spintent

SPANISH PARSE INTENTS

v0.99 2026-09-17*

©2026 by Pablo González L†

CTAN: https://www.ctan.org/pkg/spintent
https://github.com/pablgonz/spintent

Resumen

El paquete spintent proporciona una serie de utilidades para docentes de educación primaria y secundaria que necesiten crean documentos PDF *accesibles* (*tagged* PDF) en español usando LuaLaTeX.

Índice

1 Descripción 1
1.1 Carga del paquete 1
1.2 El paquete babel 2
1.3 Fuentes tipográficas 2
1.4 El paquete hyperref 2
2 El comando \setspintent 3
3 The big four 3
3.1 El comando \spnum 3
3.1.1 Llaves para \spnum 3
3.2 El comando \spunit 4
3.2.1 Llaves para \spunit 4
3.2.2 Argumentos para \spunit 4
3.2.3 El comando \spnewunit 6
3.2.4 El comando \spunitalias 6
3.3 El comando \spmoney 6
3.3.1 Llaves para \spmoney 7
3.3.2 Monedas y alias para \spmoney . 7
3.4 El comando \spshort 7
3.4.1 Llaves para \spshort 9
3.4.2 Argumentos para \spshort . . . 9
4 Utilidades generales 11
5 Utilidades para símbolos 11
6 Utilidades para números 13
7 Utilidades para geometría 17
8 Referencias 25
9 Change history 26
10 Índice de la Documentación 27
11 Implementation 29
12 Índice de la Implementación 145

Motivación y agradecimientos

Desde que desarrollé scontents[7] en 2019 y, más tarde, enumext[8] en 2024, me he ido interesando cada vez más por la accesibilidad en los documentos PDF. Al principio era un tema prácticamente desconocido para mí, pero a medida que fui aprendiendo sobre él comprendí la importancia que tiene, especialmente en el ámbito educativo.

El equipo del proyecto de LaTeX[19] ha realizado un trabajo extraordinario en este campo. Hoy es posible generar documentos etiquetados, validarlos con herramientas como veraPDF[20] o inspeccionar su estructura con ngpdf[21]. Sin embargo, siempre me quedó una duda: cuando un estudiante utiliza NVDA[11] junto con MathCAT[12], ¿realmente escucha lo que el o la docente quiso expresar?

Basta con generar un documento accesible siguiendo las recomendaciones de The LaTeX Tagged PDF Project y escucharlo con NVDA/MathCAT para darse cuenta de que, en muchos casos, la *verbalización* no coincide con lo que uno espera. Esto no resulta extraño: las reglas del español, y en particular el lenguaje matemático, dependen en gran medida del contexto en que se utilizan.

El objetivo de spintent nace precisamente de esta necesidad: proporcionar herramientas para que el o la docente pueda indicar explícitamente el contexto de determinadas expresiones matemáticas desde el propio código fuente de LaTeX, de modo que el documento PDF resultante no solo sea *accesible*, sino que también produzca una *verbalización más natural* y adecuada para el estudiante al utilizar NVDA/MathCAT

El nuevo estándar PDF 2.0[14, 18, 16], las especificaciones de WTPDF[15] y la guía más reciente de la PDF Association, «Best Practice Guide: Math in PDF»[17], coinciden en que el camino hacia una accesibilidad matemática de calidad pasa por el uso de MathML. En ese contexto, LuaLaTeX ofrece una plataforma especialmente adecuada para incorporar esta tecnología.

*Este archivo describe la documentación para la versión v0.99 revisada por última vez el 2026-09-17.

†E-mail: «pablgonz@educarchile.cl».

Licencia y requerimientos

El paquete spintent carga y requiere el paquete unicode-math[1] y que se ejecute bajo LuaLaTeX por lo que es necesario contar con una distribución moderna de TeX como TeX Live 2026. Ha sido probado con las clases estándar proporcionadas por LaTeX: book, report, article y letter en tamaños 10pt, 11pt y 12pt.

Se otorga permiso para copiar, distribuir y/o modificar este paquete bajo los términos de LaTeX Project Public License (LPPL), versión 1.3 o posterior (https://www.latex-project.org/lppl.txt). El paquete tiene la condición de mantenido al estilo *«A Work in Progress*[1]*»*.

El requerimiento mínimo es LaTeX release 2026-11-01.

Este paquete solo se puede utilizar solo con LuaLaTeX y *tagged* PDF activo. Está presente en TeX Live y MiKTeX, para instalarlo, utilice el gestor de paquetes de su distribución. Si desea una instalación manual, descargue spintent.zip y descomprímalo, luego ejecute luatex spintent.ins, mueva todos los archivos a las ubicaciones correspondientes y luego ejecute mktexlsr. Para generar la documentación, ejecute arara spintent.dtx.

```
spintent.sty  ⇒  TDS:tex/lualatex/spintent/
spintent.lua  ⇒  TDS:tex/lualatex/spintent/
spintent.pdf  ⇒  TDS:doc/lualatex/spintent/
README.md     ⇒  TDS:doc/lualatex/spintent/
spintent.dtx  ⇒  TDS:source/lualatex/spintent/
spintent.ins  ⇒  TDS:source/lualatex/spintent/
```

1 Descripción

El nombre spintent puede entenderse como *«Spanish Parse Intent»* o también como *«School Parse Intent»*. Ambas interpretaciones reflejan el objetivo del paquete y el público para el que fue diseñado: docentes de educación primaria y secundaria.
Está pensado para crear *«apuntes»*, *«hojas de ejercicios»*, *«clases escritas»* enfocadas en el estudiante. Solo un docente sabe cual es el nivel de aprendizaje del estudiante. El paquete intenta sobrescribir muchas de las reglas dictadas por la heurística de NVDA/MathCAT teniendo esto presente.

El paquete spintent usa y abusa del atributo :intent de MathML por medio del comando \MathMLintent definido en latex-lab-mathintent[30], un encapsulado de \luamml_attribute:een proveída por luamml[2]. !Gracias Marcel¡

1.1 Carga del paquete

El paquete spintent trabaja solo con LuaLaTeX y *tagged* PDF activo usando tagging=on dentro del comando \DocumentMetadata al inicio del archivo. Las llaves (⟨*keys*⟩) aceptadas por \DocumentMetadata y en general para el código de *tagged* PDF están documentadas en los paquetes y tagpdf[29] documentmetadata-support[30], es deber del usuario revisar la documentación asociada a ellos.

```
\DocumentMetadata
  {
    lang=es-CL,
    pdfversion=2.0,
    pdfstandard=ua-2,
    tagging=on,
    tagging-setup={math/setup={mathml-SE}},
  }
\documentclass{article}
\usepackage[spanish]{babel}
\usepackage{spintent}
```

1.2 El paquete babel

El paquete babel-spanish[10] está marcado como «parcialmente compatible» para *tagged* PDF, pero, es absolutamente necesario para la creación de documentos en español. En caso de encontrar problemas de compatibilidad con otros paquetes se puede desactivar usando:

```
\usepackage[spanish,shorthands=off]{babel}
```

Muchas de las «shorthands» proveidas por babel-spanish hoy no son necesarias, más allá de \sptext para las abreviaturas y ordinales el resto son solo atajos de teclas para escritura que hoy en día no se ocupan. El paquete spintent provee el comando \spshort (§3.4) que cubre estos casos, de esta manera podemos cargar babel[9] con todas sus funcionalidades sin problemas.

[1] *A Work in Progress* es un álbum en vivo de Neil Peart (∗ 1952–† 2020), baterista y principal letrista de Rush. El nombre no pretende formar parte de la terminología de la LPPL; simplemente describe el estado permanente de este paquete mientras sigo aprendiendo sobre *accesibilidad* en PDF.

1.3 Fuentes tipográficas

Como spintent es un paquete LuaLaTeX debemos distinguir entre las fuentes tipográficas `OpenType/TrueType` versus las fuentes de `8bit` estándar usadas por LaTeX para el *modo texto* y el *modo matemático*. Para evitar problemas con *tagged* `PDF` preferiremos no combinar fuentes `OpenType/TrueType` con fuentes `8bit` y trabajaremos solo con fuentes `OpenType/TrueType` que tengan soporte para matemáticas.

La carga de las fuentes tipográficas para LuaLaTeX se maneja por medio del paquete `fontspec`[3], si no se espcifica ninguna se usarán las fuentes `Latin Modern` para el *modo texto* y `Latin Modern Math` para el *modo matemático*.

El tipo de fuente tipográfica escogida es preferencia de cada usuario, en lo personal utilizo las fuente `Libertinus` para el *modo texto* por medio del paquete `libertinus-otf`[4] y la fuente `Erewhon-Math` para el modo matemático cargada por el paquete `fourier-otf`[5]. Adicionalmente cargo la fuente `Source Code Pro` para el texto tipo «`máquina de escribir`» por medio del paquete `sourcecodepro`[6].

```
% \DocumentMetadata{...}
\documentclass{article}
\usepackage[spanish]{babel}
\usepackage[osf,nomath,mono=false]{libertinus-otf}
\usepackage[osf,semibold]{sourcecodepro}
\usepackage[no-text]{fourier-otf}
\usepackage{spintent}
```

1.4 El paquete `hyperref`

El paquete `hyperref`[23] es *«casi obligatorio»* cuando se crean documentos *tagged* `PDF`, se encarga de las referencias cruzadas `\label` y `\ref`, las notas al pie `\footnote`, los hipervínculos `\href` y muchos otros elementos como marcadores, índices y metadatos.

```
\DocumentMetadata
  {
    lang=es-CL,
    pdfversion=2.0,
    pdfstandard=ua-2,
    tagging=on,
    tagging-setup={math/setup={mathml-SE}},
  }
\documentclass{article}
\usepackage[spanish]{babel}
\usepackage[osf,nomath,mono=false]{libertinus-otf}
\usepackage[osf,semibold]{sourcecodepro}
\usepackage[no-text]{fourier-otf}
\setmathfont{Erewhon-Math}[range={cal,bfcal},StylisticSet=1]
\usepackage{spintent, hyperref}
\hypersetup
  {
    colorlinks         = true,
    pdfstartview       = FitH,
    pdfdisplaydoctitle = true,
    pdftitle           = {Test para el paquete spintent},
    pdfinfo            =
      {
        Title   = {Test},
        Subject = {spintent},
      },
  }
```

2 El comando \setspintent

\setspintent `\setspintent{⟨comando⟩}{⟨key-uno = valor, key-dos = valor, key-tres = valor, ...⟩}`

El comando \setspintent establece las llaves (⟨keys⟩) de forma global para los comandos provistos por spintent. Puede utilizarse tanto en el preámbulo como en el cuerpo del documento tantas veces como se desee.

Las llaves (⟨keys⟩) definidas en el *argumento opcional* [⟨key = val⟩] de los comandos tienen prioridad sobre las establecidas por este, anulando las configuraciones globales de \setspintent.

Muchas de las utilidades que provee el paquete spintent implementan el sistema [⟨key = val⟩] y están pensadas para ser utilizadas en la generación de *«hojas de ejercicios»* o *«clases escritas»*. Se recomienda usar \setspintent y establecerlas de manera global para cada documento, para mantener la coherencia en la verbalización de NVDA/MathCAT a lo largo del mismo.

El sistema [⟨key = val⟩] utilizado por spintent se implementa usando el módulo l3keys de expl3[24]. Por diseño, las llaves que reciben un *valor* después del signo '=' NO aceptan *valores vacíos* y es obligatorio pasar uno valido; las llaves documentadas como ⟨valor prohibido⟩ no reciben *ningun valor*. En caso de omitir un *valor* valido o pasar un *valor vacío* a una llave que acepte *valor* o pasar un *valor* a una llave del tipo ⟨valor prohibido⟩ se devolverá un "error" y se detendrá la compilación del documento.

3 The big four

El paquete spintent provee cuatro grandes comandos *(«the big four[2]»)*: \spnum para el manejo de números, \spunit para el manejo de unidades, \spmoney para el manejo de dinero y \spshort para el manejo de abreviaturas, acrónimos y números ordinales.

Tambien se incluyen las utilidades \spnewunit y \spunitalias asociadas a al comando \spunit.

3.1 El comando \spnum

\spnum

`\spnum{⟨número⟩}`	`\spnum[⟨key = val⟩]{⟨número,número;número⟩}`
`\spnum[⟨key = val⟩]{⟨+número⟩}`	`\spnum[⟨key = val⟩]{⟨número.número;número⟩}`
`\spnum[⟨key = val⟩]{⟨-número⟩}`	`\spnum[⟨key = val⟩]{⟨número;número⟩}`
`\spnum[⟨key = val⟩]{⟨número,número⟩}`	`\spnum[⟨key = val⟩]{⟨número e exponente⟩}`
`\spnum[⟨key = val⟩]{⟨número.número⟩}`	`\spnum[⟨key = val⟩]{⟨número E exponente⟩}`

El comando \spnum trabaja en *modo matemático* y recibe el *argumento obligatorio* {⟨número⟩}, donde la *«parte entera»* puede iniciar con los signos '+' o '−', la *«parte decimal finita»* **debe** iniciar con una *coma* ',' o con un *punto* '.', y la *«parte decimal infinita (periódica)»* **debe** iniciar con un *punto y coma* ';'.

Si no se proporciona el *argumento opcional* [⟨key = val⟩] y se ejecuta `$\spnum{+23}$` se imprimirá '+23' en la salida PDF y NVDA/MathCAT verbalizará *«más veintitrés»*, si se ejecuta `$\spnum{3,14}$` se imprimirá '3,14' en la salida PDF y NVDA/MathCAT verbalizará *«tres coma catorce»*, si se ejecuta `$\spnum{3,1;4}$` se imprimirá '$3{,}1\overline{4}$' en la salida PDF y NVDA/MathCAT verbalizará *«tres coma uno con cuatro periódico»*.

Además, el *argumento obligatorio* {⟨número⟩} admite al final de este un sufijo de *«notación científica»*. Este se indica mediante las letras 'e' o 'E', acompañadas de los signos opcionales '+' o '−' seguido por los dígitos del exponente. Por ejemplo `$\spnum{3,14e3}$` imprimirá '$3{,}14 \cdot 10^3$' en la salida PDF y NVDA/MathCAT verbalizará *«tres coma catorce por diez al cubo»*.

La *coma* ',' o el *punto* '.' son *separadores decimales* y NO de millares, por lo tanto solo pueden aparecer una «única vez» dentro del {⟨argumento⟩}. El {⟨argumento⟩} puede contener espacios matemáticos validos, pero, NO puede contener comandos con o sin argumentos como \textstyle, \mathrm{#1} o \mathbf{#1} por ejemplo. Cualquier entrada fuera de esta sintaxis devolverá un "error" deteniendo la compilación del documento.

3.1.1 Llaves para \spnum

cifras-sep = {⟨true | false⟩} default: *true*

Establece el *espaciado* utilizado para agrupar los dígitos en bloques de tres, tanto en la parte entera como en la parte decimal finita. Si se establece en true se siguen las normas de la RAE y se utiliza un *espacio fino* '\,' entre las cifras; si se establece en false, se respetan los espacios matemáticos introducidos en el argumento. Por ejemplo `$\spnum{100000}$` imprimirá '100 000' en la salida PDF y `$\spnum[cifras-sep=false]{10\,00\,00}$` imprimirá '10 00 00' en la salida PDF.

read-sign = {⟨terse | clear⟩} default: *terse*

Establece la *forma* en la que NVDA/MathCAT verbalizará los signos '+' o '−'. Si se establece en terse, NVDA/MathCAT verbalizará *«más»* o *«menos»* **antes** de leer el número; si se establece en clear, NVDA/MathCAT verbalizará *«positivo»* o *«negativo»* **después** de leer el número. Por ejemplo `$\spnum{-23}$` imprimirá '−23' en la salida PDF y NVDA/MathCAT verbalizará *«menos veintitrés»* y `$\spnum[read-sign=clear]{-23}$` imprimirá '−23' en la salida PDF y NVDA/MathCAT verbalizará *«veintitrés negativo»*.

read-period-sep = {⟨coma | punto⟩} default: *coma*

[2] Los *«cuatro grandes»* del Thrash Metal: Metallica, Megadeth, Slayer y Anthrax, que me acompañan mientras desarrollo spintent.

Establece la *forma* en la que NVDA/MathCAT verbalizará e imprimirá el signo de separación '**;**' de la *«parte decimal infinita (periódica)»* cuando no se tiene la *«parte decimal finita»*. Si se establece en coma, se imprimirá '**,**' en la salida PDF y NVDA/MathCAT verbalizará *«coma»*; si se establece en punto, se imprimirá '**.**' en la salida PDF y NVDA/MathCAT verbalizará *«punto»*. Por ejemplo $\spnum{3;4}$ imprimirá '$3{,}\overline{4}$' en la salida PDF y NVDA/MathCAT verbalizará *«tres coma cuatro periódico»* y $\spnum[read-period-sep=punto]{3;4}$ imprimirá '$3.\overline{4}$' en la salida PDF y NVDA/MathCAT verbalizará *«tres punto cuatro periódico»*

Esta llave es necesaria SOLO cuando se utilizan *«decimales periódicos puros»*, es decir, cuando no se ha incluido en el {⟨*argumento*⟩} una *coma* '**,**' o un *punto* '**.**' antes del signo de separación '**;**' periódica.

notation-sym = {⟨*times* | *cdot*⟩} default: *cdot*

Establece el *símbolo* que se utilizará para imprimir la *«notación científica»* al usar '**e**' o '**E**'. Si se establece en times imprimirá '**×**' en la salida PDF y NVDA/MathCAT verbalizará *«por»*; si se establece en cdot imprimirá '**·**' en la salida PDF y NVDA/MathCAT verbalizará *«por»*. Por ejemplo $\spnum{3,14e3}$ imprimirá '$3{,}14 \cdot 10^3$' en la salida PDF y NVDA/MathCAT verbalizará *«tres coma catorce por diez al cubo»* y $\spnum[notation-sym=times]{3,14e3}$ imprimirá '$3{,}14 \times 10^3$' en la salida PDF y NVDA/MathCAT verbalizará *«tres coma catorce por diez al cubo»*.

Decir *«notación científica»* aquí es un poco exagerado, en realidad es *«multiplicación por potenicas de* 10*»*, no se verfica si la entrada cumple o no con la condición de estár escrita en *«notación científica»*.

3.2 El comando \spunit

\spunit

\spunit{⟨*unidad*⟩}
\spunit[⟨*key* = *val*⟩]{⟨*unidad*⟩}
\spunit[⟨*key* = *val*⟩]{⟨*número unidad*⟩}
\spunit[⟨*key* = *val*⟩]{⟨*número unidad**unidad*⟩}
\spunit[⟨*key* = *val*⟩]{⟨*número unidad**unidad*/*unidad*⟩}
\spunit[⟨*key* = *val*⟩]{⟨*número unidad**unidad*/*unidad**unidad*⟩}

El comando \spunit trabaja en *modo matemático* y recibe el *argumento obligatorio* {⟨*unidad*⟩}, el cual puede contener una *«parte numérica»* opcional la cual se procesa internamente por el comando \spnum al inicio del {⟨*argumento*⟩} seguido de una {⟨*unidad*⟩} o un {⟨*alias*⟩} valido descritos en §3.2.2 o una {⟨*unidad*⟩} creada por el comando \spnewunit (§3.2.3) o un {⟨*alias*⟩} creado por el comando \spunitalias (§3.2.4).

Si no se proporciona el *argumento opcional* [⟨*key* = *val*⟩] y se ejecuta $\spunit{23 m}$ se imprimirá '23 m' en la salida PDF y NVDA/MathCAT verbalizará *«veintitrés metros»*.

La multiplicación entre las {⟨*unidades*⟩} se realiza mediante un *asterisco* '*****' y las potencias de estas se indican con un *circunflejo seguido del exponente entre llaves* '^{⟨*exponente*⟩}'. Por ejemplo si se ejecuta $\spunit{23 m*s^{2}}$ se imprimirá '$23\,\mathrm{m\,s^2}$' en la salida PDF y NVDA/MathCAT verbalizará *«veintitrés metros segundos al cuadrado»*.

Se pueden usar *fracciones lineales* con las {⟨*unidades*⟩} utilizando una *«única barra»* '**/**' para la separación entre las {⟨*unidades*⟩} del numerador y el denominador, teniendo presente que el denominador NO puede contener números de ningún tipo. Por ejemplo si se ejecuta $\spunit{23 m/s^{2}}$ se imprimirá '$23\,\mathrm{m/s^2}$' en la salida PDF y NVDA/MathCAT verbalizará *«veintitrés metros por segundos al cuadrado»*.

Solo imprimiremos *fracciones lineales*, si se requiere otra forma fraccionaria para explicaciones y otros usos se debe usar el comando por separado. El {⟨*argumento*⟩} puede contener espacios matemáticos validos, pero, NO puede contener comandos con o sin argumentos como \textstyle, \mathrm{#1} o \mathbf{#1} por ejemplo, NO puede contener más de una *«única barra»* '**/**' y NO puede contener números en el denominador. Cualquier entrada fuera de esta sintaxis devolverá un "error" deteniendo la compilación del documento.

3.2.1 Llaves para \spunit

El comando \spunit posee las ⟨*meta-keys*⟩ cifras-sep, read-period-sep y notation-sym pasadas al comando \spnum (§3.1.1).

print-cdot = {⟨*true* | *false*⟩} default: *false*

Establece si se *imprime* \cdot '**·**' o un *espacio fino* '\,' para la multiplicación de {⟨*unidades*⟩}. Si se establece en true, se imprimirá '**·**' entre las {⟨*unidades*⟩} en la salida PDF; si se establece en false se imprimirá un *espacio fino* '\,' entre las {⟨*unidades*⟩} en la salida PDF.

read-cdot = {⟨*true* | *false*⟩} default: *false*

Establece la *forma* en la que NVDA/MathCAT verbalizará la multiplicación de {⟨*unidades*⟩}. Si se establece en true, NVDA/MathCAT verbalizará «por» entre las {⟨*unidades*⟩}; si se establece en false, NVDA/MathCAT no verbalizará nada entre las {⟨*unidades*⟩}.

read-slash = {⟨*por* | *partido-por*⟩} default: *por*

Establece la *forma* en la que NVDA/MathCAT verbalizará la *«barra»* '**/**' de separación entre el numerador y el denominador. Si se establece en por, NVDA/MathCAT verbalizará *«por»*; si se establece en partido-por, NVDA/MathCAT verbalizará *«partido por»*.

3.2.2 Argumentos para \spunit

Resumen de {⟨*unidades*⟩} y {⟨*alias*⟩} soportados por \spunit.

Entrada	Alias soportados	Nombre unidad	Salida	NVDA/MathCAT
m	metro, metros	Metro	m	*«metro»*
g	gramo, gramos	Gramo	g	*«gramo»*
s	segundo, segundos	Segundo (Tiempo)	s	*«segundo (Tiempo)»*
l	litro, l-litro	Litro (minúscula)	l	*«litro (minúscula)»*
h	hora, horas	Hora	h	*«hora»*
d	dia, dias	Día	d	*«día»*
A	amperio, ampere	Amperio	A	*«amperio»*
V	voltio, volt	Voltio	V	*«voltio»*
W	vatio, watt	Vatio	W	*«vatio»*
J	julio, joule	Julio	J	*«julio»*
N	newton	Newton	N	*«newton»*
Pa	pascal	Pascal	Pa	*«pascal»*
Hz	hercio, hertz	Hercio	Hz	*«hercio»*
Ω	ohmio, ohm, Omega	Ohmio	Ω	*«Ohmio»*
F	faradio	Faradio	F	*«Faradio»*
C	culombio	Culombio	C	*«Culombio»*
K	kelvin, Kelvin	Kelvin	K	*«Kelvin»*
°C	celsius, degC, ℃	Grados Celsius	°C	*«Grados Celsius»*
°F	fahrenheit, degF, ℉	Grados Fahrenheit	°F	*«Grados Fahrenheit»*
cal	caloria, calorias	Caloría	cal	*«Caloría»*
b	bit	Bit	b	*«Bit»*
B	byte, bytes	Byte	B	*«Byte»*
in	pulgada, pulgadas	Pulgada	in	*«Pulgada»*
ft	pie, pies	Pie	ft	*«Pie»*
yd	yarda, yardas	Yarda	yd	*«Yarda»*
mi	milla, millas	Milla	mi	*«Milla»*
lb	libra, libras	Libra	lb	*«Libra»*
oz	onza, onzas	Onza	oz	*«Onza»*
gal	galon, galones	Galón	gal	*«Galón»*
Å	angstrom	Angstrom	Å	*«Angstrom»*
µ	micro	Micro (Prefijo)	µ	*«Micro (Prefijo)»*
℧	mho	Mho / Siemens	℧	*«Mho / Siemens»*
′	minmin	Minuto (Arco)	′	*«Minuto (Arco)»*
″	segseg	Segundo (Arco)	″	*«Segundo (Arco)»*
u	u-masa	Unidad masa atómica	u	*«Unidad masa atómica»*
		Unidades directas (sin alias):		
L	-	Litro (Mayúscula)	L	*«Litro (Mayúscula)»*
ℓ	-	Litro (Cursiva)	ℓ	*«Litro (Cursiva)»*
w	-	Vatio (Minúscula)	w	*«Vatio (Minúscula)»*
°K	-	Grado Kelvin	°K	*«Grado Kelvin»*
Bq	-	Becquerel	Bq	*«Becquerel»*
Da	-	Dalton	Da	*«Dalton»*
Gy	-	Gray	Gy	*«Gray»*
S	-	Siemens	S	*«Siemens»*
Sv	-	Sievert	Sv	*«Sievert»*
T	-	Tesla	T	*«Tesla»*
Wb	-	Weber	Wb	*«Weber»*
atm	-	Atmósfera	atm	*«Atmósfera»*
au	-	Unidad astronómica	au	*«Unidad astronómica»*
bar	-	Bar	bar	*«Bar»*
cd	-	Candela	cd	*«Candela»*
dB	-	Decibelio	dB	*«Decibelio»*
dyn	-	Dina	dyn	*«Dina»*
eV	-	Electronvoltio	eV	*«Electronvoltio»*
erg	-	Ergio	erg	*«Ergio»*
ha	-	Hectárea	ha	*«Hectárea»*
kat	-	Katal	kat	*«Katal»*
kg	-	Kilogramo	kg	*«Kilogramo»*
km	-	Kilómetro	km	*«Kilómetro»*
lm	-	Lumen	lm	*«Lumen»*

Continúa en la siguiente página...

Continuación de la tabla 1

Entrada	Alias soportados	Nombre unidad	Salida	MathCAT
lx	-	Lux	lx	*«Lux»*
cm	-	Centímetro	cm	*«Centímetro»*
min	-	Minuto (Tiempo)	min	*«Minuto (Tiempo)»*
mm	-	Milímetro	mm	*«Milímetro»*
mol	-	Mol	mol	*«Mol»*
pc	-	Pársec	pc	*«Pársec»*
rad	-	Radián	rad	*«Radián»*
rpm	-	Rev. por minuto	rpm	*«Rev. por minuto»*
sr	-	Estereorradián	sr	*«Estereorradián»*
t	-	Tonelada	t	*«Tonelada»*
a	-	Año / Área (are)	a	*«Año / Área (are)»*
y	-	Año (Year)	y	*«Año (Year)»*
°	-	Grado (Arco)	°	*«Grado (Arco)»*

3.2.3 El comando `\spnewunit`

`\spnewunit`

`\spnewunit`{⟨*nueva unidad*⟩}{⟨*verbalización* NVDA⟩}

El comando `\spnewunit` trabaja en *modo texto* y recibe dos *argumentos obligatorios*: {⟨*nueva unidad*⟩} que establece lo que se imprimirá en la salida PDF y {⟨*verbalización* NVDA⟩} que establece como NVDA/MathCAT *verbalizará* la {⟨*nueva unidad*⟩}.

La {⟨*nueva unidad*⟩} no debe estar registrada como {⟨*unidad*⟩} o {⟨*alias*⟩} para `\spunit` descritos en §3.2.2. Si la {⟨*nueva unidad*⟩} ya está registrada se devolverá un "error" y se detendrá la compilación del documento.

La declaración de la {⟨*nueva unidad*⟩} es «global» y se almacena en el módulo `spintent.lua` como una {⟨*unidad*⟩} valida para pasar como {⟨*argumento*⟩} a `\spunit`. Por ejemplo: `\spnewunit{px}{píxeles}` creará la {⟨*nueva unidad*⟩} '`px`' y cuando se utilice `\spunit{100 px}` se imprimirá '100 px' en la salida PDF y NVDA/MathCAT verbalizará *«cien píxeles»*.

3.2.4 El comando `\spunitalias`

`\spunitalias`

`\spunitalias`{⟨*nuevo alias*⟩}{⟨*expresión de unidades*⟩}

El comando `\spunitalias` trabaja en *modo texto* y recibe dos *argumentos obligatorios*: {⟨*nuevo alias*⟩} que establece es el nuevo {⟨*alias*⟩} de {⟨*argumento*⟩} para `\spunit` y {⟨*expresión de unidades*⟩} que puede contener multiplicación o división de {⟨*unidades*⟩} ya registradas que se imprimirán en la salida PDF.

El {⟨*nuevo alias*⟩} no debe estar registrado como {⟨*unidad*⟩} o {⟨*alias*⟩} para `\spunit` descritos en §3.2.2 o estar definido como {⟨*unidad*⟩} por medio del comando `\spnewunit`. Si el {⟨*nuevo alias*⟩} ya está registrado se devolverá un "error" y se detendrá la compilación del documento.

La declaración de un {⟨*nuevo alias*⟩} es «global» y se almacena en el módulo `spintent.lua` como una {⟨*alias*⟩} valido para pasar como {⟨*argumento*⟩} a `\spunit`. Por ejemplo: `\spunitalias{caudal}{m^{3}/s}` creará el {⟨*alias*⟩} '`caudal`' y cuando se utilice `$\spunit{50 caudal}$` se imprimirá '50 m^3/s' en la salida PDF y NVDA/MathCAT verbalizará *«cincuenta metros al cubo por segundos»*.

3.3 El comando `\spmoney`

`\spmoney`

`\spmoney`{⟨*cifra*⟩}
`\spmoney`{⟨*cifra moneda*⟩}
`\spmoney`[⟨*key* = *val*⟩]{⟨*cifra moneda*⟩}

El comando `\spmoney` trabaja en *modo matemático* y recibe el *argumento obligatorio* {⟨*cifra*⟩} que contiene la *«cifra numérica»* positiva o negativa la se cual procesa internamente por el comando `\spnum` al inicio del {⟨*argumento*⟩} acompañada de un *«símbolo de moneda»* o *«alias de moneda»* opcional descritos en §3.3.2.

Si no se proporciona el *argumento opcional* [⟨*key* = *val*⟩] y se ejecuta `$\spmoney{500}$` se imprimirá '$500' en la salida PDF y NVDA/MathCAT verbalizará *«quinientos pesos»*.

El comportamiento de `\spmoney` depende de como se pase el *argumento obligatorio* que tiene prioridad por sobre los valores por defecto. Por ejemplo: `$\spmoney{500 dolares}$` imprimirá '$500' en la salida PDF y NVDA/MathCAT verbalizará *«quinientos dolares»* aunque la moneda por defecto sean pesos.

Si se pasa un *«alias de moneda»* del tipo ISO (código de divisas) como CLP o USD (mayúsculas) se modifica la verbalización en NVDA/MathCAT agregando el país y se ajusta la salida PDF al estandar ISO. Por ejemplo: `$\spmoney{500 CLP}$` imprimirá 'CLP500' en la salida PDF y NVDA/MathCAT verbalizará *«quinientos pesos chilenos»*.

Si se pasa un *«alias de moneda»* del tipo ISO (código de divisas) como clp o usd (minúsculas) se modifica la verbalización en NVDA/MathCAT agregando el país, pero, NO se modifica la salida PDF imprimiendo el

«símbolo de moneda» correspondiente. Por ejemplo: `$\spmoney{500 clp}$` imprimirá '$500' en la salida PDF y NVDA/MathCAT verbalizará *«quinientos pesos chilenos»*

Pasar de manera directa en el *argumento obligatorio* un *«símbolo de moneda»* como '$', '€', '£' o un *«alias de moneda»* como `peso`, `euro` o `libra` produce la salida PDF y verbalización estándar esperada para NVDA/MathCAT.

- La posición del *«símbolo de moneda»* a la izquierda o a la derecha de la *«cifra numérica»* y la verbalización para NVDA/MathCAT se manejan desde el módulo Lua `spintent.lua` y no se proveen llaves para modificar este comportamiento.
- El *«símbolo de moneda»* es tomado de la fuente tipográfica definida para el *modo matemático* al momento de ejecutar el comando, se debe tener presente en caso de querer usar un *«símbolo de moneda»* poco habitual que esté soportada por spintent. El {⟨*argumento*⟩} puede contener espacios matemáticos validos, pero, NO puede contener comandos con o sin argumentos como `\textstyle`, `\mathrm{#1}` o `\mathbf{#1}` por ejemplo. La *coma* '`,`' o el *punto* '`.`' como *separador decimal* solo pueden aparecer una «única vez». Cualquier entrada fuera de esta sintaxis devolverá un "error" deteniendo la compilación del documento.

3.3.1 Llaves para `\spmoney`

El comando `\spmoney` posee la ⟨*meta-key*⟩ `cifras-sep` pasada al comando `\spnum` (§3.1.1).

`currency` = {⟨*moneda* | *alias*⟩} default: *peso*

Establece el *símbolo de moneda* por defecto que se imprimirá en la salida PDF y cómo lo verbalizará NVDA/MathCAT cuando SOLO se pasa una *«cifra numérica»* como {⟨*argumento*⟩} al comando `\spmoney`. El *«símbolo de moneda»* o el *«alias de moneda»* pasado a esta llave debe ser alguno de los descritos en §3.3.2.

`sub-units` = {⟨*true* | *false*⟩} default: *false*

Establece si se usarán *subunidades* de la moneda en curso para la verbalización en NVDA/MathCAT. La moneda en curso debe aceptar *subunidades*. Por ejemplo: `$\spmoney[sub-units=true]{100,5 dolares}$` imprimirá '$100,5' en la salida PDF y NVDA/MathCAT verbalizará *«cien dólares con cincuenta centavos»*.

`dolar-math` = {⟨*true* | *false*⟩} default: *true*

Establece si el *símbolo* '$' se imprimirá en *modo texto* o en *modo matemático*. Si se establece en `true`, se usará la fuente tipográfica definida para el *modo matemático* en curso para imprimir '$' en la salida PDF; si se establece en `false`, se usará la la fuente tipográfica definida para el *modo texto* para imprimir '$' en la salida PDF.

3.3.2 Monedas y alias para `\spmoney`

Resumen de los {⟨*símbolos de moneda*⟩} y {⟨*alias de moneda*⟩} soportados como {⟨*argumento*⟩} para `\spmoney`.

3.4 El comando `\spshort`

`\spshort`

`\spshort`{⟨*abreviaturas* | *acrónimos* | *números ordinales*⟩}
`\spshort`[⟨*key* = *val*⟩]{⟨*abreviaturas* | *acrónimos* | *números ordinales*⟩}
`\spshort`[⟨*read-arg*={⟨*verbalización del argumento para* NVDA⟩}⟩]{⟨*comandos* LaTeX⟩}

El comando `\spshort` trabaja *solo en modo texto* y recibe *un argumento obligatorio* del tipo {⟨*abreviaturas*⟩}, {⟨*acrónimos*⟩} o {⟨*números ordinales*⟩} descritos en §3.4.2.

Si no se proporciona el *argumento opcional* [⟨*key* = *val*⟩] y se ejecuta `\spshort{dr.a}` se imprimirá 'Dr.ª' en la salida PDF y NVDA/MathCAT verbalizará *«doctora»*; si se ejecuta `\spshort{onu}` se imprimirá 'ONU' en la salida PDF y NVDA/MathCAT verbalizará *«organización de las naciones unidas»* y si se ejecuta `\spshort{mineduc}` se imprimirá 'Mineduc' en la salida PDF y NVDA/MathCAT verbalizará *«ministerio de educación»*.

Para la escritura de {⟨*números ordinales*⟩} se puede ejecutar `\spshort{1.a}` que imprimirá '1.ª' en la salida PDF y NVDA/MathCAT verbalizará *«primera»*; si se ejecuta `\spshort{1.o}` imprimirá '1.º' en la salida PDF y NVDA/MathCAT verbalizará *«primero»*.

Para crear un nueva {⟨*abreviatura*⟩}, {⟨*acrónimo*⟩} o {⟨*número ordinal*⟩} que no se encuentre registrada por spintent dentro del módulo Lua `spintent.lua` descritas en §3.4.2 se debe utilizar la llave `read-arg` y definir esta dentro del *argumento obligatorio* que puede contener {⟨*comandos* LaTeX⟩} *en modo texto* los cuales se imprimirán en salida PDF.

- Para la escritura de {⟨*números ordinales*⟩} se debe tener cuidado y no confundir la *«letra o voladita»* 'º' con el símbolo de grado '°'. Si la fuente tipográfica del *modo texto* está utilizando `Numbers=OldStyle` los números pueden quedar muy separados de la *«letra voladita»*, la solución para esto es agregar `\addfontfeatures{Numbers=Lining}` dentro de un grupo junto al comando: `{\addfontfeatures{Numbers=Lining}\spshort{1.o}}`.
- El comando `\spshort` está pensado como un reemplazo para *shorthands* y el comando `\sptext` provistos por el paquete `babel` en español compatible con *tagged* PDF. Es de las pocas utilidades que provee el paquete spintent que trabaja *solo en modo texto*. Internamente utiliza la llave `E` *(expansion text)* proveída por el paquete `tagpdf`[29].

Entrada	Alias	Nombre	Salida	NVDA/MathCAT
$	peso, pesos	peso	$	*«peso / pesos»*
€	euro, euros	euro	€	*«euro / euros»*
£	libra, libras	libra	£	*«libra / libras»*
¥	yen, yenes	yen	¥	*«yen / yenes»*
¥	yuan, yuanes	yuan	¥	*«yuan / yuanes»*
₩	won, wons	won	₩	*«won / wons»*
₪	shekel, shekels, sequel, sequeles	séquel	₪	*«séquel / séqueles»*
₹	rupia, rupias	rupia	₹	*«rupia / rupias»*
₽	rublo, rublos	rublo	₽	*«rublo / rublos»*
₺	lira, liras	lira	₺	*«lira / liras»*
₴	grivna, grivnas	grivna	₴	*«grivna / grivnas»*
¢	centavo, centavos	centavo	¢	*«centavo / centavos»*
		Divisas en formato ISO (Mayúsculas)		
CLP	—	peso chileno	CLP	*«peso chileno / pesos chilenos»*
MXN	—	peso mexicano	MXN	*«peso mexicano / pesos mexicanos»*
USD	—	dólar	USD	*«dólar estadounidense / dólares estadounidenses»*
EUR	—	euro	EUR	*«euro / euros»*
GBP	—	libra	GBP	*«libra / libras»*
JPY	—	yen	JPY	*«yen / yenes»*
CNY	—	yuan	CNY	*«yuan / yuanes»*
KRW	—	won	KRW	*«won / wons»*
ILS	—	séquel	ILS	*«séquel / séqueles»*
INR	—	rupia	INR	*«rupia / rupias»*
RUB	—	rublo	RUB	*«rublo / rublos»*
TRY	—	lira	TRY	*«lira / liras»*
UAH	—	grivna	UAH	*«grivna / grivnas»*
		Códigos ISO en minúsculas		
clp	—	peso chileno	$	*«peso chileno / pesos chilenos»*
mxn	—	peso mexicano	$	*«peso mexicano / pesos mexicanos»*
usd	dolar, dolares	dólar	$	*«dólar estadounidense / dólares estadounidenses»*
eur	—	euro	€	*«euro / euros»*
gbp	—	libra	£	*«libra / libras»*
jpy	—	yen	¥	*«yen / yenes»*
cny	—	yuan	¥	*«yuan / yuanes»*
krw	—	won	₩	*«won / wons»*
ils	—	séquel	₪	*«séquel / séqueles»*
inr	—	rupia	₹	*«rupia / rupias»*
rub	—	rublo	₽	*«rublo / rublos»*
try	—	lira	₺	*«lira / liras»*
uah	—	grivna	₴	*«grivna / grivnas»*

3.4.1 Llaves para \spshort

read-arg = {⟨*verbalización* NVDA⟩} default: *no usado*

Establece la *verbalización del argumento* {⟨*comandos* LaTeX⟩} de la nueva {⟨*abreviatura*⟩}, {⟨*acrónimo*⟩} o {⟨*número ordinal*⟩} para NVDA. El valor de esta llave *siempre* debe pasarse entre '{ }'. Por ejemplo: `\spshort[read-arg={octavo}]{8.\textsuperscript{\emph{avo}}}` creará una nueva {⟨*abreviatura*⟩} que imprimirá '8.^{avo}' en la salida PDF y NVDA verbalizará *«octavo»*.

small-caps = {⟨*true* | *false*⟩} default: *false*

Establece si se usarán *versalitas* para imprimir {⟨*acrónimos*⟩} y para la *«letra volada»* en {⟨*abreviaturas*⟩} y {⟨*números ordinales*⟩} respetando las reglas de la RAE para el uso de estas. Si se establece en `true`, las abreviaturas con mayúscula como `EE. UU.` y los acrónimos de cinco o más letras como `Unicef` no se verán afectados; si se establece en `false`, no se aplicarán *versalitas*. Por ejemplo: `\spshort[small-caps=true]{dr.a}` imprimirá 'Dr.^{A}' en la salida PDF y NVDA verbalizará *«doctora»*.

3.4.2 Argumentos para \spshort

El cuadro 2 resume las abreviaturas, expresiones latinas y títulos registrados en el módulo Lua `spintent.lua`.

Cuadro 2: Abreviaturas soportadas

Entrada	Alias / Variantes	Salida PDF	Lectura NVDA
	Abreviaturas con letras voladas		
dr.a	dr.ª	Dr.ª	*«doctora»*
dr.as	—	Dr.ªs	*«doctoras»*
prof.a	—	Prof.ª	*«profesora»*
d.a	d.ª	D.ª	*«doña»*
m.a	—	M.ª	*«maría»*
n.o	n.º	N.º	*«número»*
n.os	—	N.ºs	*«números»*
c.ia	—	C.ía	*«compañía»*
	Abreviaturas comunes y expresiones latinas		
pág.	pag.	pág.	*«página»*
vol.	—	vol.	*«volumen»*
etc.	—	etc.	*«etcétera»*
et al.	et. al.	et al.	*«y otros»*
ibíd.	ibid.	ibíd.	*«ibídem»*
op. cit.	op.cit.	op. cit.	*«obra citada»*
loc. cit.	loc.cit.	loc. cit.	*«lugar citado»*
v. gr.	v.gr.	v. gr.	*«verbigracia»*
i. e.	i.e.	i. e.	*«esto es»*
e. g.	e.g.	e. g.	*«por ejemplo»*
p. ej.	p.ej.	p. ej.	*«por ejemplo»*
	Títulos, profesiones y cargos		
sr.	—	Sr.	*«señor»*
sra.	—	Sra.	*«señora»*
srta.	—	Srta.	*«señorita»*
dr.	—	Dr.	*«doctor»*
dra.	—	Dra.	*«doctora»*
dres.	—	Dres.	*«doctores»*
dras.	—	Dras.	*«doctoras»*
prof.	—	Prof.	*«profesor»*
profa.	—	Profa.	*«profesora»*
profs.	—	Profs.	*«profesores»*
ing.	—	Ing.	*«ingeniero»*
ings.	—	Ings.	*«ingenieros»*
lic.	—	Lic.	*«licenciado»*
v. b.	v.b.	V. B.	*«visto bueno»*
	Abreviaturas compuestas e institucionales		
s. a.	s.a.	S. A.	*«sociedad anónima»*
ee. uu.	ee.uu.	EE. UU.	*«estados unidos»*
dd. hh.	dd.hh.	DD. HH.	*«derechos humanos»*
jj. oo.	jj.oo.	JJ. OO.	*«juegos olímpicos»*

Continúa en la siguiente página...

Cuadro 2: *(Continuación - Abreviaturas soportadas)*

Entrada	Alias / Variantes	Salida PDF	Lectura NVDA
`ff. aa.`	`ff.aa.`	FF. AA.	*«fuerzas armadas»*
`rr. ee.`	`rr.ee.`	RR. EE.	*«relaciones exteriores»*
`cc. aa.`	`cc.aa.`	CC. AA.	*«comunidades autónomas»*
`p. d.`	`p.d.`	P. D.	*«posdata»*
`a. c.`	`a.c.`	a. C.	*«antes de Cristo»*
`d. c.`	`d.c.`	d. C.	*«después de Cristo»*
`d. n. i.`	`d.n.i.`	D. N. I.	*«documento nacional de identidad»*
`r. u. t.`	`r.u.t.`	R. U. T.	*«rol único tributario»*
`r. u. n.`	`r.u.n.`	R. U. N.	*«rol único nacional»*

El cuadro 3 detalla las siglas y acrónimos registrados en el módulo Lua `spintent.lua`.

Cuadro 3: Siglas y acrónimos soportados

Entrada	Alias / Variantes	Salida PDF	Lectura NVDA
		Siglas (Deletreo o lectura expandida)	
`dni`	—	DNI	*«documento nacional de identidad»*
`rut`	—	RUT	*«rol único tributario»*
`run`	—	RUN	*«rol único nacional»*
`onu`	—	ONU	*«organización de las naciones unidas»*
`rae`	—	RAE	*«real academia española»*
`ong`	`ongs`	ONG	*«organización no gubernamental»*
`urss`	—	URSS	*«unión de repúblicas socialistas soviéticas»*
`oea`	—	OEA	*«organización de los estados americanos»*
`oms`	—	OMS	*«organización mundial de la salud»*
`fmi`	—	FMI	*«fondo monetario internacional»*
`bid`	—	BID	*«banco interamericano de desarrollo»*
`otan`	—	OTAN	*«organización del tratado del atlántico norte»*
`ue`	—	UE	*«unión europea»*
`ru`	—	RU	*«reino unido»*
		Acrónimos silábicos	
`unicef`	—	Unicef	*«fondo de las naciones unidas para la infancia»*
`unesco`	—	Unesco	*«organización de las naciones unidas para la educación, la ciencia y la cultura»*
`mercosur`	—	Mercosur	*«mercado común del sur»*
`cepal`	—	Cepal	*«comisión económica para américa latina»*
`mineduc`	—	Mineduc	*«ministerio de educación»*
`conicet`	—	Conicet	*«consejo nacional de investigaciones»*

El cuadro 4 muestra ejemplos de números ordinales registrados en el módulo Lua `spintent.lua`..

Cuadro 4: Números ordinales soportados

Entrada	Alias / Variantes	Salida PDF	Lectura NVDA
		Sufijos para ordinales (Ejemplos)	
`1.a`	`1.ª`	1.ª	*«primera»*
`2.o`	`2.º`	2.º	*«segundo»*
`3.er`	—	3.er	*«tercer»*
`4.os`	—	4.os	*«cuartos»*
`5.as`	—	5.as	*«quintas»*

4 Utilidades generales

Además de los *«cuatro grandes»* y sus asociados, el paquete spintent provee tres utilidades de uso general: `\spsiglo` para la escritura y lectura de siglos, `\sptime` para la escritura y lectura de tiempo y `\spdate` para la escritura y lectura de fechas.

El comando `\spsiglo`

`\spsiglo`

`\spsiglo{⟨siglo⟩}`
`\spsiglo[⟨key = val⟩]{⟨siglo⟩}`

El comando `\spsiglo` trabaja en *modo texto* y en *modo matemático*, recibe el *argumento obligatorio* {⟨*siglo*⟩}, el cual puede ser un *número arábigo* '`21`' o un *número romano* '`XXI`' mayor que '`0`' y menor que '`4000`'.

Si no se proporciona el *argumento opcional* [⟨*key* = *val*⟩] y se ejecuta `\spsiglo{21}` o `\spsiglo{XXI}` se imprimirá 'XXI' en la salida PDF y NVDA/MathCAT verbalizará *«veintiuno»*.

El *argumento obligatorio* {⟨*siglo*⟩} *«siempre»* es convertido a *número romano en versalitas* y es verbalizado por NVDA/MathCAT como *número arábigo*. Esta es la única utilidad que provee el paquete spintent que trabaja en *modo texto* y en *modo matemático*. Internamente utiliza la llave `alt` proveída por el paquete tagpdf en *modo texto* y usa `\MathMLintent` para el *modo matemático*.

Llaves para `\spsiglo`

`print-era` = {⟨*ac* | *dc* | *none*⟩} default: *none*

Establece si se *imprime* o no la abreviatura después de imprimir el argumento {⟨*siglo*⟩}. Si se establece en `ac`, `\spsiglo[print-era=ac]{XXI}` imprimirá 'XI a. C.' en la salida PDF y NVDA/MathCAT verbalizará *«veintiuno antes de cristo»*; si se establece en `dc`, `\spsiglo[print-era=dc]{XXI}` imprimirá 'XI d. C.' en la salida PDF y NVDA/MathCAT verbalizará *«veintiuno después de cristo»*; si se establece en `none`, `\spsiglo[print-era=none]{XXI}` imprimirá 'XI' en la salida PDF y NVDA/MathCAT verbalizará *«veintiuno»*.

El comando `\sptime`

`\sptime`

`\sptime{⟨horas : minutos⟩}`
`\sptime{⟨horas : minutos : segundos⟩}`

El comando `\sptime` trabaja en *modo matemático* y recibe el *argumento obligatorio* {⟨*horas* : *minutos*⟩} o {⟨*horas* : *minutos* : *segundos*⟩}. Por ejemplo: `$\sptime{23:45}$` imprimirá '23:45' en *modo texto* la salida PDF y NVDA/MathCAT verbalizará *«las veintitrés y cuarenta y cinco minutos»*.

El comando `\spdate`

`\spdate`

`\spdate{⟨DD/ MM/YYYY⟩}` `\spdate{⟨DD- MM-YYYY⟩}`
`\spdate{⟨YYYY/ MM/DD⟩}` `\spdate{⟨YYYY- MM-DD⟩}`

El comando `\spdate` trabaja en *modo matemático* y recibe el *argumento* obligatorio {⟨DD/MM/YYYY⟩}, {⟨DD-MM-YYYY⟩} o {⟨YYYY-MM-DD⟩}. Por ejemplo: `$\spdate{1981-01-02}$` y `$\spdate{02/01/1981}$` imprimirán en *modo texto* la salida PDF '1981-01-02' y '02/01/1981' respectivamente y en ambos casos, NVDA/MathCAT verbalizará *«dos de enero de mil novecientos ochenta y uno»*.

Si se pasa el argumento {⟨YYYY/MM/DD⟩}, internamente se convertirá en {⟨DD/MM/YYYY⟩} y se imprimirá de esa forma por compatibilidad con NVDA/MathCAT.

5 Utilidades para símbolos

El paquete spintent provee tres comandos `\spread`, `\spdsym` y `\spmsym` para trabajar con *«símbolos matemáticos»* descritos en esta sección. Están pensandos desde el punto de vista docente, para situaciones donde la lectura de un *«símbolo»* (acorde al contexto) es relevante.

A diferencia de las otras utilidades implementados por spintent (que analizan y dan formato a la salida), estos no procesan los *«símbolos matemáticos»*, solo afectan la *forma* (semántica) en que serán verbalizados por NVDA/MathCAT.

El comando `\spread`

`\spread`

`\spread{⟨verbalización NVDA⟩}{⟨símbolo matemático⟩}`

El comando `\spread` trabaja en *modo matemático* y recibe dos *argumentos obligatorios*: El primer *argumento* {⟨*verbalización* NVDA⟩} establece la *«forma»* en la cual NVDA/MathCAT verbalizará al segundo *argumento* {⟨*símbolo matemático*⟩}. Por ejemplo: `$\spread{es-paralela}{\parallel}$` imprimirá '‖' en la salida PDF y NVDA/MathCAT verbalizará *«es paralela»*; `$\spread{es-paralelo}{\parallel}$` imprimirá '‖' en la salida PDF y NVDA/MathCAT verbalizará *«es paralelo»*.

El comando \spusep

\spusep
\spusep{⟨separador⟩}
\spusep[⟨key = val⟩]{⟨separador⟩}

El comando \spusep trabaja en *modo matemático* y recibe el *argumento obligatorio* {⟨separador⟩}, el cual debe ser uno de los *«delimitadores»* soportados: '(', ')', '[', ']', '\{' o '\}'. Imprime el *delimitador* indicado con el tamaño establecido por la llave size y controla si NVDA/MathCAT lo verbaliza o no mediante la llave silent.

Si no se proporciona el *argumento opcional* [⟨key = val⟩], $\spusep{(}$ imprimirá '(' en la salida PDF y NVDA/MathCAT verbalizará *«paréntesis izquierdo»*.

El motivo de existencia de \spusep es que los comandos \left, \right, \Uleft y \Uright provistos por LuaTEX crean un *grupo* implícito que dificulta el código para *tagged* PDF cuando se necesita interrumpir algo entre ellos, por ejemplo al construir expresiones complejas con \spnZ, \spmQ o \spdiv. El comando \spusep imprime el *delimitador* sin crear ningún grupo.

Llaves para \spusep

size = {⟨none | big | Big | bigg | Bigg⟩} default: *none*

Establece el *tamaño* del {⟨delimitador⟩} que se imprimirá en la salida PDF. Los valores corresponden a los cuatro tamaños fijos provistos por LaTeX: si se establece en none, el {⟨delimitador⟩} se imprime en el tamaño por defecto del modo matemático en curso; si se establece en big, se usa \bigl/\bigr; si se establece en Big, se usa \Bigl/\Bigr; si se establece en bigg, se usa \biggl/\biggr; y si se establece en Bigg, se usa \Biggl/\Biggr. El tamaño se aplica automáticamente en el lado correcto (izquierdo o derecho) según el {⟨delimitador⟩} pasado.

silent = {⟨true | false⟩} default: *false*

Establece si NVDA/MathCAT *verbalizará* el {⟨delimitador⟩} impreso por \spusep. Si se establece en true, el {⟨delimitador⟩} se marcará con el *intent* :silent y NVDA/MathCAT *no* lo verbalizará; si se establece en false, NVDA/MathCAT verbalizará el {⟨delimitador⟩} de forma normal.

El comando \spdsym

\spdsym
\spdsym
\spdsym[⟨key = val⟩]

El comando \spdsym trabaja en *modo matemático* y recibe el *argumento opcional* [⟨key = val⟩] mediante el cual se establece la *«forma»* en la que se imprimirá el *símbolo de división* y como se verbalizará en NVDA/MathCAT.

