's "format" attribute will be used to format the value.

A symbol reference is entered by surrounding the symbol name, preceded by a backslash, with curly brackets. For example:

<p>The interval from {\minus}{\infty} to {\infty} represents all real numbers.</p>

See the appendix for a complete list of available symbols. NOTE: If you need a symbol that is not defined, and find yourself using a \u#### code, please inform Steve who can define a new symbol - that makes code more readable.

Note that this system of variable and symbol references makes it hard to enter the "{" and "}" characters - you would need to use \u#### escapes for those. But you can use the <fence> element to enter these symbols, and this will probably provide better semantic clarity anyway.

**Spans and Simple Spans**

Spans are sections of text that can have styles applied. Spans flow into their containing paragraph normally, with their styles applied. A span generated with a <span> element can wrap its content over multiple lines. A span generated with a <nonwrap> element will not wrap its content. A common use of the <nonwrap> element is to keep things together that would be awkward if broken, like mathematical expressions, or numbers with units. A span generated with the <math> tag is considered a run of math, which mainly affects LaTeX generation. It would mainly be used to wrap variable names in text to set them in a math font. For example: "The variable <math>x</math>.".

<span ... style properties (see above) ...>

... child elements - same list as can exist in a paragraph ...

</span>

<nonwrap ... style properties (see above) ...

bgcolor="... color name ...">

... child elements - same list as can exist in a paragraph ...

</nonwrap>

<math ... style properties (see above) ...>

... child elements - same list as can exist in a paragraph ...

</math>

A "Simple Span" is a span that is defined in an element with a different name, outside the context of a paragraph, such as within a span-valued variable.

When emitted as HTML:

* a wrapping span, math span, or simple span's content is emitted directly, with no additional wrapper
* a non-wrapping span's content is emitted wrapped in a <span> element with CSS style set to "white-space:nowrap;"

**Fractions**

A fraction construction takes two items, and draws one above the other with a horizontal line separating them. The two items can be numbers, text, tables, images, graphs, etc. Each item will be centered horizontally, and the line length will be sized to match the larger of the two items. When flowing in text, fractions try to place their separating line on the centerline of the text.

Note that the numerator and denominator are treated as nonwrapping spans (they will not break across lines).

<fraction ... style properties (see above) ...>

<numerator ... style properties (see above) ...>

... child elements - same list as can exist in a paragraph ...

</numerator>

<denominator ... style properties (see above) ...>

... child elements - same list as can exist in a paragraph ...

</denominator>

</fraction>

When emitted as HTML, a fraction is emitted as a <table> element with one row for the numerator, with a 1-px black bottom border, and a second row with the denominator. Both are centered. The numerator is preceded by "screen-reader-only" text saying "a fraction whose numerator is", and the denominator is preceded by a similar "and whose denominator is". The fraction as a whole is followed by "screen-reader-only" text saying " end of fraction".

**Radicals**

A radical, or an "nth root" symbol draws some item, then encloses it in a radical symbol, placing an optional symbol to the above left of it to specify the root.

NOTE: There is no need to include an empty <root></root> for an ordinary square root - just omit the <root> element.

<radical ... style properties (see above) ...>

<base ... style properties (see above) ...>

... child elements - same list as can exist in a paragraph ...

</base>

<root ... style properties (see above) ...>

... child elements - same list as can exist in a paragraph ...

</root>

</radical>

When emitted as HTML, a radical is emitted as a span that sets the color for the construction, containing the following:

* if there is a root, a "screen-reader-only" text saying "root" followed by the root contents, all enclosed in a <sup> element
* a <div> element that draws the surd symbol
* a <div> element with 1-px top border that contains the base contents

**Relative Offsets**

A relative offset construction allows items to be draw with a particular relative positioning. This includes superscript and subscript positioning, and over/under positioning.

It does this by allowing you to specify a "base" item, then one or more items to position relative to the base item. Each of these items can be numbers, text, images, graphs, tables, spans, etc. Only the base item is mandatory - the rest are optional, and may be included in any combination.

NOTE: this object reduces the font size for scripts and over/under constructions by a standard amount - there should be no need to set font size on these children (and in fact, doing so can make things look inconsistent, so it should be avoided if possible).

<rel-offset ... style properties (see above) ...>

<base ... style properties (see above) ...>

... child elements - same list as can exist in a paragraph ...

</base>

<super ... style properties (see above) ...>

... child elements - same list as can exist in a paragraph ...

</super>

<sub ... style properties (see above) ...>

... child elements - same list as can exist in a paragraph ...

</sub>

<over ... style properties (see above) ...>

... child elements - same list as can exist in a paragraph ...

</over>

<under ... style properties (see above) ...>

... child elements - same list as can exist in a paragraph ...

</under>

</rel-offset>

When emitted as HTML, a relative offset construction is emitted as a span that sets the CSS "white-space" property to "nowrap", and contains the following:

* if there are no over or under children, the base content is emitted without wrapper; otherwise, a <div> element with "inline-grid" display is emitted with the base, over, and under parts stacked.
* if there is a superscript child, a "screen-reader-only" text saying "raised to power" or "superscript", depending on the math mode, followed by a <sup> element with the superscript child, then a "screen-reader-only" text saying "end of power" or "end of superscript", depending on the math mode
* if there is a subscript child, a "screen-reader-only" text saying "subscript " followed by a <sub> element with the subscript child, then a "screen-reader-only" text saying "end of subscript ".