Si no se proporciona el *argumento opcional* [⟨key = val⟩], $\spdsym$ imprimirá ':' en la salida PDF y NVDA/MathCAT verbalizará *«dividido»*.

Llaves para \spdsym

prefix = {⟨prefijo⟩} default: *no usado*

Establece el *prefijo* que será *verbalizado* por NVDA/MathCAT *«antes»* de *verbalizar* el {⟨valor⟩} establecido por la llave read-sym. El valor de esta llave *siempre* debe pasarse entre llaves '{ }'.

suffix = {⟨sufijo⟩} default: *no usado*

Establece el *sufijo* que será *verbalizado* por NVDA/MathCAT *«después»* de *verbalizar* el {⟨valor⟩} establecido por la llave read-sym. El valor de esta llave *siempre* debe pasarse entre llaves '{ }'.

set-sym = {⟨old-style | two-dots | slash⟩} default: *two-dots*

Establece el *símbolo de división* que se imprimirá en la salida PDF. Si se establece en old-style, imprimirá '÷'; si se establece en two-dots, imprimirá ':' y si se establece slash, imprimirá '/'.

read-sym = {⟨dividido | dividido-por | dividido-entre⟩} default: *dividido*

Establece la *forma* con la cual NVDA/MathCAT verbalizará el *símbolo de división*. Si se establece en dividido, NVDA/MathCAT verbalizará *«dividido»*; si se establece en dividido-por, NVDA/MathCAT verbalizará *«dividido por»* y si se establece en dividido-entre, NVDA/MathCAT verbalizará *«dividido entre»*.

El comando \spmsym

\spmsym
\spmsym
\spmsym[⟨key = val⟩]

El comando \spmsym trabaja en *modo matemático* y recibe el *argumento opcional* [⟨key = val⟩] mediante el cual se establece la *«forma»* en la que se imprimirá el *símbolo de multiplicación* y como se verbalizará en NVDA/MathCAT.

Si no se proporciona el *argumento opcional* [⟨key = val⟩], $\spmsym$ imprimirá '·' en la salida PDF y NVDA/MathCAT verbalizará *«por»*.

Llaves para `\spmsym`

`prefix` = {⟨*prefijo*⟩} default: *no usado*

Establece el *prefijo* que será *verbalizado* por `NVDA/MathCAT` *«antes»* de *verbalizar* el {⟨*valor*⟩} establecido por la llave `read-sym`. El valor de esta llave *siempre* debe pasarse entre llaves '{ }'.

`suffix` = {⟨*sufijo*⟩} default: *no usado*

Establece el *sufijo* que será *verbalizado* por `NVDA/MathCAT` *«después»* de *verbalizar* el {⟨*valor*⟩} establecido por la llave `read-sym`. El valor de esta llave *siempre* debe pasarse entre llaves '{ }'.

`set-sym` = {⟨*cross* | *dot* | *asterisk*⟩} default: *dot*

Establece el *símbolo de multiplicación* que se imprimirá en la salida `PDF`. Si se establece en `cross`, imprimirá '×'; si se establece en `dot`, imprimirá '·' y si se establece en `asterisk`, imprimirá '*'.

`read-sym` = {⟨*por* | *multiplicado-por* | *veces*⟩} default: *por*

Establece la *forma* con la cual `NVDA/MathCAT` verbalizará el *símbolo de multiplicación*. Si se establece en `por`, `NVDA/MathCAT` verbalizará *«por»*; si se establece en `multiplicado-por`, `NVDA/MathCAT` verbalizará *«multiplicado por»* y si se establece en `veces`, `NVDA/MathCAT` verbalizará *«veces»*.

`not-print` = {⟨*true* | *false*⟩} default: *false*

Deshabilita la *impresión* del *símbolo de multiplicación*. Cuando se establece esta llave, `NVDA/MathCAT` verbalizará el *símbolo de multiplicación* acorde al valor establecido por la llave `read-sym`, pero, no se imprimirá este en la salida `PDF`.

6 Utilidades para números

El paquete spintent provee utilidades para trabajar con números descritas en esta sección.

El comando `\spdiv`

`\spdiv` `\spdiv`{⟨*dividendo*⟩}{⟨*divisor*⟩}
`\spdiv`[⟨*key* = *val*⟩]{⟨*dividendo*⟩}{⟨*divisor*⟩}

El comando `\spdiv` trabaja en *modo matemático* y recibe dos *argumentos obligatorios* {⟨*dividendo*⟩} y {⟨*divisor*⟩}.

Si no se proporciona el *argumento opcional* [⟨*key* = *val*⟩], `$\spdiv{20}{26}$` imprimirá '20 : 26' en la salida `PDF` y `NVDA/MathCAT` verbalizará *«veinte dividido veinte y seis»*

Llaves para `\spdiv`

El comando `\spdiv` posee las ⟨*meta-keys*⟩ `cifras-sep`, `read-sign`, `read-period-sep` y `notation-sym` pasadas al comando `\spnum` (§3.1.1) junto a las ⟨*meta-keys*⟩ `set-sym` y `read-sym` pasadas al comando `\spdsym` (pág. 12).

`wrap-parens` = {⟨*true* | *false*⟩} default: *false*

Establece si se *imprimen* paréntesis redondos '(...)' alrededor de los *argumentos* {⟨*dividendo*⟩} y {⟨*divisor*⟩} pasados a `\spdiv`. Si se establece en `true`, se imprimirán paréntesis redondos alrededor de ambos *argumentos*; si se establece en `false`, no se imprimirán los paréntesis.

`read-parens` = {⟨*true* | *false*⟩} default: *true*

Establece si `NVDA/MathCAT` *verbalizará* los paréntesis redondos '(...)' alrededor de los *argumentos* {⟨*dividendo*⟩} y {⟨*divisor*⟩} pasados a `\spdiv`. Si se establece en `true`, `NVDA/MathCAT` verbalizará la lectura de los paréntesis; si se establece en `false`, `NVDA/MathCAT` no los verbalizará.

El comando `\spmul`

`\spmul` `\spmul`{⟨*factor*⟩}{⟨*factor*⟩}
`\spmul`[⟨*key* = *val*⟩]{⟨*factor*⟩}{⟨*factor*⟩}

El comando `\spmul` trabaja en *modo matemático* y recibe dos *argumentos obligatorios* {⟨*factor*⟩} y {⟨*factor*⟩}.

Si no se proporciona el *argumento opcional* [⟨*key* = *val*⟩], `$\spmul{20}{26}$` imprimirá '20 · 26' en la salida `PDF` y `NVDA/MathCAT` verbalizará *«veinte por veinte y seis»*.

Llaves para `\spmul`

El comando `\spmul` posee las ⟨*meta-keys*⟩ `cifras-sep`, `read-sign`, `read-period-sep` y `notation-sym` pasadas al comando `\spnum` (§3.1.1) junto a las ⟨*meta-keys*⟩ `set-sym` y `read-sym` pasadas al comando `\spmsym` (pág. 13).

`wrap-parens` = {⟨*true* | *false*⟩} default: *false*

Establece si se *imprimen* paréntesis redondos '(...)' alrededor de los *argumentos* {⟨*factor*⟩} y {⟨*factor*⟩} pasados a `\spmul`. Si se establece en `true`, se imprimirán paréntesis redondos alrededor de ambos *argumentos*; si se establece en `false`, no se imprimirán los paréntesis.

`read-parens` = {⟨*true* | *false*⟩} default: *true*

Establece si NVDA/MathCAT *verbalizará* los paréntesis redondos '(...)' alrededor de los *argumentos* {⟨*factor*⟩} y {⟨*factor*⟩} pasados a \spmul. Si se establece en true, NVDA/MathCAT verbalizará la lectura de los paréntesis; si se establece en false, NVDA/MathCAT no los verbalizará.

El comando \spnD

\spnD

\spnD
\spnD(⟨*número natural*⟩)
\spnD[⟨*key* = *val*⟩](⟨*número natural*⟩)

El comando \spnD trabaja en *modo matemático* y recibe el *argumento delimitado por paréntesis* (⟨*número natural*⟩), por ejemplo: '(6)'. Si se usa sin (⟨*argumento*⟩), $\spnD$ imprimirá 'D(n)' en la salida PDF y NVDA/MathCAT verbalizará *«divisores de n»*.

Si se usa con (⟨*argumento*⟩), por ejemplo, $\spnD(6)$, imprimirá 'D(6)' en la salida PDF y NVDA/MathCAT verbalizará *«divisores de seis»*.

Llaves para \spnD

prefix = {⟨*prefijo*⟩} default: *no usado*

Establece el *prefijo* que será verbalizado por NVDA/MathCAT *«antes»* de verbalizar \spnD(⟨*argumento*⟩). El valor de esta llave *siempre* debe pasarse entre llaves '{ }'.

result = {⟨*true* | *false*⟩} default: *false*

Establece si se *imprime* el resultado de calcular los *divisores* del (⟨*número natural*⟩) pasado a \spnD. Si se establece en true, $\spnD[result=true](6)$ imprimirá 'D(6) = {1, 2, 3, 6}' en la salida PDF y NVDA/MathCAT verbalizará *«los divisores de seis son uno, dos, tres y seis»*; si se establece en false, no se imprimirá en la salida PDF el resultado de calcular los *divisores*.

result-num = {⟨*número natural*⟩} default: *5*

Establece la *cantidad* de números que se mostrarán en la salida PDF al activar result=true. Si se establece en 2, solo se imprimirán dos valores en la salida PDF; si no se establece, se imprimirán como máximo 5 valores y luego se añadirá '...' en la salida PDF.

sample-arg = {⟨*true* | *false*⟩} default: *false*

Desactiva la *validación* interna del (⟨*argumento*⟩) pasado a \spnD, permitiendo usar ejemplos *«no numéricos»* como entrada. Si se establece en true, $\spnD[sample-arg=true](m)$ imprimirá 'D(m)' en la salida PDF y NVDA/MathCAT verbalizará *«divisores de m»*; si se establece en false, el (⟨*argumento*⟩) debe ser un número natural, en caso contrario se devolverá un "error".

silent-cmd = {⟨*true* | *false*⟩} default: *false*

Establece si se *silencia* la verbalización para NVDA/MathCAT de \spnD(⟨*argumento*⟩) cuando se establece sample-arg=true o cuando \spnD se usa sin (⟨*argumento*⟩). Si se establece en true NVDA/MathCAT NO lo verbalizará; si se establece en false, NVDA/MathCAT SÍ lo verbalizará.

El comando \spnM

\spnM

\spnM
\spnM(⟨*número natural*⟩)
\spnM[⟨*key* = *val*⟩](⟨*número natural*⟩)

El comando \spnM trabaja en *modo matemático* y recibe el *argumento delimitado por paréntesis* (⟨*número natural*⟩), por ejemplo: '(6)'. Si se usa sin (⟨*argumento*⟩), $\spnM$ imprimirá 'M(n)' en la salida PDF y NVDA/MathCAT verbalizará *«múltiplos de n»*.

Si se usa con (⟨*argumento*⟩), por ejemplo, $\spnM(6)$, imprimirá 'M(6)' en la salida PDF y NVDA/MathCAT verbalizará *«múltiplos de seis»*.

Llaves para \spnM

prefix = {⟨*prefijo*⟩} default: *no usado*

Establece el *prefijo* que será verbalizado por NVDA/MathCAT *«antes»* de verbalizar \spnM(⟨*argumento*⟩). El valor de esta llave *siempre* debe pasarse entre llaves '{ }'.

result = {⟨*true* | *false*⟩} default: *false*

Establece si se *imprime* el resultado de calcular los *múltiplos* del (⟨*número natural*⟩) pasado a \spnM. Si se establece en true, $\spnM[result=true](6)$ imprimirá 'M(6) = {6, 12, 18, 24, 30, ...}' en la salida PDF y NVDA/MathCAT verbalizará *«los múltiplos de seis son seis, doce, dieciocho, veinticuatro, treinta puntos suspencivos»*; si se establece en false, no se imprimirá en la salida PDF el resultado de calcular los *múltiplos*.

result-num = {⟨*número natural*⟩} default: *5*

Establece la *cantidad* de números que se mostrarán en la salida PDF al activar result=true. Si se establece en 2, solo se imprimirán dos valores en la salida PDF seguidos de '...'; si no se establece, se imprimirán por defecto los 5 primeros valores y luego se añadirá '...' en la salida PDF.

sample-arg = {⟨*true* | *false*⟩} default: *false*

Desactiva la *validación* interna del (⟨*argumento*⟩) pasado a `\spnM`, permitiendo usar ejemplos *«no numéricos»* como entrada. Si se establece en `true`, `$\spnM[sample-arg=true](m)$` imprimirá 'M(m)' en la salida PDF y NVDA/MathCAT verbalizará *«múltiplos de m»*; si se establece en `false`, el (⟨*argumento*⟩) debe ser un número natural, en caso contrario se devolverá un "error".

`silent-cmd` = {⟨*true* | *false*⟩} default: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT de `\spnM`(⟨*argumento*⟩) cuando se establece `sample-arg=true` o cuando `\spnM` se usa sin (⟨*argumento*⟩). Si se establece en `true` NVDA/MathCAT NO lo verbalizará; si se establece en `false` NVDA/MathCAT lo verbalizará.

El comando `\spmcd`

`\spmcd` `\spmcd`
`\spmcd`(⟨*números separados por coma*⟩)
`\spmcd`[⟨*key* = *val*⟩](⟨*números separados por coma*⟩)

El comando `\spmcd` trabaja en *modo matemático* y recibe el *argumento delimitado por paréntesis* (⟨*números separados por coma*⟩), el cual debe ser una lista de *números naturales*, por ejemplo: '`(2, 3, 6)`'. Si se usa sin (⟨*argumento*⟩) `$\spmcd$` imprimirá 'mcd' en la salida PDF, y NVDA/MathCAT verbalizará *«máximo común divisor»* o *«m c d»*, acorde al {⟨*valor*⟩} establecido por la llave `read-cmd`.

Si se usa con (⟨*argumento*⟩) `$\spmcd(2, 3, 6)$` imprimirá 'mcd(2, 3, 6)' en la salida PDF, y NVDA/MathCAT verbalizará *«máximo común divisor entre dos, tres y seis»* o *«m c d entre dos, tres y seis»* acorde al {⟨*valor*⟩} establecido por la llave `read-cmd`.

Llaves para `\spmcd`

`upper-case` = {⟨*true* | *false*⟩} default: *false*

Establece si se usarán *mayúsculas* o *minúsculas* en la salida PDF para el comando `\spmcd`. Si se establece en `true`, se imprimirá 'MCD' en la salida PDF; si se establece en `false`, se imprimirá 'mcd' en la salida PDF.

`read-cmd` = {⟨*terse* | *clear*⟩} default: *clear*

Establece la *forma* en la que NVDA/MathCAT verbalizará el comando `\spmcd`. Si se establece en `terse`, NVDA/MathCAT verbalizará *«m c d»*; si se establece en `clear`, NVDA/MathCAT verbalizará *«máximo común divisor»*.

`prefix` = {⟨*prefijo*⟩} default: *no usado*

Establece el *prefijo* que será verbalizado por NVDA/MathCAT *«antes»* de verbalizar el valor establecido por la llave `read-cmd`. El valor de esta llave *siempre* debe pasarse entre llaves '`{ }`'.

`result` = {⟨*true* | *false*⟩} default: *false*

Establece si se *imprime* el resultado de calcular el *«máximo común divisor»* del (⟨*argumento*⟩) pasado a `\spmcd`. Si se establece en `true`: `$\spmcd[result=true](2, 3, 6)$` imprimirá 'mcd(2, 3, 6) = 1' en la salida PDF y NVDA/MathCAT verbalizará *«máximo común divisor entre dos, tres y seis es igual a uno»*; si se establece en `false` no se imprimirá en la salida PDF el resultado de calcular el *«máximo común divisor»*.

`sample-arg` = {⟨*true* | *false*⟩} default: *false*

Desactiva la *validación* interna del (⟨*argumento*⟩) pasado a `\spmcd`, permitiendo usar ejemplos *«no numéricos»* como entrada. Si se establece en `true`: `$\spmcd[sample-arg=true](m, n)$`, imprimirá 'mcd(m, n)' en la salida PDF y NVDA/MathCAT verbalizará *«máximo común divisor entre m y n»*; si se establece en `false` el (⟨*argumento*⟩) debe ser una lista de *números naturales*, en caso contrario se devolverá un "error".

`silent-cmd` = {⟨*true* | *false*⟩} default: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT de `\spmcd`(⟨*argumento*⟩) cuando se establece `sample-arg=true` o cuando `\spmcd` se usa sin (⟨*argumento*⟩). Si se establece en `true` NVDA/MathCAT NO lo verbalizará; si se establece en `false`, NVDA/MathCAT SÍ lo verbalizará.

El comando `\spmcm`

`\spmcm` `\spmcm`
`\spmcm`(⟨*números separados por coma*⟩)
`\spmcm`[⟨*key* = *val*⟩](⟨*números separados por coma*⟩)

El comando `\spmcm` trabaja en *modo matemático* y recibe el *argumento delimitado por paréntesis* (⟨*números separados por coma*⟩), el cual debe ser una lista de *números naturales*, por ejemplo: '`(2, 3, 6)`'. Si se usa sin (⟨*argumento*⟩) `$\spmcm$` imprimirá 'mcm' en la salida PDF, y NVDA/MathCAT verbalizará *«mínimo común múltiplo»* o *«m c m»*, acorde al {⟨*valor*⟩} establecido por la llave `read-cmd`.

Si se usa con (⟨*argumento*⟩) `$\spmcm(2, 3, 6)$` imprimirá 'mcm(2, 3, 6)' en la salida PDF, y NVDA/MathCAT verbalizará *«mínimo común múltiplo entre dos, tres y seis»* o *«m c m entre dos, tres y seis»* acorde al {⟨*valor*⟩} establecido por la llave `read-cmd`.

Llaves para `\spmcm`

`upper-case` = {⟨*true* | *false*⟩} default: *false*

Establece si se usarán *mayúsculas* o *minúsculas* en la salida PDF para el comando `\spmcm`. Si se establece en `true` se imprimirá 'MCM' en la salida PDF; si se establece en `false` se imprimirá 'mcm' en la salida PDF.

`read-cmd` = {⟨*terse* | *clear*⟩} default: *clear*

Establece la *forma* en la que NVDA/MathCAT verbalizará el comando `\spmcm`. Si se establece en `terse`, NVDA/MathCAT verbalizará *«m c m»*; si se establece en `clear`, NVDA/MathCAT verbalizará *«mínimo común múltiplo»*.

`prefix` = {⟨*prefijo*⟩} default: *no usado*

Establece el *prefijo* que será verbalizado por NVDA/MathCAT *«antes»* de verbalizar el valor establecido por la llave `read-cmd`. El valor de esta llave *siempre* debe pasarse entre llaves '{ }'.

`result` = {⟨*true* | *false*⟩} default: *false*

Establece si se *imprime* el resultado de calcular el *«mínimo común múltiplo»* del (⟨*argumento*⟩) pasado a `\spmcm`. Si se establece en `true`: `$\spmcm[result=true](2, 3, 6)$` imprimirá 'mcm$(2,3,6)=6$' en la salida PDF y NVDA/MathCAT verbalizará *«mínimo común múltiplo entre dos, tres y seis es igual a seis»*; si se establece en `false` no se imprimirá en la salida PDF el resultado de calcular el *«mínimo común múltiplo»*.

`sample-arg` = {⟨*true* | *false*⟩} default: *false*

Desactiva la *validación* interna del (⟨*argumento*⟩) pasado a `\spmcm`, permitiendo usar ejemplos *«no numéricos»* como entrada. Si se establece en `true`: `$\spmcm[sample-arg=true](m, n)$`, imprimirá 'mcm(m,n)' en la salida PDF y NVDA/MathCAT verbalizará *«mínimo común múltiplo entre m y n»*; si se establece en `false` el (⟨*argumento*⟩) debe ser una lista de *números naturales*, en caso contrario se devolverá un "error".

`silent-cmd` = {⟨*true* | *false*⟩} default: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT de `\spmcm`(⟨*argumento*⟩) cuando se establece `sample-arg=true` o cuando `\spmcm` se usa sin (⟨*argumento*⟩). Si se establece en `true` NVDA/MathCAT NO lo verbalizará; si se establece en `false` NVDA/MathCAT SÍ lo verbalizará.

El comando `\spnZ`

`\spnZ`
`\spnZ`{⟨*número entero*⟩}
`\spnZ`[⟨*key* = *val*⟩]{⟨*número entero*⟩}

El comando `\spnZ` trabaja en *modo matemático* y recibe el *argumento* obligatorio {⟨*número entero*⟩} el cual se procesa internamente bajo el comando `\spnum`. Este comando está dedicado exclusivamente al manejo de *«números enteros»*, y añade algunas utilidades prácticas para el demarcado e interacción con NVDA/MathCAT.

Llaves para `\spnZ`

El comando `\spnZ` posee las ⟨*meta-keys*⟩ `cifras-sep` y `read-sign` pasadas al comando `\spnum` (§3.1.1).

`wrap-parens` = {⟨*true* | *false*⟩} default: *false*

Establece si se *imprimen* paréntesis redondos '(...)' alrededor del argumento {⟨*número entero*⟩} pasado a `\spnZ`. Si se establece en `true`, se imprimirán paréntesis redondos alrededor del {⟨*número entero*⟩}; si se establece en `false`, no se imprimirán los paréntesis.

`read-parens` = {⟨*true* | *false*⟩} default: *false*

Establece si NVDA/MathCAT *verbalizará* los paréntesis redondos '(...)' alrededor del *argumento* {⟨*número entero*⟩} pasado a `\spnZ`. Si se establece en `true`, NVDA/MathCAT verbalizará la lectura del paréntesis '(...)'; si se establece en `false`, NVDA/MathCAT no los verbalizará.

El comando `\spmQ`

`\spmQ`
`\spmQ`{⟨*número entero*⟩}{⟨*numerador*⟩}{⟨*denominador*⟩}
`\spmQ`[⟨*key* = *val*⟩]{⟨*número entero*⟩}{⟨*numerador*⟩}{⟨*denominador*⟩}

El comando `\spmQ` trabaja en *modo matemático* y recibe *tres argumentos obligatorios*: {⟨*número entero*⟩}, {⟨*numerador*⟩} y {⟨*denominador*⟩}, los procesa internamente y devuelve un *«número mixto»*.

Si no se proporciona el *argumento opcional* [⟨*key* = *val*⟩], `$\spmQ{2}{5}{7}$` imprimirá '$2\frac{5}{7}$' en la salida PDF y NVDA/MathCAT verbalizará *«dos y cinco séptimos»*.

En el caso de las fracciones, y en especial con los *«números mixtos»*, no se pueden manejar muchas cosas desde el lado de spintent. Por ejemplo, la entrada `$3\frac{5}{7}$` con *tagged* PDF se verbaliza por NVDA/MathCAT como *«tres más cinco séptimos»*, lo cual no está mal, pero pedagógicamente hablando no es del todo correcto.

Llaves para \spmQ

join-word = {⟨*entero* | *enteros* | *y*⟩} default: *y*

Establece la *forma* en la que NVDA/MathCAT verbaliza la conexión entre el entero y la fracción que forman el *número mixto*. Si se establece en entero, al usar $\spmQ[join-word=entero]{1}{5}{7}$ imprimirá '$1\frac{5}{7}$' en la salida PDF, y NVDA/MathCAT verbalizará *«un entero cinco séptimos»*; si se establece en enteros, al usar $\spmQ[join-word=enteros]{2}{5}{7}$ imprimirá '$2\frac{5}{7}$' en la salida PDF, y NVDA/MathCAT verbalizará *«dos enteros cinco séptimos»*; si se establece en y, al usar $\spmQ[join-word=y]{2}{5}{7}$ imprimirá '$2\frac{5}{7}$' en la salida PDF, y NVDA/MathCAT verbalizará *«dos y cinco séptimos»*.

use-spnZ = {⟨*true* | *false*⟩} default: *true*

Establece si el {⟨*numerador*⟩} y el {⟨*denominador*⟩} que forman la fracción del *número mixto* se procesan internamente bajo el comando \spnZ, añadiendo así *semántica de entero* en la estructura MathML de la salida PDF. Si se establece en true, {⟨*numerador*⟩} y {⟨*denominador*⟩} se pasarán a \spnZ y NVDA/MathCAT los verbalizará correctamente como *números ordinales*; si se establece en false, los argumentos se tipografían directamente sin procesamiento semántico adicional.

print-dfrac = {⟨*true* | *false*⟩} default: *false*

Establece si la fracción del *número mixto* se imprime usando \dfrac en lugar de \frac. Si se establece en true, al usar $\spmQ[print-dfrac=true]{2}{5}{7}$ imprimirá la fracción en tamaño *display* '$2\dfrac{5}{7}$' en la salida PDF; si se establece en false, la fracción se imprime con \frac siguiendo el tamaño matemático del contexto en curso.

wrap-parens = {⟨*true* | *false*⟩} default: *false*

Establece si se *imprimen* paréntesis redondos '(...)' alrededor del *número mixto* completo devuelto por \spmQ. Si se establece en true, se imprimirán paréntesis redondos cuyo tamaño se ajusta automáticamente al estilo matemático en curso; si se establece en false, no se imprimirán los paréntesis.

read-parens = {⟨*true* | *false*⟩} default: *true*

Establece si NVDA/MathCAT *verbalizará* los paréntesis redondos '(...)' alrededor del *número mixto* cuando la llave wrap-parens está activa. Si se establece en true, NVDA/MathCAT verbalizará la lectura de los paréntesis; si se establece en false, NVDA/MathCAT no los verbalizará.

7 Utilidades para geometría

El paquete spintent provee comandos para trabajar con los objetos y medidas de la *geometría plana* descritos en esta sección. Todos los comandos trabajan en *modo matemático* y comparten el mismo módulo spintent.lua para construir la verbalización correcta.

Todos los comandos de esta sección aceptan como puntos solo *letras mayúsculas* como base, con modificadores opcionales de subíndice '_{...}' o prima '′'. La única excepción es \specta cuando el *argumento* es de la forma {⟨*letra*⟩}, que acepta *letras minúsculas*. Los subíndices siempre deben escribirse de la forma correcta usando _{...} como A_{...} y nunca como A_n sin llaves que funciona solo por coincidencia.

El comando \spPoint

\spPoint \spPoint{⟨*punto*⟩}
\spPoint[⟨*key* = *val*⟩]{⟨*punto*⟩}

El comando \spPoint trabaja en *modo matemático* y recibe el *argumento obligatorio* {⟨*punto*⟩} el cual es una *única letra mayúscula* con un modificador opcional de subíndice '_{...}' o prima '′'.

Si no se proporciona el *argumento opcional* [⟨*key* = *val*⟩], $\spPoint{A}$ imprimirá 'A' en la salida PDF y NVDA/MathCAT verbalizará *«punto A»*; $\spPoint{A_{1}}$ imprimirá 'A_1' y verbalizará *«punto A sub uno»*.

El comando \spPoint solo acepta *un punto*. Para pares o listas de puntos (rectas, rayos, lados, ángulos, arcos, polígonos), véanse los comandos correspondientes de esta sección.

Llaves para \spPoint

read-cmd = {⟨*school* | *mathcat*⟩} default: *school*

Establece la *forma* en la que NVDA/MathCAT verbalizará el argumento {⟨*punto*⟩}. Si se establece en school, NVDA/MathCAT verbalizará el argumento {⟨*punto*⟩} sin el prefijo «mayúscula», $\spPoint{A}$ se verbaliza *«punto A»*; si se establece en mathcat, NVDA/MathCAT aplica sus propias reglas, que pueden incluir el prefijo «mayúscula» dependiendo de la configuración del usuario.

prefix = {⟨*prefijo*⟩} default: *no usado*

Establece el *prefijo* que será verbalizado por NVDA/MathCAT *antes* de verbalizar el término *«punto»*. El valor de esta llave *siempre* debe pasarse entre llaves '{ }'.

concept = {⟨*true* | *false*⟩} default: *true*

Establece si se *silencia* la verbalización en NVDA/MathCAT del concepto *«punto»* al ejecutar el comando sin modificar la impresión de este. Si se establece en true, NVDA/MathCAT verbalizará el concepto; si se establece en false, NVDA/MathCAT solo verbalizará el {⟨*argumento*⟩} acorde al valor establecido por la llave read-cmd.

silent-cmd = {⟨*true* | *false*⟩} default: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT del concepto *«punto»* y del {⟨*argumento*⟩} pasado al comando sin modificar la impresión de este. Si se establece en true, NVDA/MathCAT NO verbalizará la salida del comando; si se establece en false, NVDA/MathCAT sí verbalizará la salida del comando.

El comando \sprecta

\sprecta

\sprecta{⟨*punto, punto*⟩} \sprecta{⟨*letra*⟩}
\sprecta[⟨*key = val*⟩]{⟨*punto, punto*⟩} \sprecta[⟨*key = val*⟩]{⟨*letra*⟩}

El comando \sprecta trabaja en *modo matemático* y recibe un *argumento obligatorio* el cual puede ser de la forma {⟨*letra*⟩} o {⟨*punto, punto*⟩}. Si el *argumento* es de la forma {⟨*punto, punto*⟩} se aplica \overleftrightarrow en la impresión; si el *argumento* no contiene coma y es de la forma {⟨*letra*⟩} (mayúscula o minúscula), se imprimirá solo la {⟨*letra*⟩} sin aplicar \overleftrightarrow.

Si no se proporciona el *argumento opcional* [⟨*key = val*⟩], $\sprecta{AB}$ imprimirá '$\overleftrightarrow{AB}$' en la salida PDF y NVDA/MathCAT verbalizará *«recta A B»*; $\sprecta{m}$ imprimirá 'm' en la salida PDF y NVDA/MathCAT verbalizará *«recta m»*.

Llaves para \sprecta

read-cmd = {⟨*school* | *mathcat*⟩} default: *school*

Establece la *forma* en la que NVDA/MathCAT verbalizará el argumento {⟨*punto, punto*⟩} y el argumento {⟨*letra*⟩} cuando está en mayúscula. Si se establece en school, NVDA/MathCAT verbaliza las letras del argumento sin el prefijo «mayúscula»: $\sprecta{AB}$ se verbaliza *«recta A B»*; si se establece en mathcat, NVDA/MathCAT aplica sus propias reglas, que pueden incluir el prefijo «mayúscula» dependiendo de la configuración del usuario.

prefix = {⟨*prefijo*⟩} default: *no usado*

Establece el *prefijo* que será verbalizado por NVDA/MathCAT *antes* de verbalizar el concepto *«recta»*. El valor de esta llave *siempre* debe pasarse entre llaves '{ }'.

concept = {⟨*true* | *false*⟩} default: *true*

Establece si se *silencia* la verbalización en NVDA/MathCAT del concepto *«recta»* al ejecutar el comando sin modificar la impresión de este. Si se establece en true, NVDA/MathCAT verbalizará el concepto; si se establece en false, NVDA/MathCAT solo verbalizará el {⟨*argumento*⟩} acorde al valor establecido por la llave read-cmd.

silent-cmd = {⟨*true* | *false*⟩} default: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT del concepto *«recta»* y del {⟨*argumento*⟩} pasado al comando sin modificar la impresión de este. Si se establece en true, NVDA/MathCAT NO verbalizará la salida del comando; si se establece en false, NVDA/MathCAT sí verbalizará la salida del comando.

El comando \sprayo

\sprayo

\sprayo{⟨*punto, punto*⟩}
\sprayo[⟨*key = val*⟩]{⟨*punto, punto*⟩}

El comando \sprayo trabaja en *modo matemático* y recibe el *argumento obligatorio* {⟨*punto, punto*⟩}. Representa un *rayo* o *semirrecta* con origen en el primer punto y dirección hacia el segundo, aplicando \overrightarrow en la impresión.

Si no se proporciona el *argumento opcional* [⟨*key = val*⟩], $\sprayo{AB}$ imprimirá '$\overrightarrow{AB}$' en la salida PDF y NVDA/MathCAT verbalizará *«rayo A B»*.

Llaves para \sprayo

read-cmd = {⟨*school* | *mathcat*⟩} default: *school*

Establece la *forma* en la que NVDA/MathCAT verbalizará el argumento {⟨*punto, punto*⟩}. Si se establece en school, NVDA/MathCAT verbaliza las letras del argumento sin el prefijo «mayúscula»: $\sprayo{AB}$ se verbaliza *«rayo A B»*; si se establece en mathcat, NVDA/MathCAT aplica sus propias reglas, que pueden incluir el prefijo «mayúscula» dependiendo de la configuración del usuario.

read-sty = {⟨*rayo* | *semirrecta*⟩} default: *rayo*

Establece el *concepto* que verbalizará NVDA/MathCAT el ejecutar el comando sin modificar la impresión de este. Si se utiliza $\sprayo[read-sty=rayo]{AB}$, NVDA/MathCAT verbalizará *«rayo A B»*; si se utiliza $\sprayo[read-sty=semirrecta]{AB}$, NVDA/MathCAT verbalizará *«semirrecta A B»*.

prefix = {⟨*prefijo*⟩} default: *no usado*

Establece el *prefijo* que será verbalizado por NVDA/MathCAT *antes* de verbalizar el *«concepto»* establecido por la llave read-sty. El valor de esta llave *siempre* debe pasarse entre llaves '{ }'.

concept = {⟨*true* | *false*⟩} default: *true*

Establece si se *silencia* la verbalización en NVDA/MathCAT del *«concepto»* establecido por la llave read-sty al ejecutar el comando sin modificar la impresión de este. Si se establece en true, NVDA/MathCAT verbalizará el

concepto; si se establece en `false`, `NVDA/MathCAT` solo verbalizará el {⟨*argumento*⟩} acorde al valor establecido por la llave `read-cmd`.

`silent-cmd` = {⟨*true* | *false*⟩} default: *false*

Establece si se *silencia* la verbalización en `NVDA/MathCAT` del *«concepto»* establecido por la llave `read-sty` y del {⟨*argumento*⟩} pasado al comando sin modificar la impresión de este. Si se establece en `true`, `NVDA/MathCAT` NO verbalizará la salida del comando; si se establece en `false`, `NVDA/MathCAT` sí verbalizará la salida del comando.

La impresión '$\overrightarrow{AB}$' en la salida PDF es idéntica en ambos casos. La elección depende de la terminología del libro de texto o del currículo nacional: en Chile y España se prefiere «semirrecta»; en México y otros países de América Latina se usa «rayo».

El comando `\spside`

`\spside`

`\spside`{⟨*punto, punto*⟩}
`\spside`[⟨*key* = *val*⟩]{⟨*punto, punto*⟩}

El comando `\spside` trabaja en *modo matemático* y recibe el *argumento obligatorio* {⟨*punto, punto*⟩}. Representa un *lado*, *segmento* o *trazo* determinado por dos puntos, aplicando `\overline` en la impresión.

Si no se proporciona el *argumento opcional* [⟨*key* = *val*⟩], `$\spside{AB}$` imprimirá '$\overline{AB}$' en la salida PDF y `NVDA/MathCAT` verbalizará *«lado A B»*.

Llaves para `\spside`

`read-cmd` = {⟨*school* | *mathcat*⟩} default: *school*

Establece la *forma* en la que `NVDA/MathCAT` verbalizará al argumento {⟨*punto, punto*⟩}. Si se establece en `school`, `NVDA/MathCAT` verbaliza las letras del argumento sin el prefijo «mayúscula»: `$\spside{AB}$` se verbaliza *«lado A B»*; si se establece en `mathcat`, `NVDA/MathCAT` aplica sus propias reglas, que pueden incluir el prefijo «mayúscula» dependiendo de la configuración del usuario.

`read-sty` = {⟨*lado* | *segmento* | *trazo*⟩} default: *lado*

Establece el *concepto* que verbalizará `NVDA/MathCAT` el ejecutar el comando sin modificar la impresión de este. Si se utiliza `$\spside[read-sty=lado]{AB}$`, `NVDA/MathCAT` verbalizará *«lado A B»*; si se utiliza `$\spside[read-sty=segmento]{AB}$`, `NVDA/MathCAT` verbalizará *«segmento A B»* y si se utiliza `$\spside[read-sty=trazo]{AB}$`, `NVDA/MathCAT` verbalizará *«trazo A B»*

`prefix` = {⟨*prefijo*⟩} default: *no usado*

Establece el *prefijo* que será verbalizado por `NVDA/MathCAT` *antes* de verbalizar el *«concepto»* establecido por la llave `read-sty`. El valor de esta llave *siempre* debe pasarse entre llaves '`{ }`'.

`concept` = {⟨*true* | *false*⟩} default: *true*

Establece si se *silencia* la verbalización en `NVDA/MathCAT` del *«concepto»* establecido por la llave `read-sty` al ejecutar el comando sin modificar la impresión de este. Si se establece en `true`, `NVDA/MathCAT` verbalizará el concepto; si se establece en `false`, `NVDA/MathCAT` solo verbalizará el {⟨*argumento*⟩} acorde al valor establecido por la llave `read-cmd`.

`read-arg` = {⟨*verbalización* NVDA⟩} default: *no usado*

Sobrescribe el *concepto* establecido por la llave `read-sty` que será verbalizado por `NVDA/MathCAT` al ejecutar el comando sin modificar la impresión de este. El valor de esta llave *siempre* debe pasarse entre llaves '`{ }`'. Por ejemplo: `$\spside[read-arg={hipotenusa}]{AB}$` imprimirá '$\overline{AB}$' en la salida PDF y `NVDA/MathCAT` verbalizará *«hipotenusa A B»*.

`silent-cmd` = {⟨*true* | *false*⟩} default: *false*

Establece si se *silencia* la verbalización en `NVDA/MathCAT` del *«concepto»* establecido por la llave `read-sty` y del {⟨*argumento*⟩} pasado al comando sin modificar la impresión de este. Si se establece en `true`, `NVDA/MathCAT` NO verbalizará la salida del comando; si se establece en `false`, `NVDA/MathCAT` sí verbalizará la salida del comando.

El comando `\spmside`

`\spmside`

`\spmside`{⟨*punto, punto*⟩}
`\spmside`[⟨*key* = *val*⟩]{⟨*punto, punto*⟩}

El comando `\spmside` trabaja en *modo matemático* y recibe el *argumento obligatorio* {⟨*punto, punto*⟩}. Representa la *medida* del *lado*, *segmento* o *trazo* determinado por dos puntos. La distinción entre `\spside` y `\spmside` es semántica: `\spside` representa el *objeto geométrico* '$\overline{AB}$'; `\spmside` representa su *medida* (un número real positivo).

Si no se proporciona el *argumento opcional* [⟨*key* = *val*⟩], `$\spmside{AB}$` imprimirá 'AB' en la salida PDF y `NVDA/MathCAT` verbalizará *«medida del lado A B»*.

Llaves para `\spmside`

`read-cmd` = {⟨*school* | *mathcat*⟩} default: *school*

Establece la *forma* en la que `NVDA/MathCAT` verbalizará al argumento {⟨*punto, punto*⟩}. Si se establece en `school`, `NVDA/MathCAT` verbaliza las letras del argumento sin el prefijo «mayúscula»: `$\spmside{AB}$` se verbaliza *«medida del lado A B»*; si se establece en `mathcat`, `NVDA/MathCAT` aplica sus propias reglas, que pueden incluir el prefijo «mayúscula» dependiendo de la configuración del usuario.

`read-sty` = {⟨*lado* | *segmento* | *trazo*⟩} default: *lado*

Establece el *concepto* que verbalizará `NVDA/MathCAT` después de verbalizar el término *«medida del»* al ejecutar el comando sin modificar la impresión de este. Si se utiliza `$\spmside[read-sty=lado]{AB}$`, `NVDA/MathCAT` verbalizará *«medida del lado A B»*; si se utiliza `$\spmside[read-sty=segmento]{AB}$`, `NVDA/MathCAT` verbalizará *«medida del segmento A B»* y si se utiliza `$\spmside[read-sty=trazo]{AB}$`, `NVDA/MathCAT` verbalizará *«medida del trazo A B»*

`prefix` = {⟨*prefijo*⟩} default: *no usado*

Establece el *prefijo* que será verbalizado por `NVDA/MathCAT` *antes* de verbalizar el término *«medida del»* seguido por el *«concepto»* establecido por la llave `read-sty`. El valor de esta llave *siempre* debe pasarse entre llaves '`{ }`'.

`concept` = {⟨*true* | *false*⟩} default: *true*

Establece si se *silencia* la verbalización en `NVDA/MathCAT` del término *«medida del»* y del *«concepto»* establecido por la llave `read-sty` al ejecutar el comando sin modificar la impresión de este. Si se establece en `true`, `NVDA/MathCAT` verbalizará el término *«medida del»* antes del concepto; si se establece en `false`, `NVDA/MathCAT` solo verbalizará el {⟨*argumento*⟩} acorde al valor establecido por la llave `read-cmd`.

`read-arg` = {⟨*verbalización* NVDA⟩} default: *no usado*

Sobrescribe el *concepto* establecido por la llave `read-sty` que será verbalizado por `NVDA/MathCAT` al ejecutar el comando sin modificar la impresión de este. El valor de esta llave *siempre* debe pasarse entre llaves '`{ }`' y se debe usar en conjunto con la llave `concept`.
Por ejemplo: `$\spmside[concept=false,read-arg={medida de la cuerda}]{AB}$` imprimirá '$\overline{AB}$' en la salida `PDF` y `NVDA/MathCAT` verbalizará *«medida de la cuerda A B»*.

`print-m` = {⟨*true* | *false*⟩} default: *false*

Establece *si se imprime* la notación visual 'm' antes aplicar `\overline` al {⟨*argumento*⟩}. Si se establece en `true`, `$\spmside[print-m=true]{AB}$` imprimirá '$\mathrm{m}\,\overline{AB}$' en la salida `PDF`, si se establece en `false` imprimirá 'AB' en la salida `PDF`. En ambos casos `NVDA/MathCAT` verbalizará *«medida del lado A B»*.

`silent-cmd` = {⟨*true* | *false*⟩} default: *false*

Establece si se *silencia* la verbalización en `NVDA/MathCAT` del término *«medida del»*, del *«concepto»* establecido por la llave `read-sty` y del {⟨*argumento*⟩} pasado al comando sin modificar la impresión de este. Si se establece en `true`, `NVDA/MathCAT` NO verbalizará la salida del comando; si se establece en `false`, `NVDA/MathCAT` sí verbalizará la salida del comando.

El comando `\spangle`

`\spangle`

`\spangle`{⟨*punto* | *letra griega* | *número*⟩} `\spangle`{⟨*punto, punto, punto*⟩}
`\spangle`[⟨*key* = *val*⟩]{⟨*punto* | *letra griega* | *número*⟩} `\spangle`[⟨*key* = *val*⟩]{⟨*punto, punto, punto*⟩}

El comando `\spangle` trabaja en *modo matemático* y recibe un *argumento obligatorio* el cual puede ser de la forma {⟨*punto* | *letra griega* | *número*⟩} o {⟨*punto, punto, punto*⟩} aplicando `\angle` o `\rightangle` en la impresión.

Si no se proporciona el *argumento opcional* [⟨*key* = *val*⟩], `$\spangle{A}$` imprimirá '$\angle A$' en la salida `PDF` y `NVDA/MathCAT` verbalizará *«ángulo en A»*; `$\spangle{\alpha}$` imprimirá '$\angle\alpha$' en la salida `PDF` y `NVDA/MathCAT` verbalizará *«ángulo alfa»* y `$\spangle{A, O, B}$` imprimirá '$\angle AOB$' en la salida `PDF` y `NVDA/MathCAT` verbalizará *«ángulo A O B»*.

Llaves para `\spangle`

`read-cmd` = {⟨*school* | *mathcat*⟩} default: *school*

Establece la *forma* en la que `NVDA/MathCAT` verbalizará al argumento {⟨*punto*⟩} o {⟨*punto, punto, punto*⟩}. Si se establece en `school`, `NVDA/MathCAT` verbaliza las letras del argumento sin el prefijo «mayúscula»: `$\spangle{A, O, B}$` se verbaliza *«ángulo A O B»*; si se establece en `mathcat`, `NVDA/MathCAT` aplica sus propias reglas, que pueden incluir el prefijo «mayúscula» dependiendo de la configuración del usuario.

`prefix` = {⟨*prefijo*⟩} default: *no usado*

Establece el *prefijo* que será verbalizado por `NVDA/MathCAT` *antes* de verbalizar el término *«ángulo»*. El valor de esta llave *siempre* debe pasarse entre llaves '`{ }`'.

`silent-cmd` = {⟨*true* | *false*⟩} default: *false*

Establece si se *silencia* la verbalización en `NVDA/MathCAT` del término *«ángulo»*. Si se establece en `true`, `NVDA/MathCAT` NO verbalizará el término; si se establece en `false`, `NVDA/MathCAT` SÍ lo verbalizará.

`recto` = {⟨*true* | *false*⟩} default: *false*

Establece si se usará `\rightangle` en la impresión añadiendo el término *«recto»* a la verbalización en `NVDA/MathCAT`. Si se establece en `true`, `$\spangle[recto=true]{A}$` imprimirá '∟A' en la salida `PDF` y `NVDA/MathCAT` verbalizará *«ángulo recto en A»*, `$\spangle[recto=true]{A, O, B}$` imprimirá '∟AOB' en la salida `PDF` y `NVDA/MathCAT` verbalizará *«ángulo recto A O B»*; si se establece en `false`, imprimirá usando `\angle` en la salida `PDF` y `NVDA/MathCAT` omitirá el término *«recto»*.

El comando `\spmangle`

`\spmangle`

`\spmangle`{⟨*punto* | *letra griega* | *número*⟩} `\spmangle`{⟨*punto, punto, punto*⟩}
`\spmangle`[⟨*key* = *val*⟩]{⟨*punto* | *letra griega* | *número*⟩} `\spmangle`[⟨*key* = *val*⟩]{⟨*punto, punto, punto*⟩}

El comando `\spmangle` trabaja en *modo matemático* y recibe un *argumento obligatorio* el cual puede ser de la forma {⟨*punto* | *letra griega* | *número*⟩} o {⟨*punto, punto, punto*⟩} aplicando `\measuredangle` o `\measuredrightangle` en la impresión. La distinción entre `\spangle` y `\spmangle` es semántica: `\spangle` representa el *objeto geométrico* '∠AOB'; `\spmangle` representa su *medida* (un valor en grados).

Si no se proporciona el *argumento opcional* [⟨*key* = *val*⟩], `$\spmangle{A}$` imprimirá '∡A' en la salida `PDF` y `NVDA/MathCAT` verbalizará *«medida del ángulo en A»*; `$\spmangle{\alpha}$` imprimirá '∡α' en la salida `PDF` y `NVDA/MathCAT` verbalizará *«medida del ángulo alfa»* y `$\spmangle{A, O, B}$` imprimirá '∡AOB' en la salida `PDF` y `NVDA/MathCAT` verbalizará *«medida del ángulo A O B»*.

Llaves para `\spmangle`

`read-cmd` = {⟨*school* | *mathcat*⟩} default: *school*

Establece la *forma* en la que `NVDA/MathCAT` verbalizará al argumento {⟨*punto*⟩} o {⟨*punto, punto, punto*⟩}. Si se establece en `school`, `NVDA/MathCAT` verbaliza las letras del argumento sin el prefijo «mayúscula»: `$\spmangle{A, O, B}$` se verbaliza *«medida del ángulo A O B»*; si se establece en `mathcat`, `NVDA/MathCAT` aplica sus propias reglas, que pueden incluir el prefijo «mayúscula» dependiendo de la configuración del usuario.

`prefix` = {⟨*prefijo*⟩} default: *no usado*

Establece el *prefijo* que será verbalizado por `NVDA/MathCAT` *antes* de verbalizar el término *«medida del ángulo»*. El valor de esta llave *siempre* debe pasarse entre llaves '`{ }`'.

`silent-cmd` = {⟨*true* | *false*⟩} default: *false*

Establece si se *silencia* la verbalización en `NVDA/MathCAT` del término *«medida del ángulo»*. Si se establece en `true`, `NVDA/MathCAT` NO verbalizará el término; si se establece en `false`, `NVDA/MathCAT` SÍ lo verbalizará.

`recto` = {⟨*true* | *false*⟩} default: *false*

Establece si se usará `\measuredrightangle` en la impresión añadiendo el término *«recto»* a la verbalización en `NVDA/MathCAT`. Si se establece en `true`, `$\spmangle[recto=true]{A}$` imprimirá '⊾A' en la salida `PDF` y `NVDA/MathCAT` verbalizará *«medida del ángulo recto en A»*, `$\spmangle[recto=true]{A, O, B}$` imprimirá '⊾AOB' en la salida `PDF` y `NVDA/MathCAT` verbalizará *«medida del ángulo recto A O B»*; si se establece en `false`, imprimirá usando `\measuredangle` en la salida `PDF` y `NVDA/MathCAT` omitirá el término *«recto»*.

`print-m` = {⟨*true* | *false*⟩} default: *false*

Establece *si se imprime* la notación visual 'm' antes del argumento. Si se establece en `true`, `$\spmangle[print-m=true]{A,` imprimirá 'm∠AOB' en la salida `PDF`, si se establece en `false` imprimirá '∡AOB' en la salida `PDF`. En ambos casos `NVDA/MathCAT` verbalizará *«medida del ángulo A O B»*.

El comando `\spang`

`\spang`

`\spang`{⟨*grados*⟩}
`\spang`{⟨*grados* : *minutos*⟩}
`\spang`{⟨*grados* : *minutos* : *segundos*⟩}

El comando `\spang` trabaja en *modo matemático* y recibe el *argumento obligatorio* {⟨*grados*⟩}, {⟨*grados* : *minutos*⟩} o {⟨*grados* : *minutos* : *segundos*⟩}. Los procesa internamente bajo los comandos `\spnum` y `\spunit` para imprimir y verbalizar el *«ángulo sexagesimal»*. Por ejemplo, `$\spang{45:30:15}$` imprimirá '45°30′15″' en la salida `PDF` y `NVDA/MathCAT` verbalizará *«cuarenta y cinco grados treinta minutos quince segundos»*.

El comando `\sparc`

`\sparc`

`\sparc`{⟨*punto, punto*⟩} `\sparc`{⟨*punto, punto, punto*⟩}
`\sparc`[⟨*key* = *val*⟩]{⟨*punto, punto*⟩} `\sparc`[⟨*key* = *val*⟩]{⟨*punto, punto, punto*⟩}

El comando `\sparc` trabaja en *modo matemático* y recibe un *argumento obligatorio* el cual puede ser de la forma {⟨*punto, punto*⟩} o {⟨*punto, punto, punto*⟩}. Representa un arco determinado por dos o tres puntos aplicando `\widearc` en la impresión.

Si no se proporciona el *argumento opcional* [⟨*key* = *val*⟩], `$\sparc{A, B}$` imprimirá '$\widehat{AB}$' en la salida `PDF` y `NVDA/MathCAT` verbalizará *«arco A B»*.

Llaves para `\sparc`

`read-cmd` = {⟨*school* | *mathcat*⟩} default: *school*

Establece la *forma* en la que `NVDA/MathCAT` verbalizará al argumento {⟨*punto*, *punto*⟩} o {⟨*punto*, *punto*, *punto*⟩}. Si se establece en `school`, `NVDA/MathCAT` verbaliza las letras del argumento sin el prefijo «mayúscula»: `$\sparc{A, B}$` se verbaliza *«arco A B»*; si se establece en `mathcat`, `NVDA/MathCAT` aplica sus propias reglas, que pueden incluir el prefijo «mayúscula» dependiendo de la configuración del usuario.

`prefix` = {⟨*prefijo*⟩} default: *no usado*

Establece el *prefijo* que será verbalizado por `NVDA/MathCAT` *antes* de verbalizar el término *«arco»*. El valor de esta llave *siempre* debe pasarse entre llaves '`{ }`'.

`silent-cmd` = {⟨*true* | *false*⟩} default: *false*

Establece si se *silencia* la verbalización en `NVDA/MathCAT` del término *«arco»*. Si se establece en `true`, `NVDA/MathCAT` NO verbalizará el término; si se establece en `false`, `NVDA/MathCAT` SÍ lo verbalizará.

El comando `\spmarc`

`\spmarc`

`\spmarc`{⟨*punto*, *punto*⟩} `\spmarc`{⟨*punto*, *punto*, *punto*⟩}
`\spmarc`[⟨*key* = *val*⟩]{⟨*punto*, *punto*⟩} `\spmarc`[⟨*key* = *val*⟩]{⟨*punto*, *punto*, *punto*⟩}

El comando `\spmarc` trabaja en *modo matemático* y recibe un *argumento obligatorio* el cual puede ser de la forma {⟨*punto*, *punto*⟩} o {⟨*punto*, *punto*, *punto*⟩}. Representa la *medida* del *arco* determinado por dos o tres puntos. La distinción entre `\sparc` y `\spmarc` es semántica: `\sparc` representa el *objeto geométrico* '$\widehat{AB}$'; `\spmarc` representa su *medida* (un valor en grados).

Si no se proporciona el *argumento opcional* [⟨*key* = *val*⟩], `$\spmarc{A, B}$` imprimirá 'm$\widehat{AB}$' en la salida `PDF` y `NVDA/MathCAT` verbalizará *«medida del arco A B»*.

Llaves para `\spmarc`

`read-cmd` = {⟨*school* | *mathcat*⟩} default: *school*

Establece la *forma* en la que `NVDA/MathCAT` verbalizará al argumento {⟨*punto*, *punto*⟩} o {⟨*punto*, *punto*, *punto*⟩}. Si se establece en `school`, `NVDA/MathCAT` verbaliza las letras del argumento sin el prefijo «mayúscula»: `$\spmarc{A, B}$` se verbaliza *«medida del arco A B»*; si se establece en `mathcat`, `NVDA/MathCAT` aplica sus propias reglas, que pueden incluir el prefijo «mayúscula» dependiendo de la configuración del usuario.

`prefix` = {⟨*prefijo*⟩} default: *no usado*

Establece el *prefijo* que será verbalizado por `NVDA/MathCAT` *antes* de verbalizar el término *«medida del arco»*. El valor de esta llave *siempre* debe pasarse entre llaves '`{ }`'.

`silent-cmd` = {⟨*true* | *false*⟩} default: *false*

Establece si se *silencia* la verbalización en `NVDA/MathCAT` del término *«medida del arco»*. Si se establece en `true`, `NVDA/MathCAT` NO verbalizará el término; si se establece en `false`, `NVDA/MathCAT` SÍ lo verbalizará.

El comando `\spPoly`

`\spPoly`

`\spPoly`{⟨*punto*, *punto*, *punto*, ...⟩}
`\spPoly`[⟨*key* = *val*⟩]{⟨*punto*, *punto*, *punto*, ...⟩}

El comando `\spPoly` trabaja en *modo matemático* y recibe el *argumento obligatorio* {⟨*punto*, *punto*, *punto*, ...⟩}, una lista de al menos tres {⟨*puntos*⟩} separados por comas. El tipo de *símbolo* en la impresión y la verbalización se determinan acorde al número de puntos en el argumento: Si son tres puntos se aplica `\triangle` y añade el término *«triángulo»* a la verbalización; si son cuatro puntos se aplica `\mdlgwhtsquare` y añade el término *«cuadrado»* a la verbalización; si son cinco o más puntos no se aplica ningún símbolo ni se añade verbalización.

Si no se proporciona el *argumento opcional* [⟨*key* = *val*⟩], `$\spPoly{A, B, C}$` imprimirá '$\triangle ABC$' en la salida `PDF` y `NVDA/MathCAT` verbalizará *«triángulo A B C»*; `$\spPoly{A, B, C, D}$` imprimirá '$\square ABCD$' y verbalizará *«cuadrado A B C D»*; `$\spPoly{A, B, C, D, E}$` imprimirá '$ABCDE$' y verbalizará *«A B C D E»*.

Llaves para `\spPoly`

`read-cmd` = {⟨*school* | *mathcat*⟩} default: *school*

Establece la *forma* en la que `NVDA/MathCAT` verbalizará al argumento {⟨*punto*, *punto*, *punto*, ...⟩}. Si se establece en `school`, `NVDA/MathCAT` verbaliza las letras del argumento sin el prefijo «mayúscula»: `$\spPoly{A, B, C}$` se verbaliza *«triángulo A B C»*; si se establece en `mathcat`, `NVDA/MathCAT` aplica sus propias reglas, que pueden incluir el prefijo «mayúscula» dependiendo de la configuración del usuario.

`prefix` = {⟨*prefijo*⟩} default: *no usado*

Establece el *prefijo* que será verbalizado por `NVDA/MathCAT` *antes* de verbalizar el argumento o los términos *«tríangulo»* y *«cuadrado»* establecido por la llave `print-sym`. El valor de esta llave *siempre* debe pasarse entre llaves '`{ }`'.

`print-sym` = {⟨*true* | *false*⟩} default: *true*

Establece si se usará `\triangle` o `\mdlgwhtsquare` en la impresión añadiendo el término correspondiente a la verbalización en `NVDA/MathCAT`. Si se establece en `true`, `$\spPoly[print-sym=true]{A, B, C}$` imprimirá '$\triangle ABC$' en la salida `PDF` y `NVDA/MathCAT` verbalizará *«triángulo A B C»*; si se establece en `false`, imprimirá 'ABC' en la salida `PDF` y `NVDA/MathCAT` omitirá el término *«triángulo»*, verbalizando solo *«A B C»*.

`silent-cmd` = {⟨*true* | *false*⟩} default: *false*

Alias de `print-sym=false`, incluido por consistencia con el resto de los comandos de geometría plana.

El comando `\spAfig`

`\spAfig`

`\spAfig` ⟨*punto, punto, punto, ...*⟩)
`\spAfig`(⟨*número* | *letra* | *letra_{número}*⟩)
`\spAfig`[⟨*key = val*⟩](⟨*punto, punto, punto, ...*⟩)

El comando `\spAfig` trabaja en *modo matemático* y recibe el *argumento delimitado por paréntesis*, el cual es *completamente opcional*, representando el *área* de una figura poligonal en la impresión aplicando `\mathcal{A}`.

El argumento admite tres formas: una *lista de vértices* (⟨*punto, punto, punto, ...*⟩) (al menos tres); un *número o letra sueltos* (⟨*1*⟩), (⟨*n*⟩); o una *letra de una tabla fija* (`F`, `P`, `B`, `L`) con subíndice numérico opcional, (⟨*F*⟩), (⟨*F_{1}*⟩).

Si no se proporciona el *argumento opcional* [⟨*key = val*⟩] ni el argumento delimitado por paréntesis, `$\spAfig$` imprimirá '$\mathcal{A}$' en la salida `PDF` y `NVDA/MathCAT` verbalizará *«área»*. Con vértices, `$\spAfig(A, B, C)$` imprimirá '$\mathcal{A}_{\triangle ABC}$' y verbalizará *«área del triángulo A B C»*. Con un número o letra sueltos, `$\spAfig(1)$` imprimirá '$\mathcal{A}_1$' y verbalizará *«área uno»*; `$\spAfig(n)$` imprimirá '$\mathcal{A}_n$' y verbalizará *«área n»*. Con una letra de la tabla, `$\spAfig(F)$` imprimirá '$\mathcal{A}_F$' y verbalizará *«área de la figura»*; `$\spAfig(F_{1})$` imprimirá '$\mathcal{A}_{F_1}$' y verbalizará *«área de la figura uno»*.

Llaves para `\spAfig`

`read-cmd` = {⟨*school* | *mathcat*⟩} default: *school*

Establece la *forma* en la que `NVDA/MathCAT` verbalizará al argumento. Si se establece en `school`, `NVDA/MathCAT` verbaliza las letras del argumento sin el prefijo «mayúscula»: `$\spAfig(A, B, C)$` se verbaliza *«área del triángulo A B C»*; si se establece en `mathcat`, `NVDA/MathCAT` aplica sus propias reglas, que pueden incluir el prefijo «mayúscula» dependiendo de la configuración del usuario.

`prefix` = {⟨*prefijo*⟩} default: *no usado*

Establece el *prefijo* que será verbalizado por `NVDA/MathCAT` *antes* de verbalizar el término *«área»*. El valor de esta llave *siempre* debe pasarse entre llaves '`{ }`'.

`print-sym` = {⟨*auto* | *triangle* | *square* | *none*⟩} default: *auto*

Establece si se usará `\triangle` o `\mdlgwhtsquare` en la impresión. Con vértices, `auto` infiere el símbolo según el número de puntos (tres o cuatro); `none` lo omite y `NVDA/MathCAT` verbaliza solo los vértices sin nombrar la figura: `$\spAfig[print-sym=none](A, B, C)$` imprimirá '$\mathcal{A}_{ABC}$' y verbalizará *«área A B C»*. Forzar `triangle`/`square` exige que coincida con el número real de vértices; en caso contrario se produce un error. Con un número o letra sueltos, `auto` y `none` se comportan igual (sin símbolo); `triangle`/`square` anida el argumento bajo el símbolo: `$\spAfig[print-sym=triangle](1)$` imprimirá '$\mathcal{A}_{\triangle_1}$' y verbalizará *«área del triángulo uno»*. Esta llave no puede combinarse con una letra de la tabla (`F`, `P`, `B`, `L`).

`read-sub` = {⟨*true* | *false*⟩} default: *false*

Establece si `NVDA/MathCAT` antepone la palabra «sub» al verbalizar el número o letra del argumento único. Si se establece en `true`, `$\spAfig[read-sub=true](1)$` verbaliza *«área sub uno»*; si se establece en `false`, verbaliza *«área uno»*. No tiene efecto en la notación de vértices ni cuando el argumento es una letra de la tabla sin subíndice.

`silent-cmd` = {⟨*true* | *false*⟩} default: *false*

Establece si se *silencia por completo* la verbalización en `NVDA/MathCAT`, sin afectar la impresión. Si se establece en `true`, `NVDA/MathCAT` NO verbalizará nada; si se establece en `false`, `NVDA/MathCAT` verbalizará normalmente.

`read-arg` = {⟨*nombre*⟩} default: *no usado*

Sobrescribe el *nombre de la figura* determinado automáticamente por el número de puntos en el argumento, que será verbalizado por `NVDA/MathCAT` al ejecutar el comando sin modificar la impresión de este. El valor de esta llave *siempre* debe pasarse entre llaves '`{ }`'. Por ejemplo: `$\spAfig[read-arg={trapecio}](A, B, C, D)$` imprimirá '$\mathcal{A}_{\square ABCD}$' en la salida `PDF` y `NVDA/MathCAT` verbalizará *«área del trapecio A B C D»*.

El comando \spPfig

\spPfig

\spPfig
\spPfig(⟨*punto, punto, punto, ...*⟩)
\spPfig(⟨*número* | *letra* | *letra_{número}*⟩)
\spPfig[⟨*key = val*⟩](⟨*punto, punto, punto, ...*⟩)

El comando \spPfig trabaja en *modo matemático* y recibe el *argumento delimitado por paréntesis*, el cual es *completamente opcional*, representando el *perímetro* de una figura poligonal en la impresión aplicando \mathcal{P}.

El argumento admite las mismas tres formas que \spAfig (§7): una *lista de vértices*, un *número o letra sueltos*, o una *letra de la tabla fija* (F, P, B, L) con subíndice numérico opcional.

Si no se proporciona el *argumento opcional* [⟨*key = val*⟩] ni el argumento delimitado por paréntesis, $\spPfig$ imprimirá '$\mathcal{P}$' en la salida PDF y NVDA/MathCAT verbalizará *«perímetro»*. Con vértices, $\spPfig(A, B, C)$ imprimirá '$\mathcal{P}_{\triangle ABC}$' y verbalizará *«perímetro del triángulo A B C»*. Con un número o letra sueltos, $\spPfig(2)$ imprimirá '$\mathcal{P}_2$' y verbalizará *«perímetro dos»*. Con una letra de la tabla, $\spPfig(P_{2})$ imprimirá '$\mathcal{P}_{P_2}$' y verbalizará *«perímetro del polígono dos»*.

Llaves para \spPfig

read-cmd = {⟨*school* | *mathcat*⟩} default: *school*

Establece la *forma* en la que NVDA/MathCAT verbalizará al argumento. Si se establece en school, NVDA/MathCAT verbaliza las letras del argumento sin el prefijo «mayúscula»: $\spPfig(A, B, C)$ se verbaliza *«perímetro del triángulo A B C»*; si se establece en mathcat, NVDA/MathCAT aplica sus propias reglas, que pueden incluir el prefijo «mayúscula» dependiendo de la configuración del usuario.

prefix = {⟨*prefijo*⟩} default: *no usado*

Establece el *prefijo* que será verbalizado por NVDA/MathCAT *antes* de verbalizar el término *«perímetro»*. El valor de esta llave *siempre* debe pasarse entre llaves '{ }'.

print-sym = {⟨*auto* | *triangle* | *square* | *none*⟩} default: *auto*

Establece si se usará \triangle o \mdlgwhtsquare en la impresión, con la misma semántica que en \spAfig (§7).

read-sub = {⟨*true* | *false*⟩} default: *false*

Establece si NVDA/MathCAT antepone la palabra «sub» al verbalizar el número o letra del argumento único, con la misma semántica que en \spAfig (§7).

silent-cmd = {⟨*true* | *false*⟩} default: *false*

Establece si se *silencia por completo* la verbalización en NVDA/MathCAT, sin afectar la impresión. Si se establece en true, NVDA/MathCAT NO verbalizará nada; si se establece en false, NVDA/MathCAT verbalizará normalmente.

read-arg = {⟨*nombre*⟩} default: *no usado*

Sobrescribe el *nombre de la figura* determinado automáticamente por el número de puntos en el argumento, que será verbalizado por NVDA/MathCAT al ejecutar el comando sin modificar la impresión de este. El valor de esta llave *siempre* debe pasarse entre llaves '{ }'. Por ejemplo: $\spPfig[read-arg={trapecio}](A, B, C, D)$ imprimirá '$\mathcal{P}_{\square ABCD}$' en la salida PDF y NVDA/MathCAT verbalizará *«perímetro del trapecio A B C D»*.

8 Referencias

[1] Robertson, Will . "unicode-math - Unicode mathematics support for XeTeX and LuaTeX". Available from CTAN, https://www.ctan.org/pkg/unicode-math, 2023.

[2] Krueger, Marcel. "LuaMML - Automatically generate MathML from LuaTeX math mode material". Available from CTAN, https://ctan.org/pkg/luamml, 2026.

[3] The LaTeX Project. "fontspec - Advanced font selection for XeLaTeX and LuaTeX". Available from CTAN, https://www.ctan.org/pkg/fontspec, 2025.

[4] Voß, Herbert. "libertinus-otf - Support for Libertinus OpenType". Available from CTAN, https://www.ctan.org/pkg/libertinus-otf, 2025.

[5] Flipo, Daniel. "erewhon-math - Utopia based OpenType Math font". Available from CTAN, https://ctan.org/pkg/erewhon-math, 2026.

[6] Hofstra, Silke. "sourcecodepro - Use SourceCodePro with TeX(-alike) systems". Available from CTAN, https://ctan.org/pkg/sourcecodepro, 2025.

[7] González, Pablo. "scontents - Stores LaTeX contents in memory or files". Available from CTAN, https://www.ctan.org/pkg/scontents, 2026.

[8] González, Pablo. "enumext - Enumerate exercise sheets". Available from CTAN, https://www.ctan.org/pkg/enumext, 2026.

[9] Braams, Johannes and Bezos, Javier. "babel - Multilingual support for LaTeX, LuaLaTeX, XeLaTeX, and Plain TeX". Available from CTAN, https://www.ctan.org/pkg/babel, 2026.

[10] Bezos, Javier. "babel-spanish - Babel support for Spanish". Available from CTAN, https://www.ctan.org/pkg/babel-spanish, 2026.

[11] NV Access. "NVDA - NonVisual Desktop Access screen reader". Available from NV ACCESS, https://www.nvaccess.org/, 2026.

[12] Soiffer, Neil. "MathCAT - Math to speech and braille translation". Available from GITHUB, https://nsoiffer.github.io/MathCAT/, 2026.

[13] Adobe Inc. "Adobe Acrobat Reader - Reliable PDF viewer". Available from ADOBE, https://get.adobe.com/reader/, 2026.

[14] ISO. "ISO 32000-2:2020 - Portable Document Format (PDF 2.0)". Available from PDF ASSOCIATION, https://pdfa.org/resource/iso-32000-2/, 2020.

[15] PDF association. "WTPDF - Well-Tagged PDF specification". Available from PDFA, https://pdfa.org/wtpdf/, 2024.

[16] PDF association. "PDF 2.0 Errata Collection 3". Available from PDFA, https://pdfa.org/PDF-2-0-errata-collection-3-now-available/, 2026.

[17] LaTeX Project LWG. "Best Practice Guide: Math in PDF". Available from PDFA, https://pdfa.org/resource/best-practice-guide-math-in-PDF/, 2026.

[18] Johnson, Duff. "PDF 2.0: New Features, Real-World Impact". Available from PDFA, https://pdfa.org/PDF-2-0-new-features-real-world-impact/, 2026.

[19] The LaTeX Project. "LaTeX Tagged PDF Project - Documentation and resources". Available from LATEX PROJECT, https://latex3.github.io/tagging-project/, 2026.

[20] veraPDF Consortium. "veraPDF - Industry supported PDF/A and PDF/UA validator". Available from VERAPDF, https://verapdf.org/, 2026.

[21] Dual Lab. "ngPDF - Next Generation PDF demonstrator and validator". Available from NGPDF, https://ngpdf.com/, 2026.

[22] Berry, Karl. "LaTeX 2_ε: An Unofficial Reference Manual". Available from CTAN, https://ctan.org/pkg/latex2e-help-texinfo, 2026.

[23] The LaTeX Project. "Extensive support for hypertext in LaTeX". Available from CTAN, https://www.ctan.org/pkg/hyperref, 2026.

[24] The LaTeX Project. "The expl3 package". Available from CTAN, https://www.ctan.org/pkg/l3kernel, 2026.

[25] The LaTeX Project. "The LaTeX3 Interfaces". Available from CTAN, https://www.ctan.org/pkg/l3kernel, 2026.

[26] The LaTeX Project. "The LaTeX 2_ε sources". Available from CTAN, https://ctan.org/tex-archive/macros/latex/base, 2026.

[27] The LaTeX Project. "LaTeX News, Issue 41, June 2025". Available from CTAN, https://ctan.org/tex-archive/macros/latex/base, 2025.

[28] The LaTeX Project. "LaTeX for authors current version". Available from CTAN, https://ctan.org/pkg/latex-base, 2026.

[29] Fischer, Ulrike. "tagpdf – LaTeX kernel code for PDF tagging". Available from CTAN, https://www.ctan.org/pkg/tagpdf, 2026.

[30] The LaTeX Project. "latex-lab – LaTeX laboratory". Available from CTAN, https://www.ctan.org/pkg/latex-lab, 2026.

9 Change history

v0.99 (ctan), 2026-09-17 – First public release.

10 Índice de la Documentación

Los números en cursiva indican las páginas donde se describe la entrada correspondiente.

C

Document class:
- article 1
- book 1
- letter 1
- report 1

Commands provide by spintent:
- \setspintent 3
- \spAfig 23
- \spPfig 24
- \spPoint 17
- \spPoly 22
- \spangle 20
- \spang 21
- \sparc 21
- \spdate 11
- \spdiv 13
- \spdsym 12
- \spmQ 16
- \spmangle 21
- \spmarc 22
- \spmcd 15
- \spmcm 15
- \spmoney 6
- \spmside 19
- \spmsym 12
- \spmul 13
- \spnD 14
- \spnM 14
- \spnZ 16
- \spnewunit 6
- \spnum 3
- \sprayo 18
- \spread 11
- \sprecta 18
- \spshort 7
- \spside 19
- \spsiglo 11
- \sptime 11
- \spunitalias 6
- \spunit 4
- \spusep 12

K

Keys for \spAfig provide by spintent:
- prefix 23
- print-sym 23
- read-arg 23
- read-cmd 23
- read-sub 23
- silent-cmd 23

Keys for \spangle provide by spintent:
- prefix 20
- read-cmd 20
- recto 21
- silent-cmd 20

Keys for \sparc provide by spintent:
- prefix 22
- read-cmd 22
- silent-cmd 22

Keys for \spdiv provide by spintent:
- read-parens 13
- wrap-parens 13

Keys for \spdsym provide by spintent:
- prefix 12
- read-sym 12
- set-sym 12
- suffix 12

Keys for \spmangle provide by spintent:
- prefix 21
- print-m 21
- read-cmd 21
- recto 21
- silent-cmd 21

Keys for \spmarc provide by spintent:
- prefix 22
- read-cmd 22
- silent-cmd 22

Keys for \spmcd provide by spintent:
- prefix 15
- read-cmd 15
- result 15
- sample-arg 15
- silent-cmd 15
- upper-case 15

Keys for \spmcm provide by spintent:
- prefix 16
- read-cmd 16
- result 16
- sample-arg 16
- silent-cmd 16
- upper-case 16

Keys for \spmoney provide by spintent:
- currency 7
- dolar-math 7
- sub-units 7

Keys for \spmQ provide by spintent:
- join-word 17
- print-dfrac 17
- read-parens 17
- use-spnZ 17
- wrap-parens 17

Keys for \spmside provide by spintent:
- concept 20
- prefix 20
- print-m 20
- read-arg 20
- read-cmd 20
- read-sty 20
- silent-cmd 20

Keys for \spmsym provide by spintent:
- not-print 13
- prefix 13
- read-sym 13
- set-sym 13
- suffix 13

Keys for \spmul provide by spintent:
- read-parens 13
- wrap-parens 13

Keys for \spnD provide by spintent:
- prefix 14

result-num 14
result 14
sample-arg 14
silent-cmd 14
Keys for \spnM provide by spintent:
prefix 14
result-num 14
result 14
sample-arg 14
silent-cmd 15
Keys for \spnum provide by spintent:
cifras-sep 3
notation-sym 4
read-period-sep 3
read-sign 3
Keys for \spnZ provide by spintent:
read-parens 16
wrap-parens 16
Keys for \spPfig provide by spintent:
prefix 24
print-sym 24
read-arg 24
read-cmd 24
read-sub 24
silent-cmd 24
Keys for \spPoint provide by spintent:
concept 17
prefix 17
read-cmd 17
silent-cmd 18
Keys for \spPoly provide by spintent:
prefix 22
print-sym 22
read-cmd 22
silent-cmd 23
Keys for \sprayo provide by spintent:
concept 18
prefix 18
read-cmd 18
read-sty 18
silent-cmd 19
Keys for \sprecta provide by spintent:
concept 18
prefix 18
read-cmd 18
silent-cmd 18
Keys for \spshort provide by spintent:
read-arg 9
small-caps 9
Keys for \spside provide by spintent:
concept 19
prefix 19
read-arg 19
read-cmd 19
read-sty 19
silent-cmd 19
Keys for \spsiglo provide by spintent:
print-era 11
Keys for \spunit provide by spintent:
print-cdot 4
read-cdot 4
read-slash 4
Keys for \spusep provide by spintent:
silent 12
size 12
Keys for \spdiv provide by spintent:
cifras-sep 13
notation-sym 13
read-period-sep 13
read-sign 13
read-sym 13
set-sym 13
Keys for \spdsym provide by spintent:
read-sym 12
Keys for \spmcd provide by spintent:
read-cmd 15
Keys for \spmcm provide by spintent:
read-cmd 16
Keys for \spmoney provide by spintent:
cifras-sep 7
Keys for \spmQ provide by spintent:
wrap-parens 17
Keys for \spmsym provide by spintent:
read-sym 13
Keys for \spmul provide by spintent:
cifras-sep 13
notation-sym 13
read-period-sep 13
read-sign 13
read-sym 13
set-sym 13
Keys for \spnZ provide by spintent:
cifras-sep 16
read-sign 16
Keys for \spPoly provide by spintent:
print-sym 23
Keys for \spside provide by spintent:
read-sty 19, 20
Keys for \spunit provide by spintent:
cifras-sep 4
notation-sym 4
read-period-sep 4
Keys for \spusep provide by spintent:
silent 12
size 12

M

Lua module provide by spintent:
spintent.lua 6, 7, 9, 10, 17

P

Packages :
babel-spanish 2
babel 2, 7
documentmetadata-support 1
enumext 1
expl3 3
fontspec 2
fourier-otf 2
hyperref 2
l3keys 3
latex-lab-mathintent 1
libertinus-otf 2
luamml 1
scontents 1
sourcecodepro 2
tagpdf 1, 7, 11
unicode-math 1

11 Implementation

The most recent publicly released version of spintent is available at CTAN: https://www.ctan.org/pkg/spintent. While general feedback via email is welcomed, specific bugs or feature requests should be reported through the issue tracker: https://github.com/pablgonz/spintent/issues.

The documentation presented here is far from professional, it contains a lot of obvious information that to the eye of a TEXpert are superfluous, but, after so many years developing this project is the only way to remember what does what.

11.1 General conventions

Functions of the form `\__spintent_luafun_some:n` or `\__spintent_luafun_some:nn` are defined (created) in the Lua module `spintent.lua` and only their variants are declared in `expl3`.

Similarly, variables of the form `\l__spintent_luaset_some_tl` or `\l__spintent_luaset_some_str` are defined in the Lua module `spintent.lua` and only declared in `expl3`.

All variables and functions defined in this package are private and are NOT intended to work or be used by another package or module.

11.2 Initial set up

Start the DocStrip guards.

```
⟨*package⟩
```

Identify the internal prefix (LaTeX3 DocStrip convention) for `l3doc` class.

```
⟨@@=spintent⟩
```

11.3 Declaration of the package

First we will make sure we have a minimum (super updated) release version of LaTeX to work correctly, otherwise, we will return an "error".

```
\NeedsTeXFormat{LaTeX2e}[2026-11-01]
\IfFormatAtLeastF { 2026-11-01 }
  {
    \PackageError { spintent }
      { LaTeX~kernel~too~old~need~2026-11-01~or~later }
      {
        spintent~requires~a~LaTeX2e~release~2026-11-01~or~
        later.~Update~your~TeX~distribution.
      }
  }
```

Now declare the spintent package and add only two keys: `mathcal` and `AFghost`, which can be passed as package options.

```
\ProvidesExplPackage {spintent} {2026-09-17} {0.99} {Spanish parse intents}
\keys_define:nn { spintent }
  {
    mathcal .usage:n    = load,
    mathcal .bool_set:N = \l__spintent_mathcal_bool,
    mathcal .initial:n  = true,
    mathcal .value_required:n = true,
    AFghost .usage:n    = load,
    AFghost .bool_set:N = \l__spintent_debug_ghost_bool,
    AFghost .initial:n  = false,
    AFghost .value_required:n = true,
  }
\ProcessKeyOptions [ spintent ]
```

We check if the requirements for running the spintent package are met; that is: `\DocumentMetadata` must be active, LuaTEX must be running, and the `unicode-math` or `lua-unicode-math` package must be loaded. If neither is present, we will load `unicode-math`. If the conditions cannot be met, we will return a "fatal error".

```
\IfDocumentMetadataTF
  {
    \sys_if_engine_luatex:TF
      {
        \msg_new:nnn { spintent } { package-not-load }
          {
            The~'#1'~package~will~be~loaded~as~a~dependency.
          }
        \IfPackageLoadedF { unicode-math }
          {
            \IfPackageLoadedF { lua-unicode-math }
              {
                \msg_info:nnn { spintent } { package-not-load } { unicode-math }
                \RequirePackage{unicode-math}
```

```
            }
          }
        \msg_new:nnn { spintent } { env-success }
          { Everything~looks~correct,~spintent~will~make~the~intent! }
        \msg_info:nn { spintent } { env-success }
      }
      {
        \msg_new:nnnn { spintent } { fatal-engine-requires-luatex }
          { The~spintent~package~requires~LuaLaTeX. }
          { This~package~relies~on~Lua~module.~Compile~with~lualatex. }
        \msg_fatal:nn { spintent } { fatal-engine-requires-luatex }
      }
  }
  {
    \msg_new:nnnn { spintent } { fatal-metadata-missing }
      { \token_to_str:N \DocumentMetadata \c_space_tl is~missing. }
      {
        The~spintent~package~requires~tagged~PDF~active.\\
        You~must~declare~\token_to_str:N \DocumentMetadata { tagging=on } ~
      }
    \msg_fatal:nn { spintent } { fatal-metadata-missing }
  }
```

11.4 Some kernel variants

`\str_if_eq:nVTF` `\str_if_eq:nVF` `\str_uppercase:V` `\str_lowercase:V` `\seq_set_from_clist:Ne`

Non-standard kernel variants.

```
\cs_generate_variant:Nn \str_if_eq:nnTF { V }
\cs_generate_variant:Nn \str_if_eq:nnF { V }
\cs_generate_variant:Nn \str_uppercase:n { V }
\cs_generate_variant:Nn \str_lowercase:n { V }
\cs_generate_variant:Nn \seq_set_from_clist:Nn { Ne }
```

(*End of definition for* `\str_if_eq:nVTF` *and others.*)

11.5 Variables for the key `mathcal` and `\rightanglesqr`

`\l__spintent_luaset_cal_fam_ok_str` `\l__spintent_luaset_sym_fam_ok_str`

First, we declare the string variable `\l__spintent_luaset_cal_fam_ok_str` used by the function `\__spintent_luafun_classic_mathcal:nn` (controlled by the key `mathcal`) alongside the string variable `\l__spintent_luaset_sym_fam_ok_str` used by the function `\__spintent_luafun_right_angle_sqr:` to copy `\rightanglesqr` before loading the Lua module `spintent.lua`. This is necessary due to the implementation of the functions within the Lua module; otherwise, `expl3` would issue a warning regarding duplicate variable declarations.

```
\str_new:N \l__spintent_luaset_cal_fam_ok_str
\str_new:N \l__spintent_luaset_sym_fam_ok_str
```

(*End of definition for* `\l__spintent_luaset_cal_fam_ok_str` *and* `\l__spintent_luaset_sym_fam_ok_str`.)

11.6 Loading the Lua module and functions

`spintent.lua`

Loads the Lua module `spintent.lua`, where every `_luafun_` function used throughout the package is registered.

```
\lua_load_module:n { spintent.lua }
```

(*End of definition for* `spintent.lua`.)

`\__spintent_luafun_set_debug_ghost:n`

Passes `AFghost` to Lua as a plain string. When `true`, both ghost characters are marked `sptmp="outer"` or `sptmp="inner"` in the file `-luamml-mathml.html`, so an external script can identify and strip them before generating the `Associated~Files` structure for *tagged* `PDF` — `math-AF` mode does not want either ghost present.

```
\bool_if:NTF \l__spintent_debug_ghost_bool
  { \__spintent_luafun_set_debug_ghost:n { true } }
  { \__spintent_luafun_set_debug_ghost:n { false } }
```

(*End of definition for* `\__spintent_luafun_set_debug_ghost:n`.)

`\l__spintent_scope_attr_int` `\l__spintent_scope_luaset_attr_str`

A LuaTEX attribute number, allocated collision-free from Lua via `luatexbase.new_attribute` and exposed here as an integer. Every command that uses the shared function `\__spintent_luafun_mrow_intent_arg:nn` activates it, via `\attribute \l__spintent_scope_attr_int = 1`, inside its own group — marking every node typeset within as eligible for that queue to claim. Without this mark, the queue could claim content from an unrelated command sitting in the same formula.

```
\int_new:N  \l__spintent_scope_attr_int
\str_new:N  \l__spintent_scope_luaset_attr_str
\int_set:Nn \l__spintent_scope_attr_int { \l__spintent_scope_luaset_attr_str }
```

(*End of definition for* `\l__spintent_scope_attr_int` *and* `\l__spintent_scope_luaset_attr_str`.)

`\__spintent_luafun_spunit_lookup_alias:V`
`\__spintent_luafun_spmoney_lookup_data:V`
`\__spintent_luafun_spmQ_scale_int:Vn`
`\__spintent_luafun_clean_split_and_set:V`

Variants generated for `\__spintent_luafun_spunit_lookup_alias:n` (used by `\spunit`), `\__spintent_luafun_` (used by `\spmoney`), `\__spintent_luafun_spmQ_scale_int:nn` (used by `\spmQ`) and `\__spintent_luafun_clea` (used by `\spnum`).

```
\cs_generate_variant:Nn \__spintent_luafun_spunit_lookup_alias:n { V }
\cs_generate_variant:Nn \__spintent_luafun_spmoney_lookup_data:n { V }
\cs_generate_variant:Nn \__spintent_luafun_spmQ_scale_int:nn { Vn }
\cs_generate_variant:Nn \__spintent_luafun_clean_split_and_set:n { V }
```

(*End of definition for* `\__spintent_luafun_spunit_lookup_alias:V` *and others.*)

11.7 Internal implementation for `\mathcal` and `\rightanglesqr`

To simplify user configuration and obtain the expected output style pdfTEX for `\mathcal`, we will provide a fallback to the font `NewCMMath-Regular.otf`, which contains the glyphs for these characters. Using this same font, we will obtain the symbol `\rightanglesqr` for the key `recto`.

`mathcal-fallback`
`right-angle-sqr-fallback`

First, we'll define two error messages in case our fallback fails.

```
\hook_gput_code:nnn { begindocument } { spintent }
  {
    \__spintent_luafun_init_math_fallbacks:
  }
\msg_new:nnn { spintent } { mathcal-fallback }
  {
    Using~classic~\token_to_str:N \mathcal \c_space_tl as~fallback~
    (font~not~found~or~no~free~math~family~available).~\msg_line_context:
  }
\msg_new:nnn { spintent } { right-angle-sqr-fallback }
  {
    Using~\token_to_str:N \rightangle \c_space_tl as~fallback~
    for~the~right-angle~ glyph.~\msg_line_context:
  }
```

(*End of definition for* `mathcal-fallback` *and* `right-angle-sqr-fallback`.)

`\__spintent_mathcal:n`

Now we define the internal function `\__spintent_mathcal:n` for `\mathcal` which checks the state of the variable `\l__spintent_mathcal_bool` set by the package key `mathcal`; if its state is true, we check the string variable `\l__spintent_luaset_cal_fam_ok_str` and if the latter has the value true, we call the function `\__spintent_luafun_classic_mathcal:n` and otherwise (we cannot find the source in the system) we return a warning message and directly use `\mathcal`. Otherwise, that is, `mathcal=false` we simply use `\mathcal`.

```
\cs_new_protected:Nn \__spintent_mathcal:n
  {
    \bool_if:NTF \l__spintent_mathcal_bool
      {
        \str_if_eq:VnTF \l__spintent_luaset_cal_fam_ok_str { true }
          { \__spintent_luafun_classic_mathcal:n {#1} }
          {
            \msg_warning:nn { spintent } { mathcal-fallback }
            \mathcal {#1}
          }
      }
      { \mathcal {#1} }
  }
```

(*End of definition for* `\__spintent_mathcal:n`.)

`\__spintent_right_angle_sqr:`

The internal function `\__spintent_right_angle_sqr:` establishes the internal copy of the command `\rightanglesqr`. First, we check the string variable `\l__spintent_luaset_sym_fam_ok_str`. If it is true, we call the function `\__spintent__luafun_right_angle_sqr:`; otherwise (we do not find the font in the system), we return a warning message and use `\rightangle` instead.

```
\cs_new_protected:Nn \__spintent_right_angle_sqr:
  {
    \str_if_eq:VnTF \l__spintent_luaset_sym_fam_ok_str { true }
      { \__spintent_luafun_right_angle_sqr: }
      {
        \msg_warning:nn { spintent } { right-angle-sqr-fallback }
        \rightangle
      }
  }
```

(*End of definition for* `\__spintent_right_angle_sqr:`.)

11.8 Shared annotation mechanisms: the queue and the ghosts

Almost all commands in the spintent package share three annotation functions, the mechanism of which is defined in the Lua module spintent.lua.

__spintent_luafun_mrow_intent_arg:nn

The internal function __spintent_luafun_mrow_intent_arg:nn — called directly, with no expl3 wrapper — pushes one entry onto a queue: a concept intent ('#1', possibly empty) and a Lua table literal arg=/intent= ('#2'). A pre_mlist_to_hlist_filter callback (installed once, before luamml[2]'s own) walks the formula's top-level noads in order and, for each one whose nucleus or subscript was typeset inside a spintent command's own group (marked via a LuaTEX \attribute activated by every command that uses this queue), claims it for the next pending entry: arg/a literal intent land on that leaf, and, when the concept intent is non-empty, the leaf is wrapped in a native <mrow> carrying it — no token, no ghost involved. Entries are consumed strictly in order; any left unclaimed when the formula ends are discarded rather than carried into the next one. A native \MathMLarg/\MathMLintent claim is invisible to this queue: mixing the two for the same content risks the queue claiming something already tagged natively, or never being claimed at all.

(*End of definition for* __spintent_luafun_mrow_intent_arg:nn.)

The other two are "ghost" characters (an invisible U+2063, self-tagged :silent through their own mathml_filter — so the queue above never mistakes either for one of its own free targets) dropped in as an extra sibling so the resulting <mrow> is not the formula's sole top-level content, which luamml would otherwise promote straight to <math intent=...> instead of leaving it nested (issue #26).

__spintent_invisible_sep:

Placed once, before opening the formula's own outer \MathMLintent — protects the *whole formula* from collapsing when that outer group is its only top-level content. Fires at most once per document: past the first call, in any command, it does nothing, since one sibling anywhere in the formula is already enough.

```
\cs_new_protected:Nn \__spintent_invisible_sep:
  {
    \__spintent_luafun_invisible_sep:
  }
```

Executing this function writes the following line directly to the output file -luamml-mathml.html:

```
<mi intent=":silent" mathvariant="normal"></mi>
```

and if the package key AFghost=true is used, the output is:

```
<mi intent=":silent" mathvariant="normal" sptmp="outer"></mi>
```

If you wish to use math-AF mode, you can remove the line generated by AFghost=true using the Lua script spiafcls.

(*End of definition for* __spintent_invisible_sep:.)

__spintent_inner_sep:

Placed just inside a native \MathMLintent group — protects *that specific group* when it too would otherwise hold only one child. Never deduplicates: called once per such group, however many appear, nested or sibling, in the same formula.

```
\cs_new_protected:Nn \__spintent_inner_sep:
  {
    \__spintent_luafun_inner_sep:
  }
```

Executing this function writes the following line directly to the output file -luamml-mathml.html:

```
<mi intent=":silent" mathvariant="normal"></mi>
```

and if the package key AFghost=true is used, the output is:

```
<mi intent=":silent" mathvariant="normal" sptmp="inner"></mi>
```

If you wish to use math-AF mode, you can remove the line generated by AFghost=true using the Lua script spiafcls.

(*End of definition for* __spintent_inner_sep:.)

\c__spintent_invisible_sep_int

A raw, untagged copy of the same U+2063 character, via the LuaTEX primitive \Umathchardef — for use as ordinary content *inside* an explicit \MathMLintent{⟨*intent*⟩} call (typically :silent), not as a sibling on its own.

```
\hook_gput_code:nnn { begindocument } { spintent }
  {
    \Umathchardef \c__spintent_invisible_sep_int = "0 "0 "2063
  }
```

Documented in source3 on page 887 for \int_const:Nn: "As stated, most constants can be defined as \chardef or \mathchardef but that's engine dependent." —so this appears to be the correct option here.

(*End of definition for* \c__spintent_invisible_sep_int.)

11.9 Normalize argument and affixes

We normalize the {⟨*argument*⟩} to support affixes (prefixes/suffixes) by trimming leading and trailing spaces, converting internal spaces to hyphens, collapsing consecutive hyphens, and removing leading or trailing hyphens.

__spintent_collapse_hyphens:N

A recursive helper to collapse multiple hyphens into a single one.

```
\cs_new_protected:Nn \__spintent_collapse_hyphens:N
  {
    \tl_if_in:NnT #1 { -- }
      {
        \tl_replace_all:Nnn #1 { -- } { - }
        \__spintent_collapse_hyphens:N #1
      }
  }
```

(*End of definition for* `\__spintent_collapse_hyphens:N`.)

__spintent_sanitize_affix:Nn
__spintent_sanitize_affix:cn

The function `\__spintent_sanitize_affix:Nn` checks if the {⟨*argument*⟩} passed in '`#2`' is *blank*. If so, it clears the target token list variable passed in '`#1`'. Otherwise, trimming bounds, converting spaces '␣' to hyphens '-', collapsing multiple hyphens, and safely stripping leading/trailing hyphens. For example, if you pass a target variable and raw text:

```
\__spintent_sanitize_affix:Nn \l__spintent_spsomesym_prefix_tl{ foo bar baz }
```

The function will *"store"* the normalized string `foo-bar-baz` inside `\l__spintent_spsomesym_prefix_tl`, ready to be conditionally appended.

```
\cs_new_protected:Nn \__spintent_sanitize_affix:Nn
  {
    \tl_if_blank:nTF {#2}
      { \tl_clear:N #1 }
      {
        \tl_set:Ne #1 {#2}
        \tl_trim_spaces:N #1
        \tl_replace_all:Nnn #1 { ~ } { - }
        \__spintent_collapse_hyphens:N #1
        \tl_put_left:Nn  #1 { \q_mark }
        \tl_put_right:Nn #1 { \q_stop }
        \tl_replace_all:Nnn #1 { \q_mark - } { \q_mark }
        \tl_replace_all:Nnn #1 { - \q_stop } { \q_stop }
        \tl_remove_all:Nn #1 { \q_mark }
        \tl_remove_all:Nn #1 { \q_stop }
      }
  }
\cs_generate_variant:Nn \__spintent_sanitize_affix:Nn { cn }
```

(*End of definition for* `\__spintent_sanitize_affix:Nn`.)

11.10 The command `\setspintent`

The `\setspintent` command is used to configure ⟨*keys*⟩ *"globally"* for the utilities provided by spintent.

11.10.1 Internal variables and error messages for `\setspintent`

\l__spintent_setspintent_input_arg_cmd_tl
unknown-command-setup

The variable `\l__spintent_setspintent_input_arg_cmd_tl` will *store* the command or command name passed to the *"first"* {⟨*argument*⟩} '`#1`' of the command `\setspintent`.

```
\tl_new:N \l__spintent_setspintent_input_arg_cmd_tl
```

Error message if the {⟨*argument*⟩} '`#1`' passed to `\setspintent` is a *"non-existent"* command.

```
\msg_new:nnnn { spintent } { unknown-command-setup }
  {
    Cannot~configure~unknown~command~'\exp_not:c{#1}'.
  }
  {
    You~attempted~to~use~\token_to_str:N \setspintent{#1}{...},~but~the~
    command~\exp_not:c{#1}~is~not~defined.~Please~check~for~typos.
  }
```

(*End of definition for* `\l__spintent_setspintent_input_arg_cmd_tl` *and* `unknown-command-setup`.)

__spintent_setspintent_set_keys:nn
__spintent_setspintent_set_keys:en

The function __spintent_setspintent_set_keys:nn will check if the command passed as {⟨*argument*⟩} exists in '#1' and set the ⟨*keys*⟩ passed as {⟨*argument*⟩} in '#2' to the correct path; otherwise, it will return an error.

```
\cs_new_protected:Nn \__spintent_setspintent_set_keys:nn
  {
    \cs_if_exist:cTF { #1 }
      {
        \keys_set:nn { spintent / #1 } { #2 }
      }
      {
        \msg_error:nnn { spintent } { unknown-command-setup } { #1 }
      }
  }
\cs_generate_variant:Nn \__spintent_setspintent_set_keys:nn { en }
```

(*End of definition for* __spintent_setspintent_set_keys:nn.)

__spintent_setspintent_parse_args:nn
__spintent_setspintent_parse_args:Vn

The function __spintent_setspintent_parse_args:nn will check if the argument passed in #1 begins with '\' or is just a *command name*. If it begins with '\', it will remove it using \cs_to_str:N and then execute __spintent_setspintent_set_keys:en. Otherwise, i.e., if '\' was not passed, it will directly execute __spintent_setspintent_set_keys:nn.

```
\cs_new_protected:Nn \__spintent_setspintent_parse_args:nn
  {
    \tl_if_single:nTF { #1 }
      {
        \token_if_cs:NTF #1
          {
            \__spintent_setspintent_set_keys:en { \cs_to_str:N #1 } { #2 }
          }
          {
            \__spintent_setspintent_set_keys:nn { #1 } { #2 }
          }
      }
      {
        \__spintent_setspintent_set_keys:nn { #1 } { #2 }
      }
  }
\cs_generate_variant:Nn \__spintent_setspintent_parse_args:nn { Vn }
```

(*End of definition for* __spintent_setspintent_parse_args:nn.)

\setspintent

Now we define the user command \setspintent.

```
\NewDocumentCommand \setspintent { m m }
  {
    \tl_if_blank:nTF { #1 }
      {
        \msg_error:nnn { spintent } { unknown-command-setup } { <empty> }
      }
      {
        \tl_set:Nn \l__spintent_setspintent_input_arg_cmd_tl { #1 }
        \tl_trim_spaces:N \l__spintent_setspintent_input_arg_cmd_tl
        \__spintent_setspintent_parse_args:Vn \l__spintent_setspintent_input_arg_cmd_tl { #2 }
      }
  }
```

(*End of definition for* \setspintent. *This function is documented on page* 3.)

11.11 Handling unknown keys

\l__spintent_command_name_str
unknown-choice
cmd-key-unknown
cmd-key-value-unknown

We define the variable \l__spintent_command_name_str which will *store* the *"name"* of each command within the implementation along with the messages unknown-choice used by ⟨*keys*⟩ that use .choice:n in their implementation and the messages cmd-key-unknown, cmd-key-value-unknown used by the ⟨*key*⟩ unknown in commands that use [⟨*key* = *val*⟩].

```
\str_new:N  \l__spintent_command_name_str
\msg_new:nnn { spintent } { unknown-choice }
  {
    The~value~'#3'~for~'#1'~key~is~invalid~use~('#2').
  }
\msg_new:nnnn { spintent } { cmd-key-unknown }
  {
    The~key~'#1'~is~unknown~by~command~
```

```
    \c_backslash_str \l__spintent_command_name_str \c_space_tl
    and~is~being~ignored~\msg_line_context:.
  }
  {
    The~command~\c_backslash_str \l__spintent_command_name_str \c_space_tl
    does~not~have~a~key~called~'#1'.\\
    Check~that~you~have~spelled~the~key~name~correctly.
  }
\msg_new:nnnn { spintent } { cmd-key-value-unknown }
  {
    The~key~'#1=#2'~is~unknown~by~command~
    \c_backslash_str \l__spintent_command_name_str \c_space_tl
    and~is~being~ignored~\msg_line_context:.
  }
  {
    The~command~\c_backslash_str \l__spintent_command_name_str \c_space_tl
    does~not~have~a~key~called~'#1'.\\
    Check~that~you~have~spelled~the~key~name~correctly.
  }
```

(*End of definition for* `\l__spintent_command_name_str` *and others.*)

`\__spintent_unknown_keys:nn`
`\__spintent_unknown_keys:Vn`
`\__spintent_unknown_keys:n`

The internal functions `\__spintent_unknown_keys:nn` and `\__spintent_unknown_keys:n` used by the ⟨*key*⟩ `unknown`.

```
\cs_new_protected:Nn \__spintent_unknown_keys_cmd:nn
  {
    \tl_if_blank:nTF {#2}
      { \msg_error:nnn { spintent } { cmd-key-unknown } {#1} }
      { \msg_error:nnnn { spintent } { cmd-key-value-unknown } {#1} {#2} }
  }
\cs_generate_variant:Nn \__spintent_unknown_keys_cmd:nn { Vn }
\cs_new_protected:Nn \__spintent_unknown_keys_cmd:n
  {
    \__spintent_unknown_keys_cmd:Vn \l_keys_key_str {#1}
  }
```

(*End of definition for* `\__spintent_unknown_keys:nn` *and* `\__spintent_unknown_keys:n`.)

11.12 Definition of core base variables

`\__spintent_luafun_clean_split_and_set:n`
`\__spintent_luafun_clean_split_and_set:V`

The function `\__spintent_luafun_clean_split_and_set:n` defined in the Lua module `spintent.lua`, is common to several commands in this implementation. It analyzes, processes, cleans and splits the {⟨*argument*⟩} passed to commands using internal functions defined in the Lua module `spintent.lua`. It isolates signs, integer components, decimal separators, decimal parte, decimal periodic part, exponents and final physical *"units"*, assigning them directly to `expl3` variables.

This function assigns the states of the variables '`_str`' or '`_tl`' described below to `true`, `false`, `error` or *stores* information according to the role it fulfills within the implementation.

(*End of definition for* `\__spintent_luafun_clean_split_and_set:n`.)

`\l__spintent_luaset_sign_tl`
`\l__spintent_luaset_part_int_tl`
`\l__spintent_luaset_dec_sep_tl`
`\l__spintent_luaset_part_dec_tl`
`\l__spintent_luaset_part_period_tl`
`\l__spintent_luaset_has_integer_str`
`\l__spintent_luaset_has_natural_str`

The variable `\l__spintent_luaset_sign_tl` *stores* the explicit '+' or '−' sign if present. The variable `\l__spintent_luaset_part_int_tl` *stores* the raw digit sequence of the *"integer part"*. The variable `\l__spintent_luaset_dec_sep_tl` *stores* the matched '`,`' or '`.`' used as the decimal separator. The variable `\l__spintent_luaset_part_dec_tl` *stores* the raw digits of the *"finite decimal part"* that follow the separator, and the variable `\l__spintent_luaset_part_period_tl` *stores* the digits of the *"infinite periodic decimal part"* that follow a semicolon '`;`'.

The variable `\l__spintent_luaset_has_integer_str` *store* the state `true` when it is *only* an *"integer"*. The variable `\l__spintent_luaset_has_natural_str` *store* the state `true` when it is *only* an *"natural number"*.

```
\tl_new:N  \l__spintent_luaset_sign_tl
\tl_new:N  \l__spintent_luaset_part_int_tl
\tl_new:N  \l__spintent_luaset_dec_sep_tl
\tl_new:N  \l__spintent_luaset_part_dec_tl
\tl_new:N  \l__spintent_luaset_part_period_tl
\str_new:N \l__spintent_luaset_has_integer_str
\str_new:N \l__spintent_luaset_has_natural_str
```

(*End of definition for* `\l__spintent_luaset_sign_tl` *and others.*)

\l__spintent_luaset_only_part_int_str
\l__spintent_luaset_only_part_dec_str
\l__spintent_luaset_only_part_period_str
\l__spintent_luaset_format_part_int_str
\l__spintent_luaset_format_part_dec_str

The literal entries of the numbers that we will pass to the *intent function* `:number` of `\MathMLintent` are *stored* in `\l__spintent_luaset_only_part_int_str` for the *"integer part"*, in `\l__spintent_luaset_only_part_dec_` for the *"finite decimal part"*, and in `\l__spintent_luaset_only_part_period_str` for the *"infinite periodic decimal part"*.

Formatted versions (`cifras-sep=true`) are *stored* in `\l__spintent_luaset_format_part_int_str` for the *"integer part"* and `\l__spintent_luaset_format_part_dec_str` for the *"finite decimal part"*.

```
\str_new:N \l__spintent_luaset_only_part_int_str
\str_new:N \l__spintent_luaset_only_part_dec_str
\str_new:N \l__spintent_luaset_only_part_period_str
\str_new:N \l__spintent_luaset_format_part_int_str
\str_new:N \l__spintent_luaset_format_part_dec_str
```

(*End of definition for* `\l__spintent_luaset_only_part_int_str` *and others.*)

\l__spintent_luaset_millons_str
\l__spintent_luaset_decimal_period_str
l__spintent_luaset_not_decimal_period_str
pintent_luaset_not_decimal_not_period_str
\l__spintent_luaset_dec_is_all_zeros_str
\l__spintent_luaset_has_exponent_str
\l__spintent_luaset_exponent_tl

The flag variable `\l__spintent_luaset_millons_str` indicates whether the integer part translates exactly to a multiple of one million when passed to `\MathMLintent`. To properly structure the reading and writing of numbers in `MathML`, we use the flag variables `\l__spintent_luaset_decimal_period_str`, `\l__spintent_luaset_not_decimal_period_str`, and `\l__spintent_luaset_not_decimal_not_period_st` The flag variable `\l__spintent_luaset_dec_is_all_zeros_str` monitors whether the *"finite decimal part"* is completely zero (useful for *"pure periodic decimals"*).

The flag variable `\l__spintent_luaset_has_exponent_str` indicates whether active *"scientific notation"* was detected at the end of the entered number, passed with '`e`' or '`E`'. If true, it will store the digits of the exponent in `\l__spintent_luaset_exponent_tl`.

```
\str_new:N \l__spintent_luaset_millons_str
\str_new:N \l__spintent_luaset_decimal_period_str
\str_new:N \l__spintent_luaset_not_decimal_period_str
\str_new:N \l__spintent_luaset_not_decimal_not_period_str
\str_new:N \l__spintent_luaset_dec_is_all_zeros_str
\str_new:N \l__spintent_luaset_has_exponent_str
\tl_new:N  \l__spintent_luaset_exponent_tl
```

(*End of definition for* `\l__spintent_luaset_millons_str` *and others.*)

\l__spintent_luaset_has_units_str
\l__spintent_luaset_status_str
\l__spintent_luaset_arg_above_tl
\l__spintent_luaset_arg_below_tl
\l__spintent_luaset_denom_has_number_str

Since the function `\__spintent_luafun_clean_split_and_set:n` is common to `\spnum` (§11.14) and `\spunit` (§11.15) (as well as others), it determines the following unit-related variables after processing the {⟨*arguments*⟩} via the Lua module `spintent.lua`.

First, it checks if there is any residual text *after* the processed number that is a candidate for physical *"units"*. It sets the flag variable `\l__spintent_luaset_has_units_str` accordingly. Following this, it verifies if this text contains more than one "/". It then sets the variable `\l__spintent_luaset_status_str` to `valid` or `multislash`. If the structure is `valid`, it splits the unit into `\l__spintent_luaset_arg_above_tl` and `\l__spintent_luaset_arg_below_tl`. Finally, it analyzes `\l__spintent_luaset_arg_below_tl`; if the denominator starts with a number, it sets `\l__spintent_luaset_denom_has_number_str` to `true`, otherwise to `false`.

```
\str_new:N \l__spintent_luaset_has_units_str
\str_new:N \l__spintent_luaset_status_str
\tl_new:N  \l__spintent_luaset_arg_above_tl
\tl_new:N  \l__spintent_luaset_arg_below_tl
\str_new:N \l__spintent_luaset_denom_has_number_str
```

(*End of definition for* `\l__spintent_luaset_has_units_str` *and others.*)

11.13 Common messages for text mode and math mode

Messages shared by the different commands for text mode or math mode.

```
\msg_new:nnnn { spintent } { need-math-mode }
  {
    Math~mode~required~for~'#1'~\msg_line_context:.
  }
  {
    Use~inside~$...$~or~in~math~environment.
  }
\msg_new:nnnn { spintent } { need-text-mode }
  {
    Text~mode~required~for~'#1'~\msg_line_context:.
  }
  {
    Use~outside~of~$...$~or~math~environment.
  }
```

11.14 The command \spnum

The user command \spnum is *first* of *"the big four"* provided by the spintent package and is fundamental to the implementation of other commands. It handles the processing of numerical input, scientific notation, and units of measurement, using the Lua module spintent.lua, which is responsible for separating and assigning variables defined in expl3.

From the expl3 side, we handle the printing logic and the intent *functions* passed to \MathMLintent to achieve a valid MathML structure that is accepted by MathCAT.

11.14.1 Definition of keys and variables for \spnum

\l__spintent_spnum_read_terse_bool
\l__spintent_spnum_read_period_sep_str
\l__spintent_spnum_notation_sym_tl
\l__spintent_spnum_intent_read_sign_str
\l__spintent_spnum_intent_read_dec_sep_str

The variable \l__spintent_spnum_read_terse_bool handled by the key read-sign, controls the intent passed to \MathMLintent to spoken the '+' or '−'. If set to clear, it will say *"positivo"* or *"negativo" after* reading the number; if set to terse, it will say *"más"* or *"menos" before* reading the number.

The variable \l__spintent_spnum_read_period_sep_str handled by the key read-period-sep, controls how the decimal separator ';' (periodic part) is printed and spoken when the {⟨*argument*⟩} is a *pure decimal periodic* and is passed the intent to \MathMLintent. If set to coma, it will say *"coma"* and print ',', if set to punto it will say *"punto"* and print '.'.

The variable \l__spintent_spnum__notation_sym_tl, handled by the key notation-sym, determines whether '×' will be printed when set to times or '·' when set to cdot.

The variable \l__spintent_spnum_intent_read_sign_str will *store* the spoken translation used in the intent function passed to \MathMLintent for the '+' or '−' according to the value passed to the key read-sign.