**Fence**

A fence construction wraps its content in auto-sizing "fence" characters like parentheses, brackets, vertical bars, etc.

A fence can be configured to align the centerline of its fence characters to the centerline of the current line of text, or can align the baseline of the fence characters to the text baseline.

<fence ... style properties (see above) ...

type='parentheses|brackets|braces|bars|lbrace'

valign='center|baseline'>

... child elements - same list as can exist in a paragraph ...

</fence>

When emitted as HTML, a fence offset construction is emitted as a <div> of type "inline-block" that sets the CSS "white-space" property to "nowrap", provides a .5em left and right fence symbol as a CSS border, and contains its contents, preceded and followed by "screen-reader-only" text such as "open parenthesis"/"close parenthesis" that matches the fence type.

**Horizontal Space**

A horizontal space of fixed width in text. The width expression evaluates to a real number in units of the width of a decimal "0" digit. This is useful for lining up columns of digits.

<h-space width='... a numeric constant...'>

<width>... A Real expr. (see Expression Reference) ...</width>

</h-space>

Either a constant width or a formula-based width should be provided (not both). If neither is provided, a value of 1.0 is assumed. This object is emitted in HTML as a <span> element with the specified width.

**Images**

A raster image can be included, where the image comes from a URL.

<image width="... real ..." (see note on real formats in Problem reference section)

height="... real ..." (see note on real formats in Problem reference section)

src="... URL ...">

<width> ... a Real expr. (see Expression Reference) ... </width>

<height> ... a Real expr. (see Expression Reference) ... </height>

</image>

When emitted as HTML, images are emitted as <img> elements with the source used as the 'src' attribute.

**Tables**

A table is a collection of rows, each row being a collection of cells.

<table ... style properties (see above) ...

box-width="... thickness of box to surround the table ..."

v-line-width="... thickness of vertical lines between cells ..."

h-line-width="... thickness of horizontal lines between cells ..."

column-width="uniform|nonuniform"

justification="left|right|center" (justification within each cell)

bgcolor="... a color name ..." (background of whole table)

cell-margins="... one integer of 4 comma-separated integers ...">

... any number of <tr> elements, one per row ...

</table>

If "column-width" attribute is "uniform", all columns will be the same width (the width of the widest cell). Of "nonuniform", then each column is just wide enough to contain its cells.

If "cell-margins" has a single integer, that size (in pixels) is used around all cells. If it has a list of 4 comma-separated integers, those are used as the top, left, bottom, and right margin size, respectively.

<tr>

... any number of <td> elements ...

</tr>

<td lines="..." bgcolor="... color name ...">

... child elements - same list as can exist in a paragraph ...

</td>

The "lines" attribute can have a comma-separated list of any of these tokens:  
 left right top bottom  
This will draw the corresponding lines around the cell box. The default is to draw all lines - to draw no lines at all, set the "lines" attribute to an empty string.

When emitted as HTML, a table is emitted as a span that sets the font size and color and that contains a <table> element with child <tr> and <td> elements as needed.

**Drawings and Graphs**

There are two drawing-related constructions: a **Drawing** and a **GraphXY**. A **Drawing** provides a coordinate system and draws a collection of "primitives". A **GraphXY** adds to this the ability to draw x-y axes, and to plot functions.

<drawing ... style properties (see above) ...

width="... integer ..."

height"... integer ...">

<width>... integer expression ...</width>

<height>... integer expression ...</height>

... any number of drawing primitives - see below ...

</drawing>

<graphxy ... style properties (see above) ...

width="... integer ..."

height"... integer ..."

minx="... real ..." (see note on real formats in Problem reference section)

miny="... real ..." (see note on real formats in Problem reference section)

maxx="... real ..." (see note on real formats in Problem reference section)

maxy="... real ..." (see note on real formats in Problem reference section)

xtickinterval="... real ..." (see note on real formats in Problem reference section)

ytickinterval="... real ..." (see note on real formats in Problem reference section)

bgcolor="... color name ..."

bordercolor="... color name ..."

gridcolor="... color name ..."

tickcolor="... color name ..."

axiscolor="... color name ..."

borderwidth="... integer ..."

gridwidth="... integer ..."

tickwidth="... integer ..."

ticksize="... integer ..."

axiswidth="... integer ..."

axislabelfontsize="... integer ..."

ticklabelfontsize="... integer ..."

xaxislabel="... text ..."

yaxislabel="... text ..."

name="... unique name among all inputs or variables ...">

... any number of <formula> elements or drawing primitives - see below ...

</graphxy>

<formula color="... color name ..."

style="curve"

minx="... real ..." (see note on real formats in Problem reference section)

maxx="... real ..."> (see note on real formats in Problem reference section)

<minx>... real expression ...</minx>

<maxx>... real expression ...</maxx>

<expr>... real-valued expression using {t} as domain variable ...</expr>

</formula>

There are nine types of drawing primitive that a **Drawing** or **GraphXY** may contain:

* <line>A line specified by start point and offsets to end point.
* <arc>A circular arc specified by center, radius, start angle, and arc angle.
* <oval>An elliptical oval specified by center, x-radius, and y-radius.
* <rectangle>A rectangle specified by (x, y) and width and height.
* <polygon>A polygon specified by a sequence of points.
* <raster>A raster image.
* <text>A text string.
* <span>A formatted span (this can include variable references)
* <protractor>A rendered protractor, in either degree or radian units of measure

<line x="... real ..."

y="... real ..."