The variable \l__spintent_spnum_intent_read_dec_sep_str *stores* the spoken translation that will be used in the intent function passed to \MathMLintent for reading the decimal separation as a *"coma"* or a *"punto"* according to the value found in the variable \l__spintent_luaset_dec_sep_tl.

```
\bool_new:N \l__spintent_spnum_read_terse_bool
\str_new:N  \l__spintent_spnum_read_period_sep_str
\tl_new:N   \l__spintent_spnum_notation_sym_tl
\str_new:N  \l__spintent_spnum_intent_read_sign_str
\str_new:N  \l__spintent_spnum_intent_read_dec_sep_str
```

(*End of definition for* \l__spintent_spnum_read_terse_bool *and others.*)

cifras-sep
read-sign
read-period-sep
notation-sym

Now we add the keys cifras-sep, read-sign, read-period-sep and notation-sym.

```
\keys_define:nn { spintent/spnum }
  {
    cifras-sep                .bool_set:N = \l__spintent_spnum_cifras_sep_bool,
    cifras-sep                .initial:n = true,
    cifras-sep                .value_required:n = true,
    read-sign                 .choices:nn =
      { terse , clear }
      {
        \int_case:nn { \l_keys_choice_int }
          {
            {1} { \bool_set_true:N  \l__spintent_spnum_read_terse_bool }
            {2} { \bool_set_false:N \l__spintent_spnum_read_terse_bool }
          }
      },
    read-sign                 .initial:n  = terse,
    read-sign                 .value_required:n = true,
    read-sign / unknown       .code:n =
      \msg_error:nneee { spintent } { unknown-choice }
        { read-sign } { terse , ~clear } { \exp_not:n {#1} },
    read-period-sep           .choices:nn =
      { coma , punto }
      {
        \int_case:nn { \l_keys_choice_int }
          {
            {1} { \str_set:Nn \l__spintent_spnum_read_period_sep_str { coma } }
            {2} { \str_set:Nn \l__spintent_spnum_read_period_sep_str { punto } }
          }
      },
    read-period-sep           .initial:n  = coma,
    read-period-sep           .value_required:n = true,
    read-period-sep / unknown .code:n =
      \msg_error:nneee { spintent } { unknown-choice }
        { read-period-sep } { coma, ~punto } { \exp_not:n {#1} },
    notation-sym              .choices:nn =
      { times , cdot }
```

```
      {
        \int_case:nn { \l_keys_choice_int }
          {
            {1} { \tl_set:Nn \l__spintent_spnum_notation_sym_tl { \times } }
            {2} { \tl_set:Nn \l__spintent_spnum_notation_sym_tl { \cdot } }
          }
      },
    notation-sym              .initial:n  = cdot,
    notation-sym              .value_required:n = true,
    notation-sym / unknown    .code:n =
      \msg_error:nneee { spintent } { unknown-choice }
        { notation-sym } { times , ~ cdot } { \exp_not:n {#1} },
    unknown                   .code:n =
      {
        \str_set:Nn \l__spintent_command_name_str { spnum }
        \__spintent_unknown_keys_cmd:n {#1}
      },
  }
```

(*End of definition for* `cifras-sep` *and others.*)

11.14.2 Internal functions for \spnum

`\__spintent_spnum_render_part:n`
`\__spintent_spnum_render_int:`
`\__spintent_spnum_render_dec:`

The function `\__spintent_spnum_render_part:n` processes the {⟨*argument*⟩} passed in '`#1`' used by the functions `\__spintent_spnum_render_int:` and `\__spintent_spnum_render_dec:` to define the *intent function* `:number` for the integer and decimal part passed to `\MathMLintent`. It evaluates whether the key `cifras-sep` is active or not.

```
\cs_new_protected:Nn \__spintent_spnum_render_part:n
  {
    \MathMLintent { \str_use:c { l__spintent_luaset_only_part_#1_str } :number }
      {
        \bool_if:NTF \l__spintent_spnum_cifras_sep_bool
          { \str_use:c { l__spintent_luaset_format_part_#1_str } }
          { \tl_use:c { l__spintent_luaset_part_#1_tl } }
      }
  }
\cs_new_protected:Nn \__spintent_spnum_render_int:
  {
    \__spintent_spnum_render_part:n { int }
  }
\cs_new_protected:Nn \__spintent_spnum_render_dec:
  {
    \__spintent_spnum_render_part:n { dec }
  }
```

(*End of definition for* `\__spintent_spnum_render_part:n`, `\__spintent_spnum_render_int:`, *and* `\__spintent_spnum_render_dec:`.)

`\__spintent_spnum_render_only_period:`
`\__spintent_spnum_print_period_bar:`
`\__spintent_spnum_render_period_sep:`

The function `\__spintent_spnum_render_only_period:` isolates the *"infinite periodic decimal part"* from the *"finite decimal part"*, assigning the *intent function* `:number` passed to `\MathMLintent` which will be used by `\MathMLarg` in the function `\__spintent_spnum_print_period_bar:`.

```
\cs_new_protected:Nn \__spintent_spnum_render_only_period:
  {
    \MathMLintent { \l__spintent_luaset_only_part_period_str :number }
      {
        \l__spintent_luaset_part_period_tl
      }
  }
```

The function `\__spintent_spnum_print_period_bar:` inserts the function `\__spintent_spnum_render_only_p` as an argument within the *intent function* `_periódico:postfix($dp)` passed to `\MathMLintent`, generating the correct spoken for the *"infinite periodic decimal part"*.

```
\cs_new_protected:Nn \__spintent_spnum_print_period_bar:
  {
    \MathMLintent { _periódico:postfix($dp) }
      {
        {
          \overline
            {
              \MathMLarg { dp } { \__spintent_spnum_render_only_period: }
            }
        }
      }
```

```
}
```

The function `\__spintent_spnum_render_period_sep:` sets how the decimal separator handled by the key `read-period-sep` will be printed and spoken when a *"pure periodic decimal"* is passed.

```
\cs_new_protected:Nn \__spintent_spnum_render_period_sep:
  {
    \str_if_eq:VnTF \l__spintent_spnum_read_period_sep_str { coma }
      {
        \MathMLintent { _coma:prefix($pp) }
          {
            \mathord{,}
            \MathMLarg { pp } { \__spintent_spnum_print_period_bar: }
          }
       }
       {
         \MathMLintent { _punto:prefix($pp) }
           {
             \mathord{.}
             \MathMLarg { pp } { \__spintent_spnum_print_period_bar: }
           }
       }
  }
```

(*End of definition for* `\__spintent_spnum_render_only_period:`, `\__spintent_spnum_print_period_bar:`, *and* `\__spintent_spnum_render_period_sep:`.)

`\__spintent_spnum_render_decimal_sep:`

The function `\__spintent_spnum_render_decimal_sep:` parses the value of the variable `\l__spintent_luaset_` and sets the variable `\l__spintent_spnum_intent_read_dec_sep_str` with the function `_coma:prefix($dp)` for reading a *"coma"* or `_punto:prefix($dp)` for reading a *"punto"* to `\MathMLintent`. We print the decimal separator stored in `\l__spintent_luaset_dec_sep_tl` using `\mathord` to remove spaces around it and then use the command `\MathMLarg` where we insert `\__spintent_invisible_sep:` for compatibility with `MathML` and finally call the function `\__spintent_spnum_render_dec:` to print the finite decimal part.

```
\cs_new_protected:Nn \__spintent_spnum_render_decimal_sep:
  {
    \str_if_eq:VnTF \l__spintent_luaset_dec_sep_tl { , }
      { \str_set:Nn \l__spintent_spnum_intent_read_dec_sep_str { _coma } }
      { \str_set:Nn \l__spintent_spnum_intent_read_dec_sep_str { _punto } }
    \MathMLintent { \l__spintent_spnum_intent_read_dec_sep_str :prefix($dp) }
      {
        \mathord { \l__spintent_luaset_dec_sep_tl }
        \MathMLarg { dp }
          {
            \__spintent_inner_sep:
            \__spintent_spnum_render_dec:
          }
      }
    % period part
    \str_if_eq:VnT \l__spintent_luaset_decimal_period_str { true }
      {
        \str_if_eq:VnTF \l__spintent_luaset_dec_is_all_zeros_str { true }
          {
            \__spintent_inner_sep:
            \__spintent_spnum_print_period_bar:
          }
          {
            \MathMLintent { _con:prefix($pnum) }
              {
                \__spintent_inner_sep:
                \MathMLarg { pnum }
                  {
                    \__spintent_spnum_print_period_bar:
                  }
              }
          }
      }
  }
```

(*End of definition for* `\__spintent_spnum_render_decimal_sep:`.)

`\__spintent_spnum_print_part_dec:`
`\__spintent_spnum_print_part_period:`

The function `\__spintent_spnum_print_part_dec:` checks if a decimal part exists. If it does, it determines whether the decimal separator is a comma or a period to set the correct speech intent (*"coma"* or *"punto"*).

It then prints the separator, renders the decimal digits, and if a periodic sequence follows, it decides how to attach the periodic bar.

The function __spintent_spnum_print_part_period: handles cases where a periodic part exists. It evaluates if it should render the periodic separator directly, particularly when the number has a periodic part but lacks a standard decimal part.

```
\cs_new_protected:Nn \__spintent_spnum_print_part_dec:
  {
    \tl_if_empty:NF \l__spintent_luaset_part_dec_tl
      {
        \__spintent_spnum_render_decimal_sep:
      }
  }
\cs_new_protected:Nn \__spintent_spnum_print_part_period:
  {
    \tl_if_empty:NF \l__spintent_luaset_part_period_tl
      {
        \str_if_eq:VnTF \l__spintent_luaset_not_decimal_period_str { true }
          { \__spintent_spnum_render_period_sep: }
          {
            \tl_if_empty:NT \l__spintent_luaset_part_dec_tl
              {
                \__spintent_spnum_render_period_sep:
              }
          }
      }
  }
```

(*End of definition for* __spintent_spnum_print_part_dec: *and* __spintent_spnum_print_part_period:.)

__spintent_spnum_print_integer:
__spintent_spnum_print_num_start:
__spintent_spnum_print_number_core:

The function __spintent_spnum_print_integer: checks if the number consists only of an integer. If so, it simply renders it. If a decimal or periodic part exists, it sets up a MathML intent to link the integer part (respecting digit separators if enabled) to the subsequent decimal or periodic functions.

The function __spintent_spnum_print_num_start: checks if an explicit sign is present. If it finds one, it evaluates the terse reading flag to decide whether the sign should be read as a prefix (*"más"* or *"menos"*) or as a postfix (*"positivo"* or *"negativo"*), and wraps the intent around the rest of the number.

The function __spintent_spnum_print_number_core: acts as the main wrapper. It triggers the sign and base number evaluation, and then checks if a scientific exponent is present. If it is, it appends the correct base-10 notation symbol and exponent to complete the sequence.

Note for __spintent_invisible_sep:.

```
\cs_new_protected:Nn \__spintent_spnum_print_integer:
  {
    \str_if_eq:VnTF \l__spintent_luaset_not_decimal_not_period_str { true }
      { \__spintent_spnum_render_int: }
      {
        \MathMLintent { \l__spintent_luaset_only_part_int_str :number:prefix($next) }
          {
            \bool_if:NTF \l__spintent_spnum_cifras_sep_bool
              { \l__spintent_luaset_format_part_int_str }
              { \l__spintent_luaset_part_int_tl }
            \MathMLarg { next }
              {
                \tl_if_empty:NTF \l__spintent_luaset_part_dec_tl
                  { \__spintent_spnum_print_part_period: }
                  { \__spintent_spnum_print_part_dec: }
              }
          }
      }
  }
\cs_new_protected:Nn \__spintent_spnum_print_num_start:
  {
    \tl_if_empty:NTF \l__spintent_luaset_sign_tl
      {
        \__spintent_invisible_sep: % need
        \__spintent_spnum_print_integer:
      }
      {
        \bool_if:NTF \l__spintent_spnum_read_terse_bool
          {
            \str_if_eq:VnTF \l__spintent_luaset_sign_tl { + }
```

```
              { \str_set:Nn \l__spintent_spnum_intent_read_sign_str { _más:prefix($n)} }
              { \str_set:Nn \l__spintent_spnum_intent_read_sign_str { _menos:prefix($n)} }
          }
          {
            \str_if_eq:VnTF \l__spintent_luaset_sign_tl { + }
              { \str_set:Nn \l__spintent_spnum_intent_read_sign_str { _positivo:postfix($n) } }
              { \str_set:Nn \l__spintent_spnum_intent_read_sign_str { _negativo:postfix($n) } }
          }
        \__spintent_invisible_sep: % need
        \MathMLintent { \l__spintent_spnum_intent_read_sign_str }
          {
            \l__spintent_luaset_sign_tl
            \MathMLarg { n } { \__spintent_spnum_print_integer: }
          }
      }
  }
\cs_new_protected:Nn \__spintent_spnum_print_number_core:
  {
    \__spintent_spnum_print_num_start:
    \str_if_eq:VnT \l__spintent_luaset_has_exponent_str { true }
      {
        \MathMLintent { _por } { \l__spintent_spnum_notation_sym_tl }
        10^{ \l__spintent_luaset_exponent_tl }
      }
  }
```

(*End of definition for* `\__spintent_spnum_print_integer:`, `\__spintent_spnum_print_num_start:`, *and* `\__spintent_spnum_print_number_core:`.)

11.14.3 Definition of error messages

spnum-has-units
spnum-no-digits

We define two error messages for the `\spnum` command. The first "error" triggers if the user includes *"units"* in the {⟨*argument*⟩} passed in '`#1`', advising them to use the `\spunit` command instead. The second "error" is raised if the provided {⟨*argument*⟩} contains no numerical digits.

```
\msg_new:nnnn { spintent } { spnum-has-units }
  {
    The~\token_to_str:N \spnum \c_space_tl command~does~not~accept~units~('#1').
  }
  {
    Use~\token_to_str:N \spunit \c_space_tl if~you~need~to~print~numbers~with~units.
  }
\msg_new:nnnn { spintent } { spnum-no-digits }
  {
    The~\token_to_str:N \spnum \c_space_tl command~requires~a~valid~numerical~value.
  }
  {
    You~provided~'#1',~which~contains~no~digits.
  }
```

(*End of definition for* `spnum-has-units` *and* `spnum-no-digits`.)

11.14.4 Implementation of the `\spum` command

`\spnum`

The public user command `\spnum` accepts an optional [⟨*key* = *val*⟩] argument in '`#1`' and a mandatory {⟨*rational number*⟩} in '`#2`'. It first checks if it is executed inside math mode, raising an error if it is not. If correct, it opens a local group to apply the options and passes the input to Lua for parsing via `\__spintent_luafun_clean_split_and_set:n`.

Before printing, it validates the extracted data. It throws an error if it finds trailing units in the input or if the integer part is completely empty (no valid digits). If all checks pass, it calls `\__spintent_spnum_print_number_core:` to render the final number, and finally closes the group.

```
\NewDocumentCommand \spnum { O{} m }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \keys_set:nn { spintent/spnum } { #1 }
          \__spintent_luafun_clean_split_and_set:n { \exp_not:n { #2 } }
          \str_if_eq:VnTF \l__spintent_luaset_has_units_str { true }
            { \msg_error:nnn { spintent } { spnum-has-units } { #2 } }
            {
              \tl_if_empty:NTF \l__spintent_luaset_part_int_tl
                { \msg_error:nnn { spintent } { spnum-no-digits } { #2 } }
                { \__spintent_spnum_print_number_core: }
```

```
            }
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spnum } }
  }
```

(*End of definition for* `\spnum`. *This function is documented on page* 3.)

11.15 The command \spunit

The user command `\spunit` is the *second* of *"the big four"* provided by the spintent package and is used to parse and print physical units and angular systems. It sends the {⟨*argument*⟩} to Lua module `spintent.lua` via `\__spintent_luafun_clean_split_and_set:n` to separate and clean the {⟨*argument*⟩}.

This ensures that all *"units"* (such as SÍ fractions, products, or sexagesimal angles) are processed consistently by the `spintent.lua`.

11.15.1 Definition of variables

`\l__spintent_spunit_luaset_canonical_str`
`\l__spintent_spunit_luaset_lookup_status_str`
`\l__spintent_spunit_luaset_read_str`
`\l__spintent_spunit_luaset_is_compact_str`
`\l__spintent_spunit_luaset_compact_exp_str`
`\l__spintent_spunit_luaset_is_sexagesimal_str`
`\l__spintent_spunit_luaset_status_str`

These string variables store the data returned by the Lua module `spintent.lua`. They hold the canonical *"unit symbol"*, the dictionary lookup status, the correct speech text for `NVDA`/`MathCAT`, and specific flags that indicate if the *"unit"* is compact or represents a sexagesimal angle.

```
\str_new:N  \l__spintent_spunit_luaset_canonical_str
\str_new:N  \l__spintent_spunit_luaset_lookup_status_str
\str_new:N  \l__spintent_spunit_luaset_read_str
\str_new:N  \l__spintent_spunit_luaset_is_compact_str
\str_new:N  \l__spintent_spunit_luaset_compact_exp_str
\str_new:N  \l__spintent_spunit_luaset_is_sexagesimal_str
\str_new:N  \l__spintent_spunit_luaset_status_str
```

(*End of definition for* `\l__spintent_spunit_luaset_canonical_str` *and others.*)

`\l__spintent_spunit_exponent_tl`
`\l__spintent_spunit_unit_name_tl`
`\l__spintent_spunit_read_slash_str`
`\l__spintent_spunit_exp_seq`
`\l__spintent_spunit_mult_seq`
`\l__spintent_spunit_is_mult_bool`
`\l__spintent_spunit_read_cdot_bool`

These variables are used to process and format the *"unit"*. The sequences `\l__spintent_spunit_mult_seq` and `\l__spintent_spunit_exp_seq` split compound *"units"* at multiplication '`*`' or exponent '`^`' boundaries.

The boolean variables control specific rendering options, such as whether the multiplication dot '`\cdot`' is explicitly read aloud or silently skipped.

```
\tl_new:N   \l__spintent_spunit_exponent_tl
\tl_new:N   \l__spintent_spunit_unit_name_tl
\str_new:N  \l__spintent_spunit_read_slash_str
\seq_new:N  \l__spintent_spunit_exp_seq
\seq_new:N  \l__spintent_spunit_mult_seq
\bool_new:N \l__spintent_spunit_is_mult_bool
\bool_new:N \l__spintent_spunit_read_cdot_bool
```

(*End of definition for* `\l__spintent_spunit_exponent_tl` *and others.*)

11.15.2 Definition of error messages

`spunit-invalid-unit`
`spunit-no-units`
`inline-numeric-denominator`

We define three error messages for the `\spunit` command. The first "error" triggers if the *"unit expression"* provided in '`#1`' is unknown or malformed. The second "error" occurs if no *"units"* are found in the {⟨*argument*⟩}, advising the user to use `\spnum` for pure numbers. The third "error" is raised if a numeric coefficient is illegally placed in the denominator of an inline fraction (after a slash).

```
\msg_new:nnnn { spintent } { spunit-invalid-unit }
  {
    The~expression~'#1'~contains~an~unknown~or~malformed~unit~structure.
  }
  {
    Check~spelling~or~verify~the~unit~is~defined~in~the~dictionary.
  }
\msg_new:nnnn { spintent } { spunit-no-units }
  {
    The~\token_to_str:N \spunit \c_space_tl command~requires~at~least~one~unit~('#1').
  }
  {
    Use~\token_to_str:N \spnum \c_space_tl if~you~only~need~to~format~a~pure~number.
  }
\msg_new:nnnn { spintent } { inline-numeric-denominator }
  {
    Illegal~numeric~coefficient~in~inline~denominator~'#1'.
  }
  {
    The~inline~mode~of~\token_to_str:N \spunit \c_space_tl
    does~not~allow~numbers~after~the~slash~(/).
  }
```

(*End of definition for* `spunit-invalid-unit`, `spunit-no-units`, *and* `inline-numeric-denominator`.)

11.15.3 Definition of keys for \spunit

read-cdot
read-slash
cifras-sep
read-period-sep
notation-sym

Now we add the keys print-cdot, read-cdot, read-slash, cifras-sep, read-period-sep and notation-sym.

```
\keys_define:nn { spintent/spunit }
  {
    print-cdot           .bool_set:N = \l__spintent_spunit_print_cdot_bool,
    print-cdot           .initial:n  = false,
    print-cdot           .value_required:n = true,
    read-cdot            .bool_set:N = \l__spintent_spunit_read_cdot_bool,
    read-cdot            .initial:n  = false,
    read-cdot            .value_required:n = true,
    read-slash           .choices:nn =
      { por , partido-por }
      {
        \int_case:nn { \l_keys_choice_int }
          {
            {1} { \str_set:Nn \l__spintent_spunit_read_slash_str { por } }
            {2} { \str_set:Nn \l__spintent_spunit_read_slash_str { partido-por } }
          }
      },
    read-slash           .initial:n  = por,
    read-slash           .value_required:n = true,
    read-slash / unknown .code:n =
      \msg_error:nneee { spintent } { unknown-choice }
        { read-slash } { por, ~partido-por } { \exp_not:n {#1} },
    cifras-sep           .meta:nn = { spintent/spnum } { cifras-sep = {#1} },
    cifras-sep           .value_required:n = true,
    read-period-sep      .meta:nn = { spintent/spnum } { read-period-sep = {#1} },
    read-period-sep      .value_required:n = true,
    notation-sym         .meta:nn = { spintent/spnum } { notation-sym = {#1} },
    notation-sym         .value_required:n = true,
    unknown              .code:n =
      {
        \str_set:Nn \l__spintent_command_name_str { spunit }
        \__spintent_unknown_keys_cmd:n {#1}
      },
  }
```

(*End of definition for* read-cdot *and others.*)

11.15.4 Internal functions for \spunit

__spintent_spunit_format_canonical:

The function __spintent_spunit_format_canonical: handles the visual formatting of the *"unit symbol"*. It specifically checks for the *liter symbol* 'ℓ' to print it as \ell, and formats all other valid *"units"* in standard upright text using \mathrm.

```
\cs_new_protected:Nn \__spintent_spunit_format_canonical:
  {
    \str_case:VnF \l__spintent_spunit_luaset_canonical_str
      {
        { ℓ } { \ell }
      }
      { \mathrm { \l__spintent_spunit_luaset_canonical_str } }
  }
```

(*End of definition for* __spintent_spunit_format_canonical:.)

__spintent_spunit_process_base_exp:n

The function __spintent_spunit_process_base_exp:n evaluates a *"single unit"* along with its exponent, '#1'. It first looks up the unit in the dictionary; if not found, flags it as invalid.
Otherwise, unless this is the first unit in a product (\l__spintent_spunit_is_mult_bool), it handles the spacing and multiplication dot: \cdot when print-cdot is true (read as por when read-cdot is also true, silent otherwise), or a thin space \, when print-cdot is false (still read as por under read-cdot).

```
\cs_new_protected:Nn \__spintent_spunit_process_base_exp:n
  {
    \tl_clear:N \l__spintent_spunit_exponent_tl
    \__spintent_luafun_spunit_lookup_alias:n { #1 }
    \str_if_eq:VnTF \l__spintent_spunit_luaset_lookup_status_str { notfound }
      {
        \str_set:Nn \l__spintent_luaset_status_str { invalid }
      }
      {
        \bool_if:NF \l__spintent_spunit_is_mult_bool
```

```
          {
            \bool_if:NTF \l__spintent_spunit_print_cdot_bool
              {
                \bool_if:NTF \l__spintent_spunit_read_cdot_bool
                  { \MathMLintent { por } { \cdot } }
                  { \MathMLintent { :silent } { \cdot } }
              }
              {
                \bool_if:NTF \l__spintent_spunit_read_cdot_bool
                  { \MathMLintent { por } { \, } }
                  { \, }
              }
          }
```

Then renders the base unit and its exponent: either a pre-defined compact exponent from Lua (`\l__spintent_spunit_l` or by manually splitting '`#1`' at the '`^`' symbol otherwise.

```
        \str_if_eq:VnTF \l__spintent_spunit_luaset_is_compact_str { true }
          {
            {
              \MathMLintent { \l__spintent_spunit_luaset_read_str }
                {
                  \__spintent_spunit_format_canonical:
                }
            }
            ^ { \l__spintent_spunit_luaset_compact_exp_str }
          }
          {
            \seq_set_split:Nnn \l__spintent_spunit_exp_seq { ^ } { #1 }
            \seq_pop_left:NN \l__spintent_spunit_exp_seq \l__spintent_spunit_unit_name_tl
            \seq_if_empty:NF \l__spintent_spunit_exp_seq
              {
                \seq_pop_left:NN \l__spintent_spunit_exp_seq \l__spintent_spunit_exponent_tl
              }
            \MathMLintent { \l__spintent_spunit_luaset_read_str }
              {
                \__spintent_spunit_format_canonical:
              }
            \tl_if_empty:NF \l__spintent_spunit_exponent_tl
              {
                ^ { \l__spintent_spunit_exponent_tl }
              }
          }
        \bool_set_false:N \l__spintent_spunit_is_mult_bool
      }
  }
```

(*End of definition for* `\__spintent_spunit_process_base_exp:n`.)

`\__spintent_spunit_process_mult:n`
`\__spintent_spunit_process_mult:V`

The function `\__spintent_spunit_process_mult:n` handles compound *"units"* joined by an explicit multiplication star '`*`'. It splits the {⟨*argument*⟩} at each *"star symbol"* and applies `\__spintent_spunit_process_base_ex` to evaluate and render each resulting unit individually.

```
\cs_new_protected:Nn \__spintent_spunit_process_mult:n
  {
    \bool_set_true:N \l__spintent_spunit_is_mult_bool
    \seq_set_split:Nnn \l__spintent_spunit_mult_seq { * } { #1 }
    \seq_map_function:NN \l__spintent_spunit_mult_seq \__spintent_spunit_process_base_exp:n
  }
\cs_generate_variant:Nn \__spintent_spunit_process_mult:n { V }
```

(*End of definition for* `\__spintent_spunit_process_mult:n`.)

`\__spintent_print_spunit_hierarchy:`

The function `\__spintent_print_spunit_hierarchy:` handles the layout of inline *"unit"* fractions. It first processes the *"units"* before the slash (the numerator). If a denominator exists, it prints the slash / '/' with its corresponding speech intent, and then processes the remaining *"units"*.

```
\cs_new_protected:Nn \__spintent_print_spunit_hierarchy:
  {
    \tl_if_empty:NF \l__spintent_luaset_arg_above_tl
      {
        \__spintent_spunit_process_mult:V \l__spintent_luaset_arg_above_tl
        \tl_if_empty:NF \l__spintent_luaset_arg_below_tl
          {
            \MathMLintent { \l__spintent_spunit_read_slash_str } { / }
```

```
            \__spintent_spunit_process_mult:V \l__spintent_luaset_arg_below_tl
          }
      }
  }
```

(*End of definition for* `\__spintent_print_spunit_hierarchy:`.)

`\__spintent_spunit_safe_exec:n`

The function `\__spintent_spunit_safe_exec:n` performs the main validation and layout for *"units"*. It first checks if the {⟨*argument*⟩} has units, is a valid expression, and does not contain illegal numbers in the denominator, throwing the appropriate "errors" if any check fails.

If the unit is valid, it checks if it is a sexagesimal angle (degrees, minutes, or seconds). For sexagesimal *"units"*, it prints the number and the symbol *"without any space in between"*, applying a slight negative space '`\!`' for `arcseconds`. For standard units, it prints the number followed by a thin space '`\,`' and then processes the unit sequence.

```
\cs_new_protected:Nn \__spintent_spunit_safe_exec:n
  {
    \str_if_eq:VnTF \l__spintent_luaset_has_units_str { false }
      { \msg_error:nnn { spintent } { spunit-no-units } { #1 } }
      {
        \str_if_eq:VnTF \l__spintent_luaset_status_str { valid }
          {
            \str_if_eq:VnTF
              \l__spintent_luaset_denom_has_number_str { true }
              {
                \msg_error:nnn { spintent }
                  { inline-numeric-denominator } { #1 }
              }
              {
                \__spintent_luafun_spunit_lookup_alias:V \l__spintent_luaset_arg_above_tl
                \str_if_eq:VnTF \l__spintent_spunit_luaset_is_sexagesimal_str { true }
                  {
                    \tl_if_empty:NF \l__spintent_luaset_part_int_tl
                      {
                        \__spintent_spnum_print_number_core:
                      }
                    \MathMLintent { \l__spintent_spunit_luaset_read_str }
                      {
                        \mathord
                          {
                            \str_case:Vn \l__spintent_spunit_luaset_canonical_str
                              {
                                { ″ } { \prime \! \prime }
                                { ′ } { \prime           }
                                { ° } { °                }
                              }
                          }
                      }
                  }
                  {
                    \tl_if_empty:NF \l__spintent_luaset_part_int_tl
                      {
                        \__spintent_spnum_print_number_core: \,
                      }
                    \__spintent_print_spunit_hierarchy:
                  }
              }
          }
          { \msg_error:nnn { spintent } { spunit-invalid-unit } { #1 } }
      }
  }
```

(*End of definition for* `\__spintent_spunit_safe_exec:n`.)

11.15.5 Implementation of the `\spunit` command

`\spunit`

The public user command `\spunit` accepts an optional [⟨*key* = *val*⟩] argument in '`#1`' and a mandatory {⟨*unit expression*⟩} in '`#2`'. It first checks if it is executed inside *math mode*, raising an "error" if it is not. If correct, it opens a local group to apply the options and passes the {⟨*argument*⟩} to Lua module for parsing via `\__spintent_luafun_clean_split_and_set:n` function.

Finally, it delegates the execution to the function `\__spintent_spunit_safe_exec:n` to perform the safety checks and render the parsed data, before closing the group.

```
\NewDocumentCommand \spunit { O{} m }
```

```
{
  \mode_if_math:TF
    {
      \group_begin:
        \keys_set:nn { spintent/spunit } { #1 }
        \__spintent_luafun_clean_split_and_set:n { \exp_not:n { #2 } }
        \__spintent_spunit_safe_exec:n { #2 }
      \group_end:
    }
    { \msg_error:nnn { spintent } { need-math-mode } { \spunit } }
}
```

(*End of definition for* `\spunit`. *This function is documented on page* 4.)

11.16 The commands `\spnewunit` and `\spunitalias`

The user commands `\spnewunit` and `\spunitalias` allow users to add new custom *"units"* or *"aliases"* to the package's dictionary. Similar to `\spunit`, these commands delegate the processing and validation to the `spintent.lua` via `\__spintent_luafun_define_custom_unit:nn` and `\__spintent_luafun_define_spunit`

During the definition, the Lua module checks if the new *"unit"* or *"alias"* is valid and ensures it does not conflict with existing *"units"*. It returns the result of this operation to TEX using the `\l__spintent_spunit_luaset_status_st` variable. If a collision is detected or the {⟨*argument*⟩} is invalid, the package throws the corresponding "error" message.

11.16.1 Definition of error messages

spnewunit-duplicate
spunitalias-duplicate
spunitalias-invalid-expr

We define three error messages for the creation of *"units"* and *"aliases"*. The first two "errors" trigger if the user attempts to define a *"new unit"* or alias passed in '`#1`' that already exists in the dictionary. The third "error" is raised by `\spunitalias` if the target expression provided in '`#2`' is invalid (for example, if it contains unknown *"units"* or incorrect syntax).

```
\msg_new:nnnn { spintent } { spnewunit-duplicate }
  {
    The~unit~symbol~or~alias~'#1'~is~already~defined!
  }
  {
    You~cannot~redefine~with~\token_to_str:N \spnewunit.
  }
\msg_new:nnnn { spintent } { spunitalias-duplicate }
  {
    The~alias~'#1'~is~already~defined~or~reserved!
  }
  {
    You~cannot~redefine~an~existing~unit~or~another~alias.
  }
\msg_new:nnnn { spintent } { spunitalias-invalid-expr }
  {
    The~expression~'#2'~for~alias~'#1'~is~invalid.
  }
  {
    It~must~contain~only~defined~units,~exponents~and~one~'/'.
  }
```

(*End of definition for* `spnewunit-duplicate`, `spunitalias-duplicate`, *and* `spunitalias-invalid-expr`.)

`\spnewunit`

The public command `\spnewunit` defines a new custom unit. It passes the new unit symbol '`#1`' and its speech text '`#2`' to Lua via `\__spintent_luafun_define_custom_unit:nn`. If the module reports that the unit already exists, it throws the corresponding duplicate error.

```
\NewDocumentCommand \spnewunit { m m }
  {
    \mode_if_math:TF
      {
        \msg_error:nnn { spintent } { need-text-mode } { \spnewunit }
      }
      {
        \__spintent_luafun_define_custom_unit:nn { #1 } { #2 }
        \str_if_eq:VnT \l__spintent_spunit_luaset_status_str { duplicate }
          {
            \msg_error:nnn { spintent } { spnewunit-duplicate } { #1 }
          }
      }
  }
```

(*End of definition for* `\spnewunit`. *This function is documented on page* 6.)

\spunitalias

The public command \spunitalias creates a shortcut for a complex unit expression. It passes the new alias name '#1' and the target expression '#2' to Lua via __spintent_luafun_define_spunit_alias:nn. It throws an error if the alias already exists, or if the provided target expression contains invalid syntax or unknown units.

```
\NewDocumentCommand \spunitalias { m m }
  {
    \mode_if_math:TF
      {
        \msg_error:nnn { spintent } { need-text-mode } { \spunitalias }
      }
      {
        \__spintent_luafun_define_spunit_alias:nn { #1 } { #2 }
        \str_if_eq:VnT \l__spintent_spunit_luaset_status_str { duplicate }
          {
            \msg_error:nnn { spintent } { spunitalias-duplicate } { #1 }
          }
        \str_if_eq:VnT \l__spintent_spunit_luaset_status_str { invalid-expr }
          {
            \msg_error:nnnn { spintent } { spunitalias-invalid-expr } { #1 } { #2 }
          }
      }
  }
```

(*End of definition for* \spunitalias. *This function is documented on page* 6.)

11.17 The command \spmoney

The user command \spmoney is the *third* of *"the big four"* provided by the spintent package, designed to format monetary values and currencies. It delegates {⟨*argument*⟩} parsing to the Lua module spintent.lua via __spintent_luafun_clean_split_and_set:n. The module then retrieves the correct *"currency symbol"*, pluralization rules, and layout structures from the database.

The command accepts *optional arguments* to override the default *"currency symbol"* or enable *"subunit"* rendering. Finally, it constructs the MathML intent for MathCAT.

\l__spintent_spmoney_luaset_symbol_str
\l__spintent_spmoney_luaset_position_str
\l__spintent_spmoney_luaset_sing_str
\l__spintent_spmoney_luaset_plur_str
\l__spintent_spmoney_luaset_conde_str
\l__spintent_spmoney_luaset_subsing_str
\l__spintent_spmoney_luaset_subplur_str
\l__spintent_spmoney_luaset_print_iso_str
\l__spintent_spmoney_luaset_currency_arg_str
\l__spintent_spmoney_luaset_status_str

These string variables store the currency data returned by the Lua module spintent.lua. Upon a successful database lookup, the module populates these variables with the *"currency symbol"*, its layout position (pre or post), the appropriate singular and plural names for both the main unit and subunits, and the execution status flags. TEX then uses this information to render the visual layout and construct the correct MathML speech intents.

```
\str_new:N  \l__spintent_spmoney_luaset_symbol_str
\str_new:N  \l__spintent_spmoney_luaset_position_str
\str_new:N  \l__spintent_spmoney_luaset_sing_str
\str_new:N  \l__spintent_spmoney_luaset_plur_str
\str_new:N  \l__spintent_spmoney_luaset_conde_str
\str_new:N  \l__spintent_spmoney_luaset_subsing_str
\str_new:N  \l__spintent_spmoney_luaset_subplur_str
\str_new:N  \l__spintent_spmoney_luaset_print_iso_str
\str_new:N  \l__spintent_spmoney_luaset_currency_arg_str
\str_new:N  \l__spintent_spmoney_luaset_status_str
```

(*End of definition for* \l__spintent_spmoney_luaset_symbol_str *and others.*)

\l__spintent_spmoney_sub_units_bool
\l__spintent_spmoney_intent_money_str
\l__spintent_spmoney_intent_sign_str
\l__spintent_spmoney_intent_sub_unit_str
\l__spintent_spmoney_res_main_voice_str
\l__spintent_spmoney_subnum_int
\l__spintent_spmoney_extracted_curr_str
\l__spintent_spmoney_extracted_num_tl

The boolean variable tracks whether subunit rendering is enabled. The string variables assemble the specific speech text (intents) for NVDA/MathCAT, including the sign, the main currency, and the subunits. Finally, the remaining variables temporarily store the numeric value and currency code extracted from the user's input.

```
\bool_new:N \l__spintent_spmoney_sub_units_bool
\str_new:N  \l__spintent_spmoney_intent_money_str
\str_new:N  \l__spintent_spmoney_intent_sign_str
\str_new:N  \l__spintent_spmoney_intent_sub_unit_str
\str_new:N  \l__spintent_spmoney_res_main_voice_str
\int_new:N  \l__spintent_spmoney_subnum_int
\str_new:N  \l__spintent_spmoney_extracted_curr_str
\tl_new:N   \l__spintent_spmoney_extracted_num_tl
```

(*End of definition for* \l__spintent_spmoney_sub_units_bool *and others.*)

11.17.1 Definition of error messages

currency-no-subunits
invalid-currency

We define two messages to handle invalid currency configurations. The first "error" warns the user if subunit rendering is requested for a currency that does not support fractional divisions (such as JPY or CLP), safely ignoring the flag. The second "error" is triggered if the provided currency identifier is not registered in the Lua database.

```
\msg_new:nnnn { spintent } { currency-no-subunits }
  {
    The~currency~'#1'~does~not~support~subunits.~
    The~option~'sub-units=true'~is~ignored.
  }
  {
    Monies~like~the~Japanese~Yen~(JPY)~or~Chilean~Peso~(CLP)~
    do~not~have~fractional~subunits~registered~in~Lua~module.~
  }
\msg_new:nnnn { spintent } { invalid-currency }
  {
    Invalid~currency~identifier~'#1'~in~\token_to_str:N \spmoney .
  }
  {
    The~provided~symbol,~alias,~or~ISO~code~does~not~exist~in~
    the~Lua~module.
  }
```

(*End of definition for* `currency-no-subunits` *and* `invalid-currency`.)

11.17.2 Auxiliary functions for \spmoney

__spintent_spmoney_set_currency:n
__spintent_spmoney_normalize_key:n
__spintent_spmoney_normalize_key:V

The function `\__spintent_spmoney_set_currency:n` validates the currency identifier provided in '`#1`' against the database. It delegates the lookup to the helper function `\__spintent_spmoney_normalize_key:n`, which interfaces with the Lua module `spintent.lua`. If the module returns a `notfound` status, an "error" is triggered. Otherwise, it stores the valid currency identifier in `\l__spintent_spmoney_luaset_currency_arg_str` for subsequent processing.

```
\cs_new_protected:Nn \__spintent_spmoney_set_currency:n
  {
    \__spintent_spmoney_normalize_key:n {#1}
    \str_if_eq:VnTF \l__spintent_spmoney_luaset_status_str { notfound }
      {
        \msg_error:nnn { spintent } { invalid-currency } {#1}
      }
      { \str_set:Nn \l__spintent_spmoney_luaset_currency_arg_str {#1} }
  }
\cs_new_protected:Nn \__spintent_spmoney_normalize_key:n
  {
    \__spintent_luafun_spmoney_normalize_key:n { #1 }
  }
\cs_generate_variant:Nn \__spintent_spmoney_normalize_key:n { V }
```

(*End of definition for* `\__spintent_spmoney_set_currency:n` *and* `\__spintent_spmoney_normalize_key:n`.)

11.17.3 Definition of keys for \spmoney

currency
sub-units
dolar-math
cifras-sep

Now we add the keys `currency`, `sub-units`, `dolar-math` and `cifras-sep`.

```
\keys_define:nn { spintent / spmoney }
  {
    currency      .code:n      = { \__spintent_spmoney_set_currency:n {#1} },
    currency      .initial:n   = { pesos },
    currency      .value_required:n = true,
    sub-units     .bool_set:N  = \l__spintent_spmoney_sub_units_bool,
    sub-units     .value_required:n = true,
    dolar-math    .bool_set:N  = \l__spintent_spmoney_dolar_math_bool,
    dolar-math    .initial:n   = { true },
    dolar-math    .value_required:n = true,
    cifras-sep    .meta:nn     = { spintent/spnum } { cifras-sep = {#1} },
    cifras-sep    .value_required:n = true,
    unknown       .code:n =
      {
        \str_set:Nn \l__spintent_command_name_str { spmoney }
        \__spintent_unknown_keys_cmd:n {#1}
      },
  }
```

(*End of definition for* `currency` *and others.*)

11.17.4 Internal functions for \spmoney

__spintent_spmoney_validate_money:

The function __spintent_spmoney_validate_money: verifies if the parsed input contains an embedded currency identifier by checking \l__spintent_luaset_has_units_str. If a currency is detected, it extracts and normalizes the key from \l__spintent_luaset_arg_above_tl. It throws an "error" if the currency is not found in the database. If valid, it updates \l__spintent_spmoney_luaset_currency_arg_str and retrieves the full currency metadata from the spintent.lua via __spintent_luafun_spmoney_lookup_data:V.

```
\cs_new_protected:Nn \__spintent_spmoney_validate_money:
  {
    \str_if_eq:VnT \l__spintent_luaset_has_units_str { true }
      {
        \__spintent_spmoney_normalize_key:V \l__spintent_luaset_arg_above_tl
        \str_if_eq:VnTF \l__spintent_spmoney_luaset_status_str { notfound }
          { \msg_error:nne { spintent } { invalid-currency } { \l__spintent_luaset_arg_above_tl }
          {
            \str_set:NV
              \l__spintent_spmoney_luaset_currency_arg_str
              \l__spintent_luaset_arg_above_tl
          }
      }
    \str_set:Ne \l__spintent_spmoney_luaset_status_str { found }
    \__spintent_luafun_spmoney_lookup_data:V \l__spintent_spmoney_luaset_currency_arg_str
  }
```

(*End of definition for* __spintent_spmoney_validate_money:.)

__spintent_spmoney_print_symbol:

The function __spintent_spmoney_print_symbol: renders the *"currency symbol"*. If \l__spintent_spmoney_lu is true, it outputs the ISO code using \mathrm. Otherwise, it prints the *"standard symbol"*. For dollar signs '$', it checks \l__spintent_spmoney_dolar_math_bool to determine whether to render it directly as a math character '$' or wrap it inside \text.

```
\cs_new_protected:Nn \__spintent_spmoney_print_symbol:
  {
    \str_if_eq:VnTF \l__spintent_spmoney_luaset_print_iso_str { true }
      { \mathrm { \l__spintent_spmoney_luaset_currency_arg_str } }
      {
        \str_if_eq:VnTF \l__spintent_spmoney_luaset_symbol_str { $ }
          {
            \bool_if:NTF \l__spintent_spmoney_dolar_math_bool
              { \$ } { \text { \$ } }
          }
          { \l__spintent_spmoney_luaset_symbol_str }
      }
  }
```

(*End of definition for* __spintent_spmoney_print_symbol:.)

__spintent_spmoney_render_layout:

The function __spintent_spmoney_render_layout: determines the layout order of the *"currency symbol"* relative to the numeric value. It evaluates \l__spintent_spmoney_luaset_position_str; if set to pre, it renders the *"currency symbol" before* the number node. Otherwise, it places the number first, followed by the *"currency symbol"*.

```
\cs_new_protected:Nn \__spintent_spmoney_render_layout:
  {
    \str_if_eq:VnTF \l__spintent_spmoney_luaset_position_str { pre }
      {
        \__spintent_spmoney_print_symbol:
        \MathMLarg { n } { \__spintent_spmoney_render_number_node: }
      }
      {
        \MathMLarg { n } { \__spintent_spmoney_render_number_node: }
        \__spintent_spmoney_print_symbol:
      }
  }
```

(*End of definition for* __spintent_spmoney_render_layout:.)

__spintent_spmoney_render_number_node:
pintent_spmoney_render_only_integer_node:

The function __spintent_spmoney_render_number_node: renders the full numeric value, encompassing both the integer and decimal components. Conversely, the function __spintent_spmoney_render_only_integer_ outputs exclusively the integer portion, omitting any decimal data.

```
\cs_new_protected:Nn \__spintent_spmoney_render_number_node:
  {
    \__spintent_spnum_render_int:
```

```
    \__spintent_spnum_print_part_dec:
  }
\cs_new_protected:Nn \__spintent_spmoney_render_only_integer_node:
  {
    \__spintent_spnum_render_int:
  }
```

(*End of definition for* `\__spintent_spmoney_render_number_node:` *and* `\__spintent_spmoney_render_only_integer_node:`.)

`\__spintent_spmoney_print_int:`

The function `\__spintent_spmoney_print_int:` evaluates the presence of an explicit sign in `\l__spintent_luase` If a sign is detected, it assigns a prefix speech intent 'más' or 'menos' and embeds the sign and subsequent layout inside a `\MathMLintent`. If no sign is present, it routes execution based on whether subunits are enabled.

```
\cs_new_protected:Nn \__spintent_spmoney_print_int:
  {
    \tl_if_empty:NTF \l__spintent_luaset_sign_tl
      {
        \bool_if:NTF \l__spintent_spmoney_sub_units_bool
          { \__spintent_spmoney_set_and_print_sub_units: }
          { \__spintent_spmoney_intent_int: }
      }
      {
        \str_if_eq:VnTF \l__spintent_luaset_sign_tl { + }
          {
            \str_set:Nn \l__spintent_spmoney_intent_sign_str { _más:prefix ($n) }
          }
          {
            \str_set:Nn \l__spintent_spmoney_intent_sign_str { _menos:prefix ($n) }
          }
        \__spintent_invisible_sep:
        \MathMLintent { \l__spintent_spmoney_intent_sign_str }
          {
            \l__spintent_luaset_sign_tl
            \MathMLarg { n }
              {
                \bool_if:NTF \l__spintent_spmoney_sub_units_bool
                  { \__spintent_spmoney_set_and_print_sub_units: }
                  { \__spintent_spmoney_intent_int: }
              }
          }
      }
  }
```

(*End of definition for* `\__spintent_spmoney_print_int:`.)

`\__spintent_spmoney_intent_int:`

The function `\__spintent_spmoney_intent_int:` determines the correct grammatical inflection for the currency string (singular, plural, or the prepositional 'de' variant for exact millions). It evaluates the integer component, decimal boundaries, and zero-fills to select the appropriate string register, applying it as a postfix `MathML` intent over the layout rendered by `\__spintent_spmoney_render_layout:`.

```
\cs_new_protected:Nn \__spintent_spmoney_intent_int:
  {
    \tl_if_empty:NTF \l__spintent_luaset_dec_sep_tl
      {
        \str_if_eq:VnTF \l__spintent_luaset_only_part_int_str { 1 }
          {
            \str_set:NV \l__spintent_spmoney_intent_money_str \l__spintent_spmoney_luaset_sing_st
          }
          {
            \str_set:NV \l__spintent_spmoney_intent_money_str \l__spintent_spmoney_luaset_plur_st
          }
        \str_if_eq:VnT \l__spintent_luaset_millons_str { true }
          {
            \str_set:NV \l__spintent_spmoney_intent_money_str \l__spintent_spmoney_luaset_conde_s
          }
      }
      {
        \str_set:NV \l__spintent_spmoney_intent_money_str \l__spintent_spmoney_luaset_plur_str
        \str_if_eq:VnT \l__spintent_luaset_millons_str { true }
          {
            \str_if_eq:VnTF \l__spintent_luaset_dec_is_all_zeros_str { true }
              {
```

```
                \str_set:NV \l__spintent_spmoney_intent_money_str \l__spintent_spmoney_luaset_cond
              }
              {
                \str_set:NV \l__spintent_spmoney_intent_money_str \l__spintent_spmoney_luaset_plu
              }
          }
      }
    \__spintent_invisible_sep:
    \MathMLintent { \l__spintent_spmoney_intent_money_str :postfix($n) }
      {
        \__spintent_spmoney_render_layout:
      }
  }
```

(*End of definition for* `\__spintent_spmoney_intent_int:`.)

`\__spintent_spmoney_check_sub_units:`
`spintent_spmoney_set_and_print_sub_units:`

The function `\__spintent_spmoney_check_sub_units:` verifies whether the selected currency supports *"subunits"*. If the database returns empty subunit data, it triggers a warning and automatically disables the *"subunit"* rendering flag.

The function `\__spintent_spmoney_set_and_print_sub_units:` processes and formats the speech intents when *"subunits"* are active. It normalizes the decimal component (for instance, padding a single-digit decimal like '.5' to '50'), evaluates the correct singular or plural inflections for both the main *"currency"* and the *"subunits"*, and delegates the final `MathML` tree assembly to `\__spintent_spmoney_build_tree:`.

```
\cs_new_protected:Nn \__spintent_spmoney_check_sub_units:
  {
    \str_if_empty:NT \l__spintent_spmoney_luaset_subsing_str
      {
        \msg_warning:nne { spintent } { currency-no-subunits }
          { \l__spintent_spmoney_luaset_currency_arg_str }
        \bool_set_false:N \l__spintent_spmoney_sub_units_bool
      }
  }
\cs_new_protected:Nn \__spintent_spmoney_set_and_print_sub_units:
  {
    \int_compare:nTF { \str_count:N \l__spintent_luaset_only_part_dec_str == 1 }
      {
        \int_set:Nn \l__spintent_spmoney_subnum_int
          { \l__spintent_luaset_only_part_dec_str * 10 }
        \str_set:Ne \l__spintent_luaset_only_part_dec_str
          { \l__spintent_luaset_only_part_dec_str 0 }
      }
      {
        \int_set:Nn \l__spintent_spmoney_subnum_int
          { \l__spintent_luaset_only_part_dec_str }
      }
    \int_compare:nTF { \l__spintent_spmoney_subnum_int == 1 }
      {
        \str_set:Ne \l__spintent_spmoney_intent_sub_unit_str
          { \l__spintent_spmoney_luaset_subsing_str :postfix($sub) }
      }
      {
        \str_set:Ne \l__spintent_spmoney_intent_sub_unit_str
          { \l__spintent_spmoney_luaset_subplur_str :postfix($sub) }
      }
    \str_if_eq:VnTF \l__spintent_luaset_only_part_int_str { 1 }
      {
        \str_set:NV \l__spintent_spmoney_res_main_voice_str
          \l__spintent_spmoney_luaset_sing_str
      }
      {
        \str_set:NV \l__spintent_spmoney_res_main_voice_str
          \l__spintent_spmoney_luaset_plur_str
      }
    \str_if_eq:VnT \l__spintent_luaset_millons_str { true }
      {
        \str_set:NV \l__spintent_spmoney_res_main_voice_str
          \l__spintent_spmoney_luaset_conde_str
      }
    \__spintent_spmoney_build_tree:
  }
```

(*End of definition for* `\__spintent_spmoney_check_sub_units:` *and* `\__spintent_spmoney_set_and_print_sub_units:`.)

`\__spintent_spmoney_build_tree:`
`\__spintent_spmoney_print_sep_con:`
`\__spintent_spmoney_print_dec_sub_unit:`

The function `\__spintent_spmoney_build_tree:` structures the final `MathML` tree assembly for fractional *"currencies"*. It arranges the integer component, the decimal separator, and the *"subunit"* component based on the layout position (`pre` or `post`) of the *"currency symbol"*.

Furthermore, `\__spintent_spmoney_print_sep_con:` applies the specific speech intent '`con`' to the decimal separator, while `\__spintent_spmoney_print_dec_sub_unit:` binds the computed *"subunit"* intent to the decimal digits, ensuring accurate accessibility output.

```
\cs_new_protected:Nn \__spintent_spmoney_build_tree:
  {
    \str_if_eq:VnTF \l__spintent_spmoney_luaset_position_str { pre }
      {
        \MathMLintent { \l__spintent_spmoney_res_main_voice_str :postfix($n) }
          {
            \__spintent_spmoney_print_symbol:
            \MathMLarg { n } { \__spintent_spmoney_render_only_integer_node: }
          }
        \__spintent_spmoney_print_sep_con:
        \__spintent_spmoney_print_dec_sub_unit:
      }
      {
        \MathMLintent { \l__spintent_spmoney_res_main_voice_str :postfix($n) }
          {
            \__spintent_inner_sep:
            \MathMLarg { n } { \__spintent_spmoney_render_only_integer_node: }
          }
        \__spintent_spmoney_print_sep_con:
        \__spintent_spmoney_print_dec_sub_unit:
        \MathMLintent {:silent } { \__spintent_spmoney_print_symbol: }
      }
  }
\cs_new_protected:Nn \__spintent_spmoney_print_sep_con:
  {
    \MathMLintent { _con }
      {
        \mathord { \l__spintent_luaset_dec_sep_tl }
      }
  }
\cs_new_protected:Nn \__spintent_spmoney_print_dec_sub_unit:
  {
    \MathMLintent { \l__spintent_spmoney_intent_sub_unit_str }
      {
        \__spintent_inner_sep:
        \MathMLarg { sub }
          {
            \__spintent_spnum_render_dec:
          }
      }
  }
```

(*End of definition for* `\__spintent_spmoney_build_tree:`, `\__spintent_spmoney_print_sep_con:`, *and* `\__spintent_spmoney_print_dec_sub_unit:`.)

`\__spintent_spmoney_parse_and_route:n`

The internal function `\__spintent_spmoney_parse_and_route:n` processes the {⟨*argument*⟩} passed in '`#1`' using Lua module to extract the numeric value and any *"currency"* identifier. If a *"currency"* is found (`\l__spintent_spmoney_extracted_curr_str`), it sets the corresponding variables. It then calls `\__spintent_spmoney_validate_money:` to verify the *"currency"*. If *"subunit"* rendering is enabled (`\l__spintent_spmoney_sub_units_bool`), it calls `\__spintent_spmoney_check_sub_units:`. Finally, it calls `\__spintent_spmoney_print_int:` to format the output.

```
\cs_new_protected:Nn \__spintent_spmoney_parse_and_route:n
  {
    \__spintent_luafun_spmoney_prepare_input:n { #1 }
    \__spintent_luafun_clean_split_and_set:V \l__spintent_spmoney_extracted_num_tl
    \str_if_empty:NF \l__spintent_spmoney_extracted_curr_str
      {
        \str_set:Ne \l__spintent_luaset_has_units_str { true }
        \str_set:NV \l__spintent_luaset_arg_above_tl \l__spintent_spmoney_extracted_curr_str
      }
    \__spintent_spmoney_validate_money:
    \bool_if:NT \l__spintent_spmoney_sub_units_bool
```

```
      { \__spintent_spmoney_check_sub_units: }
    \__spintent_spmoney_print_int:
  }
```

(*End of definition for* `\__spintent_spmoney_parse_and_route:n`.)

`\spmoney` The public document command `\spmoney` formats monetary values with accessible speech intents. It accepts an optional argument '`#1`' for local key-value configurations and a mandatory argument '`#2`' containing the numeric value and optional embedded currency identifier. It first evaluates the typesetting context, triggering a fatal error if executed outside of math mode. Upon validation, it establishes a local TEX group to safely isolate the key assignments, delegates the raw input string to `\__spintent_spmoney_parse_and_route:n` for parsing and rendering, and subsequently closes the group to prevent scope leakage.

```
\NewDocumentCommand \spmoney { O{} m }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \keys_set:nn { spintent / spmoney } { #1 }
          \__spintent_spmoney_parse_and_route:n { #2 }
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spmoney } }
  }
```

(*End of definition for* `\spmoney`. *This function is documented on page 6.*)

11.18 The command `\spshort`

The user command `\spshort` is the *fourth* and *final* of *"the big four"*. It manages the semantic injection and typographical formatting of *text mode* abbreviations, ordinals, and acronyms. Unlike *math mode* commands, which rely on `MathML` attributes, this command uses the `PDF` *tagging tools* provided by the `tagpdf`[29] package. Specifically, it uses the `E` (expanded) key to ensure that `NVDA` interpret the full form of the abbreviation, ordinal, or acronym in *text mode*.

`\l__spintent_spshort_luaset_status_str`
`\l__spintent_spshort_luaset_layout_str`
`\l__spintent_spshort_luaset_expanded_str`
`\l__spintent_spshort_luaset_output_str`
`\l__spintent_spshort_luaset_base_str`
`\l__spintent_spshort_luaset_suffix_str`

These internal string variables serve as the primary data interface with the Lua module `spintent.lua`. During processing, the Lua module *"stores"* within these registers the execution status, the targeted typographical layout parameters, the phonetic replacement text for the `E` key intended for `NVDA`, the final sanitized output string, and the isolated structural components for ordinals (base and suffix).

```
\str_new:N  \l__spintent_spshort_luaset_status_str
\str_new:N  \l__spintent_spshort_luaset_layout_str
\str_new:N  \l__spintent_spshort_luaset_expanded_str
\str_new:N  \l__spintent_spshort_luaset_output_str
\str_new:N  \l__spintent_spshort_luaset_base_str
\str_new:N  \l__spintent_spshort_luaset_suffix_str
```

(*End of definition for* `\l__spintent_spshort_luaset_status_str` *and others.*)

11.18.1 Definition of error messages for `\spshort`

`unknown-abbreviation` The internal "error" message `unknown-abbreviation` is triggered when the command `\spshort` encounters an invalid lookup state from the Lua module. It acts as a strict fallback exception when the provided {⟨*argument*⟩} cannot be mapped to any registered database entry in `spintent.lua` nor successfully resolved through the grammatical ordinal expansion rules.

```
\msg_new:nnnn { spintent } { unknown-abbreviation }
  { The~abbreviation~or~ordinal~'#1'~is~invalid~or~not~recognized. }
  { Verify~the~spelling~or~check~the~grammar~rules~for~ordinals. }
```

(*End of definition for* `unknown-abbreviation`.)

11.18.2 Accessibility interface

`\__spintent_spshort_tag_span:nn`
`\__spintent_spshort_tag_span:en`

The internal function `\__spintent_spshort_tag_span:nn` processes the {⟨*argument*⟩} passed in '`#2`' using the tools provided by `tagpdf`. By utilizing the `tag` and `E` keys, it creates a native `PDF /E (#1)` structure and expands the {⟨*argument*⟩} passed in '`#1`' for proper reading with `NVDA`.

```
\cs_new_protected:Nn \__spintent_spshort_tag_span:nn
  {
    \tag_mc_end_push:
      \tag_struct_begin:n { tag=Span, E={ #1 } }
        \tag_mc_begin:n { tag=Span }
          \tag_suspend:n { \spshort }
            #2
          \tag_resume:n { \spshort }
        \tag_mc_end:
```

```
      \tag_struct_end:
    \tag_mc_begin_pop:n {}
  }
\cs_generate_variant:Nn \__spintent_spshort_tag_span:nn { en }
```

(*End of definition for* `\__spintent_spshort_tag_span:nn`.)

11.18.3 Definition of keys for `\spshort`

small-caps
read-arg

Now we add the keys `small-caps` and `read-arg`.

```
\keys_define:nn { spintent / spshort }
  {
    small-caps .bool_set:N = \l__spintent_spshort_smallcaps_bool,
    small-caps .initial:n  = false,
    small-caps .value_required:n = true,
    read-arg   .str_set:N  = \l__spintent_spshort_read_arg_str,
    read-arg   .initial:n  = {},
    read-arg   .value_required:n = true,
    unknown    .code:n =
      {
        \str_set:Nn \l__spintent_command_name_str { spshort }
        \__spintent_unknown_keys_cmd:n {#1}
      },
  }
```

(*End of definition for* `small-caps` *and* `read-arg`.)

11.18.4 Internal functions for `\spshort`

`\__spintent_spshort_print:nn`

The internal function `\__spintent_spshort_print:nn` acts as the primary formatter for text abbreviations. It clears the string variable `\l__spintent_spshort_read_arg_str` and evaluates the optional ⟨*keys*⟩ passed in '`#1`'. It then checks if a manual reading override was provided via those keys. If that string variable remains empty, it calls `\__spintent_spshort_process_and_render:n` to process the {⟨*argument*⟩} passed in '`#2`'. Otherwise, if an override is detected, it directly delegates the execution to `\__spintent_spshort_render_bypass:n`.

```
\cs_new_protected:Nn \__spintent_spshort_print:nn
  {
    \str_clear:N \l__spintent_spshort_read_arg_str
    \keys_set:nn { spintent / spshort } {#1}
    \str_if_empty:NTF \l__spintent_spshort_read_arg_str
      {
        \__spintent_spshort_process_and_render:n {#2}
      }
      {
        \__spintent_spshort_render_bypass:n {#2}
      }
  }
```

(*End of definition for* `\__spintent_spshort_print:nn`.)

`\__spintent_spshort_process_and_render:n`

The internal function `\__spintent_spshort_process_and_render:n` interfaces with the Lua module `spintent.lua` function `\__spintent_luafun_spshort_process:n` to resolve the abbreviation or ordinal {⟨*argument*⟩} passed in '`#1`'.

It subsequently evaluates the execution state stored within the string variable `\l__spintent_spshort_luaset_statu` and dynamically routes execution to the corresponding rendering subroutine `success`, `fallback` or `error`. Any unrecognized status automatically defaults to the "error" handler.

```
\cs_new_protected:Nn \__spintent_spshort_process_and_render:n
  {
    \__spintent_luafun_spshort_process:n {#1}
    \str_case:VnF \l__spintent_spshort_luaset_status_str
      {
        { success }  { \__spintent_spshort_render_success: }
        { fallback } { \__spintent_spshort_render_fallback: }
        { error }    { \__spintent_spshort_render_error:n {#1} }
      }
      { \__spintent_spshort_render_error:n {#1} }
  }
```

(*End of definition for* `\__spintent_spshort_process_and_render:n`.)

__spintent_spshort_render_success:
__spintent_spshort_render_fallback:
__spintent_spshort_render_bypass:n
__spintent_spshort_render_error:n

The function __spintent_spshort_render_success: injects the key E and executes the targeted typographical layout. The __spintent_spshort_render_fallback: function handles default cases, applying optional formatting such as SMALL CAPS to unrecognized or partially resolved terms.

Conversely, the function __spintent_spshort_render_bypass:n directly maps the user-defined *"speech"* to the {⟨*argument*⟩}, bypassing the automated lookup Lua module. Finally, the function __spintent_spshort_rende triggers a structural exception while yielding the unformatted {⟨*argument*⟩} to prevent fatal compilation.

```
\cs_new_protected:Nn \__spintent_spshort_render_success:
  {
    \__spintent_spshort_tag_span:en
      { \l__spintent_spshort_luaset_expanded_str }
      { \__spintent_spshort_execute_layout: }
  }
\cs_new_protected:Nn \__spintent_spshort_render_fallback:
  {
    \__spintent_spshort_tag_span:en
      { \l__spintent_spshort_luaset_expanded_str }
      {
        \bool_if:NTF \l__spintent_spshort_smallcaps_bool
          { \textsc{ \l__spintent_spshort_luaset_output_str } }
          { \l__spintent_spshort_luaset_output_str }
      }
  }
\cs_new_protected:Nn \__spintent_spshort_render_bypass:n
  {
    \__spintent_spshort_tag_span:en { \l__spintent_spshort_read_arg_str } { #1 }
  }
\cs_new_protected:Nn \__spintent_spshort_render_error:n
  {
    \msg_error:nnn { spintent } { unknown-abbreviation } {#1}
  }
```

(*End of definition for* __spintent_spshort_render_success: *and others.*)

__spintent_spshort_execute_layout:

The internal function __spintent_spshort_execute_layout: executes the typographical layout. It evaluates the descriptor *"stored"* within the string variable \l__spintent_spshort_luaset_layout_str (such as linear_regular, superscript, or small_caps_pure) and maps it to the corresponding LaTeX formatting. Furthermore, it manages complex nested configurations, dynamically applying combinations such as superscripts integrated with conditional SMALL CAPS based on the active boolean variables.

```
\cs_new_protected:Nn \__spintent_spshort_execute_layout:
  {
    \str_case:VnF \l__spintent_spshort_luaset_layout_str
      {
        { linear_regular }
          {
            \bool_if:NTF \l__spintent_spshort_smallcaps_bool
              { \textsc{ \l__spintent_spshort_luaset_output_str } }
              { \l__spintent_spshort_luaset_output_str }
          }
        { linear_caps }
          { \l__spintent_spshort_luaset_output_str }
        { linear_mixed }
          { \l__spintent_spshort_luaset_output_str }
        { superscript }
          {
            \l__spintent_spshort_luaset_base_str
            \bool_if:NTF \l__spintent_spshort_smallcaps_bool
              {
                \textsuperscript{
                  \textsc{
                    \str_lowercase:V \l__spintent_spshort_luaset_suffix_str
                  }
                }
              }
              {
                \textsuperscript{ \l__spintent_spshort_luaset_suffix_str }
              }
          }
        { small_caps_pure }
          {
            \bool_if:NTF \l__spintent_spshort_smallcaps_bool
```

```
                {
                  \textsc{
                    \str_lowercase:V \l__spintent_spshort_luaset_output_str
                  }
                }
                { \l__spintent_spshort_luaset_output_str }
            }
          { acronym_long }
            { \l__spintent_spshort_luaset_output_str }
        }
        { \l__spintent_spshort_luaset_output_str }
    }
```

(*End of definition for* `\__spintent_spshort_execute_layout:`.)

`\spshort` The public document command `\spshort` manages the accessible formatting of *text mode* abbreviations, acronyms, and ordinals. It first triggers an error if invoked within a *math mode*, upon validation, puts `\mode_leave_vertical:` and establishes a local group to call the function `\__spintent_spshort_print:nn` for processing and printing.

```
\NewDocumentCommand \spshort { O{} m }
  {
    \mode_if_math:TF
      {
        \msg_error:nnn { spintent } { need-text-mode } { \spshort }
      }
      {
        \mode_leave_vertical:
        \group_begin:
          \__spintent_spshort_print:nn {#1} {#2}
        \group_end:
      }
  }
```

(*End of definition for* `\spshort`. *This function is documented on page* 7.)

11.19 The command `\spsiglo`

The public document command `\spsiglo` provides a semantic interface for typesetting century designations (e.g., siglo XXI). It accepts both Arabic and Roman numerals as {⟨*argument*⟩}, interfacing with the Lua module `spintent.lua` to validate, parse, and normalize chronological values.

This command work in dual-mode context awareness, operating uniformly across both *text mode* and *math mode*. When executed within *math mode*, it constructs an accessible `MathML` `intent` expression tree. Conversely, when invoked in standard *text mode*, it encapsulates the rendered output within a native `PDF` tagging structure (such as `/Alt (#1)`) containing an `alt` key.

`\l__spintent_spsiglo_luaset_error_str`
`\l__spintent_spsiglo_luaset_arabic_str`
`\l__spintent_spsiglo_luaset_roman_min_str`
`\l__spintent_spsiglo_print_era_str`

These internal string variables handle data transfer and synchronization with the Lua module `spintent.lua`. They isolate the exception evaluation states, the Arabic numeral required for speech synthesis, the lowercase Roman numeral designated for small caps typesetting via `\textsc`, and the localized chronological era modifier.

```
\str_new:N \l__spintent_spsiglo_luaset_error_str
\str_new:N \l__spintent_spsiglo_luaset_arabic_str
\str_new:N \l__spintent_spsiglo_luaset_roman_min_str
\str_new:N \l__spintent_spsiglo_print_era_str
```

(*End of definition for* `\l__spintent_spsiglo_luaset_error_str` *and others.*)

11.19.1 Definition of error messages for `\spsiglo`

`invalid-century-format` This internal error message is issued when the provided century identifier fails both Arabic and Roman numeral validation. It is triggered when the Lua module `spintent.lua` cannot map the {⟨*argument*⟩} to an integer within the valid range of `1` to `4000`, preventing incorrect speech synthesis.

```
\msg_new:nnnn { spintent } { invalid-century-format }
  {
    Invalid~century~format~in~'#1'.
  }
  {
    The~argument~'#2'~is~not~a~valid~Arabic~or~Roman~numeral.~
    Ensure~the~value~is~between~1~and~4000.
  }
```

(*End of definition for* `invalid-century-format`.)

11.19.2 Definition of keys for \spsiglo

print-era

Now we add the key print-era. User configuration determining if an era designation is appended, available values are none, ac (antes de cristo) and dc (después de cristo).

```
\keys_define:nn { spintent / spsiglo }
  {
    print-era           .choices:nn =
      { none , ac , dc }
      {
        \int_case:nn { \l_keys_choice_int }
          {
            {1} { \str_set:Nn \l__spintent_spsiglo_print_era_str { none } }
            {2} { \str_set:Nn \l__spintent_spsiglo_print_era_str { ac } }
            {3} { \str_set:Nn \l__spintent_spsiglo_print_era_str { dc } }
          }
      },
    print-era           .initial:n  = none,
    print-era           .value_required:n = true,
    print-era / unknown .code:n =
      \msg_error:nneee { spintent } { unknown-choice }
        { print-era } { none, ~ac, ~dc } { \exp_not:n {#1} },
    unknown             .code:n =
      {
        \str_set:Nn \l__spintent_command_name_str { spsiglo }
        \__spintent_unknown_keys_cmd:n {#1}
      },
  }
```

(*End of definition for* print-era.)

11.19.3 Internal functions for \spsiglo

__spintent_spsiglo_render_simple:nn
__spintent_spsiglo_render_simple:VV
__spintent_spsiglo_render_with_era:nnn
__spintent_spsiglo_render_with_era:VVV

The functions __spintent_spsiglo_render_simple:nn and __spintent_spsiglo_render_with_era:nnn generate MathML structures when century designations are evaluated within *math mode*. They construct MathML intent structures, mapping the Arabic numerals to their phonetic equivalents. Furthermore, __spintent_spsiglo_render_with_era:nnn integrates the era modifiers ac or dc. It adds invisible spacing and postfix intents to ensure correct NVDA/MathCAT pronunciation.

```
\cs_new_protected:Nn \__spintent_spsiglo_render_simple:nn
  {
    \MathMLintent { #1:number } { \text { \textsc {#2} } }
  }
\cs_generate_variant:Nn \__spintent_spsiglo_render_simple:nn { VV }
\cs_new_protected:Nn \__spintent_spsiglo_render_with_era:nnn
  {
    \str_if_eq:nnTF { #1 } { ac }
      {
        \MathMLintent { antes-de-cristo:postfix($z) }
          {
            \__spintent_inner_sep:
            \MathMLarg { z }
              {
                \MathMLintent { #2:number }
                  {
                    \text { \textsc {#3}~a.~C. }
                  }
              }
          }
      }
      {
        \MathMLintent { después-de-cristo:postfix($z) }
          {
            \__spintent_inner_sep:
            \MathMLarg { z }
              {
                \MathMLintent { #2:number }
                  {
                    \text { \textsc {#3}~d.~C. }
                  }
              }
          }
      }
  }
```

```
\cs_generate_variant:Nn \__spintent_spsiglo_render_with_era:nnn { VVV }
```

(*End of definition for* `\__spintent_spsiglo_render_simple:nn` *and* `\__spintent_spsiglo_render_with_era:nnn`.)

`\__spintent_spsiglo_tag_span:nn`
`\__spintent_spsiglo_tag_span:en`

The function `\__spintent_spsiglo_tag_span:nn` handles century formatting within *text mode*. It utilizes the `tagpdf` package to enclose the content inside a `PDF /Alt (#1)` structure. By passing the {⟨*argument*⟩} to the `alt` key, it embeds the chronological value directly into the `PDF`, ensuring correct `NVDA/MathCAT` pronunciation without relying on `MathML` intents.

```
\cs_new_protected:Nn \__spintent_spsiglo_tag_span:nn
  {
    \tag_mc_end_push:
      \tag_struct_begin:n { tag=Span, alt={ #1 } }
        \tag_mc_begin:n { tag=Span }
          \tag_suspend:n { \spsiglo }
            #2
          \tag_resume:n { \spsiglo }
        \tag_mc_end:
      \tag_struct_end:
    \tag_mc_begin_pop:n {}
  }
\cs_generate_variant:Nn \__spintent_spsiglo_tag_span:nn { en }
```

(*End of definition for* `\__spintent_spsiglo_tag_span:nn`.)

`\__spintent_spsiglo_print_math:n`
`\__spintent_spsiglo_print_text:n`

The functions `\__spintent_spsiglo_print_math:n` and `\__spintent_spsiglo_print_text:n` format century designations based on the active mode. Both first check the "error" status in the variable `\l__spintent_spsiglo_luaset_error_str` returned by the Lua module `spintent.lua` and issue an "error" if validation fails.

If the {⟨*argument*⟩} is valid, the function `\__spintent_spsiglo_print_math:n` operates in *math mode*, applying `MathML` structures depending on whether an era modifier is present. Conversely, the function `\__spintent_spsiglo_print_text:n` operates in *text mode*, utilizing the `tag` and `alt` keys to expand the value of the `print-era` option into full text (e.g., "antes de cristo") for `NVDA/MathCAT`.

```
\cs_new_protected:Nn \__spintent_spsiglo_print_math:n
  {
    \str_if_eq:VnTF \l__spintent_spsiglo_luaset_error_str { true }
      {
        \msg_error:nnnn { spintent } { invalid-century-format }
          { \spsiglo } { #1 }
      }
      {
        \str_if_eq:VnTF \l__spintent_spsiglo_print_era_str { none }
          {
            \__spintent_spsiglo_render_simple:VV
              \l__spintent_spsiglo_luaset_arabic_str
              \l__spintent_spsiglo_luaset_roman_min_str
          }
          {
            \__spintent_spsiglo_render_with_era:VVV
              \l__spintent_spsiglo_print_era_str
              \l__spintent_spsiglo_luaset_arabic_str
              \l__spintent_spsiglo_luaset_roman_min_str
          }
      }
  }
\cs_new_protected:Nn \__spintent_spsiglo_print_text:n
  {
    \str_if_eq:VnTF \l__spintent_spsiglo_luaset_error_str { true }
      {
        \msg_error:nnnn { spintent } { invalid-century-format }
          { \spsiglo } { #1 }
      }
      {
         \str_case:Vn \l__spintent_spsiglo_print_era_str
          {
            { none }
            {
              \__spintent_spsiglo_tag_span:en
                { \l__spintent_spsiglo_luaset_arabic_str }
                { \textsc { \l__spintent_spsiglo_luaset_roman_min_str } }
            }
            { ac }
```

```
            {
              \__spintent_spsiglo_tag_span:en
                { \l__spintent_spsiglo_luaset_arabic_str ~ antes ~ de ~ cristo }
                { \textsc { \l__spintent_spsiglo_luaset_roman_min_str } ~ a.~C. }
            }
            { dc }
            {
              \__spintent_spsiglo_tag_span:en
                { \l__spintent_spsiglo_luaset_arabic_str ~ después ~ de ~ cristo }
                { \textsc { \l__spintent_spsiglo_luaset_roman_min_str } ~ d.~C. }
            }
          }
      }
  }
```

(*End of definition for* `\__spintent_spsiglo_print_math:n` *and* `\__spintent_spsiglo_print_text:n`.)

`\spsiglo` The public command `\spsiglo` is the main interface for century typesetting. It opens a local group and evaluates any optional ⟨*keys*⟩ passed in '`#1`', then executes the function `\__spintent_luafun_spsiglo_parse:n` defined in the Lua module `spintent.lua` passed in '`#2`' and formats the output for *math mode* or *text mode*.

```
\NewDocumentCommand \spsiglo { O{} m }
  {
    \group_begin:
      \keys_set:nn { spintent / spsiglo } { #1 }
      \__spintent_luafun_spsiglo_parse:n { #2 }
      \mode_if_math:TF
        { \__spintent_spsiglo_print_math:n { #2 } }
        {
          \mode_leave_vertical:
          \__spintent_spsiglo_print_text:n { #2 }
        }
    \group_end:
  }
```

(*End of definition for* `\spsiglo`. *This function is documented on page* 11.)

11.20 The command `\sptime`

The command `\sptime` is the main public interface for typesetting and tagging time expressions. It parses time string formats (such as `HH:MM` or `HH:MM:SS`) using the Lua function `\__spintent_luafun_sptime_parse:n`. This separates the hours, minutes, seconds, and fractions to construct a `MathML intent` tree for `NVDA/MathCAT`.

`\l__spintent_sptime_luaset_seconds_str`
`\l__spintent_sptime_luaset_has_seconds_str`
`\l__spintent_sptime_luaset_error_str`
`\l__spintent_sptime_luaset_is_fraction_str`
`\l__spintent_sptime_luaset_base_str`

These internal string variables store the values returned from the Lua module `spintent.lua`. They hold the extracted time units, error flags, and boolean indicators (such as the presence of seconds or decimal fractions) used during formatting.

```
\str_new:N \l__spintent_sptime_luaset_seconds_str
\str_new:N \l__spintent_sptime_luaset_has_seconds_str
\str_new:N \l__spintent_sptime_luaset_error_str
\str_new:N \l__spintent_sptime_luaset_is_fraction_str
\str_new:N \l__spintent_sptime_luaset_base_str
```

(*End of definition for* `\l__spintent_sptime_luaset_seconds_str` *and others.*)

11.20.1 Definition of error messages

`invalid-time-format` The error message `invalid-time-format` defines the text displayed when an invalid time input is detected. It is triggered when an argument fails the format checks or contains values outside normal ranges, such as hours exceeding 23 or minutes and seconds exceeding 59.

```
\msg_new:nnnn { spintent } { invalid-time-format }
  {
    Invalid~time~format~in~#1. \\
    The~argument~'#2'~does~not~match~any~supported~time~format.
  }
  {
    Allowed~formats:~HH:MM,~HH:MM:SS,~or~HH:MM:SS.ss. \\
    Please~verify~that~hours~are~0-23~and~minutes/seconds~are~0-59.
  }
```

(*End of definition for* `invalid-time-format`.)

11.20.2 Internal functions for \sptime

__spintent_sptime_build_seconds:nn

The internal function __spintent_sptime_build_seconds:nn builds the MathML structure for the seconds component of a time expression. It wraps the numeric value '#2' in speech intent tags, applying a connector prefix (passed in '#1') and the "segundos" postfix so it is read correctly by NVDA/MathCAT.

```
\cs_new_protected:Nn \__spintent_sptime_build_seconds:nn
  {
    \MathMLintent { #1:prefix($z) }
      {
        \text { : }
        \MathMLarg { z }
          {
            \MathMLintent { _segundos:postfix($m) }
              {
                \__spintent_inner_sep:
                \MathMLarg { m }
                  {
                    \text { #2 }
                  }
              }
          }
      }
  }
```

(*End of definition for* __spintent_sptime_build_seconds:nn.)

__spintent_sptime_build_tree:

The internal function __spintent_sptime_build_tree: builds the complete MathML structure for the time expression. It checks the string variables \l__spintent_sptime_luaset_has_seconds_str and \l__spintent_sptime_luaset_is_fraction_str to determine if the time includes seconds or decimal fractions. Based on these values, it inserts the base time string and either calls __spintent_sptime_build_seconds: with the appropriate prefix ('con' or 'minutos-con'), or simply appends the 'minutos' intent tag if no seconds are present.

```
\cs_new_protected:Nn \__spintent_sptime_build_tree:
  {
    \str_if_eq:VnTF \l__spintent_sptime_luaset_has_seconds_str { true }
      {
        \MathMLintent { :time }
          {
            \text { \l__spintent_sptime_luaset_base_str }
          }
        \str_if_eq:VnTF \l__spintent_sptime_luaset_is_fraction_str { true }
          {
            \__spintent_sptime_build_seconds:nn { con }
              { \l__spintent_sptime_luaset_seconds_str }
          }
          {
            \__spintent_sptime_build_seconds:nn { _minutos-con }
              { \l__spintent_sptime_luaset_seconds_str }
          }
      }
      {
        \MathMLintent { :time }
          {
            \text { \l__spintent_sptime_luaset_base_str }
          }
        \str_if_eq:VnT \l__spintent_sptime_luaset_is_fraction_str { false }
          {
            \MathMLintent { _minutos }
              {
                \c__spintent_invisible_sep_int
              }
          }
      }
  }
```

(*End of definition for* __spintent_sptime_build_tree:.)

\sptime

The command \sptime is the public interface for formatting time expressions. It first checks if it is being used in math mode, issuing an error if not. Inside a local group, it parses the input using __spintent_luafun_sptime_parse:n and checks the error status in \l__spintent_sptime_luaset_error_str. If

an invalid format is detected, it issues the `invalid-time-format` error; otherwise, it calls `\__spintent_sptime_build_tree:` to assemble the MathML output.

```
\NewDocumentCommand \sptime { m }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \__spintent_luafun_sptime_parse:n { #1 }
          \str_if_eq:VnTF \l__spintent_sptime_luaset_error_str { true }
            {
               \msg_error:nnnn { spintent } { invalid-time-format }
                 { \sptime } { #1 }
            }
            {
               \__spintent_sptime_build_tree:
            }
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \sptime } }
  }
```

(*End of definition for* `\sptime`. *This function is documented on page 11.*)

11.21 The command \spdate

The command `\spdate` is the main public command for typesetting and tagging date expressions. It parses date string formats (such as `DD/MM/YYYY` or `YYYY-MM-DD`) using the Lua function `\__spintent_luafun_spdate_parse:n`. This verifies the input and constructs a MathML `intent` tree for `NVDA/MathCAT`.

`\l__spintent_spdate_luaset_output_str`
`\l__spintent_spdate_luaset_error_str`

These internal string variables store the values returned from the Lua module `spintent.lua`. They hold the formatted date string and the error status used during formatting.

```
\str_new:N \l__spintent_spdate_luaset_output_str
\str_new:N \l__spintent_spdate_luaset_error_str
```

(*End of definition for* `\l__spintent_spdate_luaset_output_str` *and* `\l__spintent_spdate_luaset_error_str`.)

11.21.1 Definition of error messages

`invalid-date-format`

The error message `invalid-date-format` defines the text displayed when an invalid date input is detected. It is triggered when an argument fails the format checks or represents an invalid calendar date, such as `31/02/2024` or using an unsupported pattern like `YYYY/MM/DD`.

```
\msg_new:nnnn { spintent } { invalid-date-format }
  {
    Invalid~date~or~unsupported~format~in~#1. \\
    The~argument~'#2'~does~not~match~a~supported~format.
  }
  {
    Allowed~formats:~DD\slash MM\slash YYYY,~DD-MM-YYYY,~or~YYYY-MM-DD. \\
    Ensure~the~day~and~month~are~valid~(e.g.,~avoid~31\slash 02\slash 2024).
  }
```

(*End of definition for* `invalid-date-format`.)

11.21.2 Internal functions for \spdate

`\__spintent_spdate_process:n`

The internal function `\__spintent_spdate_process:n` parses the input using `\__spintent_luafun_spdate_parse:n` and checks the error status in `\l__spintent_spdate_luaset_error_str`. If an invalid format is detected, it issues the `invalid-date-format` error. Otherwise, it builds the MathML `intent` structure using the formatted date stored in `\l__spintent_spdate_luaset_output_str`.

```
\cs_new_protected:Nn \__spintent_spdate_process:n
  {
    \__spintent_luafun_spdate_parse:n { #1 }
    \str_if_eq:VnTF \l__spintent_spdate_luaset_error_str { true }
      {
        \msg_error:nnnn { spintent } { invalid-date-format }
          { \spdate } { #1 }
      }
      {
        \MathMLintent { :date }
          {
            \text { \l__spintent_spdate_luaset_output_str }
          }
      }
  }
```

(*End of definition for* `\__spintent_spdate_process:n`.)

`\spdate` The command `\spdate` is the public interface for formatting date expressions. It first checks if it is being used in math mode, issuing an error if not. Inside a local group, it calls `\__spintent_spdate_process:n` to process the input.

```
\NewDocumentCommand \spdate { m }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \__spintent_spdate_process:n { #1 }
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spdate } }
  }
```

(*End of definition for* `\spdate`. *This function is documented on page* 11.)

11.22 Semantic Operators

This section provides purely semantic operators. Unlike their formatting counterparts (which parse and format numbers), these commands do not process mandatory arguments. Instead, they solely inject a mathematical glyph wrapped in its corresponding regional `MathCAT` intent. This grants teachers absolute control when building complex algebraic expressions, mixed numbers, or fraction divisions.

11.23 The command `\spusep`

For the use of parentheses, we will provide the command `\spusep` with only two ⟨*keys*⟩ `size` and `silent`. Most of the time we can use `\left` and `\right` or `\Uleft` and `\Uright`, but both forms create a group, and this hinders the code for *tagged* `PDF` when we want to interrupt something between them.

`spusep-empty-arg` `spusep-invalid-arg` The first "error" message `spusep-empty-arg` is raised when `\spusep` receives an empty or blank {⟨*argument*⟩}, the second this is when an input is invalid.

```
\msg_new:nnnn { spintent } { spusep-empty-arg }
  {
    Empty~argument~'#1'~for~\token_to_str:N \spusep.
  }
  {
    \token_to_str:N \spusep~requires~a~non-empty~argument.
  }
\msg_new:nnn { spintent } { spusep-invalid-arg }
  {
    Invalid~argument~'#1'~for~\token_to_str:N \spusep.
  }
```

(*End of definition for* `spusep-empty-arg` *and* `spusep-invalid-arg`.)

`\l__spintent_spusep_arg_tl` `\l__spintent_spusep_left_tl` `\l__spintent_spusep_right_tl` The variable `\l__spintent_spusep_arg_tl` stores the {⟨*argument*⟩}, the variable `\l__spintent_spusep_left_tl` and `\l__spintent_spusep_right_tl` stores the parentheses.

```
\tl_new:N \l__spintent_spusep_arg_tl
\tl_new:N \l__spintent_spusep_left_tl
\tl_new:N \l__spintent_spusep_right_tl
```

(*End of definition for* `\l__spintent_spusep_arg_tl`, `\l__spintent_spusep_left_tl`, *and* `\l__spintent_spusep_right_tl`.)

`size` `silent` `unknown` We define the ⟨*keys*⟩ `size` and `silent`.

```
\keys_define:nn { spintent / spusep }
  {
    size           .choices:nn =
      { none , big , Big, bigg, Bigg }
      {
        \int_case:nn { \l_keys_choice_int }
          {
            { 1 }
              {
                \tl_clear:N \l__spintent_spusep_left_tl
                \tl_clear:N \l__spintent_spusep_right_tl
              }
            { 2 }
              {
                \tl_set:Nn \l__spintent_spusep_left_tl  { \bigl }
                \tl_set:Nn \l__spintent_spusep_right_tl { \bigr }
              }
```

```
            { 3 }
              {
                \tl_set:Nn \l__spintent_spusep_left_tl  { \Bigl }
                \tl_set:Nn \l__spintent_spusep_right_tl { \Bigr }
              }
            { 4 }
              {
                \tl_set:Nn \l__spintent_spusep_left_tl  { \biggl }
                \tl_set:Nn \l__spintent_spusep_right_tl { \biggr }
              }
            { 5 }
              {
                \tl_set:Nn \l__spintent_spusep_left_tl  { \Biggl }
                \tl_set:Nn \l__spintent_spusep_right_tl { \Biggr }
              }
          }
      },
    size           .initial:n  = none,
    size           .value_required:n = true,
    size / unknown .code:n =
      \msg_error:nneee { spintent } { unknown-choice }
        { size } { none,~big,~Big,~bigg,~Bigg } { \exp_not:n {#1} },
    silent         .bool_set:N = \l__spintent_spusep_silent_bool,
    silent         .initial:n = false,
    silent         .value_required:n = true,
    unknown        .code:n =
      {
        \str_set:Nn \l__spintent_command_name_str { spusep }
        \__spintent_unknown_keys_cmd:n {#1}
      },
  }
```

(*End of definition for* `size`, `silent`, *and* `unknown`.)

`\__spintent_spusep_set_parens:nn`

The function `\__spintent_spusep_set_parens:nn` will place the argument passed in '`#2`' to the right of the variables `\l__spintent_spusep_left_tl` and `\l__spintent_spusep_right_tl` and then check the state of the variable `\l__spintent_spusep_silent_bool` handled by the key `silent` to print.

```
\cs_new_protected:Nn \__spintent_spusep_set_parens:nn
  {
    \tl_put_right:cn { l__spintent_spusep_ #1 _tl } { #2 }
    \bool_if:NTF \l__spintent_spusep_silent_bool
      {
         \MathMLintent { :silent } { \tl_use:c { l__spintent_spusep_ #1 _tl } }
      }
      {
        \tl_use:c { l__spintent_spusep_ #1 _tl }
      }
  }
```

(*End of definition for* `\__spintent_spusep_set_parens:nn`.)

`\__spintent_spusep_convert:n`

The function `\__spintent_spusep_convert:n` captures and stores the {⟨*argument*⟩} passed in '`#1`' in the variable `\l__spintent_spusep_arg_tl`, validates the cases by converting them to the macros provided by `unicode-math`, and passes them to the function `\__spintent_spusep_set_parens:nn` for printing. If the input is not supported or is empty, we return an "error".

```
\cs_new_protected:Nn \__spintent_spusep_convert:n
  {
    \tl_set:Ne \l__spintent_spusep_arg_tl { \tl_trim_spaces:n {#1} }
    \tl_if_empty:NT \l__spintent_spusep_arg_tl
      {
        \msg_error:nnn { spintent } { spusep-empty-arg } {#1}
      }
    \str_case:VnF \l__spintent_spusep_arg_tl
      {
        { (  }
          {
            \__spintent_spusep_set_parens:nn { left } { \lparen }
          }
        { )  }
          {
            \__spintent_spusep_set_parens:nn { right } { \rparen }
```

```
            }
          { [  }
            {
              \__spintent_spusep_set_parens:nn { left } { \lbrack }
            }
          { ]  }
            {
              \__spintent_spusep_set_parens:nn { right } { \rbrack }
            }
          { \{ }
            {
              \__spintent_spusep_set_parens:nn { left } { \lbrace }
            }
          { \} }
            {
              \__spintent_spusep_set_parens:nn { right } { \rbrace }
            }
        }
        {
          \msg_error:nnn { spintent } { spusep-invalid-arg } {#1}
        }
    }
```

(*End of definition for* `\__spintent_spusep_convert:n`.)

`\spusep` The command `\spusep` checks that it is running in math mode, opens a local group, sets the keys passed in '`#1`' and processes the {⟨*argument*⟩} passed in '`#2`' by means of the function `\__spintent_spusep_convert:n`.

```
\NewDocumentCommand \spusep { O{} m }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \keys_set:nn { spintent / spusep } {#1}
          \__spintent_spusep_convert:n {#2}
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spusep } }
  }
```

(*End of definition for* `\spusep`. *This function is documented on page 12.*)

11.23.1 The command `\spread`

The command `\spread` provides a semantic read for `MathCAT` of mathematical symbols for situations in which it is necessary to specify gender or plurality in Spanish.

`spread-empty-args` The error message `spread-empty-args` is raised when `\spread` receives an empty or blank {⟨*argument*⟩}.

```
\msg_new:nnnn { spintent } { spread-empty-args }
  { Empty~argument~in~\token_to_str:N \spread. }
  { \token_to_str:N \spread~requires~two~non-empty~arguments. }
```

(*End of definition for* `spread-empty-args`.)

`\l__spintent_spread_intent_tl` The variable `\l__spintent_spread_intent_tl` stores the sanitized semantic reading {⟨*argument*⟩} for symbol to be injected into the `\MathMLintent`.

```
\tl_new:N \l__spintent_spread_intent_tl
```

(*End of definition for* `\l__spintent_spread_intent_tl`.)

`\__spintent_spread_exec:nn` The function `\__spintent_spread_exec:nn` implements the core logic for `\spread`. It verifies that neither {⟨*argument*⟩} is empty or blank. If valid, it evaluates the *reading* {⟨*argument*⟩} '`#1`' using `\__spintent_sanitize_affix:Nn` and embeds it via `\MathMLintent` into the *math symbol* '`#2`'.

```
\cs_new_protected:Nn \__spintent_spread_exec:nn
  {
    \bool_lazy_or:nnTF { \tl_if_blank_p:n { #1 } } { \tl_if_blank_p:n { #2 } }
      { \msg_error:nn { spintent } { spread-empty-args } }
      {
        \__spintent_sanitize_affix:Nn \l__spintent_spread_intent_tl { #1 }
        \MathMLintent { \l__spintent_spread_intent_tl } { #2 }
      }
  }
```

(*End of definition for* `\__spintent_spread_exec:nn`.)

\spread The command \spread assigns a custom semantic *reading* '#1' to any mathematical *symbol* '#2'. It enforces math mode and passes execution to __spintent_spread_exec:nn.

```
\NewDocumentCommand \spread { m m }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \__spintent_spread_exec:nn { #1 } { #2 }
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spread } }
  }
```

(*End of definition for* \spread. *This function is documented on page* 11.)

11.23.2 The command \spdsym

\l__spintent_spdsym_read_sym_tl
\l__spintent_spdsym_symbol_tl
\l__spintent_spdsym_prefix_tl
\l__spintent_spdsym_suffix_tl

Variables to store the regional reading string, the visual operator, and the optional semantic affixes specifically for \spdsym.

```
\tl_new:N \l__spintent_spdsym_read_sym_tl
\tl_new:N \l__spintent_spdsym_symbol_tl
\tl_new:N \l__spintent_spdsym_prefix_tl
\tl_new:N \l__spintent_spdsym_suffix_tl
```

(*End of definition for* \l__spintent_spdsym_read_sym_tl *and others.*)

prefix
suffix
set-sym
read-sym
unknown

We define the ⟨*keys*⟩ prefix, suffix, set-sym and read-sym for \spdsym.

```
\keys_define:nn { spintent / spdsym }
  {
    prefix             .code:n =
      { \__spintent_sanitize_affix:Nn \l__spintent_spdsym_prefix_tl {#1} },
    prefix             .initial:n = ,
    prefix             .value_required:n = true,
    suffix             .code:n =
      { \__spintent_sanitize_affix:Nn \l__spintent_spdsym_suffix_tl {#1} },
    suffix             .initial:n = ,
    suffix             .value_required:n = true,
    set-sym            .choices:nn =
      { old-style , two-dots , slash }
      {
        \int_case:nn { \l_keys_choice_int }
          {
            {1} { \tl_set:Nn \l__spintent_spdsym_symbol_tl { \div } }
            {2} { \tl_set:Nn \l__spintent_spdsym_symbol_tl {   :  } }
            {3} { \tl_set:Nn \l__spintent_spdsym_symbol_tl {   /  } }
          }
      },
    set-sym            .initial:n  = two-dots,
    set-sym            .value_required:n = true,
    set-sym / unknown  .code:n =
      \msg_error:nneee { spintent } { unknown-choice }
        { set-sym } { old-style, ~two-dots, ~slash } { \exp_not:n {#1} },
    read-sym           .choices:nn =
      { dividido , dividido-por , dividido-entre }
      {
        \int_case:nn { \l_keys_choice_int }
          {
            {1} { \tl_set:Nn \l__spintent_spdsym_read_sym_tl { dividido       } }
            {2} { \tl_set:Nn \l__spintent_spdsym_read_sym_tl { dividido-por   } }
            {3} { \tl_set:Nn \l__spintent_spdsym_read_sym_tl { dividido-entre } }
          }
      },
    read-sym           .initial:n  = dividido,
    read-sym           .value_required:n = true,
    read-sym / unknown .code:n =
      \msg_error:nneee { spintent } { unknown-choice }
        { read-sym } { dividido, ~dividido-por, ~dividido-entre }
        { \exp_not:n {#1} },
    unknown            .code:n =
      {
        \str_set:Nn \l__spintent_command_name_str { spdsym }
        \__spintent_unknown_keys_cmd:n {#1}
```

```
      },
  }
```

(*End of definition for* `prefix` *and others.*)

\spdsym Defines the \spdsym operator. It takes a single optional argument for local setup and outputs the configured intent structure, safely attaching prefixes and suffixes if present.

```
\NewDocumentCommand \spdsym { O{} }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \keys_set:nn { spintent / spdsym } { #1 }
          \MathMLintent
            {
              \tl_if_empty:NF \l__spintent_spdsym_prefix_tl
                { \l__spintent_spdsym_prefix_tl - }
              \l__spintent_spdsym_read_sym_tl
              \tl_if_empty:NF \l__spintent_spdsym_suffix_tl
                { - \l__spintent_spdsym_suffix_tl }
            }
            { \l__spintent_spdsym_symbol_tl }
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spdsym } }
  }
```

(*End of definition for* `\spdsym`*. This function is documented on page 12.*)

11.23.3 The command \spmsym

\l__spintent_spmsym_read_sym_tl
\l__spintent_spmsym_symbol_tl
\l__spintent_spmsym_prefix_tl
\l__spintent_spmsym_suffix_tl

Variables to store the regional reading string, the visual operator, and the optional semantic affixes specifically for \spmsym.

```
\tl_new:N \l__spintent_spmsym_read_sym_tl
\tl_new:N \l__spintent_spmsym_symbol_tl
\tl_new:N \l__spintent_spmsym_prefix_tl
\tl_new:N \l__spintent_spmsym_suffix_tl
```

(*End of definition for* `\l__spintent_spmsym_read_sym_tl` *and others.*)

prefix
suffix
set-sym
read-sym
not-print
unknown

We define the ⟨*keys*⟩ prefix, suffix, set-sym, read-sym and not-print for \spmsym.

```
\keys_define:nn { spintent / spmsym }
  {
    prefix              .code:n =
      { \__spintent_sanitize_affix:Nn \l__spintent_spmsym_prefix_tl {#1} },
    prefix              .initial:n = ,
    prefix              .value_required:n = true,
    suffix              .code:n =
      { \__spintent_sanitize_affix:Nn \l__spintent_spmsym_suffix_tl {#1} },
    suffix              .initial:n = ,
    suffix              .value_required:n = true,
    set-sym             .choices:nn =
      { cross , dot , asterisk }
      {
        \int_case:nn { \l_keys_choice_int }
          {
            {1} { \tl_set:Nn \l__spintent_spmsym_symbol_tl { \times } }
            {2} { \tl_set:Nn \l__spintent_spmsym_symbol_tl { \cdot  } }
            {3} { \tl_set:Nn \l__spintent_spmsym_symbol_tl { \ast   } }
          }
      },
    set-sym             .initial:n  = dot,
    set-sym             .value_required:n = true,
    set-sym / unknown   .code:n =
      \msg_error:nneee { spintent } { unknown-choice }
        { set-sym } { cross, ~dot, ~asterisk } { \exp_not:n {#1} },
    read-sym            .choices:nn =
      { por , multiplicado-por , veces }
      {
        \int_case:nn { \l_keys_choice_int }
          {
            {1} { \tl_set:Nn \l__spintent_spmsym_read_sym_tl { por } }
            {2} { \tl_set:Nn \l__spintent_spmsym_read_sym_tl { multiplicado-por } }
```

```
            {3} { \tl_set:Nn \l__spintent_spmsym_read_sym_tl { veces } }
          }
      },
    read-sym           .initial:n  = por,
    read-sym           .value_required:n = true,
    read-sym / unknown .code:n =
      \msg_error:nneee { spintent } { unknown-choice }
        { read-sym } { por, ~multiplicado-por, ~veces } { \exp_not:n {#1} },
    not-print          .bool_set:N = \l__spintent_spmsym_not_print_bool,
    not-print          .initial:n  = false,
    not-print          .value_required:n = true,
    unknown            .code:n =
      {
        \str_set:Nn \l__spintent_command_name_str { spmsym }
        \__spintent_unknown_keys_cmd:n {#1}
      },
  }
```

(*End of definition for* `prefix` *and others.*)

\spmsym Defines the \spmsym operator. Similar to \spdsym, it outputs the correct mathematical glyph based on the local ⟨*keys*⟩, safely attaching prefixes and suffixes if present.

```
\NewDocumentCommand \spmsym { O{} }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \keys_set:nn { spintent / spmsym } { #1 }
          \MathMLintent
            {
              \tl_if_empty:NF \l__spintent_spmsym_prefix_tl
                { \l__spintent_spmsym_prefix_tl - }
              \l__spintent_spmsym_read_sym_tl
              \tl_if_empty:NF \l__spintent_spmsym_suffix_tl
                { - \l__spintent_spmsym_suffix_tl }
            }
            {
              \bool_if:NTF \l__spintent_spmsym_not_print_bool
                {
                  \invisibletimes
                }
                { \l__spintent_spmsym_symbol_tl }
            }
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spmsym } }
  }
```

(*End of definition for* \spmsym*. This function is documented on page 12.*)

11.24 The command \spdiv

The user-level command \spdiv formats division representations. It takes two mandatory arguments: the {⟨*dividend*⟩} and the {⟨*divisor*⟩}. Both {⟨*arguments*⟩} are internally processed by the \spnum command to ensure proper digit separation. This command enforces strict validation, ensuring that neither the {⟨*dividend*⟩} nor the {⟨*divisor*⟩} is *blank*.

11.24.1 Definition of keys for \spdiv

cifras-sep
read-sign
read-period-sep
notation-sym
set-sym
read-sym
wrap-parens
read-parens

We define the ⟨*keys*⟩ wrap-parens and read-parens for \spdiv command. Number-related ⟨*keys*⟩ cifras-sep, read-sign, read-period-sep and notation-sym are forwarded to \spnum command using .meta:nn, while semantic operator ⟨*keys*⟩ set-sym and read-sym are forwarded to the \spdsym command using .meta:nn.

```
\keys_define:nn { spintent / spdiv }
  {
    cifras-sep      .meta:nn = { spintent / spnum  } { cifras-sep      = {#1} },
    read-sign       .meta:nn = { spintent / spnum  } { read-sign       = {#1} },
    read-period-sep .meta:nn = { spintent / spnum  } { read-period-sep = {#1} },
    notation-sym    .meta:nn = { spintent / spnum  } { notation-sym    = {#1} },
    set-sym         .meta:nn = { spintent / spdsym } { set-sym         = {#1} },
    read-sym        .meta:nn = { spintent / spdsym } { read-sym        = {#1} },
    wrap-parens     .bool_set:N = \l__spintent_spdiv_wrap_parens_bool,
    wrap-parens     .initial:n  = false,
```

```
    wrap-parens     .value_required:n = true,
    read-parens     .bool_set:N = \l__spintent_spdiv_read_parens_bool,
    read-parens     .initial:n  = true,
    read-parens     .value_required:n = true,
    unknown         .code:n =
      {
        \str_set:Nn \l__spintent_command_name_str { spdiv }
        \__spintent_unknown_keys_cmd:n {#1}
      },
  }
```

(*End of definition for* `cifras-sep` *and others.*)

11.24.2 Definition of error messages

`spdiv-empty-argument` Raised when either the {⟨*dividend*⟩} or the {⟨*divisor*⟩} is missing or *blank*.

```
\msg_new:nnnn { spintent } { spdiv-empty-argument }
  { The~arguments~for~\token_to_str:N \spdiv~cannot~be~empty. }
  {
    You~must~provide~both~a~dividend~and~a~divisor.\\
  }
```

(*End of definition for* `spdiv-empty-argument`.)

11.24.3 Internal functions for \spdiv

`\__spintent_spdiv_render_op:nn` The function `\__spintent_spdiv_render_op:nn` parses both {⟨*arguments*⟩} through the `\spnum` command and embeds the semantic operator from `\spdsym`.

```
\cs_new_protected:Nn \__spintent_spdiv_render_op:nn
  {
    \spnum {#1}
    \MathMLintent { \l__spintent_spdsym_read_sym_tl }
      {
        \l__spintent_spdsym_symbol_tl
      }
    \spnum {#2}
  }
```

(*End of definition for* `\__spintent_spdiv_render_op:nn`.)

`\__spintent_spdiv_render_parens:nn` The function `\__spintent_spdiv_render_parens:nn` handles the structural parentheses around the division.

```
\cs_new_protected:Nn \__spintent_spdiv_render_parens:nn
  {
    \bool_if:NTF \l__spintent_spdiv_read_parens_bool
      {
        ( \__spintent_spdiv_render_op:nn {#1} {#2} )
      }
      {
        \MathMLintent { :silent } { ( }
        \__spintent_spdiv_render_op:nn {#1} {#2}
        \MathMLintent { :silent } { ) }
      }
  }
```

(*End of definition for* `\__spintent_spdiv_render_parens:nn`.)

`\__spintent_spdiv_process:nnn` The function `\__spintent_spdiv_process:nnn` sets ⟨*keys*⟩, validates {⟨*arguments*⟩}, and decides on parenthesis wrapping.

```
\cs_new_protected:Nn \__spintent_spdiv_process:nnn
  {
    \keys_set:nn { spintent / spdiv } { #1 }
    \tl_if_blank:nTF { #2 }
      { \msg_error:nnn { spintent } { spdiv-empty-argument } }
      {
        \tl_if_blank:nTF { #3 }
          { \msg_error:nnn { spintent } { spdiv-empty-argument } }
          {
            \bool_if:NTF \l__spintent_spdiv_wrap_parens_bool
              { \__spintent_spdiv_render_parens:nn {#2} {#3} }
              { \__spintent_spdiv_render_op:nn {#2} {#3} }
          }
      }
  }
```

(*End of definition for* `\__spintent_spdiv_process:nnn`.)

`\spdiv` Defines the user command `\spdiv`. It strictly relies on *math mode*.

```
\NewDocumentCommand \spdiv { O{} m m }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \__spintent_spdiv_process:nnn { #1 } { #2 } { #3 }
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spdiv } }
  }
```

(*End of definition for* `\spdiv`. *This function is documented on page 13.*)

11.25 The command `\spmul`

The user-level command `\spmul` formats multiplication representations. It takes two mandatory arguments: the {⟨*first factor*⟩} and the {⟨*second factor*⟩}. Both {⟨*arguments*⟩} are internally processed by the `\spnum` command to ensure proper digit separation. This command enforces strict validation, ensuring that neither the {⟨*first factor*⟩} nor the {⟨*second factor*⟩} is *blank*.

11.25.1 Definition of keys for `\spmul`

`cifras-sep` `read-sign` `read-period-sep` `notation-sym` `set-sym` `read-sym` `wrap-parens` `read-parens` We define the ⟨*keys*⟩ `wrap-parens` and `read-parens` for `\spmul` command. Number-related ⟨*keys*⟩ `cifras-sep`, `read-sign`, `read-period-sep` and `notation-sym` are forwarded to `\spnum` command using `.meta:nn`, while semantic operator ⟨*keys*⟩ `set-sym` and `read-sym` are forwarded to the `\spmsym` command using `.meta:nn`.

```
\keys_define:nn { spintent / spmul }
  {
    cifras-sep      .meta:nn = { spintent / spnum  } { cifras-sep       = {#1} },
    read-sign       .meta:nn = { spintent / spnum  } { read-sign        = {#1} },
    read-period-sep .meta:nn = { spintent / spnum  } { read-period-sep  = {#1} },
    notation-sym    .meta:nn = { spintent / spnum  } { notation-sym     = {#1} },
    set-sym         .meta:nn = { spintent / spmsym } { set-sym          = {#1} },
    read-sym        .meta:nn = { spintent / spmsym } { read-sym         = {#1} },
    wrap-parens     .bool_set:N = \l__spintent_spmul_wrap_parens_bool,
    wrap-parens     .initial:n  = false,
    wrap-parens     .value_required:n = true,
    read-parens     .bool_set:N = \l__spintent_spmul_read_parens_bool,
    read-parens     .initial:n  = true,
    read-parens     .value_required:n = true,
    unknown         .code:n =
      {
        \str_set:Nn \l__spintent_command_name_str { spmul }
        \__spintent_unknown_keys_cmd:n {#1}
      },
  }
```

(*End of definition for* `cifras-sep` *and others.*)

11.25.2 Definition of error messages

`spmul-empty-argument` Raised when either the {⟨*first factor*⟩} or the {⟨*second factor*⟩} is missing or *blank*.

```
\msg_new:nnnn { spintent } { spmul-empty-argument }
  { The~arguments~for~\token_to_str:N \spmul~cannot~be~empty. }
  {
    You~must~provide~both~first~factor~and~second~factor.\\
  }
```

(*End of definition for* `spmul-empty-argument`.)

11.25.3 Internal functions for `\spmul`

`\__spintent_spmul_render_op:nn` The function `\__spintent_spmul_render_op:nn` parses both {⟨*arguments*⟩} through the `\spnum` command and embeds the semantic operator from `\spmsym`.

```
\cs_new_protected:Nn \__spintent_spmul_render_op:nn
  {
    \spnum {#1}
    \MathMLintent { \l__spintent_spmsym_read_sym_tl }
      {
        \l__spintent_spmsym_symbol_tl
      }
    \spnum {#2}
  }
```

(*End of definition for* `\__spintent_spmul_render_op:nn`.)