width="... real ..." (see note on real formats in Problem reference section)

height="... real ..." (see note on real formats in Problem reference section)

color="... color name ..."

stroke-width="... real ..." (see note on real formats in Problem reference section)

dash="... real, real, ..." (see note on real formats in Problem reference section)

alpha="... real ..."> (see note on real formats in Problem reference section)

<x> ... a Real expr. (see Expression Reference) ... </x>

<y> ... a Real expr. (see Expression Reference) ... </y>

<width> ... a Real expr. (see Expression Reference) ... </width>

<height> ... a Real expr. (see Expression Reference) ... </height>

</line>

<arc x="... real ..." (see note on real formats in Problem reference section)

y="... real ..." (see note on real formats in Problem reference section)

width="... real ..." (see note on real formats in Problem reference section)

height="... real ..." (see note on real formats in Problem reference section)

cx="... real ..." (see note on real formats in Problem reference section)

cy="... real ..." (see note on real formats in Problem reference section)

r="... real ..." (see note on real formats in Problem reference section)

rx="... real ..." (see note on real formats in Problem reference section)

ry="... real ..." (see note on real formats in Problem reference section)

start-angle="... real ..." (see note on real formats in Problem reference section)

arc-angle="... real ..." (see note on real formats in Problem reference section)

stroke-width="... real ..." (see note on real formats in Problem reference section)

stroke-color="... color name ..."

stroke-dash="... real, real, ..." (see note on real formats in Problem reference section)

stroke-alpha="... real ..." (see note on real formats in Problem reference section)

fill-style="none|pie|chord"

fill-color="... color name ..."

fill-alpha="... real ..." (see note on real formats in Problem reference section)

rays-shown="none|start|end|both"

ray-length="... real ..." (see note on real formats in Problem reference section)

ray-width="... real ..." (see note on real formats in Problem reference section)

ray-color="... color name ..."

ray-dash="... real, real, ..." (see note on real formats in Problem reference section)

ray-alpha="... real ..." (see note on real formats in Problem reference section)

label="... text ..." (may include variable references, special characters)

label-color="... color name ..."

label-alpha="... real ..." (see note on real formats in Problem reference section)

label-offset="... real ..." (see note on real formats in Problem reference section)

fontname="... font name ..."

fontsize="... real ..." (see note on real formats in Problem reference section)

fontstyle="... see below ...">

<x> ... a Real expr. (see Expression Reference) ... </x>

<y> ... a Real expr. (see Expression Reference) ... </y>

<width> ... a Real expr. (see Expression Reference) ... </width>

<height> ... a Real expr. (see Expression Reference) ... </height>

<cx> ... a Real expr. (see Expression Reference) ... </cx>

<cy> ... a Real expr. (see Expression Reference) ... </cy>

<r> ... a Real expr. (see Expression Reference) ... </r>

<rx> ... a Real expr. (see Expression Reference) ... </rx>

<ry> ... a Real expr. (see Expression Reference) ... </ry>

<start-angle> ... a Real expr. (see Expression Reference) ... </start-angle>

<arc-angle> ... a Real expr. (see Expression Reference) ... </arc-angle>

<label> ... any elements that can exist within a span ... </label>

</arc>

An arc, which may be configured to draw its starting or ending rays, and may have an optional label, which can be a simple string or a span. The label is drawn centered on the arc, further from the center than the arc itself.

The arc’s position and size can be specified one of three ways:

* With the ‘x’, ‘y’, ‘width’, and ‘height’ attributes (any of which can be specified as a formula in a child element rather than as an attribute with a fixed value), which specify a bounding box in which the arc’s circle or ellipse is inscribed.
* With the ‘cx’, ‘cy’, and ‘r’ attributes (any of which can be specified as a formula in a child element rather than as an attribute with a fixed value), which specifies a circular arc’s center point and radius.
* With the ‘cx’, ‘cy’, ‘rx’, and ‘ry’ attributes (any of which can be specified as a formula in a child element rather than as an attribute with a fixed value), which specify an elliptical arc’s center point and radius in each of the x- and y-axis directions.

By default, an arc is not filled. To fill the arc, specify the ‘fill-style’ attribute, which fills the arc in the arc color. To alter fill settings, use the ‘fill-color’, and ‘fill-alpha’ attributes.

By default, no rays are drawn. To draw a ray at the start or end of the arc, fill the arc, set the ‘rays-shown’ attribute to something other than ‘none’. By default, the specified rays are drawn with a length such that they end on the arc. To alter the length, specify the ‘ray-length’ attribute as a scaling factor (2 to draw the ray twice the arc length, etc.) To alter ray line style, use the ‘ray-width’, ‘ray-color’, ‘ray-dash’, and ‘ray-alpha’ attributes.

By default, no label is drawn. To add a label, specify either a string label, in the ‘label’ attribute, or a span label as a <label> child element. A string label can be styled with the ‘fontname’, ‘fontsize’, ‘fontstyle’, ‘label-color’, and ‘label-alpha’ attributes. The label’s position will be calculated so the center of the label will fall on the bisecting ray of the arc, at a distance that keeps the label close to, but not overlapping, the arc itself. The distance of the label’s center from the arc’s center can be adjusted using the ‘label-offset’ attribute, which is a constant distance that is added to the label’s distance to the arc center (it can be negative to move the label closer, even so it displays inside the arc).

TODO: I hope to add support for arrowheads at either or both ends of the arc, and at the ends of rays, and the ability to add “highlight” or “shadow” effects to arcs and rays.

<oval x="... real ..." (see note on real formats in Problem reference section)

y="... real ..." (see note on real formats in Problem reference section)

width="... real ..." (see note on real formats in Problem reference section)

height="... real ..." (see note on real formats in Problem reference section)

filled="TRUE|FALSE"

color="... color name ..."

stroke-width="... real ..." (see note on real formats in Problem reference section)

dash="... real, real, ..." (see note on real formats in Problem reference section)

alpha="... real ..."> (see note on real formats in Problem reference section)

<x> ... a Real expr. (see Expression Reference) ... </x>

<y> ... a Real expr. (see Expression Reference) ... </y>

<width> ... a Real expr. (see Expression Reference) ... </width>

<height> ... a Real expr. (see Expression Reference) ... </height>

</oval>

<rectangle x="... real ..." (see note on real formats in Problem reference section)

y="... real ..." (see note on real formats in Problem reference section)

width="... real ..." (see note on real formats in Problem reference section)

height="... real ..." (see note on real formats in Problem reference section)

filled="TRUE|FALSE"

color="... color name ..."

stroke-width="... real ..." (see note on real formats in Problem reference section)

dash="... real, real, ..." (see note on real formats in Problem reference section)

alpha="... real ..."> (see note on real formats in Problem reference section)

<x> ... a Real expr. (see Expression Reference) ... </x>

<y> ... a Real expr. (see Expression Reference) ... </y>

<width> ... a Real expr. (see Expression Reference) ... </width>

<height> ... a Real expr. (see Expression Reference) ... </height>

</rectangle>

<polygon x-list="... real, real, ..." (see note on real formats in Problem reference section)

y-list="... real, real, ..." (see note on real formats in Problem reference section)

filled="TRUE|FALSE"

color="... color name ..."

stroke-width="... real ..." (see note on real formats in Problem reference section)

dash="... real, real, ..." (see note on real formats in Problem reference section)

alpha="... real ..."> (see note on real formats in Problem reference section)

<x> ... a Real expr. (see Expression Reference) ... </x> (one child element per point)

<y> ... a Real expr. (see Expression Reference) ... </y> (one child element per point)

</polygon>

<text x="... real ..." (see note on real formats in Problem reference section)

y="... real ..." (see note on real formats in Problem reference section)

anchor="NW|N|NE|W|C|E|SW|S|SE"

color="... color name ..."

value="... text, can include symbol references ..."

fontname="... font name ..."

fontsize="... real ..." (see note on real formats in Problem reference section)

fontstyle="... see below ..."

alpha="... real ..."> (see note on real formats in Problem reference section)

<x> ... a Real expr. (see Expression Reference) ... </x>

<y> ... a Real expr. (see Expression Reference) ... </y>

</text>

The "fontstyle" attribute can be "plain", or a comma-separated list of any of the following:

* "bold", "italic"

<span x="... real ..." (see note on real formats in Problem reference section)

y="... real ..." (see note on real formats in Problem reference section)

anchor="NW|N|NE|W|C|E|SW|S|SE"

color="... color name ..."

value="... text, can include symbol references ..."

fontname="... font name ..."

fontsize="... real ..." (see note on real formats in Problem reference section)

fontstyle="... see below ..."

alpha="... real ..."> (see note on real formats in Problem reference section)

<x> ... a Real expr. (see Expression Reference) ... </x>

<y> ... a Real expr. (see Expression Reference) ... </y>

<content>... child elements - same list as can exist in a paragraph ...</content>

</span>

The "fontstyle" attribute can be "plain", or a comma-separated list of any of the following:

* "bold", "italic"

<raster x="... real ..." (see note on real formats in Problem reference section)

y="... real ..." (see note on real formats in Problem reference section)

width="... real ..." (see note on real formats in Problem reference section)

height="... real ..." (see note on real formats in Problem reference section)

src="... URL ..."

alpha="... real ..."> (see note on real formats in Problem reference section)

<x> ... a Real expr. (see Expression Reference) ... </x>

<y> ... a Real expr. (see Expression Reference) ... </y>

<width> ... a Real expr. (see Expression Reference) ... </width>

<height> ... a Real expr. (see Expression Reference) ... </height>

</raster>

<protractor cx="... real ..." (see note on real formats in Problem reference section)

cy="... real ..." (see note on real formats in Problem reference section)

r="... real ..." (see note on real formats in Problem reference section)

orientation="... real ..." (see note on real formats in Problem reference section)

units="deg|rad"

quadrants=”1|2|3|4"

color="... color name ..."

text-color="... color name ..."

alpha="... real ..."> (see note on real formats in Problem reference section)

<cx> ... a Real expr. (see Expression Reference) ... </cx>

<cy> ... a Real expr. (see Expression Reference) ... </cy>

<r> ... a Real expr. (see Expression Reference) ... </r>

<orientation> ... a Real expr. (see Expression Reference) ... </orientation>

</protractor>

When emitted as HTML, a drawing or graph is emitted as an inline image with the rendered contents. The content may be rendered at a higher resolution so browsers can "zoom in" to a certain level and see more detail rather than pixelated content. The "alt-text" for the image is simply "Drawing" or "Graph".

**Symbol Palette**

A symbol palette acts like a virtual keyboard to allow the user to enter symbols like "π".

<symbol-palette symbols="pi"/>

NOTE: At this point, only "Pi" is defined.

Symbol palettes are not emitted in HTML.

**Inputs**

Document columns within the <question> element of an "EmbeddedInput" problem may contain inputs.