`\__spintent_spmul_render_parens:nn`

The function `\__spintent_spmul_render_parens:nn` handles the structural parentheses around the multiplication.

```
\cs_new_protected:Nn \__spintent_spmul_render_parens:nn
  {
    \bool_if:NTF \l__spintent_spmul_read_parens_bool
      {
        ( \__spintent_spmul_render_op:nn {#1} {#2} )
      }
      {
        \MathMLintent { :silent } { ( }
        \__spintent_spmul_render_op:nn {#1} {#2}
        \MathMLintent { :silent } { ) }
      }
  }
```

(*End of definition for* `\__spintent_spmul_render_parens:nn`.)

`\__spintent_spmul_process:nnn`

The function `\__spintent_spmul_process:nnn` sets ⟨*keys*⟩, validates {⟨*arguments*⟩}, and decides on parenthesis wrapping.

```
\cs_new_protected:Nn \__spintent_spmul_process:nnn
  {
    \keys_set:nn { spintent / spmul } { #1 }
    \tl_if_blank:nTF { #2 }
      { \msg_error:nnn { spintent } { spmul-empty-argument } }
      {
        \tl_if_blank:nTF { #3 }
          { \msg_error:nnn { spintent } { spmul-empty-argument } }
          {
            \bool_if:NTF \l__spintent_spmul_wrap_parens_bool
              { \__spintent_spmul_render_parens:nn {#2} {#3} }
              { \__spintent_spmul_render_op:nn {#2} {#3} }
          }
      }
  }
```

(*End of definition for* `\__spintent_spmul_process:nnn`.)

`\spmul`

Defines the user command `\spmul`. It strictly relies on *math mode*.

```
\NewDocumentCommand \spmul { O{} m m }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \__spintent_spmul_process:nnn { #1 } { #2 } { #3 }
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spmul } }
  }
```

(*End of definition for* `\spmul`. *This function is documented on page 13.*)

11.26 The commands `\spmcd` and `\spmcm`

The user-level commands `\spmcd` and `\spmcm` provide a semantic interface for the Greatest Common Divisor (MCD in Spanish) and the Least Common Multiple (MCM in Spanish). They delegate argument validation and calculation to the Lua module `spintent.lua`, constructing a valid `MathCAT` intent.

11.26.1 Variables for `\spmcd` and `\spmcm`

`\l__spintent_spmcm_spmcd_luaset_error_str`
`\l__spintent_mc_luaset_types_str`
`\l__spintent_mc_luaset_all_numeric_str`
`\l__spintent_mc_args_seq`
`\l__spintent_mc_raw_args_seq`
`\l__spintent_spmcd_print_result_bool`
`\l__spintent_spmcm_print_result_bool`
`\l__spintent_spmcd_read_terse_bool`
`\l__spintent_spmcm_read_terse_bool`
`\l__spintent_spmcd_print_upper_bool`
`\l__spintent_spmcm_print_upper_bool`
`\l__spintent_spmcd_prefix_tl`
`\l__spintent_spmcm_prefix_tl`

Internal variables generated dynamically via a comma-separated list to avoid code duplication for both symmetric commands.

```
\str_new:N   \l__spintent_spmcm_spmcd_luaset_error_str
\clist_new:N \l__spintent_mc_luaset_types_clist
\str_new:N   \l__spintent_mc_luaset_all_numeric_str
\seq_new:N   \l__spintent_mc_args_seq
\seq_new:N   \l__spintent_mc_raw_args_seq
\seq_new:N   \l__spintent_mc_types_seq
\int_new:N   \l__spintent_mc_idx_int
\clist_map_inline:nn { mcd , mcm }
  {
    \tl_new:c   { l__spintent_sp #1 _luaset_ #1 _value_tl }
```

```
    \bool_new:c { l__spintent_sp #1 _read_terse_bool }
    \tl_new:c   { l__spintent_sp #1 _prefix_tl }
  }
```

(*End of definition for* `\l__spintent_spmcm_spmcd_luaset_error_str` *and others.*)

11.26.2 Definition of error messages

mc-invalid-arguments

Exception triggered when the comma-separated sequence contains negative integers, floats, or alphabetical characters.

```
\msg_new:nnnn { spintent } { mc-invalid-arguments }
  {
    Invalid~arguments~provided~to~#1.
  }
  {
    The~arguments~must~be~a~comma-separated~list. \\
  }
```

(*End of definition for* `mc-invalid-arguments`.)

11.26.3 Keys for \spmcd and \spmcm

result
prefix
read-cmd
upper-case
concept
silent-cmd
unknown

We define the shared ⟨*keys*⟩ `result`, `prefix`, `read-cmd`, `upper-case`, `concept` and `silent-cmd`.

```
\clist_map_inline:nn { mcd , mcm }
  {
    \keys_define:nn { spintent / sp #1 }
      {
        result             .bool_set:c = { l__spintent_sp #1 _print_result_bool },
        result             .initial:n  = false,
        result             .value_required:n = true,
        prefix             .code:n =
          { \__spintent_sanitize_affix:cn { l__spintent_sp #1 _prefix_tl } {##1} },
        prefix             .initial:n = ,
        prefix             .value_required:n = true,
        read-cmd           .choices:nn =
          { terse , clear }
          {
            \int_case:nn { \l_keys_choice_int }
              {
                {1} { \bool_set_true:c  { l__spintent_sp #1 _read_terse_bool } }
                {2} { \bool_set_false:c { l__spintent_sp #1 _read_terse_bool } }
              }
          },
        read-cmd           .initial:n = clear,
        read-cmd           .value_required:n = true,
        read-cmd / unknown .code:n =
          \msg_error:nneee { spintent } { unknown-choice }
            { read-cmd } { terse , ~clear } { \exp_not:n {##1} },
        upper-case         .bool_set:c = { l__spintent_sp #1 _print_upper_bool },
        upper-case         .initial:n  = false,
        upper-case         .value_required:n = true,
        concept            .bool_set:c = { l__spintent_sp #1 _concept_bool },
        concept            .initial:n  = true,
        concept            .value_required:n = true,
        silent-cmd         .bool_set:c = { l__spintent_sp #1 _silent_cmd_bool },
        silent-cmd         .initial:n  = false,
        silent-cmd         .value_required:n = true,
        unknown            .code:n =
          {
            \str_set:Nn \l__spintent_command_name_str { sp #1 }
            \__spintent_unknown_keys_cmd:n {##1}
          },
      }
  }
```

(*End of definition for* `result` *and others.*)

11.26.4 Internal functions for \spmcd and \spmcm

__spintent_mc_process_args:nn

The function `\__spintent_mc_process_args:nn` parses the operation '`#1`' and its raw argument list '`#2`'. Calls `\__spintent_luafun_mc_classify_args:n`, splits '`#2`' into `\l__spintent_mc_raw_args_seq` and `\l__spintent_mc_luaset_types_clist` into `\l__spintent_mc_types_seq`, then walks the former by position, fetching the matching type from the latter through `\seq_item:Nn`, to build `\l__spintent_mc_args_seq`:

each argument wrapped through \spnum when its type is number, verbatim otherwise. Injects MathML intents around the already-formatted \l__spintent_mc_args_seq.

```
\cs_new_protected:Nn \__spintent_mc_process_args:nn
  {
    \__spintent_luafun_mc_classify_args:n {#2}
    \seq_set_from_clist:Nn \l__spintent_mc_raw_args_seq {#2}
    \seq_set_from_clist:NN \l__spintent_mc_types_seq \l__spintent_mc_luaset_types_clist
    \seq_clear:N \l__spintent_mc_args_seq
    \int_zero:N \l__spintent_mc_idx_int
    \seq_map_inline:Nn \l__spintent_mc_raw_args_seq
      {
        \int_incr:N \l__spintent_mc_idx_int
        \str_if_eq:eeTF { \seq_item:Nn \l__spintent_mc_types_seq { \l__spintent_mc_idx_int } } { n
          { \seq_put_right:Nn \l__spintent_mc_args_seq { \spnum {##1} } }
          { \seq_put_right:Nn \l__spintent_mc_args_seq {##1} }
      }
```

Injects MathML intents around the already-formatted \l__spintent_mc_args_seq, exactly as before.

```
    \bool_if:cTF { l__spintent_sp #1 _silent_cmd_bool }
      {
        \seq_set_map:NNn \l__spintent_mc_args_seq \l__spintent_mc_args_seq
          { \MathMLintent { :silent } { ##1 } }
        \MathMLintent { _entre } { \c__spintent_invisible_sep_int }
        \MathMLintent { :silent } { ( }
        \seq_use:Nnnn \l__spintent_mc_args_seq
          { \MathMLintent { _y:silent } { , } }
          { \MathMLintent { :silent }   { , } }
          { \MathMLintent { _y:silent } { , } }
      }
      {
        \MathMLintent { _entre } { \c__spintent_invisible_sep_int }
        \MathMLintent { :silent } { ( }
        \seq_use:Nnnn \l__spintent_mc_args_seq
          { \MathMLintent { _y }      { , } }
          { \MathMLintent { :silent } { , } }
          { \MathMLintent { _y }      { , } }
      }
    \MathMLintent { :silent } { ) }
  }
```

(*End of definition for* __spintent_mc_process_args:nn.)

__spintent_spmc_process:nnnn

The internal function __spintent_spmc_process:nnnn is the shared handler for both \spmcd and \spmcm. Takes the short identifier in '#1', its full reading string in '#2', the local ⟨*keys*⟩ in '#3', and the optional arguments in '#4'.

The operator name (through \operatorname, per upper-case) is silenced with silent-cmd true — __spintent_mc_process_args:nn silences the arguments too, in that case — or with concept false alone, leaving the arguments untouched. With concept true, it carries the literal intent instead — #1 or #2 per read-cmd, prefixed with prefix when given.

With '#4' given, classifies it through __spintent_luafun_mc_classify_args:n. Only when every argument is a number does it call the Lua calculation (calculate_#1), reporting mc-invalid-arguments on failure, and, when result is true, printing the computed value. __spintent_mc_process_args:n prints the argument list itself either way.

```
\cs_new_protected:Nn \__spintent_spmc_process:nnnn
  {
    \keys_set:nn { spintent / sp #1 } { #3 }
    \bool_if:cTF { l__spintent_sp #1 _silent_cmd_bool }
      {
        \MathMLintent { :silent }
          {
            \operatorname
              {
                \bool_if:cTF { l__spintent_sp #1 _print_upper_bool }
                  { \text_uppercase:n {#1} } { #1 }
              }
          }
      }
      {
        \bool_if:cTF { l__spintent_sp #1 _concept_bool }
          {
            \MathMLintent
```

```
                {
                  \tl_if_empty:cF { l__spintent_sp #1 _prefix_tl }
                    {
                      \tl_use:c { l__spintent_sp #1 _prefix_tl } -
                    }
                  \bool_if:cTF { l__spintent_sp #1 _read_terse_bool } { #1 :nofix} { #2 :nofix}
                }
                {
                  \operatorname
                    {
                      \bool_if:cTF { l__spintent_sp #1 _print_upper_bool }
                        { \text_uppercase:n {#1} } { #1 }
                    }
                }
            }
            {
              \MathMLintent { :silent }
                {
                  \operatorname
                    {
                      \bool_if:cTF { l__spintent_sp #1 _print_upper_bool }
                        { \text_uppercase:n {#1} } { #1 }
                    }
                }
            }
        }
```

With '#4' given, classifies it through `\__spintent_luafun_mc_classify_args:n`. Only when every argument is a number does it call the Lua calculation (`calculate_#1`), reporting `mc-invalid-arguments` on failure — which should not happen, since every argument already passed classification — and, when `result` is `true`, printing the computed value. With any non-numeric argument, no calculation is attempted at all. Either way, `\__spintent_mc_process_args:n` prints the argument list itself.

```
    \tl_if_novalue:nF { #4 }
      {
        \__spintent_luafun_mc_classify_args:n {#4}
        \str_if_eq:VnT \l__spintent_mc_luaset_all_numeric_str { true }
          {
            \use:c { __spintent_luafun_calculate_ #1 :n } { #4 }
            \str_if_eq:VnT \l__spintent_spmcm_spmcd_luaset_error_str { true }
              {
                \msg_error:nne { spintent } { mc-invalid-arguments } { \exp_not:c { sp #1 } }
              }
          }
        \__spintent_mc_process_args:nn {#1} {#4}
        \str_if_eq:VnT \l__spintent_mc_luaset_all_numeric_str { true }
          {
            \bool_if:cT { l__spintent_sp #1 _print_result_bool }
              {
                \MathMLintent { es-igual-a } { = }
                \tl_use:c { l__spintent_sp #1 _luaset_ #1 _value_tl }
              }
          }
      }
  }
```

(*End of definition for* `\__spintent_spmc_process:nnnn`.)

`\spmcd` `\spmcm` The commands `\spmcd` and `\spmcm` act as the public interface wrappers for typesetting the greatest common divisor and least common multiple. They enforce math mode, open a local group, and defer execution to the main processing macro `\__spintent_spmc_process:nnnn`, passing along the appropriate reading strings and arguments.

```
\NewDocumentCommand \spmcd { O{} d() }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \__spintent_spmc_process:nnnn { mcd } { máximo-común-divisor } {#1} {#2}
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spmcd } }
  }
```

```
\NewDocumentCommand \spmcm { O{} d() }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \__spintent_spmc_process:nnnn { mcm } { mínimo-común-múltiplo } {#1} {#2}
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spmcm } }
  }
```

(*End of definition for* `\spmcd` *and* `\spmcm`*. These functions are documented on page 15.*)

11.27 The commands \spnM and \spnD

The user-level commands `\spnM` and `\spnD` provide a semantic interface for the Multiples of a number (*"múltiplos"* in Spanish) and the Divisors of a number (*"divisores"* in Spanish). They delegate argument validation and calculation to the Lua module `spintent.lua`, constructing a valid `MathCAT` intent.

11.27.1 Definition of variables for \spnM and \spnD

`\l__spintent_md_out_seq`
`\l__spintent_spnmd_luaset_is_number_str`
`\l__spintent_spnM_luaset_result_tl`
`\l__spintent_spnD_luaset_result_tl`
`\l__spintent_spnM_luaset_is_truncated_str`
`\l__spintent_spnD_luaset_is_truncated_str`
`\l__spintent_spnM_prefix_tl`
`\l__spintent_spnD_prefix_tl`
`\l__spintent_spnM_print_result_num_tl`
`\l__spintent_spnD_print_result_num_tl`

Internal variables generated dynamically for multiples and divisors.

```
\seq_new:N \l__spintent_md_out_seq
\str_new:N \l__spintent_spnmd_luaset_is_number_str
\clist_map_inline:nn { nM , nD }
  {
    \tl_new:c   { l__spintent_sp #1 _luaset_result_tl        }
    \str_new:c  { l__spintent_sp #1 _luaset_is_truncated_str }
    \tl_new:c   { l__spintent_sp #1 _prefix_tl               }
  }
```

(*End of definition for* `\l__spintent_md_out_seq` *and others.*)

11.27.2 Keys for \spnM and \spnD

`result`
`silent-cmd`
`result-num`
`prefix`
`concept`
`unknown`

We define the shared ⟨*keys*⟩ `result`, `silent-cmd`, `result-num`, `prefix` and `concept`.

```
\clist_map_inline:nn { nM , nD }
  {
    \keys_define:nn { spintent / sp #1 }
      {
        result     .bool_set:c = { l__spintent_sp #1 _print_result_bool },
        result     .initial:n  = false,
        result     .value_required:n = true,
        silent-cmd .bool_set:c = { l__spintent_sp #1 _silent_cmd_bool },
        silent-cmd .initial:n  = false,
        silent-cmd .value_required:n = true,
        result-num .tl_set:c   = { l__spintent_sp #1 _print_result_num_tl },
        result-num .initial:n  = ,
        prefix     .code:n =
          { \__spintent_sanitize_affix:cn { l__spintent_sp #1 _prefix_tl } {##1} },
        prefix     .initial:n = ,
        prefix     .value_required:n = true,
        concept    .bool_set:c = { l__spintent_sp #1 _concept_bool },
        concept    .initial:n  = true,
        concept    .value_required:n = true,
        unknown    .code:n =
          {
            \str_set:Nn \l__spintent_command_name_str { sp #1 }
            \__spintent_unknown_keys_cmd:n {##1}
          },
      }
  }
```

(*End of definition for* `result` *and others.*)

11.27.3 Definition of error messages for \spnM and \spnD

`md-invalid-argument`

Exception triggered when the argument for multiples or divisors is invalid.

```
\msg_new:nnnn { spintent } { md-invalid-argument }
  { Invalid~argument~provided~to~#1. }
  {
    The~argument~must~be~a~single~positive~integer. \\
  }
```

(*End of definition for* `md-invalid-argument`*.*)

11.27.4 Internal functions for \spnM and \spnD

__spintent_spnM_spnD_process:nnnnn

The internal function __spintent_spnM_spnD_process:nnnnn acts as the primary entry point for the multiples and divisors commands. It takes the command identifier '#1', its printed mathematical symbol '#2', its base reading string '#3', the local key-value options '#4', and the optional argument '#5'. Its sole purpose is to evaluate and set the provided keys within the current scope before delegating the core branching and typesetting logic to __spintent_spn_M_and_D_exec:nnnn.

```
\cs_new_protected:Nn \__spintent_spnM_spnD_process:nnnnn
  {
    \keys_set:nn { spintent / sp #1 } { #4 }
    \__spintent_spnM_spnD_exec:nnnn { #1 } { #2 } { #3 } { #5 }
  }
```

(*End of definition for* __spintent_spnM_spnD_process:nnnnn.)

__spintent_spnM_spnD_intent:nn

The internal function __spintent_spnM_spnD_intent:nn resolves the main reading intent string. It takes the command identifier '#1' (such as "nM" or "nD") and the base reading string '#2'. It automatically prepends the localized prefix if defined, normalizes accentuation for "múltiplos", and appends '-de'.

```
\cs_new:Nn \__spintent_spnM_spnD_intent:nn
  {
    \tl_if_empty:cF { l__spintent_sp #1 _prefix_tl }
      { \tl_use:c { l__spintent_sp #1 _prefix_tl } - }
    \str_case:nnF { #2 }
      { { multiplos } { múltiplos } }
      { #2 }
    -de
  }
```

(*End of definition for* __spintent_spnM_spnD_intent:nn.)

__spintent_spnM_spnD_silent:nn

The internal function __spintent_spn_M_and_D_silent:nn prints the atomized notation structure in full silent mode. It takes the mathematical symbol '#1' (such as "D" or "M") and the variable or explicit argument '#2'. Every component is wrapped in a silent intent, and an invisible separator is injected between the parentheses to prevent assistive technologies from applying default contextual reading rules like "de".

```
\cs_new_protected:Nn \__spintent_spnM_spnD_silent:nn
  {
    \MathMLintent { :silent } { \operatorname { #1 } }
    \MathMLintent { :silent } { ( }
    \MathMLintent { :silent } { \c__spintent_invisible_sep_int }
    \MathMLintent { :silent } { #2 }
    \MathMLintent { :silent } { ) }
  }
```

(*End of definition for* __spintent_spnM_spnD_silent:nn.)

__spintent_spnM_spnD_result:n

The internal function __spintent_spn_M_and_D_result:n handles the calculation and accessible formatting of the computed mathematical set. It takes the command identifier '#1'. It invokes the corresponding Lua backend calculation, maps the resulting clist into a sequence, and formats the final printed output utilizing custom accessibility intents for both separators and conjunctions. It also safely manages truncation indicators.

```
\cs_new_protected:Nn \__spintent_spnM_spnD_result:n
  {
    \use:c { __spintent_luafun_calcular_ #1 :nn }
      { \l__spintent_luaset_part_int_tl }
      { \tl_use:c { l__spintent_sp #1 _print_result_num_tl } }
    \MathMLintent { _son } { = \{ }
    \seq_set_from_clist:Ne
      \l__spintent_md_out_seq { \tl_use:c { l__spintent_sp #1 _luaset_result_tl } }
    \seq_use:Nnnn \l__spintent_md_out_seq
      { \MathMLintent { _y }      { , } }
      { \MathMLintent { :silent } { , } }
      { \MathMLintent { _y }      { , } }
    \str_if_eq:vnT
      { l__spintent_sp #1 _luaset_is_truncated_str } { true }
        {
          \MathMLintent { :silent } { , } ~ \dots
        }
    \MathMLintent { :silent } { \} }
  }
```

(*End of definition for* __spintent_spnM_spnD_result:n.)

__spintent_spnM_spnD_exec:nnnn

The function __spintent_spnM_spnD_exec:nnnn takes the command name in '#1', the symbol in '#2', the base reading string in '#3', and the optional argument in '#4'. The parentheses are always silenced. With '#4' absent, the placeholder n stands in for the argument. With '#4' given, __spintent_luafun_spnmd_classif decides whether it is wrapped through \spnum — and, when it is, whether __spintent_spnM_spnD_result:n also prints the computed set, per result — or left verbatim, with no result attempted.
Either way, silent-cmd silences the whole thing through __spintent_spnM_spnD_silent:nn; otherwise concept true carries the literal intent from __spintent_spnM_spnD_intent:nn, false silences only the operator symbol, leaving the argument to read normally.

```
\cs_new_protected:Nn \__spintent_spnM_spnD_exec:nnnn
  {
    \tl_if_novalue:nTF { #4 }
      {
        \bool_if:cTF { l__spintent_sp #1 _silent_cmd_bool }
          { \__spintent_spnM_spnD_silent:nn { #2 } { n } }
          {
            \bool_if:cTF { l__spintent_sp #1 _concept_bool }
              {
                \MathMLintent
                  { \__spintent_spnM_spnD_intent:nn { #1 } { #3 } }
                  { \operatorname { #2 } }
              }
              { \MathMLintent { :silent } { \operatorname { #2 } } }
            \MathMLintent { :silent } { ( }
            n
            \MathMLintent { :silent } { ) }
          }
      }
      {
        \__spintent_luafun_spnmd_classify:n { \exp_not:n { #4 } }
        \bool_if:cTF { l__spintent_sp #1 _silent_cmd_bool }
          {
            \str_if_eq:VnTF \l__spintent_spnmd_luaset_is_number_str { true }
              { \__spintent_spnM_spnD_silent:nn { #2 } { \spnum { #4 } } }
              { \__spintent_spnM_spnD_silent:nn { #2 } { #4 } }
          }
          {
            \bool_if:cTF { l__spintent_sp #1 _concept_bool }
              {
                \MathMLintent
                  { \__spintent_spnM_spnD_intent:nn { #1 } { #3 } }
                  { \operatorname { #2 } }
              }
              { \MathMLintent { :silent } { \operatorname { #2 } } }
            \MathMLintent { :silent } { ( }
            \str_if_eq:VnTF \l__spintent_spnmd_luaset_is_number_str { true }
              { \spnum { #4 } }
              { #4 }
            \MathMLintent { :silent } { ) }
          }
        \str_if_eq:VnT \l__spintent_spnmd_luaset_is_number_str { true }
          {
            \bool_if:cT { l__spintent_sp #1 _print_result_bool }
              { \__spintent_spnM_spnD_result:n { #1 } }
          }
      }
  }
```

(*End of definition for* __spintent_spnM_spnD_exec:nnnn.)

\spnM
\spnD

The commands \spnM and \spnD act as the public interface wrappers for typesetting multiples and divisors, respectively. They enforce math mode, open a local group, and defer execution to the function __spintent_spnM_spnD_process:nnnnn, passing along the optional ⟨*keys*⟩ '#1' and the numeric {⟨*argument*⟩} '#2'.

```
\NewDocumentCommand \spnM { O{} d() }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \__spintent_spnM_spnD_process:nnnnn { nM } { M } { multiplos } {#1} {#2}
        \group_end:
```

```
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spnM } }
  }
\NewDocumentCommand \spnD { O{} d() }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \__spintent_spnM_spnD_process:nnnnn { nD } { D } { divisores } {#1} {#2}
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spnD } }
  }
```

(*End of definition for* \spnM *and* \spnD*. These functions are documented on page 14.*)

11.28 The command \sprazon

The user command \sprazon typesets a ratio a:b as a fraction (\dfrac or \frac, per use-dfrac), read natively — no Lua parsing involved.

11.28.1 Definition of keys for \sprazon

silent-cmd
use-dfrac
unknown

We define the ⟨*keys*⟩ silent-cmd and use-dfrac.

```
\keys_define:nn { spintent / sprazon }
  {
    silent-cmd .bool_set:N = \l__spintent_sprazon_silent_cmd_bool,
    silent-cmd .initial:n  = false,
    silent-cmd .value_required:n = true,
    use-dfrac  .bool_set:N = \l__spintent_sprazon_use_dfrac_bool,
    use-dfrac  .initial:n  = false,
    use-dfrac  .value_required:n = true,
    unknown    .code:n =
      {
        \str_set:Nn \l__spintent_command_name_str { sprazon }
        \__spintent_unknown_keys_cmd:n {#1}
      },
  }
```

(*End of definition for* silent-cmd*,* use-dfrac*, and* unknown*.*)

__spintent_sprazon_exec:nnn

The function __spintent_sprazon_exec:nnn receives the optional ⟨*keys*⟩ in '#1', the numerator in '#2' and the denominator in '#3'. Under silent-cmd, the whole fraction is silenced as one unit. Otherwise, numerator and denominator are each tagged through a native \MathMLarg (num/ denom), and the outer \MathMLintent reads es-a:infix($num,$denom).

```
\cs_new_protected:Nn \__spintent_sprazon_exec:nnn
  {
    \keys_set:nn { spintent / sprazon } { #1 }
    \__spintent_invisible_sep:
    \bool_if:NTF \l__spintent_sprazon_silent_cmd_bool
      {
        \MathMLintent { _- }
          {
            \__spintent_inner_sep:
            \bool_if:NTF \l__spintent_sprazon_use_dfrac_bool
              { \dfrac { #2 } { #3 } }
              { \frac  { #2 } { #3 } }
          }
      }
      {
        \MathMLintent { es-a:infix($num,$denom) }
          {
            \__spintent_inner_sep:
            \bool_if:NTF \l__spintent_sprazon_use_dfrac_bool
              {
                \dfrac { \MathMLarg { num } { #2 } } { \MathMLarg { denom } { #3 } }
              }
              { \frac  { \MathMLarg { num } { #2 } } { \MathMLarg { denom } { #3 } } }
          }
      }
  }
```

(*End of definition for* __spintent_sprazon_exec:nnn*.*)

\sprazon The public command \sprazon enforces *math mode* and delegates execution to __spintent_sprazon_exec:nnn.

```
\NewDocumentCommand \sprazon { O{} m m }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \__spintent_sprazon_exec:nnn { #1 } { #2 } { #3 }
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \sprazon } }
  }
```

(*End of definition for* \sprazon. *This function is documented on page* ??.)

11.29 The command \spProp

The user command \spProp typesets a proportion a:b = c:d as two fractions joined by =, reusing __spintent_sprazon_exec:nnn directly (not the public \sprazon command) for each ratio.

\spProp The public command \spProp enforces *math mode*, and forwards the same ⟨*keys*⟩ in '#1' to both ratios: a/b ('#2'/'#3') and c/d ('#4'/'#5'), joined by a native \MathMLintent reading _como over the = sign.

```
\NewDocumentCommand \spProp { O{} m m m m }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \__spintent_sprazon_exec:nnn { #1 } { #2 } { #3 }
          \bool_if:NTF \l__spintent_sprazon_silent_cmd_bool
            {
              \MathMLintent { :silent } { = }
            }
            { \MathMLintent { _como } { = } }
          \__spintent_sprazon_exec:nnn { #1 } { #4 } { #5 }
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spProp } }
  }
```

(*End of definition for* \spProp. *This function is documented on page* ??.)

11.30 The command \spang

The command \spang is the main public command for typesetting and tagging sexagesimal angle expressions (degrees, minutes, and seconds). It parses the input using the Lua function __spintent_luafun_spang_parse:n. This separates the degrees, minutes, and seconds to construct a MathML intent tree with the correct postfixes for NVDA/MathCAT.

\l__spintent_spang_luaset_grado_str
\l__spintent_spang_luaset_minuto_str
\l__spintent_spang_luaset_segundo_str
\l__spintent_spang_luaset_error_str

These internal string variables store the values returned from the Lua module spintent.lua. They hold the extracted numerical values for degrees, minutes, and seconds, as well as the error status used during formatting.

```
\str_new:N  \l__spintent_spang_luaset_grado_str
\str_new:N  \l__spintent_spang_luaset_minuto_str
\str_new:N  \l__spintent_spang_luaset_segundo_str
\str_new:N  \l__spintent_spang_luaset_error_str
```

(*End of definition for* \l__spintent_spang_luaset_grado_str *and others.*)

11.30.1 Definition of error messages for \spang

invalid-angle-format The error message invalid-angle-format defines the text displayed when an invalid angle input is detected. It is triggered when an argument fails the format checks, does not match the expected sexagesimal pattern, or contains invalid measurement symbols.

```
\msg_new:nnnn { spintent } { invalid-angle-format }
  { Invalid~angle~format~in~#1. }
  {
    The~argument~'#2'~could~not~be~parsed. \\
  }
```

(*End of definition for* invalid-angle-format.)

11.30.2 Internal functions for \spang

__spintent_spang_process:n

The internal function __spintent_spang_process:n parses the angle input using __spintent_luafun_spang_parse:n after resetting the error flag in \l__spintent_spang_luaset_error_str. If an invalid format is detected, it issues the invalid-angle-format error. Otherwise, it checks the extracted string variables \l__spintent_spang_luaset_grado_str, \l__spintent_spang_luaset_minuto_str, and \l__spintent_spang_luaset_segundo_str. For each non-empty variable, it constructs the corresponding MathML intent structure with the correct measurement symbols and postfixes for NVDA/MathCAT.

```
\cs_new_protected:Nn \__spintent_spang_process:n
  {
    \str_set:Nn \l__spintent_spang_luaset_error_str { false }
    \__spintent_luafun_spang_parse:n { #1 }
    \str_if_eq:VnTF \l__spintent_spang_luaset_error_str { true }
      {
        \msg_error:nnnn { spintent } { invalid-angle-format } { \spang } { #1 }
      }
      {
        \str_if_empty:NF \l__spintent_spang_luaset_grado_str
          {
            \spunit{ \l__spintent_spang_luaset_grado_str °}
          }
        \str_if_empty:NF \l__spintent_spang_luaset_minuto_str
          {
            \MathMLintent { minutos : postfix($m) }
              {
                \MathMLarg { m }
                  {
                    \spunit{ \l__spintent_spang_luaset_minuto_str '}
                  }
              }
          }
        \str_if_empty:NF \l__spintent_spang_luaset_segundo_str
          {
            \MathMLintent { segundos : postfix($s) }
              {
                \MathMLarg { s }
                  {
                    \spunit { \l__spintent_spang_luaset_segundo_str "}
                  }
              }
          }
      }
  }
```

(*End of definition for* __spintent_spang_process:n.)

\spang

The command \spang is the public interface for formatting sexagesimal angle expressions. It first checks if it is being used in math mode, issuing an error if not. Inside a local group, it calls __spintent_spang_process:n to process the {⟨*argument*⟩}.

```
\NewDocumentCommand \spang { m }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \__spintent_spang_process:n { #1 }
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spang } }
  }
```

(*End of definition for* \spang. *This function is documented on page* 21.)

11.31 The command \spnZ

The user-level command \spnZ formats integers (the set $\mathbb{Z}$). It includes specific options to automatically enclose numbers (particularly negative ones) in visual parentheses and allows you to control whether these parentheses are read by MathCAT.

This command verifies that the {⟨*argument*⟩} is indeed an *integer*, checking for the complete absence of decimal separators or attached units of measurement.

11.31.1 Definition of keys for \spnZ

cifras-sep
read-sign
wrap-parens
read-parens

Now we define the ⟨*keys*⟩ cifras-sep and read-sign. These are .meta:nn ⟨*keys*⟩ passed directly to the \spnum command. We also introduce wrap-parens, which determines whether the number is enclosed in parentheses, and read-parens, which controls if these parentheses are read by MathCAT.

```
\keys_define:nn { spintent / spnZ }
  {
    cifras-sep  .meta:nn = { spintent / spnum } { cifras-sep = {#1} },
    read-sign   .meta:nn = { spintent / spnum } { read-sign  = {#1} },
    wrap-parens .bool_set:N = \l__spintent_spnz_wrap_parens_bool,
    wrap-parens .initial:n  = false,
    wrap-parens .value_required:n = true,
    read-parens .bool_set:N = \l__spintent_spnz_read_parens_bool,
    read-parens .initial:n  = true,
    read-parens .value_required:n = true,
    unknown     .code:n =
      {
        \str_set:Nn \l__spintent_command_name_str { spnZ }
        \__spintent_unknown_keys_cmd:n {#1}
      },
  }
```

(*End of definition for* cifras-sep *and others.*)

11.31.2 Definition of error messages

spnz-not-integer

Raised when the {⟨*argument*⟩} contains decimal separators or attached units, which conflict with the constraints of a pure integer representation.

```
\msg_new:nnnn { spintent } { spnz-not-integer }
  { The~value~'#1'~is~not~a~valid~integer. }
  {
    Command~\token_to_str:N \spnZ~only~accepts~integers.\\
   }
```

(*End of definition for* spnz-not-integer.)

11.31.3 Internal functions for \spnZ

__spintent_spnz_render_parens:

The function __spintent_spnz_render_parens: first checks the state of the boolean variable \l__spintent_spnz (controlled by the read-parens key). If it is true, it prints the number surrounded by parentheses. Otherwise, it applies the :silent *function intent* to the parentheses while printing the number.

```
\cs_new_protected:Nn \__spintent_spnz_render_parens:
  {
    \bool_if:NTF \l__spintent_spnz_read_parens_bool
      {
        ( \__spintent_spnum_print_num_start: )
      }
      {
        \MathMLintent { :silent } { ( }
        \__spintent_spnum_print_num_start:
        \MathMLintent { :silent } { ) }
      }
  }
```

(*End of definition for* __spintent_spnz_render_parens:.)

__spintent_spnz_process:nn

The function __spintent_spnz_process:nn first processes the ⟨*keys*⟩ passed in '#1' and then calls __spintent_luafun_clean_split_and_set:n on '#2'. It verifies that the input is a valid integer by checking if the variable \l__spintent_luaset_has_integer_str is true. If not true, an "error" is raised.

If it is true, it checks whether \l__spintent_luaset_part_int_tl is *empty* and raises an "error" if so. Otherwise, it checks \l__spintent_spnz_wrap_parens_bool set by wrap-parens and calls either the function __spintent_spnz_render_parens: or __spintent_spnum_print_num_start:.

```
\cs_new_protected:Nn \__spintent_spnz_process:nn
  {
    \keys_set:nn { spintent / spnZ } {#1}
    \__spintent_luafun_clean_split_and_set:n { \exp_not:n {#2} }
    \str_if_eq:VnTF \l__spintent_luaset_has_integer_str { true }
      {
        \tl_if_empty:NTF \l__spintent_luaset_part_int_tl
          { \msg_error:nnn { spintent } { spnum-no-digits } {#2} }
          {
            \bool_if:NTF \l__spintent_spnz_wrap_parens_bool
```

```
                { \__spintent_spnz_render_parens: }
                { \__spintent_spnum_print_num_start: }
            }
        }
        { \msg_error:nnn { spintent } { spnz-not-integer } {#2} }
    }
```

(*End of definition for* `\__spintent_spnz_process:nn`.)

`\spnZ` Defines the user command `\spnZ`, which is executed within a *group* and strictly enforces *math mode*.

```
\NewDocumentCommand \spnZ { O{} m }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \__spintent_spnz_process:nn { #1 } { #2 }
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spnZ } }
  }
```

(*End of definition for* `\spnZ`. *This function is documented on page 16.*)

11.32 The command `\spmQ`

The command `\spmQ` typesets *"mixed numbers"* within Spanish mathematical contexts (e.g., $3\frac{1}{2}$). It delegates the scaling of the integer part to the Lua module `spintent.lua` via `\__spintent_luafun_spmQ_scale_int:Vn`, which measures both the integer glyph and the fraction box at the node level and computes a scaling factor to align their heights visually. Finally, it constructs the correct `MathML` intent for `MathCAT`.

11.32.1 Definition of variables for `\spmQ`

`\l__spintent_spmQ_join_word_str`
`\l__spintent_spnQ_intent_read_sign_str`
`\l__spintent_spnQ_wrap_parens_l_tl`
`\l__spintent_spnQ_wrap_parens_r_tl`
`\l__spintent_saved_luamml_flag_int`

The variable `\l__spintent_spmQ_join_word_str` stores the value set by the key `join-word` which will be verbalized by `MathCAT` as a grammatical connector between the integer and fractional parts.

The variable `\l__spintent_spnQ_intent_read_sign_str` stores the `:intent` passed to `\MathMLintent` when the signs '+' or '−' are present in the integer part, used by the function `\__spintent_spmQ_print_scale_int:n`.

The variables `\l__spintent_spnQ_wrap_parens_l_tl` and `\l__spintent_spnQ_wrap_parens_r_tl` store the sized parentheses set by the function `\__spintent_spmQ_set_wrap_parens_size:` and used by the key `wrap-parens`.

The integer variable `\l__spintent_saved_luamml_flag_int` temporarily stores the current value of `\l__luamml_flag_int` before disabling it during the injection of the scaled integer node, and restores it afterwards. This prevents `luamml` from generating a spurious `<math>` element around the `sub_box` processed in isolation, before `\frac` has expanded.

```
\str_new:N \l__spintent_spmQ_join_word_str
\str_new:N \l__spintent_spnQ_intent_read_sign_str
\tl_new:N  \l__spintent_spnQ_wrap_parens_l_tl
\tl_new:N  \l__spintent_spnQ_wrap_parens_r_tl
\int_new:N \l__spintent_saved_luamml_flag_int
```

(*End of definition for* `\l__spintent_spmQ_join_word_str` *and others.*)

11.32.2 Definition of error messages for `\spmQ`

`spmQ-zero-denominator`
`spmQ-not-integer`

The error message `spmQ-zero-denominator` is raised when a zero denominator is passed to `\spmQ`. The second error `spmQ-not-integer` is returned when the integer part passed in {⟨*argument*⟩} is not actually an integer.

```
\msg_new:nnnn { spintent } { spmQ-zero-denominator }
  {
    Zero~denominator~in~\token_to_str:N \spmQ~\msg_line_context:.
  }
  {
    The~denominator~of~a~fraction~cannot~be~zero.\\
  }
\msg_new:nnnn { spintent } { spmQ-not-integer }
  {
    The~value~'#1'~is~not~a~valid~integer~\msg_line_context:.
  }
  {
    Command~\token_to_str:N \spmQ \c_space_tl only~accepts~integers.\\
  }
```

(*End of definition for* `spmQ-zero-denominator` *and* `spmQ-not-integer`.)

11.32.3 Definition of keys for \spmQ

join-word
use-spnZ
wrap-parens
read-parens
print-dfrac
unknown

Now we add the keys join-word, use-spnZ, wrap-parens, read-parens and print-dfrac.

```
\keys_define:nn { spintent / spmQ }
  {
    join-word         .choices:nn =
      { entero , enteros , y }
      {
        \int_case:nn { \l_keys_choice_int }
          {
            {1} { \str_set:Nn \l__spintent_spmQ_join_word_str { entero } }
            {2} { \str_set:Nn \l__spintent_spmQ_join_word_str { enteros } }
            {3} { \str_set:Nn \l__spintent_spmQ_join_word_str { y } }
          }
      },
    join-word         .initial:n  = y,
    join-word         .value_required:n = true,
    join-word/unknown .code:n =
      \msg_error:nneee { spintent } { unknown-choice }
        { join-word } { entero, ~enteros, ~y } { \exp_not:n {#1} },
    use-spnZ          .bool_set:N = \l__spintent_spmQ_use_spnZ_bool,
    use-spnZ          .initial:n = true,
    use-spnZ          .value_required:n = true,
    print-dfrac       .bool_set:N = \l__spintent_spmQ_print_dfrac_bool,
    print-dfrac       .initial:n = false,
    print-dfrac       .value_required:n = true,
    wrap-parens       .bool_set:N = \l__spintent_spnQ_wrap_parens_bool,
    wrap-parens       .initial:n  = false,
    wrap-parens       .value_required:n = true,
    read-parens       .bool_set:N = \l__spintent_spnQ_read_parens_bool,
    read-parens       .initial:n  = true,
    read-parens       .value_required:n = true,
    unknown           .code:n =
      {
        \str_set:Nn \l__spintent_command_name_str { spmQ }
        \__spintent_unknown_keys_cmd:n {#1}
      },
  }
```

(*End of definition for* join-word *and others.*)

11.32.4 Internal functions for \spmQ

__spintent_spmQ_set_wrap_parens_size:

The function __spintent_spmQ_set_wrap_parens_size: captures the current mathematical style using the primitive \mathstyle from LuaTEX and sets the variables \l__spintent_spnQ_wrap_parens_l_tl and \l__spintent_spnQ_wrap_parens_r_tl with the appropriate size for the parentheses. Display styles (0 and 1) receive \Bigl\lparen and \Bigr\rparen; text styles (2 and 3) receive \bigl\lparen and \bigr\rparen; any other style falls back to unsized \lparen and \rparen.

If the key print-dfrac is active, the size is always forced to \Bigl/\Bigr regardless of the current mathematical style, since \dfrac always produces a display-height fraction.

```
\cs_new_protected:Nn \__spintent_spmQ_set_wrap_parens_size:
  {
    \int_case:nnF { \mathstyle }
      {
        { 0 }
          {
            \tl_set:Nn \l__spintent_spnQ_wrap_parens_l_tl { \Bigl \lparen }
            \tl_set:Nn \l__spintent_spnQ_wrap_parens_r_tl { \Bigr \rparen }
          }
        { 1 }
          {
            \tl_set:Nn \l__spintent_spnQ_wrap_parens_l_tl { \Bigl \lparen }
            \tl_set:Nn \l__spintent_spnQ_wrap_parens_r_tl { \Bigr \rparen }
          }
        { 2 }
          {
            \tl_set:Nn \l__spintent_spnQ_wrap_parens_l_tl { \bigl \lparen }
            \tl_set:Nn \l__spintent_spnQ_wrap_parens_r_tl { \bigr \rparen }
          }
        { 3 }
          {
            \tl_set:Nn \l__spintent_spnQ_wrap_parens_l_tl { \bigl \lparen }
```

```
            \tl_set:Nn \l__spintent_spnQ_wrap_parens_r_tl { \bigr \rparen }
          }
      }
      {
        \tl_set:Nn \l__spintent_spnQ_wrap_parens_l_tl { \lparen }
        \tl_set:Nn \l__spintent_spnQ_wrap_parens_r_tl { \rparen }
      }
```

If the key `print-dfrac` is active, we must reset the variables `\l__spintent_spnQ_wrap_parens_l_tl` and `\l__spintent_spnQ_wrap_parens_r_tl` to `\Bigl`/`\Bigr` unconditionally.

```
    \bool_if:NT \l__spintent_spmQ_print_dfrac_bool
      {
        \tl_set:Nn \l__spintent_spnQ_wrap_parens_l_tl { \Bigl \lparen }
        \tl_set:Nn \l__spintent_spnQ_wrap_parens_r_tl { \Bigr \rparen }
      }
  }
```

(*End of definition for* `\__spintent_spmQ_set_wrap_parens_size:`.)

`\__spintent_spmQ_start_wrap_parens:`
`\__spintent_spmQ_stop_wrap_parens:`

The functions `\__spintent_spmQ_start_wrap_parens:` and `\__spintent_spmQ_stop_wrap_parens:` open and close the parentheses around the *"mixed number"* when the key `wrap-parens` is active. They first call `\__spintent_spmQ_set_wrap_parens_size:` to determine the correct size for the current mathematical style, then print the appropriate parenthesis stored in `\l__spintent_spnQ_wrap_parens_l_tl` or `\l__spintent_spnQ_wrap_parens_r_tl`.

If the variable `\l__spintent_spnQ_read_parens_bool`, controlled by the key `read-parens`, is *false*, each parenthesis is wrapped inside `\MathMLintent{:silent}` so that `NVDA`/`MathCAT` does not verbalize them.

```
\cs_new_protected:Nn \__spintent_spmQ_start_wrap_parens:
  {
    \bool_if:NT \l__spintent_spnQ_wrap_parens_bool
      {
        \__spintent_spmQ_set_wrap_parens_size:
        \bool_if:NTF \l__spintent_spnQ_read_parens_bool
          { \l__spintent_spnQ_wrap_parens_l_tl }
          { \MathMLintent { :silent } { \l__spintent_spnQ_wrap_parens_l_tl } }
      }
  }
\cs_new_protected:Nn \__spintent_spmQ_stop_wrap_parens:
  {
    \bool_if:NT \l__spintent_spnQ_wrap_parens_bool
      {
        \__spintent_spmQ_set_wrap_parens_size:
        \bool_if:NTF \l__spintent_spnQ_read_parens_bool
          { \l__spintent_spnQ_wrap_parens_r_tl }
          { \MathMLintent { :silent } { \l__spintent_spnQ_wrap_parens_r_tl } }
      }
  }
```

(*End of definition for* `\__spintent_spmQ_start_wrap_parens:` *and* `\__spintent_spmQ_stop_wrap_parens:`.)

`\__spintent_spmQ_set_join_word:`

The function `\__spintent_spmQ_set_join_word:` injects the grammatical connector between the integer part and the fraction by dispatching on the value stored in `\l__spintent_spmQ_join_word_str`, set by the key `join-word`. In all three cases the visual output is the Unicode invisible separator `U+2063` via `\c__spintent_invisible_sep_tl`, which produces a `<mi>` node in the `MathML` structure; what differs is the `:intent` text attached to it via `\MathMLintent`, which `NVDA`/`MathCAT` will verbalize as *«entero»*, *«enteros»* or *«y»* respectively.

```
\cs_new_protected:Nn \__spintent_spmQ_set_join_word:
  {
    \str_case:Vn \l__spintent_spmQ_join_word_str
      {
        { entero  } { \MathMLintent { _entero  } { \c__spintent_invisible_sep_int } }
        { enteros } { \MathMLintent { _enteros } { \c__spintent_invisible_sep_int } }
        { y       } { \MathMLintent { _y-      } { \c__spintent_invisible_sep_int } }
      }
  }
```

(*End of definition for* `\__spintent_spmQ_set_join_word:`.)

`\__spintent_spmQ_print_scale_int:`

The function `\__spintent_spmQ_print_scale_int:` prints the integer part of the *"mixed number"* scaled to match the visual height of the accompanying fraction. Before calling the Lua function `\__spintent_luafun_spmQ_sca` it temporarily disables `luamml` by saving the current value of `\l__luamml_flag_int` into `\l__spintent_saved_luamml_flag_int` and setting it to `0` (skip completely). This prevents `luamml` from generating a

spurious <math> wrapper around the sub_box node that holds the scaled integer, which is processed in isolation before \frac has expanded. Once the Lua function returns, the original flag value is restored via \l__spintent_saved_luamml_flag_int.

The Lua function receives the integer digits from \l__spintent_luaset_part_int_tl and the fraction command name (\frac or \dfrac, determined by the key print-dfrac) as its two arguments, and inserts the appropriately scaled sub_box node directly into the current horizontal list.

```
\cs_new_protected:Nn \__spintent_spmQ_print_scale_int:
  {
    \int_set_eq:Nc \l__spintent_saved_luamml_flag_int { l__luamml_flag_int }
    \int_set:Nn \l__luamml_flag_int { 0 }
    \bool_if:NTF \l__spintent_spmQ_print_dfrac_bool
      { \__spintent_luafun_spmQ_scale_int:Vn \l__spintent_luaset_part_int_tl { dfrac } }
      { \__spintent_luafun_spmQ_scale_int:Vn \l__spintent_luaset_part_int_tl { frac  } }
    \int_set_eq:cN { l__luamml_flag_int } \l__spintent_saved_luamml_flag_int
  }
```

(*End of definition for* __spintent_spmQ_print_scale_int:.)

__spintent_spmQ_print_scale_int:n

The function __spintent_spmQ_print_scale_int:n takes the integer part of the *"mixed number"* passed in '#1'. It first calls __spintent_luafun_clean_split_and_set:n on '#1' to parse and populate the shared Lua variables, then checks the status of \l__spintent_luaset_has_integer_str.

If the status is true, it inspects \l__spintent_luaset_sign_tl. When the sign is *empty*, it delegates directly to __spintent_spmQ_print_scale_int: to render the scaled integer. When a sign is present, it sets \l__spintent_spnQ_intent_read_sign_str to either _más:prefix($e) or _menos:prefix($e) and wraps the sign glyph together with a MathML argument ($e) containing the scaled integer inside a \MathMLintent, so that NVDA/MathCAT verbalizes the sign as a prefix before the integer value. An __spintent_invisible_sep: call between the sign and the MathMLarg forces the opening of a new <mrow> in the MathML structure, which is required for the intent tree to be well-formed.

If the status is false, the spmQ-not-integer "error" is raised immediately.

```
\cs_new_protected:Nn \__spintent_spmQ_print_scale_int:n
  {
    \__spintent_luafun_clean_split_and_set:n { \exp_not:n {#1} }
    \str_if_eq:VnTF \l__spintent_luaset_has_integer_str { true }
      {
        \tl_if_empty:NTF \l__spintent_luaset_sign_tl
          {
            \__spintent_spmQ_print_scale_int:
          }
          {
            \str_if_eq:VnTF \l__spintent_luaset_sign_tl { + }
              {
                \str_set:Nn
                  \l__spintent_spnQ_intent_read_sign_str { _más:prefix($e) }
              }
              {
                \str_set:Nn
                  \l__spintent_spnQ_intent_read_sign_str { _menos:prefix($e) }
              }
            \MathMLintent { \l__spintent_spnQ_intent_read_sign_str }
              {
                \l__spintent_luaset_sign_tl \c__spintent_invisible_sep_int % \__spintent_invisible_s
                \MathMLarg { e } { \__spintent_spmQ_print_scale_int: }
              }
          }
      }
      { \msg_error:nnn { spintent } { spmQ-not-integer } { #1 } }
  }
```

(*End of definition for* __spintent_spmQ_print_scale_int:n.)

__spintent_spmQ_print_spnZ_dfrac:nn

The function __spintent_spmQ_print_spnZ_dfrac:nn typesets the fractional part of the *"mixed number"*. It takes the numerator in '#1' and the denominator in '#2'. It first calls __spintent_luafun_clean_split_and_set: on '#1' to validate that the numerator is an integer; if \l__spintent_luaset_has_integer_str is false, the spmQ-not-integer "error" is raised immediately.

If the validation passes, the function dispatches on the two boolean variables \l__spintent_spmQ_use_-spnZ_bool and \l__spintent_spmQ_print_dfrac_bool, controlled respectively by the keys use-spnZ and print-dfrac. When use-spnZ is true, both the numerator and denominator are passed through \spnZ to gain MathML integer semantics; otherwise, the raw token lists '#1' and '#2' are used directly. In both

branches, the fraction command is selected dynamically via \use:c: if print-dfrac is true, dfrac is assembled; otherwise, frac is used.

```
\cs_new_protected:Nn \__spintent_spmQ_print_spnZ_dfrac:nn
  {
    \__spintent_luafun_clean_split_and_set:n { \exp_not:n {#1} }
    \str_if_eq:VnT \l__spintent_luaset_has_integer_str { false }
      {
        \msg_error:nnn { spintent } { spmQ-not-integer } { #1 }
      }
    \bool_if:NTF \l__spintent_spmQ_use_spnZ_bool
      {
        \use:c { \bool_if:NT \l__spintent_spmQ_print_dfrac_bool { d } frac }
          { \spnZ { #1 } } { \spnZ { #2 } }
      }
      {
        \use:c { \bool_if:NT \l__spintent_spmQ_print_dfrac_bool { d } frac }
          { #1 } { #2 }
      }
  }
```

(*End of definition for* __spintent_spmQ_print_spnZ_dfrac:nn.)

__spintent_spmQ_print:nnn

The function __spintent_spmQ_print:nnn is the main rendering orchestrator for \spmQ. It takes three arguments: the integer part '#1', the numerator '#2', and the denominator '#3'. It opens the optional parentheses via __spintent_spmQ_start_wrap_parens:, then calls __spintent_spmQ_print_scale_int:n to scale and render the integer part. Next, it injects the grammatical connector between the integer and the fraction via __spintent_spmQ_set_join_word:. Then it calls __spintent_spmQ_print_spnZ_dfrac:nn to typeset the fraction using the keys print-dfrac and use-spnZ. Finally, it closes the optional parentheses via __spintent_spmQ_stop_wrap_parens:.

```
\cs_new_protected:Nn \__spintent_spmQ_print:nnn
  {
    \__spintent_spmQ_start_wrap_parens:
    \__spintent_spmQ_print_scale_int:n { #1 }
    \__spintent_spmQ_set_join_word:
    \__spintent_spmQ_print_spnZ_dfrac:nn { #2 } { #3 }
    \__spintent_spmQ_stop_wrap_parens:
  }
```

(*End of definition for* __spintent_spmQ_print:nnn.)

__spintent_spmQ_exec:nnnn

The internal function __spintent_spmQ_exec:nnnn takes four arguments: the optional ⟨*keys*⟩ '#1', the integer '#2', the numerator '#3', and the denominator '#4'. First sets the optional ⟨*keys*⟩ from '#1', then processes the denominator input in '#4' using __spintent_luafun_clean_split_and_set:n to validate it is a non-zero integer.

It checks the status of \l__spintent_luaset_has_integer_str. If true and the integer part is exactly zero, it issues the spmQ-zero-denominator "error". Otherwise, whether the status is false or the integer part is *non-zero*, it calls __spintent_spmQ_print:nnn to format the *"mixed number"*.

```
\cs_new_protected:Nn \__spintent_spmQ_exec:nnnn
  {
    \keys_set:nn { spintent / spmQ } { #1 }
    \__spintent_luafun_clean_split_and_set:n { \exp_not:n { #4 } }
    \str_if_eq:VnTF \l__spintent_luaset_has_integer_str { true }
      {
        \int_compare:nNnTF { \l__spintent_luaset_part_int_tl } = { 0 }
          { \msg_error:nn { spintent } { spmQ-zero-denominator } }
          { \__spintent_spmQ_print:nnn { #2 } { #3 } { #4 } }
      }
      { \__spintent_spmQ_print:nnn { #2 } { #3 } { #4 } }
  }
```

(*End of definition for* __spintent_spmQ_exec:nnnn.)

\spmQ

The command \spmQ is the public interface for formatting *"mixed numbers"*. It first checks if it is being used in *math mode*, issuing an "error" if not. Inside a local group, it calls __spintent_spmQ_exec:nnnn to process the optional ⟨*keys*⟩ and the {⟨*arguments*⟩}.

```
\NewDocumentCommand \spmQ { O{} m m m }
  {
    \mode_if_math:TF
      {
        \group_begin:
```

```
            \__spintent_spmQ_exec:nnnn { #1 } { #2 } { #3 } { #4 }
          \group_end:
        }
        { \msg_error:nnn { spintent } { need-math-mode } { \spmQ } }
    }
```

(*End of definition for* \spmQ. *This function is documented on page 16.*)

11.33 The commands of the geo family

The commands described in this section provide a semantic interface for geometric objects in Spanish mathematical contexts. They all delegate parsing to spintent.lua, sharing the variables \l__spintent_geo_luaset_error_st \l__spintent_geo_luaset_intent_str, \l__spintent_geo_luaset_print_tl, \l__spintent_geo_luaset \l__spintent_geo_luaset_sub_str and \l__spintent_geo_luaset_tipo_str — constructing a valid MathCAT intent — and a single point is always annotated through __spintent_mathmlarg_point:nnn. Every command also activates \l__spintent_scope_attr_int within its own group, so the shared queue never claims content typeset outside a geo command in the same formula.

11.33.1 Definition of shared variables

\l__spintent_geo_luaset_error_str
\l__spintent_geo_luaset_intent_str
\l__spintent_geo_luaset_print_tl
\l__spintent_geo_luaset_is_name_str
\l__spintent_geo_luaset_poly_sym_str
\l__spintent_geo_luaset_angulo_case_str
\l__spintent_geo_luaset_concept_str
\l__spintent_geo_luaset_letra_word_str
\l__spintent_geo_luaset_sub_spoken_str
\l__spintent_geo_luaset_nombre_es_num_str
\l__spintent_geo_luaset_letra_str
\l__spintent_geo_luaset_sub_str
\l__spintent_geo_luaset_tipo_str

Internal variables shared across all commands of the family to avoid repeated calls into Lua.

```
\str_new:N    \l__spintent_geo_luaset_error_str
\str_new:N    \l__spintent_geo_luaset_intent_str
\tl_new:N     \l__spintent_geo_luaset_print_tl
\clist_new:N  \l__spintent_geo_luaset_points_clist
\str_new:N    \l__spintent_geo_luaset_is_name_str
\str_new:N    \l__spintent_geo_luaset_poly_sym_str
\str_new:N    \l__spintent_geo_luaset_angulo_case_str
\str_new:N    \l__spintent_geo_luaset_concept_str
\str_new:N    \l__spintent_geo_luaset_letra_word_str
\str_new:N    \l__spintent_geo_luaset_sub_spoken_str
\str_new:N    \l__spintent_geo_luaset_nombre_es_num_str
\str_new:N    \l__spintent_geo_luaset_letra_str
\str_new:N    \l__spintent_geo_luaset_sub_str
\str_new:N    \l__spintent_geo_luaset_tipo_str
```

(*End of definition for* \l__spintent_geo_luaset_error_str *and others.*)

11.34 The command \spcoord

The user command \spcoord typesets an ordered pair (x,y), with the comma (or semicolon) between them carrying its own :infix intent, and each side wrapped with :prefix($vp)/:prefix($vn) when it starts with +/-, so the sign is always read as más/menos — never positivo/negativo, which is not how MathCAT reads a bare sign in this position.

11.34.1 Definition of variables for \spcoord

\l__spintent_spcoord_luaset_error_str
\l__spintent_spcoord_luaset_sep_str
\l__spintent_spcoord_luaset_x_sign_str
\l__spintent_spcoord_luaset_x_type_str
\l__spintent_spcoord_luaset_x_value_str
\l__spintent_spcoord_luaset_x_has_frac_str
\l__spintent_spcoord_luaset_y_sign_str
\l__spintent_spcoord_luaset_y_type_str
\l__spintent_spcoord_luaset_y_value_str
\l__spintent_spcoord_luaset_y_has_frac_str

Set by __spintent_luafun_spcoord_parse_and_set:n. \l__spintent_spcoord_luaset_sep_str is , or ;, whichever the argument actually used. For each side, \l__spintent_spcoord_luaset_ #1 _sign_str is +, -, or empty; \l__spintent_spcoord_luaset_ #1 _type_str is number or raw; \l__spintent_spcoord_lu is the value with the sign already removed; \l__spintent_spcoord_luaset_ #1 _has_frac_str is none, frac, or dfrac.

```
\str_new:N \l__spintent_spcoord_luaset_error_str
\str_new:N \l__spintent_spcoord_luaset_sep_str
\str_new:N \l__spintent_spcoord_luaset_x_sign_str
\str_new:N \l__spintent_spcoord_luaset_x_type_str
\str_new:N \l__spintent_spcoord_luaset_x_value_str
\str_new:N \l__spintent_spcoord_luaset_x_has_frac_str % "none" / "frac" / "dfrac"
\str_new:N \l__spintent_spcoord_luaset_y_sign_str
\str_new:N \l__spintent_spcoord_luaset_y_type_str
\str_new:N \l__spintent_spcoord_luaset_y_value_str
\str_new:N \l__spintent_spcoord_luaset_y_has_frac_str % "none" / "frac" / "dfrac"
```

(*End of definition for* \l__spintent_spcoord_luaset_error_str *and others.*)

\l__spintent_spcoord_raw_arg_tl
\l__spintent_spcoord_intent_str
\l__spintent_spcoord_wrap_parens_l_tl
\l__spintent_spcoord_wrap_parens_r_tl

\l__spintent_spcoord_raw_arg_tl holds the argument exactly as given, for silent-cmd. \l__spintent_spcoor holds the :infix word for the separator (_coma:infix or _punto-y-coma:infix). \l__spintent_spcoord_wrap_ hold the opening/closing delimiter, sized by __spintent_spcoord_set_wrap_parens_size:.

```
\tl_new:N   \l__spintent_spcoord_raw_arg_tl
\str_new:N  \l__spintent_spcoord_intent_str % ,;:infix
\tl_new:N   \l__spintent_spcoord_wrap_parens_l_tl
\tl_new:N   \l__spintent_spcoord_wrap_parens_r_tl
```

(*End of definition for* \l__spintent_spcoord_raw_arg_tl *and others.*)

11.34.2 Definition of error messages for \spcoord

spcoord-invalid-arg

```
\msg_new:nnn { spintent } { spcoord-invalid-arg }
  {
    Invalid~argument~'#1'~for~\token_to_str:N \spcoord.
  }
```

(*End of definition for* spcoord-invalid-arg.)

11.34.3 Definition of keys for \spcoord

silent-cmd

We define the key silent-cmd.

```
\keys_define:nn { spintent / spcoord }
  {
    silent-cmd  .bool_set:N = \l__spintent_spcoord_silent_cmd_bool,
    silent-cmd  .initial:n  = false,
    silent-cmd  .value_required:n = true,
    unknown     .code:n =
      {
        \str_set:Nn \l__spintent_command_name_str { spcoord }
        \__spintent_unknown_keys_cmd:n {#1}
      },
  }
```

(*End of definition for* silent-cmd.)

11.34.4 Internal functions for \spcoord

__spintent_spcoord_set_wrap_parens_size:

With neither side carrying a fraction, the parentheses stay unsized. Otherwise, \mathstyle picks \Bigl/\Bigr for display, \bigl/\bigr for text — same mechanism as __spintent_spmQ_set_wrap_parens_size: When either side is dfrac, the parentheses are forced to \Bigl/\Bigr regardless of \mathstyle, since \dfrac always renders at display height — same criterion as print-dfrac in \spnQ.

```
\cs_new_protected:Nn \__spintent_spcoord_set_wrap_parens_size:
  {
    \tl_set:Nn \l__spintent_spcoord_wrap_parens_l_tl { \lparen }
    \tl_set:Nn \l__spintent_spcoord_wrap_parens_r_tl { \rparen }
    \bool_if:nT
      {
        \bool_lazy_or_p:nn
          { \bool_not_p:n { \str_if_eq_p:Vn \l__spintent_spcoord_luaset_x_has_frac_str { none } }
          { \bool_not_p:n { \str_if_eq_p:Vn \l__spintent_spcoord_luaset_y_has_frac_str { none } }
      }
      {
        \int_case:nnF { \mathstyle }
          {
            { 0 }
              {
                \tl_set:Nn \l__spintent_spcoord_wrap_parens_l_tl { \Bigl \lparen }
                \tl_set:Nn \l__spintent_spcoord_wrap_parens_r_tl { \Bigr \rparen }
              }
            { 1 }
              {
                \tl_set:Nn \l__spintent_spcoord_wrap_parens_l_tl { \Bigl \lparen }
                \tl_set:Nn \l__spintent_spcoord_wrap_parens_r_tl { \Bigr \rparen }
              }
            { 2 }
              {
                \tl_set:Nn \l__spintent_spcoord_wrap_parens_l_tl { \bigl \lparen }
                \tl_set:Nn \l__spintent_spcoord_wrap_parens_r_tl { \bigr \rparen }
              }
            { 3 }
              {
                \tl_set:Nn \l__spintent_spcoord_wrap_parens_l_tl { \bigl \lparen }
                \tl_set:Nn \l__spintent_spcoord_wrap_parens_r_tl { \bigr \rparen }
              }
          }
          { }
        \bool_if:nT
          {
            \bool_lazy_or_p:nn
              { \str_if_eq_p:Vn \l__spintent_spcoord_luaset_x_has_frac_str { dfrac } }
              { \str_if_eq_p:Vn \l__spintent_spcoord_luaset_y_has_frac_str { dfrac } }
          }
```

```
          {
            \tl_set:Nn \l__spintent_spcoord_wrap_parens_l_tl { \Bigl \lparen }
            \tl_set:Nn \l__spintent_spcoord_wrap_parens_r_tl { \Bigr \rparen }
          }
      }
  }
```

(*End of definition for* `\__spintent_spcoord_set_wrap_parens_size:`.)

`\__spintent_spcoord_value:n` With `\l__spintent_spcoord_luaset_ #1 _type_str` equal to `number`, calls `\spnum` on `\l__spintent_spcoord` Otherwise, typesets it as given — a `\frac`/`\dfrac` or a letter expression, untouched. Receives `x` or `y` in '`#1`'.

```
\cs_new_protected:Nn \__spintent_spcoord_value:n
  {
    \str_if_eq:eeTF { \str_use:c { l__spintent_spcoord_luaset_ #1 _type_str } } { number }
      { \exp_args:Ne \spnum { \str_use:c { l__spintent_spcoord_luaset_ #1 _value_str } } }
      { \str_use:c { l__spintent_spcoord_luaset_ #1 _value_str } }
  }
```

(*End of definition for* `\__spintent_spcoord_value:n`.)

`\__spintent_spcoord_print:n` With `\l__spintent_spcoord_luaset_ #1 _sign_str` equal to `+` or `-`, wraps the sign and `\__spintent_spcoord_` together with `_más:prefix($vp)`/`_menos:prefix($vn)`, referencing the value through read as `más`/`menos`, never `positivo`/`negativo`. With no sign, delegates directly to `\__spintent_spcoord_value:n`. Receives `x` or `y` in '`#1`'.

```
\cs_new_protected:Nn \__spintent_spcoord_print:n
  {
    \str_case:enF { \str_use:c { l__spintent_spcoord_luaset_ #1 _sign_str } }
      {
        { + }
          {
            \MathMLintent { _más:prefix ($vp) }
              {
                \str_use:c { l__spintent_spcoord_luaset_ #1 _sign_str }
                \MathMLarg{ vp } { \__spintent_spcoord_value:n { #1 } }
              }
          }
        { - }
          {
            \MathMLintent { _menos :prefix($vn) }
              {
                \str_use:c { l__spintent_spcoord_luaset_ #1 _sign_str }
                \MathMLarg{ vn } { \__spintent_spcoord_value:n { #1 } }
              }
          }
      }
      { \__spintent_spcoord_value:n { #1 } }
  }
```

(*End of definition for* `\__spintent_spcoord_print:n`.)

`\__spintent_spcoord_intent_infix:` Sets `\l__spintent_spcoord_intent_str` to `_coma:infix` or `_punto-y-coma:infix`, per `\l__spintent_spcoo` then wraps both sides — each referenced through with it, `$x`/`$y` standing for `\__spintent_spcoord_print:n`'s own output.

```
\cs_new_protected:Nn \__spintent_spcoord_intent_infix:
  {
    \str_if_eq:VnTF \l__spintent_spcoord_luaset_sep_str { ; }
      { \str_set:Nn \l__spintent_spcoord_intent_str { _punto-y-coma:infix } }
      { \str_set:Nn \l__spintent_spcoord_intent_str { _coma:infix } }
    \MathMLintent{ \l__spintent_spcoord_intent_str ($x, $y)}
      {
        \MathMLarg{x}{ \__spintent_spcoord_print:n { x } }
        \MathMLintent { :silent } { \l__spintent_spcoord_luaset_sep_str }
        \MathMLarg{y}{ \__spintent_spcoord_print:n { y } }
      }
  }
```

(*End of definition for* `\__spintent_spcoord_intent_infix:`.)

`\__spintent_spcoord_mathml:` With `silent-cmd true`, everything — parentheses and argument alike — is wrapped in a single `_-`, reading nothing at all: `\l__spintent_spcoord_raw_arg_tl` is typeset exactly as given, untouched by

__spintent_spcoord_print:n/__spintent_spcoord_intent_infix:. Otherwise, the parentheses are silenced individually and __spintent_spcoord_intent_infix: provides the reading in between.

```
\cs_new_protected:Nn \__spintent_spcoord_mathml:
  {
    \bool_if:NTF \l__spintent_spcoord_silent_cmd_bool
      {
        \MathMLintent { _- }
          {
            \l__spintent_spcoord_wrap_parens_l_tl
            \l__spintent_spcoord_raw_arg_tl
            \l__spintent_spcoord_wrap_parens_r_tl
          }
      }
      {
        \MathMLintent { :silent } { \l__spintent_spcoord_wrap_parens_l_tl }
          \__spintent_spcoord_intent_infix:
        \MathMLintent { :silent } { \l__spintent_spcoord_wrap_parens_r_tl }
      }
  }
```

(*End of definition for* __spintent_spcoord_mathml:.)

__spintent_spcoord_exec:nn

Sets the ⟨*keys*⟩ in '#1' and \l__spintent_spcoord_raw_arg_tl to '#2'. Calls __spintent_luafun_spcoord_par either way — even with silent-cmd true, since __spintent_spcoord_set_wrap_parens_size: still needs \l__spintent_spcoord_luaset_ #1 _has_frac_str — but with silent-cmd true, forces \l__spintent_spcoord_luaset_error_str back to false rather than reporting it: the raw argument is typeset regardless of whether it parsed. Otherwise, reports spcoord-invalid-arg on error. Either way, sizes the parentheses, then delegates to __spintent_spcoord_mathml: unless there was an error.

```
\cs_new_protected:Nn \__spintent_spcoord_exec:nn
  {
    \keys_set:nn { spintent / spcoord } {#1}
    \tl_set:Nn \l__spintent_spcoord_raw_arg_tl {#2}
    \bool_if:NTF \l__spintent_spcoord_silent_cmd_bool
      {
        \__spintent_luafun_spcoord_parse_and_set:n { \exp_not:n {#2} }
        \str_set:Nn \l__spintent_spcoord_luaset_error_str { false }
      }
      {
        \__spintent_luafun_spcoord_parse_and_set:n { \exp_not:n {#2} }
        \str_if_eq:VnT \l__spintent_spcoord_luaset_error_str { true }
          { \msg_error:nnn { spintent } { spcoord-invalid-arg } { \exp_not:n {#2} } }
      }
    \__spintent_spcoord_set_wrap_parens_size:
    \str_if_eq:VnF \l__spintent_spcoord_luaset_error_str { true }
      { \__spintent_spcoord_mathml: }
  }
```

(*End of definition for* __spintent_spcoord_exec:nn.)

\spcoord

The public command enforces *math mode*, opens a local group, and delegates to __spintent_spcoord_exec:nn.

```
\NewDocumentCommand \spcoord { O{} m }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \attribute \l__spintent_scope_attr_int = 1
          \__spintent_spcoord_exec:nn {#1} {#2}
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spcoord } }
  }
```

(*End of definition for* \spcoord. *This function is documented on page* ??.)

11.35 The function __spintent_mathmlarg_point:nnn

\l__spintent_mathmlarg_point_ref_tl
\l__spintent_mathmlarg_point_mode_str
\l__spintent_mathmlarg_point_kv_tl
\l__spintent_mathmlarg_point_silent_bool

Scratch variables for __spintent_mathmlarg_point:nnn: the reference name (#1), and the calling command's own read-cmd/silent-cmd state, looked up by name (#2).

```
\tl_new:N   \l__spintent_mathmlarg_point_ref_tl
\str_new:N  \l__spintent_mathmlarg_point_mode_str
\tl_new:N   \l__spintent_mathmlarg_point_kv_tl
\bool_new:N \l__spintent_mathmlarg_point_silent_bool
```

(*End of definition for* `\l__spintent_mathmlarg_point_ref_tl` *and others.*)

`\__spintent_mathmlarg_point:nnn`
`\__spintent_mathmlarg_point:nnV`

Annotates one point (`#3`) with `arg` and, when applicable, a literal `intent`, without wrapping it in its own `<mrow>`. Concept `intent` is passed empty on purpose: the caller applies it externally, via a native `\MathMLintent`.

With `\l__spintent_geo_luaset_tipo_str` equal to `sub`, for `read-cmd=school` the letter alone is tagged through `\__spintent_luafun_mrow_intent_arg:nn`, and the subscript is appended separately, silenced. For `read-cmd=mathcat`, letter and subscript are typeset together, tagged as a whole through a native `\MathMLarg`.

```
\cs_new_protected:Nn \__spintent_mathmlarg_point:nnn
  {
    \tl_set:Nn \l__spintent_mathmlarg_point_ref_tl {#1}
    \__spintent_luafun_geo_point_parse_and_set:n { \exp_not:n {#3} }
    \bool_set_eq:Nc \l__spintent_mathmlarg_point_silent_bool { l__spintent_ #2 _silent_cmd_bool }
    \str_set_eq:Nc  \l__spintent_mathmlarg_point_mode_str  { l__spintent_ #2 _mode_str }
    \__spintent_invisible_sep:
    \str_case:VnF \l__spintent_geo_luaset_tipo_str
      {
        { sub }
          {
            \str_if_eq:VnTF \l__spintent_mathmlarg_point_mode_str { school }
              {
                \tl_set:Ne \l__spintent_mathmlarg_point_kv_tl
                  { arg = "\l__spintent_mathmlarg_point_ref_tl", intent = "_ \l__spintent_geo_luas
                \__spintent_luafun_mrow_intent_arg:nn
                  { } { \l__spintent_mathmlarg_point_kv_tl }
                {
                  \l__spintent_geo_luaset_letra_str
                  \c_math_subscript_token
                    { \MathMLintent { :silent } { \l__spintent_geo_luaset_sub_str } }
                }
              }
              {
                {
                  \__spintent_inner_sep:
                  \MathMLarg { \l__spintent_mathmlarg_point_ref_tl }
                    {
                      {
                        \l__spintent_geo_luaset_letra_str
                        \c_math_subscript_token { \l__spintent_geo_luaset_sub_str }
                      }
                    }
                }
              }
          }
```

With `\l__spintent_geo_luaset_tipo_str` equal to `prima`, the same distinction applies: `read-cmd=school` tags the letter with a literal `intent` and appends the mark unsilenced (a prime is not spoken on the base, so nothing to repeat); `read-cmd=mathcat` tags letter and mark together through a native `\MathMLarg`.

```
        { prima }
          {
            \str_if_eq:VnTF \l__spintent_mathmlarg_point_mode_str { school }
              {
                \tl_set:Ne \l__spintent_mathmlarg_point_kv_tl
                  { arg = "\l__spintent_mathmlarg_point_ref_tl", intent = "_ \l__spintent_geo_luas
                \__spintent_luafun_mrow_intent_arg:nn
                  { } { \l__spintent_mathmlarg_point_kv_tl }
                { \l__spintent_geo_luaset_letra_str \l__spintent_geo_luaset_sub_str }
              }
              {
                {
                  \__spintent_inner_sep:
                  \MathMLarg { \l__spintent_mathmlarg_point_ref_tl }
                    {
                      { \l__spintent_geo_luaset_letra_str \l__spintent_geo_luaset_sub_str }
                    }
                }
              }
          }
      }
```

Otherwise (no subscript, no prime), the same distinction applies to the combined \l__spintent_geo_luaset_print_t as a whole — the shared queue for school (or when silent-cmd is true), a native \MathMLarg for mathcat, never mixed within the same formula, since the queue does not recognize a native \MathMLarg claim as already taken.

```
      {
        \bool_if:NTF \l__spintent_mathmlarg_point_silent_bool
          {
            \tl_set:Ne \l__spintent_mathmlarg_point_kv_tl
              { arg = "\l__spintent_mathmlarg_point_ref_tl", intent = "_-" }
            \__spintent_luafun_mrow_intent_arg:nn
              {  } % debe ir vacío
              { \l__spintent_mathmlarg_point_kv_tl }
            { \l__spintent_geo_luaset_print_tl }
          }
          {
            \str_if_eq:VnTF \l__spintent_mathmlarg_point_mode_str { school }
              {
                \tl_set:Ne \l__spintent_mathmlarg_point_kv_tl
                  { arg = "\l__spintent_mathmlarg_point_ref_tl", intent = "_ \l__spintent_geo_luas
                \__spintent_luafun_mrow_intent_arg:nn
                  {  } % debe ir vacío
                  { \l__spintent_mathmlarg_point_kv_tl }
                { \l__spintent_geo_luaset_print_tl }
              }
              {
                \MathMLarg { \l__spintent_mathmlarg_point_ref_tl }
                  { \l__spintent_geo_luaset_print_tl }
              }
          }
      }
  }
\cs_generate_variant:Nn \__spintent_mathmlarg_point:nnn { nnV }
```

(*End of definition for* __spintent_mathmlarg_point:nnn.)

11.36 The command \spPoint

The user-level command \spPoint provides a semantic interface for a single geometric point. It delegates argument validation to the Lua module spintent.lua, constructing a valid MathCAT intent.

11.36.1 Definition of variables for \spPoint

\l__spintent_spPoint_mode_str
\l__spintent_spPoint_prefix_tl
\l__spintent_spPoint_intent_arg_tl

Internal variables holding the values set by read-cmd and prefix.

```
\str_new:N  \l__spintent_spPoint_mode_str
\tl_new:N   \l__spintent_spPoint_prefix_tl
\tl_new:N   \l__spintent_spPoint_intent_arg_tl
```

(*End of definition for* \l__spintent_spPoint_mode_str, \l__spintent_spPoint_prefix_tl, *and* \l__spintent_spPoint_intent_arg_tl.)

11.36.2 Definition of error messages for \spPoint

spPoint-invalid-point

The "error" spPoint-invalid-point is raised when the argument passed in '#1' is not a single uppercase letter with an optional subscript or prime modifier.

```
\msg_new:nnnn { spintent } { spPoint-invalid-point }
  {
    Invalid~point~format~in~\token_to_str:N \spPoint:~'#1'.
  }
  {
    The~argument~must~be~a~single~uppercase~letter~with~an~optional~
    subscript~(\_{...})~or~prime~(').\\
  }
```

(*End of definition for* spPoint-invalid-point.)

11.36.3 Definition of keys for \spPoint

read-cmd
prefix
silent-cmd
unknown

We define the ⟨*keys*⟩ read-cmd, prefix, silent-cmd and concept.

```
\keys_define:nn { spintent / spPoint }
  {
    read-cmd            .choices:nn = { school , mathcat }
      {
        \int_case:nn { \l_keys_choice_int }
          {
            {1} { \str_set:Nn \l__spintent_spPoint_mode_str { school  } }
```

```
            {2} { \str_set:Nn \l__spintent_spPoint_mode_str { mathcat } }
          }
      },
    read-cmd            .initial:n  = school,
    read-cmd            .value_required:n = true,
    read-cmd / unknown  .code:n =
      \msg_error:nneee { spintent } { unknown-choice }
        { read-cmd } { school,~mathcat } { \exp_not:n {#1} },
    prefix              .code:n =
      { \__spintent_sanitize_affix:Nn \l__spintent_spPoint_prefix_tl {#1} },
    prefix              .initial:n  = ,
    prefix              .value_required:n = true,
    silent-cmd          .bool_set:N = \l__spintent_spPoint_silent_cmd_bool,
    silent-cmd          .initial:n = false,
    silent-cmd          .value_required:n = true,
    concept             .bool_set:N = \l__spintent_spPoint_concept_bool,
    concept             .initial:n  = true,
    concept             .value_required:n = true,
    unknown             .code:n =
      {
        \str_set:Nn \l__spintent_command_name_str { spPoint }
        \__spintent_unknown_keys_cmd:n {#1}
      },
  }
```

(*End of definition for* `read-cmd` *and others.*)

11.36.4 Internal functions for `\spPoint`

`\__spintent_spPoint_print:n`

The internal function `\__spintent_spPoint_print:n` processes the {⟨*argument*⟩} passed in '`#1`'. Calls `\__spintent_luafun_geo_point_parse_and_set:n`, reporting `spPoint-invalid-point` on error. If `silent-cmd` is `true` or `concept` is `false`, sets `\l__spintent_spPoint_intent_arg_tl` to `_($pt)` — forced further to `_-` when `silent-cmd` is `true` and `read-cmd` is not `school`, since `\__spintent_spPoint_mathml:n` then tags through a native `\MathMLarg` that needs the concept silenced too. Otherwise sets it to `punto:prefix($pt)`, prefixed with `prefix` when given. Finally calls `\__spintent_spPoint_mathml:n`.

```
\cs_new_protected:Nn \__spintent_spPoint_print:n
  {
    \__spintent_luafun_geo_point_parse_and_set:n { \exp_not:n {#1} }
    \str_if_eq:VnT \l__spintent_geo_luaset_error_str { true }
      {
        \msg_error:nne { spintent } { spPoint-invalid-point } { \exp_not:n {#1} }
      }
    \bool_if:nTF
      {
        \bool_lazy_or_p:nn
          { \l__spintent_spPoint_silent_cmd_bool }
          { ! \l__spintent_spPoint_concept_bool }
      }
      {
        \tl_set:Nn \l__spintent_spPoint_intent_arg_tl { _($pt) }
        \bool_if:NT \l__spintent_spPoint_silent_cmd_bool
          {
            \str_if_eq:VnF \l__spintent_spPoint_mode_str { school }
              {
                \tl_set:Nn \l__spintent_spPoint_intent_arg_tl { _- }
              }
          }
      }
      {
        \tl_set:Ne \l__spintent_spPoint_intent_arg_tl
          {
            \tl_if_empty:NF \l__spintent_spPoint_prefix_tl
              {
                \tl_use:N \l__spintent_spPoint_prefix_tl -
              }
            punto:prefix($pt)
          }
      }
    \__spintent_spPoint_mathml:n { #1 }
  }
```

(*End of definition for* `\__spintent_spPoint_print:n`.)

__spintent_spPoint_mathml:n

The function __spintent_spPoint_mathml:n calls __spintent_invisible_sep: first, then wraps __spintent_mathmlarg_point:nnn — which parses '#1', tags it arg="pt", and silences a real subscript or prime on its own — inside a native \MathMLintent carrying \l__spintent_spPoint_intent_arg_tl as the concept intent, with __spintent_inner_sep: first inside that group.

```
\cs_new_protected:Nn \__spintent_spPoint_mathml:n
  {
    \__spintent_invisible_sep:
    \MathMLintent { \l__spintent_spPoint_intent_arg_tl }
      {
        \__spintent_inner_sep:
        \__spintent_mathmlarg_point:nnn { pt } { spPoint } { #1 }
      }
  }
```

(*End of definition for* __spintent_spPoint_mathml:n.)

__spintent_spPoint_exec:nnn

The function __spintent_spPoint_exec:nnn receives the optional ⟨*keys*⟩ in '#1' and the point {⟨*argument*⟩} in '#2', delegating to __spintent_spPoint_print:n. With '#3' given (the optional coordinate), silences the gap between the point and the coordinate first (otherwise read as an implicit product), then temporarily aliases \l__spintent_spcoord_silent_cmd_bool to \l__spintent_spPoint_silent_cmd_bool — reverted when the calling group closes — and calls __spintent_luafun_spcoord_parse_and_set:n directly, reporting spcoord-invalid-arg on error unless silenced, then __spintent_spcoord_set_wrap_parens_s and __spintent_spcoord_mathml: — the same rendering functions \spcoord itself uses.

```
\cs_new_protected:Nn \__spintent_spPoint_exec:nnn
  {
    \keys_set:nn { spintent / spPoint } { #1 }
    \__spintent_spPoint_print:n { #2 }
    \tl_if_novalue:nF {#3}
      {
        \MathMLintent { :silent } { \c__spintent_invisible_sep_int } % silencia el "por"
        \bool_set_eq:NN \l__spintent_spcoord_silent_cmd_bool \l__spintent_spPoint_silent_cmd_bool
        \tl_set:Nn \l__spintent_spcoord_raw_arg_tl {#3}
        \bool_if:NTF \l__spintent_spPoint_coord_silent_cmd_bool
          {
            \__spintent_luafun_spcoord_parse_and_set:n { \exp_not:n {#3} }
            \str_set:Nn \l__spintent_spcoord_luaset_error_str { false }
          }
          {
            \__spintent_luafun_spcoord_parse_and_set:n { \exp_not:n {#3} }
            \str_if_eq:VnT \l__spintent_spcoord_luaset_error_str { true }
              { \msg_error:nnn { spintent } { spcoord-invalid-arg } { \exp_not:n {#3} } }
          }
        \__spintent_spcoord_set_wrap_parens_size:
        \str_if_eq:VnF \l__spintent_spcoord_luaset_error_str { true }
          { \__spintent_spcoord_mathml: }
      }
  }
```

(*End of definition for* __spintent_spPoint_exec:nnn.)

\spPoint

The public command \spPoint enforces *math mode*, opens a local group, and delegates execution to __spintent_spPoint_exec:nnn.

```
\NewDocumentCommand \spPoint { O{} m d() }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \attribute \l__spintent_scope_attr_int = 1
          \__spintent_spPoint_exec:nnn { #1 } { #2 } { #3 }
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spPoint } }
  }
```

(*End of definition for* \spPoint. *This function is documented on page 17.*)

11.37 The command \spvec

The user command \spvec supports two types of notation for the input {⟨*argument*⟩}: a pair of points (with \overrightarrow applied), or a one-letter name (with \vec applied instead). Both notations are automatically detected by the Lua module. An optional coordinate, in parentheses, may follow the {⟨*argument*⟩}.

11.37.1 Definition of variables for \spvec

\l__spintent_spvec_mode_str
\l__spintent_spvec_prefix_tl
\l__spintent_spvec_intent_arg_tl
\l__spintent_spvec_kv_tl
\l__spintent_spvec_pt_seq
\l__spintent_spvec_refs_tl
\l__spintent_spvec_pt_idx_int

The variable \l__spintent_spvec_mode_str stores the value set by the key read-cmd and the variable \l__spintent_spvec_prefix_tl stores the value set by the key prefix.

```
\str_new:N \l__spintent_spvec_mode_str
\tl_new:N  \l__spintent_spvec_prefix_tl
\tl_new:N  \l__spintent_spvec_intent_arg_tl
\tl_new:N  \l__spintent_spvec_kv_tl
\seq_new:N \l__spintent_spvec_pt_seq
\tl_new:N  \l__spintent_spvec_refs_tl
\int_new:N \l__spintent_spvec_pt_idx_int
```

(*End of definition for* \l__spintent_spvec_mode_str *and others.*)

11.37.2 Definition of error messages for \spvec

spvec-invalid-arg

The "error" spvec-invalid-arg is raised when the argument fails to parse, when a one-letter name carries a prime, or when the point list is not exactly two points.

```
\msg_new:nnn { spintent } { spvec-invalid-arg }
  {
    Invalid~argument~'#1'~for~\token_to_str:N \spvec.
  }
```

(*End of definition for* spvec-invalid-arg.)

11.37.3 Definition of keys for \spvec

silent-cmd
concept
autofit
read-cmd
prefix
unknown

We define the ⟨*keys*⟩ silent-cmd, concept, autofit, read-cmd and prefix.

```
\keys_define:nn { spintent / spvec }
  {
    silent-cmd    .bool_set:N = \l__spintent_spvec_silent_cmd_bool,
    silent-cmd    .initial:n = false,
    silent-cmd    .value_required:n = true,
    concept       .bool_set:N = \l__spintent_spvec_concept_bool,
    concept       .initial:n  = true,
    concept       .value_required:n = true,
    autofit       .bool_set:N = \l__spintent_spvec_autofit_bool,
    autofit       .initial:n  = false,
    autofit       .value_required:n = true,
    read-cmd      .choices:nn = { school , mathcat }
      {
        \int_case:nn { \l_keys_choice_int }
          {
            {1} { \str_set:Nn \l__spintent_spvec_mode_str { school } }
            {2} { \str_set:Nn \l__spintent_spvec_mode_str { mathcat } }
          }
      },
    read-cmd      .initial:n = school,
    read-cmd      .value_required:n = true,
    read-cmd / unknown .code:n =
      \msg_error:nneee { spintent } { unknown-choice }
        { read-cmd } { school,~mathcat } { \exp_not:n {##1} },
    prefix        .code:n =
      { \__spintent_sanitize_affix:Nn \l__spintent_spvec_prefix_tl {#1} },
    prefix        .initial:n = { },
    prefix        .value_required:n = true,
    unknown       .code:n =
      {
        \str_set:Nn \l__spintent_command_name_str { spvec }
        \__spintent_unknown_keys_cmd:n {#1}
      },
  }
```

(*End of definition for* silent-cmd *and others.*)

11.37.4 Internal functions for \spvec

__spintent_spvec_print:n

The internal function __spintent_spvec_print:n processes the {⟨*argument*⟩} passed in '#1'. Calls __spintent_luafun_geo_parse_and_set:n, reporting spvec-invalid-arg on error, on a proper name with \l__spintent_geo_luaset_tipo_str equal to prima (a one-letter vector cannot carry a prime), or on a point list that is not exactly two points. Otherwise calls __spintent_spvec_mathml:n.

```
\cs_new_protected:Nn \__spintent_spvec_print:n
  {
    \__spintent_luafun_geo_parse_and_set:n { \exp_not:n {#1} }
```

```
    \str_if_eq:VnT \l__spintent_geo_luaset_is_name_str { true }
      {
        \str_if_eq:VnT \l__spintent_geo_luaset_tipo_str { prima }
          { \str_set:Nn \l__spintent_geo_luaset_error_str { true } }
      }
    \str_if_eq:VnT \l__spintent_geo_luaset_is_name_str { false }
      {
        \int_compare:nT { \clist_count:N \l__spintent_geo_luaset_points_clist != 2 }
          { \str_set:Nn \l__spintent_geo_luaset_error_str { true } }
      }
    \str_if_eq:VnTF \l__spintent_geo_luaset_error_str { true }
      { \msg_error:nne { spintent } { spvec-invalid-arg } { \exp_not:n {#1} } }
      { \__spintent_spvec_mathml:n { #1 } }
  }
```

(*End of definition for* `\__spintent_spvec_print:n`.)

`\__spintent_spvec_mathml:n` The function `\__spintent_spvec_mathml:n` branches on `\l__spintent_geo_luaset_is_name_str`. For a proper name (a single letter, no prime), the concept `intent` is `_($pt)` when `silent-cmd` is `true` *or* `concept` is `false` — forced further to `_-` when `silent-cmd` is `true` and `read-cmd` is not `school`; otherwise `vector:prefix($pt)`. With `\l__spintent_geo_luaset_tipo_str` equal to `sub`, `read-cmd=school` tags `\vec{letra}` through `\__spintent_luafun_mrow_intent_arg:nn` with a literal `kv`, the subscript appended and silenced; `read-cmd=mathcat` tags `\vec{...}` through a native `\MathMLarg` instead, the subscript silenced only under `silent-cmd`. `autofit` inserts a `\vphantom{b}` inside `\vec{...}` to normalize its height. Otherwise (no subscript), the same `kv` logic wraps `\vec{print\_tl}` as a whole.

```
\cs_new_protected:Nn \__spintent_spvec_mathml:n
  {
    \str_if_eq:VnTF \l__spintent_geo_luaset_is_name_str { true }
      {
        \bool_if:nTF
          {
            \bool_lazy_or_p:nn
              { \l__spintent_spvec_silent_cmd_bool }
              { ! \l__spintent_spvec_concept_bool }
          }
          {
            \tl_set:Nn \l__spintent_spvec_intent_arg_tl { _($pt) }
            \bool_if:NT \l__spintent_spvec_silent_cmd_bool
              {
                \str_if_eq:VnF \l__spintent_spvec_mode_str { school }
                  { \tl_set:Nn \l__spintent_spvec_intent_arg_tl { _- } }
              }
          }
          {
            \tl_set:Nn \l__spintent_spvec_intent_arg_tl { vector:prefix($pt) }
          }
        \str_case:VnF \l__spintent_geo_luaset_tipo_str
          {
            { sub }
              {
                \str_if_eq:VnTF \l__spintent_spvec_mode_str { school }
                  {
                    \bool_if:NTF \l__spintent_spvec_silent_cmd_bool
                      { \tl_set:Nn \l__spintent_spvec_kv_tl { arg = "pt", intent = "_-" } }
                      {
                        \tl_set:Ne \l__spintent_spvec_kv_tl
                          { arg = "pt", intent = "_ \l__spintent_geo_luaset_intent_str -" }
                      }
                    \__spintent_invisible_sep:
                    \MathMLintent { \l__spintent_spvec_intent_arg_tl }
                      {
                        \__spintent_inner_sep:
                        \__spintent_luafun_mrow_intent_arg:nn
                          { } { \l__spintent_spvec_kv_tl }
                        {
                          \vec
                            {
                              \bool_if:NT \l__spintent_spvec_autofit_bool
                                { \vphantom{ b } }
                              \l__spintent_geo_luaset_letra_str
                            }
```

```
                          \c_math_subscript_token
                            { \MathMLintent { :silent } { \l__spintent_geo_luaset_sub_str } }
                        }
                      }
                  }
                  {
                    \__spintent_invisible_sep:
                    \MathMLintent { \l__spintent_spvec_intent_arg_tl }
                      {
                        \__spintent_inner_sep:
                        \vec
                          {
                            \bool_if:NT \l__spintent_spvec_autofit_bool
                              { \vphantom{ b } }
                            \MathMLarg{ pt } { \l__spintent_geo_luaset_letra_str }
                          }
                        \bool_if:NT \l__spintent_spvec_silent_cmd_bool
                          { \c_math_subscript_token { \l__spintent_geo_luaset_sub_str } }
                      }
                    \bool_if:NF \l__spintent_spvec_silent_cmd_bool
                      { \c_math_subscript_token { \l__spintent_geo_luaset_sub_str } }
                  }
              }
          }
          {
            \bool_if:NTF \l__spintent_spvec_silent_cmd_bool
              { \tl_set:Nn \l__spintent_spvec_kv_tl { arg = "pt", intent = "_-" } }
              {
                \tl_set:Ne \l__spintent_spvec_kv_tl
                  { arg = "pt", intent = "_ \l__spintent_geo_luaset_intent_str -" }
              }
            \__spintent_invisible_sep:
            \MathMLintent { \l__spintent_spvec_intent_arg_tl }
              {
                \__spintent_inner_sep:
                \__spintent_luafun_mrow_intent_arg:nn
                  { } { \l__spintent_spvec_kv_tl }
                {
                  \vec
                    {
                      \bool_if:NT \l__spintent_spvec_autofit_bool
                        { \vphantom{ b } }
                      \l__spintent_geo_luaset_print_tl
                    }
                }
              }
          }
      }
```

For a pair of points, `\l__spintent_geo_luaset_points_clist` is split into `\l__spintent_spvec_pt_seq` and walked to build `\l__spintent_spvec_refs_tl`, the `$pts1,$pts2` reference list. The concept `intent` is `_(refs)` or `vector:prefix(refs)`, same conditions as above, applied with a native `\MathMLintent` wrapping `\overrightarrow`. The sequence is walked again, calling `\__spintent_mathmlarg_point:nnn` once per point.

```
      {
        \seq_set_from_clist:NN \l__spintent_spvec_pt_seq \l__spintent_geo_luaset_points_clist
        \tl_clear:N \l__spintent_spvec_refs_tl
        \int_zero:N \l__spintent_spvec_pt_idx_int
        \seq_map_inline:Nn \l__spintent_spvec_pt_seq
          {
            \int_incr:N \l__spintent_spvec_pt_idx_int
            \tl_if_empty:NF \l__spintent_spvec_refs_tl
              { \tl_put_right:Nn \l__spintent_spvec_refs_tl { , } }
            \tl_put_right:Ne \l__spintent_spvec_refs_tl { $ pts \int_use:N \l__spintent_spvec_pt_
          }
        \bool_if:nTF
          {
            \bool_lazy_or_p:nn
              { \l__spintent_spvec_silent_cmd_bool }
              { ! \l__spintent_spvec_concept_bool }
          }
```

```
          {
            \tl_set:Ne \l__spintent_spvec_intent_arg_tl { _( \l__spintent_spvec_refs_tl ) }
          }
          {
            \tl_set:Ne \l__spintent_spvec_intent_arg_tl
              {
                \tl_if_empty:NF \l__spintent_spvec_prefix_tl
                  { \tl_use:N \l__spintent_spvec_prefix_tl - }
                vector:prefix( \l__spintent_spvec_refs_tl )
              }
          }
        \__spintent_invisible_sep:
        \MathMLintent { \l__spintent_spvec_intent_arg_tl }
          {
            \__spintent_inner_sep:
            \overrightarrow
              {
                \int_zero:N \l__spintent_spvec_pt_idx_int
                \seq_map_inline:Nn \l__spintent_spvec_pt_seq
                  {
                    \int_incr:N \l__spintent_spvec_pt_idx_int
                    \__spintent_mathmlarg_point:nnn
                      { pts \int_use:N \l__spintent_spvec_pt_idx_int }
                      { spvec }
                      { ##1 }
                  }
              }
          }
      }
  }
```

(*End of definition for* `\__spintent_spvec_mathml:n`.)

`\__spintent_spvec_exec:nnn`

The function `\__spintent_spvec_exec:nnn` receives the optional ⟨*keys*⟩ in '`#1`' and the {⟨*argument*⟩} in '`#2`', delegating to `\__spintent_spvec_print:n`. With '`#3`' given (the optional coordinate), aliases `\l__spintent_spcoord_silent_cmd_bool` to `\l__spintent_spvec_silent_cmd_bool` — reverted when the calling group closes — and calls `\__spintent_luafun_spcoord_parse_and_set:n` directly, reporting `spcoord-invalid-arg` on error unless silenced, then `\__spintent_spcoord_set_wrap_parens_size:` and `\__spintent_spcoord_mathml:` — the same rendering functions `\spcoord` itself uses.

```
\cs_new_protected:Nn \__spintent_spvec_exec:nnn
  {
    \keys_set:nn { spintent / spvec } { #1 }
    \__spintent_spvec_print:n { #2 }
    \tl_if_novalue:nF {#3}
      {
        \equal % notación de vector =( , )
        \MathMLintent { :silent } { \c__spintent_invisible_sep_int } % previene lectura de intervalo
        \bool_set_eq:NN \l__spintent_spcoord_silent_cmd_bool \l__spintent_spvec_silent_cmd_bool
        \tl_set:Nn \l__spintent_spcoord_raw_arg_tl {#3}
        \bool_if:NTF \l__spintent_spvec_silent_cmd_bool
          {
            \__spintent_luafun_spcoord_parse_and_set:n { \exp_not:n {#3} }
            \str_set:Nn \l__spintent_spcoord_luaset_error_str { false }
          }
          {
            \__spintent_luafun_spcoord_parse_and_set:n { \exp_not:n {#3} }
            \str_if_eq:VnT \l__spintent_spcoord_luaset_error_str { true }
              { \msg_error:nnn { spintent } { spcoord-invalid-arg } { \exp_not:n {#3} } }
          }
        \__spintent_spcoord_set_wrap_parens_size:
        \str_if_eq:VnF \l__spintent_spcoord_luaset_error_str { true }
          { \__spintent_spcoord_mathml: }
      }
  }
```

(*End of definition for* `\__spintent_spvec_exec:nnn`.)

`\spvec`

The public command `\spvec` enforces *math mode*, opens a local group, and delegates execution to `\__spintent_spvec_exec:nnn`.

```
\NewDocumentCommand \spvec { O{} m d() }
  {
```

```
    \mode_if_math:TF
      {
        \group_begin:
          \attribute \l__spintent_scope_attr_int = 1
          \__spintent_spvec_exec:nnn { #1 } { #2 } { #3 }
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spvec } }
  }
```

(*End of definition for* `\spvec`. *This function is documented on page* ??.)

11.38 The command \sprecta

The user command `\sprecta` supports two types of notation for the input {⟨*argument*⟩}: when the argument contains two points (separated by an optional comma), `\overleftrightarrow` is applied; when the argument is a single point (without comma) or a lowercase letter, it is formatted without any visual decorator. Both notations are automatically detected by the Lua module.

11.38.1 Definition of variables for \sprecta

`\l__spintent_sprecta_mode_str`
`\l__spintent_sprecta_prefix_tl`
`\l__spintent_sprecta_pt_seq`
`\l__spintent_sprecta_refs_tl`
`\l__spintent_sprecta_intent_arg_tl`
`\l__spintent_sprecta_kv_tl`
`\l__spintent_sprecta_pt_idx_int`

The variable `\l__spintent_sprecta_mode_str` store the values set by the key `read-cmd` and the variable `\l__spintent_sprayo_prefix_tl` store the value set by the key `prefix`.

```
\str_new:N \l__spintent_sprecta_mode_str
\tl_new:N  \l__spintent_sprecta_prefix_tl
\seq_new:N \l__spintent_sprecta_pt_seq
\tl_new:N  \l__spintent_sprecta_refs_tl
\tl_new:N  \l__spintent_sprecta_intent_arg_tl
\tl_new:N  \l__spintent_sprecta_kv_tl
\int_new:N \l__spintent_sprecta_pt_idx_int
```

(*End of definition for* `\l__spintent_sprecta_mode_str` *and others.*)

11.38.2 Definition of error messages for \sprecta

`sprecta-invalid-points`

The "error" `sprecta-invalid-points` is raised when the argument matches neither the two-point notation nor the name notation.

```
\msg_new:nnnn { spintent } { sprecta-invalid-points }
  {
    Invalid~format~in~\token_to_str:N \sprecta:~'#1'~\msg_line_context:.
  }
  {
    See~the~documentation~of~\token_to_str:N \sprecta~for~the~accepted~argument~forms.
  }
```

(*End of definition for* `sprecta-invalid-points`.)

11.38.3 Definition of keys for \sprecta

`read-cmd`
`prefix`
`silent-cmd`
`unknown`

We define the ⟨*keys*⟩ `read-cmd`, `prefix` and `silent-cmd`.

```
\keys_define:nn { spintent / sprecta }
  {
    read-cmd            .choices:nn = { school , mathcat }
      {
        \int_case:nn { \l_keys_choice_int }
          {
            {1} { \str_set:Nn \l__spintent_sprecta_mode_str { school  } }
            {2} { \str_set:Nn \l__spintent_sprecta_mode_str { mathcat } }
          }
      },
    read-cmd            .initial:n  = school,
    read-cmd            .value_required:n = true,
    read-cmd / unknown  .code:n =
      \msg_error:nneee { spintent } { unknown-choice }
        { read-cmd } { school,~mathcat } { \exp_not:n {#1} },
    prefix              .code:n =
      { \__spintent_sanitize_affix:Nn \l__spintent_sprecta_prefix_tl {#1} },
    prefix              .initial:n  = ,
    prefix              .value_required:n = true,
    silent-cmd          .bool_set:N = \l__spintent_sprecta_silent_cmd_bool,
    silent-cmd          .initial:n  = false,
    silent-cmd          .value_required:n = true,
    concept             .bool_set:N = \l__spintent_sprecta_concept_bool,
    concept             .initial:n  = true,
    concept             .value_required:n = true,
```

```
    unknown              .code:n =
      {
        \str_set:Nn \l__spintent_command_name_str { specta }
        \__spintent_unknown_keys_cmd:n {#1}
      },
  }
```

(*End of definition for* `read-cmd` *and others.*)

11.38.4 Internal functions for \specta

`\__spintent_specta_print:n`

The internal function `\__spintent_specta_print:n` processes the {⟨*argument*⟩} passed in '`#1`'. Calls `\__spintent_luafun_geo_parse_and_set:n`, reporting `specta-invalid-points` on error, then `\__spintent_`

```
\cs_new_protected:Nn \__spintent_specta_print:n
  {
    \__spintent_luafun_geo_parse_and_set:n { \exp_not:n {#1} }
    \str_if_eq:VnT \l__spintent_geo_luaset_error_str { true }
      {
        \msg_error:nne { spintent } { specta-invalid-points } { \exp_not:n {#1} }
      }
    \__spintent_specta_mathml:n { #1 }
  }
```

(*End of definition for* `\__spintent_specta_print:n`.)

`\__spintent_specta_mathml:n`

The function `\__spintent_specta_mathml:n` branches on `\l__spintent_geo_luaset_is_name_str`. For a proper name (`m`, `l`, ...), the concept `intent` is `:silent($pt)` when `silent-cmd` is `true` *or* `concept` is `false`; otherwise `recta:prefix($pt)`, prefixed with `prefix` when given.

With `\l__spintent_geo_luaset_tipo_str` equal to `sub`, `read-cmd=school` tags the base through `\__spintent_luafun_mrow_intent_arg:nn` with a literal `kv`, the subscript appended and silenced; `read-cmd=mathcat` tags base and subscript together through a native `\MathMLarg` instead.

```
\cs_new_protected:Nn \__spintent_specta_mathml:n
  {
    \str_if_eq:VnTF \l__spintent_geo_luaset_is_name_str { true }
      {
        \__spintent_invisible_sep:
        \bool_if:nTF
          {
            \bool_lazy_or_p:nn
              { \l__spintent_specta_silent_cmd_bool }
              { ! \l__spintent_specta_concept_bool }
          }
          {
            \tl_set:Nn \l__spintent_specta_intent_arg_tl { :silent($pt) }
          }
          {
            \tl_set:Ne \l__spintent_specta_intent_arg_tl
              {
                \tl_if_empty:NF \l__spintent_specta_prefix_tl
                  { \tl_use:N \l__spintent_specta_prefix_tl - }
                recta:prefix($pt)
              }
          }
        \str_case:VnF \l__spintent_geo_luaset_tipo_str
          {
            { sub }
              {
                \str_if_eq:VnTF \l__spintent_specta_mode_str { school }
                  {
                    \tl_set:Ne \l__spintent_specta_kv_tl
                      { arg = "pt", intent = "_ \l__spintent_geo_luaset_intent_str -" }
                    \MathMLintent { \l__spintent_specta_intent_arg_tl }
                      {
                        \__spintent_inner_sep:
                        \__spintent_luafun_mrow_intent_arg:nn
                          { } { \l__spintent_specta_kv_tl }
                        {
                          \l__spintent_geo_luaset_letra_str
                          \c_math_subscript_token
                            { \MathMLintent { :silent } { \l__spintent_geo_luaset_sub_str } }
                        }
                      }
```

```
                    }
```

For `read-cmd=mathcat`, the same sub case tags base and subscript together through a native `\MathMLarg` instead — never mixed with the shared queue, since the queue does not recognize a native `\MathMLarg` claim as already taken.

```
                    {
                      \MathMLintent { \l__spintent_specta_intent_arg_tl }
                        {
                          {
                            \__spintent_inner_sep:
                            \MathMLarg { pt }
                              {
                                {
                                  \l__spintent_geo_luaset_letra_str
                                  \c_math_subscript_token { \l__spintent_geo_luaset_sub_str }
                                }
                              }
                          }
                        }
                    }
                }
```

With `\l__spintent_geo_luaset_tipo_str` equal to `prima`, the same distinction applies: `read-cmd=school` tags the base with the literal `kv` and appends the mark, unsilenced (a prime is not spelled out on the base, so nothing to repeat); `read-cmd=mathcat` tags base and mark together through a native `\MathMLarg`.

```
            { prima }
              {
                \str_if_eq:VnTF \l__spintent_specta_mode_str { school }
                  {
                    \tl_set:Ne \l__spintent_specta_kv_tl
                      { arg = "pt", intent = "_ \l__spintent_geo_luaset_intent_str -" }
                    \MathMLintent { \l__spintent_specta_intent_arg_tl }
                      {
                        \__spintent_inner_sep:
                        \__spintent_luafun_mrow_intent_arg:nn
                          { } { \l__spintent_specta_kv_tl }
                        { \l__spintent_geo_luaset_letra_str \l__spintent_geo_luaset_sub_str }
                      }
                  }
                  {
                    \MathMLintent { \l__spintent_specta_intent_arg_tl }
                      {
                        {
                          \__spintent_inner_sep:
                          \MathMLarg { pt }
                            {
                              { \l__spintent_geo_luaset_letra_str \l__spintent_geo_luaset_sub_str
                            }
                          }
                      }
                  }
              }
          }
```

Otherwise (no subscript, no prime), the same kv logic wraps the combined `\l__spintent_geo_luaset_print_tl` as a whole — the shared queue for `school`, its own `arg` alone for `mathcat`.

```
          {
            \bool_if:NTF \l__spintent_specta_silent_cmd_bool
              {
                \tl_set:Nn \l__spintent_specta_kv_tl { arg = "pt", intent = "_-" }
              }
              {
                \str_if_eq:VnTF \l__spintent_specta_mode_str { school }
                  {
                    \tl_set:Ne \l__spintent_specta_kv_tl
                      { arg = "pt", intent = "_ \l__spintent_geo_luaset_intent_str -" }
                  }
                  {
                    \tl_set:Nn \l__spintent_specta_kv_tl { arg = "pt" }
                  }
              }
            \__spintent_luafun_mrow_intent_arg:nn
```

```
            { \l__spintent_specta_intent_arg_tl }
            { \l__spintent_specta_kv_tl }
          { \l__spintent_geo_luaset_print_tl }
        }
    }
```

For a list of points, \l__spintent_geo_luaset_points_clist is split into \l__spintent_specta_pt_seq and walked to build \l__spintent_specta_refs_tl, the $pt1,$pt2,... reference list. The concept intent is _(refs) or recta:prefix(refs), same conditions as above, applied with a native \MathMLintent wrapping \overleftrightarrow. The sequence is walked again, calling __spintent_mathmlarg_ once per point.

```
    {
      \seq_set_from_clist:NN \l__spintent_specta_pt_seq \l__spintent_geo_luaset_points_clist
      \tl_clear:N \l__spintent_specta_refs_tl
      \int_zero:N \l__spintent_specta_pt_idx_int
      \seq_map_inline:Nn \l__spintent_specta_pt_seq
        {
          \int_incr:N \l__spintent_specta_pt_idx_int
          \tl_if_empty:NF \l__spintent_specta_refs_tl
            { \tl_put_right:Nn \l__spintent_specta_refs_tl { , } }
          \tl_put_right:Ne \l__spintent_specta_refs_tl { $ pt \int_use:N \l__spintent_specta_
        }
      \bool_if:nTF
        {
          \bool_lazy_or_p:nn
            { \l__spintent_specta_silent_cmd_bool }
            { ! \l__spintent_specta_concept_bool }
        }
        {
          \tl_set:Ne \l__spintent_specta_intent_arg_tl { _( \l__spintent_specta_refs_tl ) }
        }
        {
          \tl_set:Ne \l__spintent_specta_intent_arg_tl
            {
              \tl_if_empty:NF \l__spintent_specta_prefix_tl
                { \tl_use:N \l__spintent_specta_prefix_tl - }
              recta:prefix( \l__spintent_specta_refs_tl )
            }
        }
      \__spintent_invisible_sep:
      \MathMLintent { \l__spintent_specta_intent_arg_tl }
        {
          \__spintent_inner_sep:
          \overleftrightarrow
            {
              \int_zero:N \l__spintent_specta_pt_idx_int
              \seq_map_inline:Nn \l__spintent_specta_pt_seq
                {
                  \int_incr:N \l__spintent_specta_pt_idx_int
                  \__spintent_mathmlarg_point:nnn
                    { pt \int_use:N \l__spintent_specta_pt_idx_int }
                    { specta }
                    { ##1 }
                }
            }
        }
    }
}
```

(*End of definition for* __spintent_specta_mathml:n.)