Every input can be "enabled" based on a variable (which must be Boolean or Integer valued). To enable this feature, the input specifies the name of that variable and the value it must take on to enable the input.

<input type="integer"

name="... unique name among all inputs or variables ..."

width="... integer ..."

default="... integer ..." (used if the user leaves input blank)

style="underline" (used to present as underline)

treat-minus-as="... integer ..." (used if the user enters only "-")

enabled-var-name="variable name"

enabled-var-value="... boolean or integer ..."/>

<input type="real"

name="... unique name among all inputs or variables ..."

width="... integer ..."

default="... real ..." (used if the user leaves input blank)  
 (see note on real formats in Problem reference section)

style="underline" (used to present as underline)

treat-minus-as="... real ..." (used if the user enters only "-")  
 (see note on real formats in Problem reference section)

enabled-var-name="variable name"

enabled-var-value="... boolean or integer ..."/>

<input type="string"

name="... unique name among all inputs or variables ..."

width="... integer ..."

style="underline" (used to present as underline)

enabled-var-name="variable name"

enabled-var-value="... boolean or integer ..."/>

<input type="radio-button"

name="... unique name among all inputs or variables ..."

value="... integer ..."

selected="true|false"

enabled-var-name="variable name"

enabled-var-value="... boolean or integer ..."/>

<input type="checkbox"

name="... unique name among all inputs or variables ..."

value="... integer ..."

selected="true|false"

enabled-var-name="variable name"

enabled-var-value="... boolean or integer ..."/>

An integer, real, or string field is emitted in HTML as an <input> element with type='text', and the "id" and "name" attributes set to "INP\_" plus the input variable name.

A radio button input is emitted in HTML as an <input> element with type='radio', "name" set to "INP\_" plus the variable name, "id" set to "INP\_" plus the variable name plus "\_" then the value associated with this input, and "value" set to the value associated with the input.

A checkbox input is emitted in HTML as an <input> element with type=checkbox, "name" set to "INP\_" plus the variable name, "id" set to "INP\_" plus the variable name plus "\_" then the value associated with this input, and "value" set to the value associated with the input.

Colors

Color names allowed by the system are defined here.

black

white

snow

ghost white

GhostWhite

white smoke

WhiteSmoke

gainsboro

floral white

FloralWhite

old lace

OldLace

linen

antique white

AntiqueWhite

papaya whip

PapayaWhip

blanched almond

BlanchedAlmond

bisque

peach puff

PeachPuff

navajo white

NavajoWhite

moccasin

cornsilk

ivory

lemon chiffon

LemonChiffon

seashell

honeydew

mint cream

MintCream

azure

alice blue

AliceBlue

lavender

lavender blush

LavenderBlush

misty rose

MistyRose

dark slate gray

DarkSlateGray

dark slate grey

DarkSlateGrey

dim gray

DimGray

dim grey

DimGrey

slate gray

SlateGray

slate grey

SlateGrey

light slate gray

LightSlateGray

light slate grey

LightSlateGrey

gray

grey

light grey

LightGrey

light gray

LightGray

midnight blue

MidnightBlue

navy

navy blue

NavyBlue

cornflower blue

CornflowerBlue

dark slate blue

DarkSlateBlue

slate blue

SlateBlue

medium slate blue

MediumSlateBlue

light slate blue

LightSlateBlue

medium blue

MediumBlue

royal blue

RoyalBlue

blue

dodger blue

DodgerBlue

deep sky blue

DeepSkyBlue

sky blue

SkyBlue

light sky blue

LightSkyBlue

steel blue

SteelBlue

light steel blue

LightSteelBlue

light blue

LightBlue

powder blue

PowderBlue

pale turquoise

PaleTurquoise

dark turquoise

DarkTurquoise

medium turquoise

MediumTurquoise

turquoise

cyan

light cyan

LightCyan

cadet blue

CadetBlue

medium aquamarine

MediumAquamarine

aquamarine

dark green

DarkGreen

dark olive green

DarkOliveGreen

dark sea green

DarkSeaGreen

sea green

SeaGreen

medium sea green

MediumSeaGreen

light sea green

LightSeaGreen

pale green

PaleGreen

spring green

SpringGreen

lawn green

LawnGreen

green

chartreuse

medium spring green

MediumSpringGreen

green yellow

GreenYellow

lime green

LimeGreen

yellow green

YellowGreen

forest green

ForestGreen

olive drab

OliveDrab

dark khaki

DarkKhaki

khaki

pale goldenrod

PaleGoldenrod

light goldenrod yellow

LightGoldenrodYellow

light yellow

LightYellow

yellow

gold

light goldenrod

LightGoldenrod

goldenrod

dark goldenrod

DarkGoldenrod

rosy brown

RosyBrown

indian red

IndianRed

saddle brown

SaddleBrown

sienna

peru

burlywood

beige

wheat

sandy brown

SandyBrown

tan

chocolate

firebrick

brown

dark salmon

DarkSalmon

salmon

light salmon

LightSalmon

orange

dark orange

DarkOrange

coral

light coral

LightCoral

tomato

orange red

OrangeRed

red

hot pink

HotPink

deep pink

DeepPink

pink

light pink

LightPink

pale violet red

PaleVioletRed

maroon

medium violet red

MediumVioletRed

violet red

VioletRed

magenta

violet

plum

orchid

medium orchid

MediumOrchid

dark orchid

DarkOrchid

dark violet

DarkViolet

blue violet

BlueViolet

purple

medium purple

MediumPurple

thistle

snow1

snow2

snow3

snow4

seashell1

seashell2

seashell3

seashell4

AntiqueWhite1

AntiqueWhite2

AntiqueWhite3

AntiqueWhite4

bisque1

bisque2

bisque3

bisque4

PeachPuff1

PeachPuff2

PeachPuff3

PeachPuff4

NavajoWhite1

NavajoWhite2

NavajoWhite3

NavajoWhite4

LemonChiffon1

LemonChiffon2

LemonChiffon3

LemonChiffon4

cornsilk1

cornsilk2

cornsilk3

cornsilk4

ivory1

ivory2

ivory3

ivory4

honeydew1

honeydew2

honeydew3

honeydew4

LavenderBlush1

LavenderBlush2

LavenderBlush3

LavenderBlush4

MistyRose1

MistyRose2

MistyRose3

MistyRose4

azure1

azure2

azure3

azure4

SlateBlue1

SlateBlue2

SlateBlue3

SlateBlue4

RoyalBlue1

RoyalBlue2

RoyalBlue3

RoyalBlue4

blue1

blue2

blue3

blue4

DodgerBlue1

DodgerBlue2

DodgerBlue3

DodgerBlue4

SteelBlue1

SteelBlue2

SteelBlue3

SteelBlue4

DeepSkyBlue1

DeepSkyBlue2

DeepSkyBlue3

DeepSkyBlue4

SkyBlue1

SkyBlue2

SkyBlue3

SkyBlue4

LightSkyBlue1

LightSkyBlue2

LightSkyBlue3

LightSkyBlue4

SlateGray1

SlateGray2

SlateGray3

SlateGray4

LightSteelBlue1

LightSteelBlue2

LightSteelBlue3

LightSteelBlue4

LightBlue1

LightBlue2

LightBlue3

LightBlue4

LightCyan1

LightCyan2

LightCyan3

LightCyan4

PaleTurquoise1

PaleTurquoise2

PaleTurquoise3

PaleTurquoise4

CadetBlue1

CadetBlue2

CadetBlue3

CadetBlue4

turquoise1

turquoise2

turquoise3

turquoise4

cyan1

cyan2

cyan3

cyan4

DarkSlateGray1

DarkSlateGray2

DarkSlateGray3

DarkSlateGray4

aquamarine1

aquamarine2

aquamarine3

aquamarine4

DarkSeaGreen1

DarkSeaGreen2

DarkSeaGreen3

DarkSeaGreen4

SeaGreen1

SeaGreen2

SeaGreen3

SeaGreen4

PaleGreen1

PaleGreen2

PaleGreen3

PaleGreen4

SpringGreen1

SpringGreen2

SpringGreen3

SpringGreen4

green1

green2

green3

green4

chartreuse1

chartreuse2

chartreuse3

chartreuse4

OliveDrab1

OliveDrab2

OliveDrab3

OliveDrab4

DarkOliveGreen1

DarkOliveGreen2

DarkOliveGreen3

DarkOliveGreen4

khaki1

khaki2

khaki3

khaki4

LightGoldenrod1

LightGoldenrod2

LightGoldenrod3

LightGoldenrod4

LightYellow1

LightYellow2

LightYellow3

LightYellow4

yellow1

yellow2

yellow3

yellow4

gold1

gold2

gold3

gold4

goldenrod1

goldenrod2

goldenrod3

goldenrod4

DarkGoldenrod1

DarkGoldenrod2

DarkGoldenrod3

DarkGoldenrod4

RosyBrown1

RosyBrown2

RosyBrown3

RosyBrown4

IndianRed1

IndianRed2

IndianRed3

IndianRed4

sienna1

sienna2

sienna3

sienna4

burlywood1

burlywood2

burlywood3

burlywood4

wheat1

wheat2

wheat3

wheat4

tan1

tan2

tan3

tan4

chocolate1

chocolate2

chocolate3

chocolate4

firebrick1

firebrick2

firebrick3

firebrick4

brown1

brown2

brown3

brown4

salmon1

salmon2

salmon3

salmon4

LightSalmon1

LightSalmon2

LightSalmon3

LightSalmon4

orange1

orange2

orange3

orange4

DarkOrange1

DarkOrange2

DarkOrange3

DarkOrange4

coral1

coral2

coral3

coral4

tomato1

tomato2

tomato3

tomato4

OrangeRed1

OrangeRed2

OrangeRed3

OrangeRed4

red1

red2

red3

red4

DeepPink1

DeepPink2

DeepPink3

DeepPink4

HotPink1

HotPink2

HotPink3

HotPink4

pink1

pink2

pink3

pink4

LightPink1

LightPink2

LightPink3

LightPink4

PaleVioletRed1

PaleVioletRed2

PaleVioletRed3

PaleVioletRed4

maroon1

maroon2

maroon3

maroon4

VioletRed1

VioletRed2

VioletRed3

VioletRed4

magenta1

magenta2

magenta3

magenta4

orchid1

orchid2

orchid3

orchid4

plum1

plum2

plum3

plum4

MediumOrchid1

MediumOrchid2

MediumOrchid3

MediumOrchid4

DarkOrchid1

DarkOrchid2

DarkOrchid3

DarkOrchid4

purple1

purple2

purple3

purple4

MediumPurple1

MediumPurple2

MediumPurple3

MediumPurple4

thistle1

thistle2

thistle3

thistle4

gray0

grey0

gray1

grey1

gray2

grey2

gray3

grey3

gray4

grey4

gray5

grey5

gray6

grey6

gray7

grey7

gray8

grey8

gray9

grey9

gray10

grey10

gray11

grey11

gray12

grey12

gray13

grey13

gray14

grey14

gray15

grey15

gray16

grey16

gray17

grey17

gray18

grey18

gray19

grey19

gray20

grey20

gray21

grey21

gray22

grey22

gray23

grey23

gray24

grey24

gray25

grey25

gray26

grey26

gray27

grey27

gray28

grey28

gray29

grey29

gray30

grey30

gray31

grey31

gray32

grey32

gray33

grey33

gray34

grey34

gray35

grey35

gray36

grey36

gray37

grey37

gray38

grey38

gray39

grey39

gray40

grey40

gray41

grey41

gray42

grey42

gray43

grey43

gray44

grey44

gray45

grey45

gray46

grey46

gray47

grey47

gray48

grey48

gray49

grey49

gray50

grey50

gray51

grey51

gray52

grey52

gray53

grey53

gray54

grey54

gray55

grey55

gray56

grey56

gray57

grey57

gray58

grey58

gray59

grey59

gray60

grey60

gray61

grey61

gray62

grey62

gray63

grey63

gray64

grey64

gray65

grey65

gray66

grey66

gray67

grey67

gray68

grey68

gray69

grey69

gray70

grey70

gray71

grey71

gray72

grey72

gray73

grey73

gray74

grey74

gray75

grey75

gray76

grey76

gray77

grey77

gray78

grey78

gray79

grey79

gray80

grey80

gray81

grey81

gray82

grey82

gray83

grey83

gray84

grey84

gray85

grey85

gray86

grey86

gray87

grey87

gray88

grey88

gray89

grey89

gray90

grey90

gray91

grey91

gray92

grey92

gray93

grey93

gray94

grey94

gray95

grey95

gray96

grey96

gray97

grey97

gray98

grey98

gray99

grey99

gray100

grey100

dark grey

DarkGrey

dark gray

DarkGray

dark blue

DarkBlue

dark cyan

DarkCyan

dark magenta

DarkMagenta

dark red

DarkRed

light green

LightGreen

CSU Green

CSU Green 1

CSU Green 2

CSU Green 3

CSU Green 4

CSU Gold

CSU Gold 1

CSU Gold 2

CSU Gold 3

CSU Gold 4

Light CSU Gold

Symbols

Within a span or paragraph, symbols can be added using these references. Uniciode values are given after the symbol description.

{\nbsp} Non-break space (00A0)

{\degree} ° Degree sign (00B0)

{\pm} ± Plus/Minus (00B1)

{\cdot} · Center dot (00B7)

{\times} × Times (00D7)

{\div} ÷ Divided by (00F7)

{\fnof} ƒ Function "f" (0192)

{\Gamma} Γ Gamma (0393)

{\Delta} Δ Delta (0394)

{\Theta} Θ Theta (0398)

{\Lamda} Λ Lamda (039B)

{\Xi} Ξ Xi (039E)

{\Pi} Π Pi (03A0)

{\Sigma} Σ Sigma (03A3)

{\Upsilon} Υ Upsilon (03A5)

{\Phi} Φ Phi (03A6)

{\Psi} Ψ Psi (03A8)

{\Omega} Ω Omega (03A9)

{\alpha} α alpha (03B1)

{\beta} β beta (03B2)

{\gamma} γ gamma (\u03B3)

{\delta} δ delta (03B4)

{\varepsilon} ε variant epsilon (03B5)

{\zeta} ζ zeta (03B6)

{\eta} η eta (03B7)

{\theta} θ theta (03B8)

{\iota} ι iota (03B9)

{\kappa} κ kappa (03BA)

{\lamda} λ lamda (03BB)

{\mu} μ mu (03BC)

{\nu} ν nu (03BD)

{\xi} ξ xi (03BE)

{\omicron} ο omicron (03BF)

{\pi} π pi (03C0)

{\rho} ρ rho (03C1)

{\varsigma} ς variant sigma (03C2)

{\sigma} σ sigma (03C3)

{\tau} τ tau (03C4)

{\upsilon} υ upsilon (03C5)

{\varphi} φ variant phi (03C6)

{\chi} χ chi (03C7)

{\psi} ψ psi (03C8)

{\omega} ω omega (03C9)

{\vartheta} ϑ variant theta (03D1)

{\phi} ϕ phi (03D5)

{\varpi} ϖ variant pi (03D6)

{\varkappa} ϰ variant kappa (03F0)

{\varrho} ϱ variant rho (03F1)

{\epsilon} ϵ epsilon (03F5)

{\minus} − minus (2212)

{\textendash} – en-dash (2013)

{\textemdash} — em-dash (2014)

{\textquoteleft} ‘ left quote (2018)

{\textquoteright} ’ right quote (2019)

{\textquotedblleft} “ left double-quote (201C)

{\textquotedblright} ” right double-quote (201D)

{\bullet} • bullet (2022)

{\prime} ′ prime (2032)

{\dprime} ″ double prime (2033)

{\tprime} ‴ triple prime (2034)

{\qprime} ⁗ quadruple prime (2057)

{\e} ⅇ double-struck e (2147)

{\i} ⅈ double-struck i (2148)

{\leftarrow} ← left arrow (2190)

{\uparrow} ↑ up arrow (2191)

{\rightarrow} → right arrow (2192)

{\downarrow} ↓ down arrow (2193)

{\leftrightarrow} ↔ left/right arrow (2194)

{\updownarrow} ↕ up/down arrow (2195)

{\circ} ∘ open circle (2218)

{\varpropto} ∝ proportional to (221D)

{\infty} ∞ infinity (221E)

{\angle} ∠ angle (2220)

{\measuredangle} ∡ angle measure (2221)

{\cap} ∩ set intersection (2229)

{\cup} ∪ set union (222A)

{\int} ∫ integral (222B)

{\simeq} ≃ similar or equal to (2243)

{\approx} ≈ approximately (2248)

{\neq} ≠ not equal (2260)

{\leq} ≤ less than or equal (2264)

{\geq} ≥ greater than or eq. (2265)

{\leqq} ≦ less than or eq. (2266)

{\geqq} ≧ greater than or eq. (2267)

{\lneqq} ≨ less than, not eq. (2268)

{\gneqq} ≩ greater than, not eq. (2269)

{\ll} ≪ much less than (226A)

{\gg} ≫ much greater than (226B)

{\between} ≬ between (226C)

{\nless} ≮ not less than (226E)

{\ngtr} ≯ not greater than (226F)

{\nleq} ≰ not less than or eq. (2270)

{\ngeq} ≱ not greater than or eq. (2271)

{\lesssim} ≲ less than or similar to (2272)

{\gtrsim} ≳ greater than or sim. to (2273)

{\lessgtr} ≶ less than or greater than (2276)

{\gtrless} ≷ greater than or less than (2277)

{\prec} ≺ precedes (227A)

{\succ} ≻ succeeds (227B)

{\preccurlyeq} ≼ precedes or equal (227C)

{\succcurlyeq} ≽ succeeds or equal (227D)

{\precsim} ≾ precedes or sim. (227E)

{\succsim} ≿ succeeds or sim. (\u227F)

{\nprec} ⊀ does not precede (2280)

{\nsucc} ⊁ does not succeed (2281)

{\lessdot} ⋖ less than with dot (22D6)

{\gtrdot} ⋗ greater than with dot (22D7)

{\lesseqgtr} ⋚ less than, eq., gtr. than (22DA)

{\gtreqless} ⋛ gtr. than, eq., less than (22DB)

{\curlyeqprec} ⋞ equal or precedes (22DE)

{\curlyeqsucc} ⋟ equal or succeeds (22DF)

{\npreceq} ⋠ does not prec. or eq. (22E0)

{\nsucceq} ⋡ does not succ. or eq. (22E1)

{\lnsim} ⋦ less than, not sim. (22E6)

{\gnsim} ⋧ greater than, not sim. (22E7)

{\precnsim} ⋨ precedes, not sim. (22E8)

{\succnsim} ⋩ succeeds, not sim. (22E9)

{\cdots} ⋯ center dots (22EF)

{\smallfrown} ⌢ frown (2322)

{\smallsmile} ⌣ smile (2323)

{\langle} 〈 left angle bracket (2329)

{\rangle} 〉 right angle bracket (232A)

{\blacksquare} ■ black square (25A0)

{\blacktriangle} ▲ black up triangle (25B2)

{\triangle} △ white up triangle (25B3)

{\blacktriangleright} ► black right triangle (25BA)

{\blacktriangledown} ▼ black down triangle (25BC)

{\blacktriangleleft} ◄ black left triangle (25C4)

{\spadesuit} ♠ spade suit (2660)

{\clubsuit} ♣ club suit (2663)

{\heartsuit} ♥ heart suit (2665)

{\diamondsuit} ♦ diamond suit (2666)

{\checkmark} ✓ check mark (2713)

{\diagup} ⟋ diagonal up (27CB)

{\diagdown} ⟍ diagonal down (27CD)

{\Longleftarrow} ⟸ long left arrow (27F8)

{\Longrightarrow} ⟹ long right arrow (27F9)

{\leqslant} ⩽ less than or equal, slant (2A7D)

{\geqslant} ⩾ greater than or eq., slant (2A7E)

{\lessapprox} ⪅ less than or approx. (2A85)

{\gtrapprox} ⪆ greater than or approx. (2A86)

{\lneq} ⪇ less than or not equal (2A87)

{\gneq} ⪈ greater than or not equal (2A88)

{\lnapprox} ⪉ less than or not approx. (2A89)

{\gnapprox} ⪊ greater than or not approx. (2A8A)

{\lesseqqgtr} ⪋ less than, eq., greater than (2A8B)

{\gtreqqless} ⪌ greater than, eq., less than (2A8C)

{\eqslantless} ⪕ equal or less than, slant (2A95)

{\eqslantgtr} ⪖ eq.or greater than, slant (2A96)

{\lll} ⪡ much less than (2AA1)

{\ggg} ⪢ much greater than (2AA2)

{\preceq} ⪯ precedes or equal (2AAF)

{\succeq} ⪰ succeeds or equal (2AB0)

{\precneqq} ⪵ precedes, not equal (2AB5)

{\succneqq} ⪶ succeeds, not equal (2AB6)

{\precapprox} ⪷ precedes, approx. (2AB7)

{\succapprox} ⪸ succeeds, approx. (2AB8)

{\precnapprox} ⪹ precedes, not approx. (2AB9)

{\succnapprox} ⪺ succeeds, not approx. (2ABA)

{\pitchfork} ⫛ pitchfork (2ADB)