__spintent_specta_exec:nn

The function __spintent_specta_exec:nn receives the optional ⟨*keys*⟩ passed in '#1' and the {⟨*argument*⟩} passed in '#2' delegating the rendering of the latter to __spintent_specta_print:n.

```
\cs_new_protected:Nn \__spintent_specta_exec:nn
  {
    \keys_set:nn { spintent / specta } { #1 }
    \__spintent_specta_print:n { #2 }
  }
```

(*End of definition for* __spintent_specta_exec:nn.)

\sprecta The public command \sprecta enforces *math mode*, opens a local group, and delegates execution to __spintent_sprecta_exec:nn.

```
\NewDocumentCommand \sprecta { O{} m }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \attribute \l__spintent_scope_attr_int = 1
          \__spintent_sprecta_exec:nn { #1 } { #2 }
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \sprecta } }
  }
```

(*End of definition for* \sprecta. *This function is documented on page 18.*)

11.39 The command \sprayo

The user command \sprayo provides a semantic interface for a ray. The key read-sty selects the verbalized term, rayo or semirrecta, both rendered identically with \overrightarrow.

11.39.1 Definition of variables for \sprayo

\l__spintent_sprayo_mode_str
\l__spintent_sprayo_prefix_tl
\l__spintent_sprayo_read_sty_str

The variable \l__spintent_sprayo_mode_str store the values set by the key read-cmd; the variable \l__spintent_sprayo_prefix_tl store the value set by the key prefix and the variable \l__spintent_sprayo_r stores the value set by the key read-sty.

The variables defined here are assigned using .choices:nn or .code:n within \keys_define:nn.

```
\str_new:N \l__spintent_sprayo_mode_str
\tl_new:N  \l__spintent_sprayo_prefix_tl
\str_new:N \l__spintent_sprayo_read_sty_str
```

(*End of definition for* \l__spintent_sprayo_mode_str, \l__spintent_sprayo_prefix_tl, *and* \l__spintent_sprayo_read_sty_str.)

11.39.2 Definition of error messages for \sprayo

sprayo-invalid-points Raised when the argument is not a comma-separated pair of uppercase letters.

```
\msg_new:nnnn { spintent } { sprayo-invalid-points }
  {
    Invalid~point~format~in~\token_to_str:N \sprayo:~'#1'~\msg_line_context:.
  }
  {
    See~the~documentation~of~\token_to_str:N \sprayo~for~the~accepted~point~formats.
  }
```

(*End of definition for* sprayo-invalid-points.)

11.39.3 Definition of keys for \sprayo

read-cmd
prefix
silent-cmd
read-sty
unknown

We define the ⟨*keys*⟩ read-cmd, prefix, silent-cmd, concept and read-sty.

```
\keys_define:nn { spintent / sprayo }
  {
    read-cmd            .choices:nn = { school , mathcat }
      {
        \int_case:nn { \l_keys_choice_int }
          {
            {1} { \str_set:Nn \l__spintent_sprayo_mode_str { school  } }
            {2} { \str_set:Nn \l__spintent_sprayo_mode_str { mathcat } }
          }
      },
    read-cmd            .initial:n  = school,
    read-cmd            .value_required:n = true,
    read-cmd / unknown  .code:n =
      \msg_error:nneee { spintent } { unknown-choice }
        { read-cmd } { school,~mathcat } { \exp_not:n {#1} },
    prefix              .code:n =
      { \__spintent_sanitize_affix:Nn \l__spintent_sprayo_prefix_tl {#1} },
    prefix              .initial:n  = ,
    prefix              .value_required:n = true,
    silent-cmd          .bool_set:N = \l__spintent_sprayo_silent_cmd_bool,
    silent-cmd          .initial:n  = false,
    silent-cmd          .value_required:n = true,
    concept             .bool_set:N = \l__spintent_sprayo_concept_bool,
    concept             .initial:n  = true,
    concept             .value_required:n = true,
```

```
    read-sty            .choices:nn = { rayo , semirrecta }
      {
        \int_case:nn { \l_keys_choice_int }
          {
            {1} { \str_set:Nn \l__spintent_sprayo_read_sty_str { rayo       } }
            {2} { \str_set:Nn \l__spintent_sprayo_read_sty_str { semirrecta } }
          }
      },
    read-sty            .initial:n  = rayo,
    read-sty            .value_required:n = true,
    read-sty / unknown  .code:n =
      \msg_error:nneee { spintent } { unknown-choice }
        { read-sty } { rayo,~semirrecta } { \exp_not:n {#1} },
    unknown             .code:n =
      {
        \str_set:Nn \l__spintent_command_name_str { sprayo }
        \__spintent_unknown_keys_cmd:n {#1}
      },
  }
```

(*End of definition for* `read-cmd` *and others.*)

11.39.4 Internal functions for \sprayo

`\l__spintent_sprayo_pt_seq`
`\l__spintent_sprayo_refs_tl`
`\l__spintent_sprayo_intent_arg_tl`
`\l__spintent_sprayo_pt_idx_int`

Auxiliary variables used by `\__spintent_sprayo_mathml:` to split `\l__spintent_geo_luaset_points_clist` and build the `$pts1,$pts2,...` reference list.

```
\seq_new:N \l__spintent_sprayo_pt_seq
\tl_new:N  \l__spintent_sprayo_refs_tl
\tl_new:N  \l__spintent_sprayo_intent_arg_tl
\int_new:N \l__spintent_sprayo_pt_idx_int
```

(*End of definition for* `\l__spintent_sprayo_pt_seq` *and others.*)

`\__spintent_sprayo_print:n`

The internal function `\__spintent_sprayo_print:n` process the {⟨*argument*⟩} passed in '`#1`'. It calls `\__spintent_luafun_geo_parse_and_set:n` to process the {⟨*argument*⟩}, returning an "error" if the input is invalid, then calls `\__spintent_sprayo_mathml:`. Unlike `\sprecta`, `\sprayo` has no proper-name argument: it always takes a list of points, so no `intent` is built here.

```
\cs_new_protected:Nn \__spintent_sprayo_print:n
  {
    \__spintent_luafun_geo_parse_and_set:n { \exp_not:n {#1} }
    \str_if_eq:VnT \l__spintent_geo_luaset_error_str { true }
      {
        \msg_error:nne { spintent } { sprayo-invalid-points } { \exp_not:n {#1} }
      }
    \__spintent_sprayo_mathml:
  }
```

(*End of definition for* `\__spintent_sprayo_print:n`.)

`\__spintent_sprayo_mathml:`

The function `\__spintent_sprayo_mathml:` splits `\l__spintent_geo_luaset_points_clist` (set by Lua, one raw point per element) into `\l__spintent_sprayo_pt_seq` and walks it once to build `\l__spintent_sprayo_refs_tl`, the comma-separated `$pts1,$pts2,...` reference list, naming each point by position.

The concept `intent` is `_(refs)` when `silent-cmd` is `true` *or* `concept` is `false`; otherwise it is `\l__spintent_sprayo_read_sty_str:prefix(refs)` — the concept word itself (`rayo` or `semirrecta`) comes from the `read-sty` key, not a literal, since `\sprayo` verbalizes under two names — prefixed with `prefix` when given. It is applied with a native `\MathMLintent` wrapping `\overrightarrow`, preceded by `\__spintent_inner_sep:` (a second sibling so the sole `<mrow>` does not collapse into `<math>`). The sequence is walked a second time, calling `\__spintent_mathmlarg_point:nnn` once per point with its `pts`*n* reference name and `silent-cmd`/`read-cmd` looked up under `sprayo`: it alone decides `arg`/literal `intent` per point, so `concept` plays no role at that level.

```
\cs_new_protected:Nn \__spintent_sprayo_mathml:
  {
    \seq_set_from_clist:NN \l__spintent_sprayo_pt_seq \l__spintent_geo_luaset_points_clist
    \tl_clear:N \l__spintent_sprayo_refs_tl
    \int_zero:N \l__spintent_sprayo_pt_idx_int
    \seq_map_inline:Nn \l__spintent_sprayo_pt_seq
      {
        \int_incr:N \l__spintent_sprayo_pt_idx_int
        \tl_if_empty:NF \l__spintent_sprayo_refs_tl
          { \tl_put_right:Nn \l__spintent_sprayo_refs_tl { , } }
```

```
        \tl_put_right:Ne \l__spintent_sprayo_refs_tl { $ pts \int_use:N \l__spintent_sprayo_pt_idx
      }
    \bool_if:nTF
      {
        \bool_lazy_or_p:nn
          { \l__spintent_sprayo_silent_cmd_bool }
          { ! \l__spintent_sprayo_concept_bool }
      }
      {
        \tl_set:Ne \l__spintent_sprayo_intent_arg_tl { _( \l__spintent_sprayo_refs_tl ) }
      }
      {
        \tl_set:Ne \l__spintent_sprayo_intent_arg_tl
          {
            \tl_if_empty:NF \l__spintent_sprayo_prefix_tl
              { \tl_use:N \l__spintent_sprayo_prefix_tl - }
            \l__spintent_sprayo_read_sty_str :prefix( \l__spintent_sprayo_refs_tl )
          }
      }
    \__spintent_invisible_sep:
    \MathMLintent { \l__spintent_sprayo_intent_arg_tl }
      {
        \__spintent_inner_sep:
        \overrightarrow
          {
            \int_zero:N \l__spintent_sprayo_pt_idx_int
            \seq_map_inline:Nn \l__spintent_sprayo_pt_seq
              {
                \int_incr:N \l__spintent_sprayo_pt_idx_int
                \__spintent_mathmlarg_point:nnn
                  { pts \int_use:N \l__spintent_sprayo_pt_idx_int }
                  { sprayo }
                  { ##1 }
              }
          }
      }
  }
```

(*End of definition for* `\__spintent_sprayo_mathml:`.)

`\__spintent_sprayo_exec:nn` The function `\__spintent_sprayo_exec:nn` receives the optional ⟨*keys*⟩ passed in '`#1`' and the {⟨*argument*⟩} passed in '`#2`' delegating the rendering of the latter to `\__spintent_sprayo_print:n`.

```
\cs_new_protected:Nn \__spintent_sprayo_exec:nn
  {
    \keys_set:nn { spintent / sprayo } { #1 }
    \__spintent_sprayo_print:n { #2 }
  }
```

(*End of definition for* `\__spintent_sprayo_exec:nn`.)

`\sprayo` The public command `\sprayo` enforces *math mode*, opens a local group, and delegates execution to `\__spintent_sprayo_exec:nn`.

```
\NewDocumentCommand \sprayo { O{} m }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \attribute \l__spintent_scope_attr_int = 1
          \__spintent_sprayo_exec:nn { #1 } { #2 }
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \sprayo } }
  }
```

(*End of definition for* `\sprayo`. *This function is documented on page 18.*)

11.40 The commands `\spside` and `\spmside`

The user commands `\spside` and `\spmside` provide a semantic interface for a side, segment or trace determined by two points. The key `read-sty` selects the verbalized term — `lado`, `segmento`, or `trazo`, the latter used in straightedge-and-compass constructions — and the key `read-arg` overrides that term entirely with a free-form string. Additionally, the command `\spmside` provides the key `print-m`, controlling whether `\operatorname{m}` is typeset before the argument.

11.40.1 Definition of variables for \spside and \spmside

\l__spintent_spside_mode_str
\l__spintent_spmside_mode_str
\l__spintent_spside_prefix_tl
\l__spintent_spmside_prefix_tl
\l__spintent_spside_read_sty_str
\l__spintent_spmside_read_sty_str
\l__spintent_spside_read_arg_tl
\l__spintent_spmside_read_arg_tl

Each pair of variables is declared once for both \spside and \spmside via \clist_map_inline:nn. The variables \l__spintent_spside_mode_str and \l__spintent_spside_mode_str store the values set by the key read-cmd; the variables \l__spintent_spside_prefix_tl and \l__spintent_spmside_prefix_tl store the values set by the key prefix.

The variables \l__spintent_spside_read_sty_str and \l__spintent_spmside_read_sty_str stores the value set by the key read-sty.

The variables \l__spintent_spside_read_arg_tl and \l__spintent_spmside_read_arg_tl stores the value set by the key read-arg.

The variables are defined here as they are assigned using .choices:nn or .code:n rather than a direct property within \keys_define:nn.

```
\clist_map_inline:nn { side , mside }
  {
    \str_new:c { l__spintent_sp #1 _mode_str     }
    \tl_new:c  { l__spintent_sp #1 _prefix_tl    }
    \str_new:c { l__spintent_sp #1 _read_sty_str }
    \tl_new:c  { l__spintent_sp #1 _read_arg_tl  }
  }
```

(*End of definition for* \l__spintent_spside_mode_str *and others.*)

11.40.2 Definition of error messages for \spside and \spmside

spside-invalid-points
spmside-invalid-points

We define an error message for the commands \spside and \spmside. The "error" is triggered if the user includes a invalid *"point"* in the {⟨*argument*⟩} passed in '#1' to the command.

```
\clist_map_inline:nn { side , mside }
  {
    \msg_new:nnnn { spintent } { sp #1 -invalid-points }
      {
        Invalid~point~format~in~##1:~'##2'~\msg_line_context:.
      }
      {
        See~the~documentation~of~##1~for~the~accepted~point~formats.
      }
  }
```

(*End of definition for* spside-invalid-points *and* spmside-invalid-points.)

11.40.3 Definition of keys for \spside and \spmside

read-cmd
prefix
silent-cmd
concept
read-sty
read-arg
print-m
unknown

We define the shared ⟨*keys*⟩ read-cmd, prefix, silent-cmd, concept, read-sty, read-arg and print-m.

```
\clist_map_inline:nn { side , mside }
  {
    \keys_define:nn { spintent / sp #1 }
      {
        read-cmd            .choices:nn = { school , mathcat }
          {
            \int_case:nn { \l_keys_choice_int }
              {
                {1} { \str_set:cn { l__spintent_sp #1 _mode_str } { school  } }
                {2} { \str_set:cn { l__spintent_sp #1 _mode_str } { mathcat } }
              }
          },
        read-cmd            .initial:n  = school,
        read-cmd            .value_required:n = true,
        read-cmd / unknown  .code:n =
          \msg_error:nneee { spintent } { unknown-choice }
            { read-cmd } { school,~mathcat } { \exp_not:n {##1} },
        prefix              .code:n =
          { \__spintent_sanitize_affix:cn { l__spintent_sp #1 _prefix_tl } {##1} },
        prefix              .initial:n  = ,
        prefix              .value_required:n = true,
        silent-cmd          .bool_set:c = { l__spintent_sp #1 _silent_cmd_bool },
        silent-cmd          .initial:n  = false,
        silent-cmd          .value_required:n = true,
        concept             .bool_set:c = { l__spintent_sp #1 _concept_bool },
        concept             .initial:n  = true,
        concept             .value_required:n = true,
        read-sty            .choices:nn = { lado , segmento , trazo }
          {
            \int_case:nn { \l_keys_choice_int }
```

```
                {
                  {1} { \str_set:cn { l__spintent_sp #1 _read_sty_str } { lado      } }
                  {2} { \str_set:cn { l__spintent_sp #1 _read_sty_str } { segmento  } }
                  {3} { \str_set:cn { l__spintent_sp #1 _read_sty_str } { trazo     } }
                }
            },
          read-sty            .initial:n  = lado,
          read-sty            .value_required:n = true,
          read-sty / unknown  .code:n =
            \msg_error:nneee { spintent } { unknown-choice }
              { read-sty } { lado,~segmento,~trazo } { \exp_not:n {##1} },
          read-arg            .code:n =
            {
              \__spintent_sanitize_affix:cn { l__spintent_sp #1 _read_arg_tl } {##1}
            },
          read-arg            .initial:n  = ,
          read-arg            .value_required:n = true,
          print-m             .bool_set:c = { l__spintent_sp #1 _print_m_bool },
          print-m             .initial:n  = false,
          print-m             .value_required:n = true,
          unknown             .code:n =
            {
              \str_set:Nn \l__spintent_command_name_str { sp #1 }
              \__spintent_unknown_keys_cmd:n {##1}
            },
        }
    }
```

The keys `print-m` and `concept` are defined for both commands for simplicity, but has no effect on `\spside`.

(*End of definition for* `read-cmd` *and others.*)

`\l__spintent_spside_intent_tl`
`\l__spintent_spside_pt_seq`
`\l__spintent_spside_refs_tl`
`\l__spintent_spside_intent_arg_tl`
`\l__spintent_spside_pt_idx_int`

`\l__spintent_spside_intent_tl` holds the concept word (`read-arg` or `read-sty`). The rest are scratch variables shared by both commands (the function itself is shared) to split `\l__spintent_geo_luaset_points_clist` and build the `$pts1,$pts2,...` reference list.

```
\tl_new:N  \l__spintent_spside_intent_tl
\seq_new:N \l__spintent_spside_pt_seq
\tl_new:N  \l__spintent_spside_refs_tl
\tl_new:N  \l__spintent_spside_intent_arg_tl
\int_new:N \l__spintent_spside_pt_idx_int
```

(*End of definition for* `\l__spintent_spside_intent_tl` *and others.*)

`\__spintent_spside_process:nn`

The internal function `\__spintent_spside_process:nn` receives the short name of the command (`side` or `mside`) in '`#1`' and the {⟨*argument*⟩} in '`#2`'. It calls `\__spintent_luafun_geo_parse_and_set:n` to process '`#2`', returning an "error" if invalid, then builds `\l__spintent_spside_intent_tl`.
When `read-arg` is empty, it is set from `read-sty`, prefixed with '`medida-del-`' for `\spmside`. When `read-arg` is given, it is used as is, and '`medida-del-`' is prepended only if `concept` is `true`: with `concept=false`, `read-arg` is the verbatim escape hatch for a full user-written phrase (e.g. `read-arg={medida de la hipotenusa}`), gender and article left to the user. No `intent` is built here: it needs the reference list, which `\__spintent_spside_math` builds after splitting the points.

```
\cs_new_protected:Nn \__spintent_spside_process:nn
  {
    \__spintent_luafun_geo_parse_and_set:n { \exp_not:n {#2} }
    \str_if_eq:VnT \l__spintent_geo_luaset_error_str { true }
      {
        \msg_error:nnee { spintent } { sp #1 -invalid-points }
          { \c_backslash_str sp #1 } { \exp_not:n {#2} }
      }
    \tl_if_empty:cTF { l__spintent_sp #1 _read_arg_tl }
      {
        \tl_set:Ne \l__spintent_spside_intent_tl { \str_use:c { l__spintent_sp #1 _read_sty_str }
        \str_if_eq:eeT {#1} { mside }
          {
            \tl_put_left:Nn \l__spintent_spside_intent_tl { medida-del- }
          }
      }
      {
        \tl_set:Ne \l__spintent_spside_intent_tl { \tl_use:c { l__spintent_sp #1 _read_arg_tl } }
        \bool_if:cT { l__spintent_sp #1 _concept_bool }
          {
            \str_if_eq:eeT {#1} { mside }
```

```
                {
                  \tl_put_left:Nn \l__spintent_spside_intent_tl { medida-del- }
                }
            }
        }
      \__spintent_spside_mathml:n { #1 }
    }
```

(*End of definition for* `\__spintent_spside_process:nn`.)

`\__spintent_spside_mathml:n` The function `\__spintent_spside_mathml:n` receives the short command name in '`#1`'. It splits `\l__spintent_geo_luaset_points_clist` into `\l__spintent_spside_pt_seq` and walks it once to build `\l__spintent_spside_refs_tl`, the comma-separated `$pts1,$pts2,...` reference list, naming each point by position.

```
\cs_new_protected:Nn \__spintent_spside_mathml:n
  {
    \seq_set_from_clist:NN \l__spintent_spside_pt_seq \l__spintent_geo_luaset_points_clist
    \tl_clear:N \l__spintent_spside_refs_tl
    \int_zero:N \l__spintent_spside_pt_idx_int
    \seq_map_inline:Nn \l__spintent_spside_pt_seq
      {
        \int_incr:N \l__spintent_spside_pt_idx_int
        \tl_if_empty:NF \l__spintent_spside_refs_tl
          { \tl_put_right:Nn \l__spintent_spside_refs_tl { , } }
        \tl_put_right:Ne \l__spintent_spside_refs_tl { $ pts \int_use:N \l__spintent_spside_pt_id
      }
```

The concept `intent` is `_(refs)` when `silent-cmd` is `true`; otherwise, if `read-arg` is empty, it is `_(refs)` when `concept` is `false` or `\l__spintent_spside_intent_tl:prefix(refs)` when `true`; if `read-arg` is given, it is always `\l__spintent_spside_intent_tl:prefix(refs)` — `concept` never silences a user-given `read-arg` (see `\__spintent_spside_process:nn`). The `:prefix(...)` form is prefixed with `prefix` when given.

```
    \bool_if:cTF { l__spintent_sp #1 _silent_cmd_bool }
      {
        \tl_set:Ne \l__spintent_spside_intent_arg_tl { _( \l__spintent_spside_refs_tl ) }
      }
      {
        \tl_if_empty:cTF { l__spintent_sp #1 _read_arg_tl }
          {
            \bool_if:cTF { l__spintent_sp #1 _concept_bool }
              {
                \tl_set:Ne \l__spintent_spside_intent_arg_tl
                  {
                    \tl_if_empty:cF { l__spintent_sp #1 _prefix_tl }
                      { \tl_use:c { l__spintent_sp #1 _prefix_tl } - }
                    \l__spintent_spside_intent_tl :prefix( \l__spintent_spside_refs_tl )
                  }
              }
              {
                \tl_set:Ne \l__spintent_spside_intent_arg_tl { _( \l__spintent_spside_refs_tl ) }
              }
          }
          {
            \tl_set:Ne \l__spintent_spside_intent_arg_tl
              {
                \tl_if_empty:cF { l__spintent_sp #1 _prefix_tl }
                  { \tl_use:c { l__spintent_sp #1 _prefix_tl } - }
                \l__spintent_spside_intent_tl :prefix( \l__spintent_spside_refs_tl )
              }
          }
      }
```

Applied with a native `\MathMLintent`, preceded by `\__spintent_inner_sep:` (a second sibling so the sole `<mrow>` does not collapse into `<math>`. Inside, for `#1=side` the points are always wrapped in `\overline`; for `#1=mside`, when `print-m` is `true` a silent `\operatorname{m}` precedes the `\overline`-wrapped points; when `false`, the points are typeset bare, with no `\overline` at all — the recursive lookup in the shared `mathml_filter` callback finds each point either way, whether nested inside `\overline` or sitting directly in the `\MathMLintent` group. In every case the sequence is walked a second time, calling `\__spintent_mathmlarg_point:nnn` once per point with its `pts`*n* reference name and `silent-cmd`/`read-cmd` looked up under `sp#1`: it alone decides `arg`/literal `intent` per point, so `concept` plays no role at that level.

```
    \__spintent_invisible_sep: % (luamml issues #26)
```

```
    \MathMLintent { \l__spintent_spside_intent_arg_tl }
      {
        \__spintent_inner_sep:
        \str_if_eq:eeTF {#1} { side }
          {
            \overline { \__spintent_spside_points_loop:n {#1} }
          }
          {
            \bool_if:cTF { l__spintent_sp #1 _print_m_bool }
              {
                \__spintent_luafun_mrow_intent_arg:nn
                  { } { intent = ":silent" }
                { \operatorname { m } }
                \overline { \__spintent_spside_points_loop:n {#1} }
              }
              {
                \__spintent_spside_points_loop:n {#1}
              }
          }
      }
  }
```

(*End of definition for* `\__spintent_spside_mathml:n`.)

`\__spintent_spside_points_loop:n`

Walks `\l__spintent_spside_pt_seq` a second time, calling `\__spintent_mathmlarg_point:nnn` once per point. Factored out of `\__spintent_spside_mathml:n` since it is needed identically in three places there (`side`, `mside` with and without `print-m`); it takes the short command name in '`#1`' again since a separate macro does not inherit its caller's own `#1`.

```
\cs_new_protected:Nn \__spintent_spside_points_loop:n
  {
    \int_zero:N \l__spintent_spside_pt_idx_int
    \seq_map_inline:Nn \l__spintent_spside_pt_seq
      {
        \int_incr:N \l__spintent_spside_pt_idx_int
        \__spintent_mathmlarg_point:nnn
          { pts \int_use:N \l__spintent_spside_pt_idx_int }
          { sp #1 }
          { ##1 }
      }
  }
```

(*End of definition for* `\__spintent_spside_points_loop:n`.)

`\__spintent_spside_exec:nnn`

The function `\__spintent_spside_exec:nnn` receives the *short name* of the command passed in '`#1`', the optional ⟨*keys*⟩ passed in '`#2`' and the {⟨*argument*⟩} passed in '`#3`', delegating the rendering to `\__spintent_spside_process:nn`.

```
\cs_new_protected:Nn \__spintent_spside_exec:nnn
  {
    \keys_set:nn { spintent / sp #1 } { #2 }
    \__spintent_spside_process:nn { #1 } { #3 }
  }
```

(*End of definition for* `\__spintent_spside_exec:nnn`.)

`\spside` `\spmside`

Both public commands `\spside` and `\spmside` enforces *math mode*, opens a local group, and delegates execution to `\__spintent_spside_exec:nnn`.

```
\NewDocumentCommand \spside { O{} m }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \attribute \l__spintent_scope_attr_int = 1
          \__spintent_spside_exec:nnn { side } { #1 } { #2 }
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spside } }
  }
\NewDocumentCommand \spmside { O{} m }
  {
    \mode_if_math:TF
      {
```

```
      \group_begin:
        \attribute \l__spintent_scope_attr_int = 1
        \__spintent_spside_exec:nnn { mside } { #1 } { #2 }
      \group_end:
    }
    { \msg_error:nnn { spintent } { need-math-mode } { \spmside } }
}
```

(*End of definition for* `\spside` *and* `\spmside`*. These functions are documented on page 19.*)

11.41 The commands \spangle and \spmangle

The user commands `\spangle` and `\spmangle`, provide a semantic interface for angles, following the same generic pattern as `\spside`/`\spmside`. The key `print-m` chooses between the `\angle`/`\rightanglesqr` pair, typeset together with an explicit `\operatorname{m}`, and the `\measuredangle`/`\measuredrightangle` pair, which already conveys the notion of measure on its own; the key `recto` independently chooses the right-angle variant of whichever pair is active.

11.41.1 Definition of variables for \spangle and \spmangle

`\l__spintent_spangle_mode_str`
`\l__spintent_spmangle_mode_str`
`\l__spintent_spangle_prefix_tl`
`\l__spintent_spmangle_prefix_tl`

Each pair of variables is declared once for both `\spangle` and `\spmangle` via `\clist_map_inline:nn`. The variables `\l__spintent_spangle_mode_str` and `\l__spintent_spmangle_mode_str` store the values set by the key `read-cmd`; the variables `\l__spintent_spangle_prefix_tl` and `\l__spintent_spmangle_prefix` store the values set by the key `prefix`.

The variables defined here are assigned using `.choices:nn` or `.code:n` within `\keys_define:nn`.

```
\clist_map_inline:nn { angle , mangle }
  {
    \str_new:c { l__spintent_sp #1 _mode_str  }
    \tl_new:c  { l__spintent_sp #1 _prefix_tl }
  }
```

(*End of definition for* `\l__spintent_spangle_mode_str` *and others.*)

11.41.2 Definition of error messages for \spangle and \spmangle

`spangle-invalid-arg`
`spmangle-invalid-arg`

We define an "error" message for the commands `\spangle` and `\spmangle`. The "error" is triggered if the user includes a invalid {⟨*argument*⟩} passed in '`#1`' to the command.

```
\clist_map_inline:nn { angle , mangle }
  {
    \msg_new:nnnn { spintent } { sp #1 -invalid-arg }
      {
        Invalid~format~in~##1:~'##2'~\msg_line_context:.
      }
      {
        See~the~documentation~of~##1~for~the~accepted~argument~forms.
      }
  }
```

(*End of definition for* `spangle-invalid-arg` *and* `spmangle-invalid-arg`*.*)

11.41.3 Definition of keys for \spangle and \spmangle

`read-cmd`
`prefix`
`recto`
`print-m`
`silent-cmd`
`concept`
`unknown`

We define the shared ⟨*keys*⟩ `read-cmd`, `prefix`, `recto`, `print-m`, `silent-cmd` and `concept`,

```
\clist_map_inline:nn { angle , mangle }
  {
    \keys_define:nn { spintent / sp #1 }
      {
        read-cmd            .choices:nn = { school , mathcat }
          {
            \int_case:nn { \l_keys_choice_int }
              {
                {1} { \str_set:cn { l__spintent_sp #1 _mode_str } { school  } }
                {2} { \str_set:cn { l__spintent_sp #1 _mode_str } { mathcat } }
              }
          },
        read-cmd            .initial:n  = school,
        read-cmd            .value_required:n = true,
        read-cmd / unknown  .code:n =
          \msg_error:nneee { spintent } { unknown-choice }
            { read-cmd } { school,~mathcat } { \exp_not:n {##1} },
        prefix              .code:n =
          { \__spintent_sanitize_affix:cn { l__spintent_sp #1 _prefix_tl } {##1} },
        prefix              .initial:n  = ,
        prefix              .value_required:n = true,
```

```
        recto              .bool_set:c = { l__spintent_sp #1 _recto_bool },
        recto              .initial:n  = false,
        recto              .value_required:n = true,
        print-m            .bool_set:c = { l__spintent_sp #1 _print_m_bool },
        print-m            .initial:n  = false,
        print-m            .value_required:n = true,
        silent-cmd         .bool_set:c = { l__spintent_sp #1 _silent_cmd_bool },
        silent-cmd         .initial:n  = false,
        silent-cmd         .value_required:n = true,
        concept            .bool_set:c = { l__spintent_sp #1 _concept_bool },
        concept            .initial:n  = true,
        concept            .value_required:n = true,
        unknown            .code:n =
          {
            \str_set:Nn \l__spintent_command_name_str { sp #1 }
            \__spintent_unknown_keys_cmd:n {##1}
          },
      }
  }
```

The keys `print-m` and `concept` are defined for both commands for simplicity, but has no effect on `\spangle`.

(*End of definition for* `read-cmd` *and others.*)

11.41.4 Internal functions for `\spangle` and `\spmangle`

`\__spintent_spangle_sym:n`

The internal function `\__spintent_spangle_sym:n` dispatches on `#1`, `print-m` and `recto` to select one of four symbols: `\angle`/`\rightanglesqr` when `print-m` is `true`; `\measuredangle`/`\measuredrightangle` otherwise. When `#1` is `angle`, only `recto` applies, choosing between `\angle` and `\rightangle`.

```
\cs_new_protected:Nn \__spintent_spangle_sym:n
  {
    \str_if_eq:eeTF {#1} { mangle }
      {
        \bool_if:cTF { l__spintent_sp #1 _print_m_bool }
          {
            \bool_if:cTF { l__spintent_sp #1 _recto_bool }
              { \__spintent_right_angle_sqr: } { \angle }
          }
          {
            \bool_if:cTF { l__spintent_sp #1 _recto_bool }
              { \measuredrightangle } { \measuredangle }
          }
      }
      {
        \bool_if:cTF { l__spintent_sp #1 _recto_bool }
          { \__spintent_right_angle_sqr: } { \angle }
      }
  }
```

(*End of definition for* `\__spintent_spangle_sym:n`.)

`\l__spintent_spangle_intent_tl`
`\l__spintent_spangle_pt_seq`
`\l__spintent_spangle_refs_tl`
`\l__spintent_spangle_intent_arg_tl`
`\l__spintent_spangle_pt_idx_int`
`\l__spintent_spangle_print_m_now_bool`

`\l__spintent_spangle_intent_tl` holds the concept word (`ángulo` or `medida-del-ángulo`, '`-recto`'-suffixed when `recto` is `true`). The rest are scratch variables shared by both commands to split `\l__spintent_geo_luas` and build the reference list — used only in the `tres-puntos` and `un-punto` cases (see `\__spintent_spangle_mathml:`

```
\tl_new:N   \l__spintent_spangle_intent_tl
\seq_new:N  \l__spintent_spangle_pt_seq
\tl_new:N   \l__spintent_spangle_refs_tl
\tl_new:N   \l__spintent_spangle_intent_arg_tl
\int_new:N  \l__spintent_spangle_pt_idx_int
\str_new:N  \l__spintent_spangle_kv_tl
\bool_new:N \l__spintent_spangle_print_m_now_bool
```

(*End of definition for* `\l__spintent_spangle_intent_tl` *and others.*)

`\__spintent_spangle_process:nn`

The internal function `\__spintent_spangle_process:nn` receives the short name of the command (`angle` or `mangle`) in '`#1`' and the {⟨*argument*⟩} in '`#2`'. Calls `\__spintent_luafun_geo_angulo_parse_and_set:n`, reporting `sp#1-invalid-arg` on error. For a proper name that is a single letter (not numeric), also reports the same error when that letter is lowercase — angle names take an uppercase letter, number, or Greek letter, never a lowercase Latin one; the length check keeps a Greek letter command such as `\alpha` out of this comparison. Then sets `\l__spintent_spangle_intent_tl` to `ángulo` (or `medida-del-ángulo` for `\spmangle`), suffixed with `-recto` when `recto` is `true`, and calls `\__spintent_spangle_mathml:n`.

```
\cs_new_protected:Nn \__spintent_spangle_process:nn
  {
```

```
    \__spintent_luafun_geo_angulo_parse_and_set:n { \exp_not:n {#2} }
    \str_if_eq:VnT \l__spintent_geo_luaset_error_str { true }
      {
        \msg_error:nnee { spintent } { sp #1 -invalid-arg }
          { \c_backslash_str sp #1 } { \exp_not:n {#2} }
      }
    \str_if_eq:VnT \l__spintent_geo_luaset_angulo_case_str { nombre }
      {
        \str_if_eq:VnF \l__spintent_geo_luaset_nombre_es_num_str { true }
          {
            \int_compare:nT { \str_count:N \l__spintent_geo_luaset_letra_str = 1 }
              {
                \str_if_eq:eeF { \l__spintent_geo_luaset_letra_str } { \str_uppercase:n { \l__spi
                  {
                    \msg_error:nnee { spintent } { sp #1 -invalid-arg }
                      { \c_backslash_str sp #1 } { \exp_not:n {#2} }
                  }
              }
          }
      }
    \str_if_eq:eeTF {#1} { mangle }
      {
        \tl_set:Nn \l__spintent_spangle_intent_tl { medida-del-ángulo }
      }
      {
        \tl_set:Nn \l__spintent_spangle_intent_tl { ángulo }
      }
    \bool_if:cT { l__spintent_sp #1 _recto_bool }
      {
        \tl_put_right:Nn \l__spintent_spangle_intent_tl { -recto }
      }
    \__spintent_spangle_mathml:n {#1}
  }
```

(*End of definition for* `\__spintent_spangle_process:nn`.)

`\__spintent_spangle_mathml:n`

The function `\__spintent_spangle_mathml:n` branches on `\l__spintent_geo_luaset_angulo_case_str`. For `nombre`, the symbol (`\__spintent_spangle_sym:n`) is tagged through `\__spintent_luafun_mrow_intent_ar` on its own, with `\l__spintent_spangle_intent_tl:prefix` (no $-reference: `nombre` never refers to more than one thing) — `_-` when `silent-cmd` is `true` *or* `concept` is `false`. With `silent-cmd true`, the name that follows is silenced as a whole.

```
\cs_new_protected:Nn \__spintent_spangle_mathml:n
  {
    \__spintent_invisible_sep: % (luamml issues #26)
    \str_if_eq:VnTF \l__spintent_geo_luaset_angulo_case_str { nombre }
      {
        \bool_if:cTF { l__spintent_sp #1 _silent_cmd_bool }
          {
            \tl_set:Nn \l__spintent_spangle_intent_arg_tl { _- }
          }
          {
            \bool_if:cTF { l__spintent_sp #1 _concept_bool }
              {
                \tl_set:Ne \l__spintent_spangle_intent_arg_tl
                  {
                    \tl_if_empty:cF { l__spintent_sp #1 _prefix_tl }
                      { \tl_use:c { l__spintent_sp #1 _prefix_tl } - }
                    \l__spintent_spangle_intent_tl :prefix
                  }
              }
              {
                \tl_set:Nn \l__spintent_spangle_intent_arg_tl { _- }
              }
          }
        \tl_set:Ne \l__spintent_spangle_kv_tl
          { intent = "\l__spintent_spangle_intent_arg_tl" }
        \__spintent_luafun_mrow_intent_arg:nn
          { } { \l__spintent_spangle_kv_tl }
        { \__spintent_spangle_sym:n {#1} }
        \bool_if:cTF { l__spintent_sp #1 _silent_cmd_bool }
          {
```

```
            \__spintent_luafun_mrow_intent_arg:nn { } { intent = "_-" }
            { \l__spintent_geo_luaset_print_tl }
          }
```

Otherwise, a numeric name is tagged as a whole, through `\__spintent_luafun_mrow_intent_arg:nn`, with a literal `intent` built from `\l__spintent_geo_luaset_intent_str`.

```
          {
            \str_if_eq:VnTF \l__spintent_geo_luaset_nombre_es_num_str { true }
              {
                \tl_set:Ne \l__spintent_spangle_kv_tl
                  { intent = "_ \l__spintent_geo_luaset_intent_str -" }
                \__spintent_luafun_mrow_intent_arg:nn
                  { } { \l__spintent_spangle_kv_tl }
                { \l__spintent_geo_luaset_print_tl }
              }
```

With `\l__spintent_geo_luaset_tipo_str` equal to sub, `read-cmd=school` tags the letter through the same function, with a literal `intent`, and the subscript is appended and silenced; `read-cmd=mathcat` leaves both letter and subscript untouched, for `MathCAT` to read the whole `<msub>` natively (no `arg` is set in either case: there is no `$`-reference to resolve).

```
              {
                \str_case:VnF \l__spintent_geo_luaset_tipo_str
                  {
                    { sub }
                      {
                        \str_if_eq:eeTF { \use:c { l__spintent_sp #1 _mode_str } } { school }
                          {
                            \tl_set:Ne \l__spintent_spangle_kv_tl
                              { intent = "_ \l__spintent_geo_luaset_intent_str -" }
                            \__spintent_luafun_mrow_intent_arg:nn
                              { } { \l__spintent_spangle_kv_tl }
                            {
                              \l__spintent_geo_luaset_letra_str
                              \c_math_subscript_token
                                { \MathMLintent { :silent } { \l__spintent_geo_luaset_sub_str } }
                            }
                          }
                          {
                            \l__spintent_geo_luaset_letra_str
                            \c_math_subscript_token { \l__spintent_geo_luaset_sub_str }
                          }
                      }
```

With `\l__spintent_geo_luaset_tipo_str` equal to `prima`, the prime is always read — spoken as part of the literal for `read-cmd=school`, native for `read-cmd=mathcat` — since a prime is not a repeated word to silence.

```
                    { prima }
                      {
                        \str_if_eq:eeTF { \use:c { l__spintent_sp #1 _mode_str } } { school }
                          {
                            \tl_set:Ne \l__spintent_spangle_kv_tl
                              { intent = "_ \l__spintent_geo_luaset_intent_str -" }
                            \__spintent_luafun_mrow_intent_arg:nn
                              { } { \l__spintent_spangle_kv_tl }
                            { \l__spintent_geo_luaset_letra_str \l__spintent_geo_luaset_sub_str }
                          }
                          {
                            \l__spintent_geo_luaset_letra_str \l__spintent_geo_luaset_sub_str
                          }
                      }
                  }
                  { \l__spintent_geo_luaset_print_tl }
              }
          }
      }
```

For `tres-puntos`/`un-punto`, `\l__spintent_geo_luaset_points_clist` is split into `\l__spintent_spangle_pt` and walked to build `\l__spintent_spangle_refs_tl`, the `$pt1,$pt2,...` reference list. A single native `\MathMLintent` wraps the symbol and the points together, with `\l__spintent_spangle_intent_tl:prefix(refs` — `_-` when `silent-cmd` is `true`, `_(refs)` when `concept` is `false` alone. For `\spmangle` with `print-m`

true, a silent \operatorname{m} precedes __spintent_spangle_sym:n inside the same \MathMLintent. The sequence is walked again, calling __spintent_mathmlarg_point:nnn once per point.

```
      {
        \seq_set_from_clist:NN \l__spintent_spangle_pt_seq \l__spintent_geo_luaset_points_clist
        \tl_clear:N \l__spintent_spangle_refs_tl
        \int_zero:N \l__spintent_spangle_pt_idx_int
        \seq_map_inline:Nn \l__spintent_spangle_pt_seq
          {
            \int_incr:N \l__spintent_spangle_pt_idx_int
            \tl_if_empty:NF \l__spintent_spangle_refs_tl
              { \tl_put_right:Nn \l__spintent_spangle_refs_tl { , } }
            \tl_put_right:Ne \l__spintent_spangle_refs_tl { $ pt \int_use:N \l__spintent_spangle_
          }
        \bool_if:cTF { l__spintent_sp #1 _silent_cmd_bool }
          {
            \tl_set:Nn \l__spintent_spangle_intent_arg_tl { _- }
          }
          {
            \bool_if:cTF { l__spintent_sp #1 _concept_bool }
              {
                \tl_set:Ne \l__spintent_spangle_intent_arg_tl
                  {
                    \tl_if_empty:cF { l__spintent_sp #1 _prefix_tl }
                      { \tl_use:c { l__spintent_sp #1 _prefix_tl } - }
                    \l__spintent_spangle_intent_tl :prefix( \l__spintent_spangle_refs_tl )
                  }
              }
              {
                \tl_set:Ne \l__spintent_spangle_intent_arg_tl { _( \l__spintent_spangle_refs_tl )
              }
          }
        \bool_set_false:N \l__spintent_spangle_print_m_now_bool
        \str_if_eq:eeT {#1} { mangle }
          {
            \bool_if:cT { l__spintent_sp #1 _print_m_bool }
              { \bool_set_true:N \l__spintent_spangle_print_m_now_bool }
          }
        \MathMLintent { \l__spintent_spangle_intent_arg_tl }
          {
            \__spintent_inner_sep:
            \bool_if:NT \l__spintent_spangle_print_m_now_bool
              {
                \__spintent_luafun_mrow_intent_arg:nn
                  { } { intent = ":silent" }
                { \operatorname { m } }
              }
            \__spintent_luafun_mrow_intent_arg:nn
              { } { intent = ":silent" }
            { \__spintent_spangle_sym:n {#1} }
            \int_zero:N \l__spintent_spangle_pt_idx_int
            \seq_map_inline:Nn \l__spintent_spangle_pt_seq
              {
                \int_incr:N \l__spintent_spangle_pt_idx_int
                \__spintent_mathmlarg_point:nnn
                  { pt \int_use:N \l__spintent_spangle_pt_idx_int }
                  { sp#1 }
                  { ##1 }
              }
          }
      }
  }
```

(*End of definition for* __spintent_spangle_mathml:n.)

__spintent_spangle_exec:nnn

The function __spintent_spangle_exec:nnn receives the *short name* of the command passed in '#1', the optional ⟨*keys*⟩ passed in '#2' and the {⟨*argument*⟩} passed in '#3', delegating the rendering to __spintent_spangle_process:nn.

```
\cs_new_protected:Nn \__spintent_spangle_exec:nnn
  {
    \keys_set:nn { spintent / sp #1 } { #2 }
    \__spintent_spangle_process:nn { #1 } { #3 }
```

```
}
```

(*End of definition for* `\__spintent_spangle_exec:nnn`.)

`\spangle` `\spmangle`

Both public commands `\spangle` and `\spmangle` enforces *math mode*, opens a local group, and delegates execution to `\__spintent_spangle_exec:nnn`.

```
\NewDocumentCommand \spangle { O{} m }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \attribute \l__spintent_scope_attr_int = 1
          \__spintent_spangle_exec:nnn { angle } { #1 } { #2 }
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spangle } }
  }
\NewDocumentCommand \spmangle { O{} m }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \attribute \l__spintent_scope_attr_int = 1
          \__spintent_spangle_exec:nnn { mangle } { #1 } { #2 }
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spmangle } }
  }
```

(*End of definition for* `\spangle` *and* `\spmangle`. *These functions are documented on page 20.*)

11.42 The commands `\sparc` and `\spmarc`

The user commands `\sparc` and `\spmarc` provide a semantic interface for arcs. Unlike `\spmside` and `\spmangle`, `\spmarc` has no `print-m` key: `\operatorname{m}` is always present, since $\mathrm{m}\,\widehat{AB}$ is the only standard notation for the measure of an arc.

11.42.1 Definition of variables for `\sparc` and `\spmarc`

`\l__spintent_sparc_mode_str` `\l__spintent_spmarc_mode_str` `\l__spintent_sparc_prefix_tl` `\l__spintent_smparc_prefix_tl`

Each pair of variables is declared once for both `\sparc` and `\spmarc` via `\clist_map_inline:nn`. The variables `\l__spintent_sparc_mode_str` and `\l__spintent_spmarc_mode_str` store the values set by the key `read-cmd`; the variables `\l__spintent_sparc_prefix_tl` and `\l__spintent_smparc_prefix_tl` store the values set by the key `prefix`.

```
\clist_map_inline:nn { arc , marc }
  {
    \str_new:c { l__spintent_sp #1 _mode_str  }
    \tl_new:c  { l__spintent_sp #1 _prefix_tl }
  }
```

(*End of definition for* `\l__spintent_sparc_mode_str` *and others.*)

11.42.2 Definition of error messages for `\sparc` and `\spmarc`

`sparc-invalid-points` `spmarc-invalid-points`

We define an "error" message for the commands `\sparc` and `\spmarc`. The "error" is triggered if the user includes a invalid {⟨*argument*⟩} passed in '`#1`' to the command.

```
\clist_map_inline:nn { arc , marc }
  {
    \msg_new:nnnn { spintent } { sp #1 -invalid-points }
      {
        Invalid~point~format~in~##1:~'##2'~\msg_line_context:.
      }
      {
        See~the~documentation~of~##1~for~the~accepted~point~formats.
      }
  }
```

(*End of definition for* `sparc-invalid-points` *and* `spmarc-invalid-points`.)

11.42.3 Definition of keys for \sparc and \spmarc

read-cmd
prefix
silent-cmd
concept
unknown

We define the shared ⟨*keys*⟩ read-cmd, prefix, silent-cmd and concept.

```
\clist_map_inline:nn { arc , marc }
  {
    \keys_define:nn { spintent / sp #1 }
      {
        read-cmd            .choices:nn = { school , mathcat }
          {
            \int_case:nn { \l_keys_choice_int }
              {
                {1} { \str_set:cn { l__spintent_sp #1 _mode_str } { school  } }
                {2} { \str_set:cn { l__spintent_sp #1 _mode_str } { mathcat } }
              }
          },
        read-cmd            .initial:n  = school,
        read-cmd            .value_required:n = true,
        read-cmd / unknown  .code:n =
          \msg_error:nneee { spintent } { unknown-choice }
            { read-cmd } { school,~mathcat } { \exp_not:n {##1} },
        prefix              .code:n =
          { \__spintent_sanitize_affix:cn { l__spintent_sp #1 _prefix_tl } {##1} },
        prefix              .initial:n  = ,
        prefix              .value_required:n = true,
        silent-cmd          .bool_set:c = { l__spintent_sp #1 _silent_cmd_bool },
        silent-cmd          .initial:n  = false,
        silent-cmd          .value_required:n = true,
        concept             .bool_set:c = { l__spintent_sp #1 _concept_bool },
        concept             .initial:n  = true,
        concept             .value_required:n = true,
        unknown             .code:n =
          {
            \str_set:Nn \l__spintent_command_name_str { sp #1 }
            \__spintent_unknown_keys_cmd:n {##1}
          },
      }
  }
```

The key print-m is defined for both commands for simplicity, but has no effect on \sparc.

(*End of definition for* read-cmd *and others.*)

11.42.4 Internal functions for \sparc and \spmarc

__spintent_sparc_wide_arc:n

The function __spintent_sparc_wide_arc:n acts as a fallback when the mathematical font used in the document does not provide the command \widearc. The definition of this function is a direct adaptation of a response given by Clea F. Rees (@cfr) to a query in the chat room of TeX-SX.

```
\cs_new_protected_nopar:Nn \__spintent_sparc_wide_arc:n
  {
    \Umathaccent 7 \symoperators "023DC \scan_stop: {#1}
  }
```

(*End of definition for* __spintent_sparc_wide_arc:n.)

__spintent_sparc_print_arc:n

The function __spintent_sparc_print_arc:n applies the command \widearc to the argument passed to \sparc or \spmarc. If the mathematical font set in the document does not provide the command \widearc, we will use the built-in function __spintent_sparc_wide_arc:n.

```
\cs_new_protected:Nn \__spintent_sparc_print_arc:n
  {
    \cs_if_exist:NTF \widearc
      { \widearc {#1} }
      { \__spintent_sparc_wide_arc:n {#1} }
  }
```

(*End of definition for* __spintent_sparc_print_arc:n.)

\l__spintent_sparc_intent_tl
\l__spintent_sparc_pt_seq
\l__spintent_sparc_refs_tl
\l__spintent_sparc_intent_arg_tl
\l__spintent_sparc_pt_idx_int

\l__spintent_sparc_intent_tl holds the concept word (arco or medida-del-arco). The rest are scratch variables to split \l__spintent_geo_luaset_points_clist and build the $pt1,$pt2,... reference list used in the concept intent.

```
\tl_new:N  \l__spintent_sparc_intent_tl
\seq_new:N \l__spintent_sparc_pt_seq
\tl_new:N  \l__spintent_sparc_refs_tl
\tl_new:N  \l__spintent_sparc_intent_arg_tl
\int_new:N \l__spintent_sparc_pt_idx_int
```

(*End of definition for* `\l__spintent_sparc_intent_tl` *and others.*)

`\__spintent_sparc_process:nn`

The internal function `\__spintent_sparc_process:nn` receives the short name of the command (`arc` or `marc`) in '`#1`' and the {⟨*argument*⟩} in '`#2`'. It calls `\__spintent_luafun_geo_parse_and_set:n` to process '`#2`', returning an "error" if invalid, then sets `\l__spintent_sparc_intent_tl` to `arco` (or `medida-del-arco` for `\spmarc`). No `intent` is built here: it needs the reference list, which `\__spintent_sparc_math` builds after splitting the points.

```
\cs_new_protected:Nn \__spintent_sparc_process:nn
  {
    \__spintent_luafun_geo_parse_and_set:n { \exp_not:n {#2} }
    \str_if_eq:VnT \l__spintent_geo_luaset_error_str { true }
      {
        \msg_error:nnee { spintent } { sp #1 -invalid-points }
          { \c_backslash_str sp #1 } { \exp_not:n {#2} }
      }
    \str_if_eq:eeTF {#1} { marc }
      { \tl_set:Nn \l__spintent_sparc_intent_tl { medida-del-arco } }
      { \tl_set:Nn \l__spintent_sparc_intent_tl { arco } }
    \__spintent_sparc_mathml:n {#1}
  }
```

(*End of definition for* `\__spintent_sparc_process:nn`.)

`\__spintent_sparc_mathml:n`

The function `\__spintent_sparc_mathml:n` receives the short command name in '`#1`'. It splits `\l__spintent_geo_` into `\l__spintent_sparc_pt_seq` and walks it once to build `\l__spintent_sparc_refs_tl`, the comma-separated `$pt1,$pt2,...` reference list, naming each point by position.

```
\cs_new_protected:Nn \__spintent_sparc_mathml:n
  {
    \seq_set_from_clist:NN \l__spintent_sparc_pt_seq \l__spintent_geo_luaset_points_clist

    \tl_clear:N \l__spintent_sparc_refs_tl
    \int_zero:N \l__spintent_sparc_pt_idx_int
    \seq_map_inline:Nn \l__spintent_sparc_pt_seq
      {
        \int_incr:N \l__spintent_sparc_pt_idx_int
        \tl_if_empty:NF \l__spintent_sparc_refs_tl
          { \tl_put_right:Nn \l__spintent_sparc_refs_tl { , } }
        \tl_put_right:Ne \l__spintent_sparc_refs_tl { $ pt \int_use:N \l__spintent_sparc_pt_idx_i
      }
```

The concept `intent` is `_-` when `silent-cmd` is `true` or `concept` is `false`; otherwise `\l__spintent_sparc_intent_` prefixed with `prefix` when given.

```
    \bool_if:cTF { l__spintent_sp #1 _silent_cmd_bool }
      {
        \tl_set:Nn \l__spintent_sparc_intent_arg_tl { _- }
      }
      {
        \bool_if:cTF { l__spintent_sp #1 _concept_bool }
          {
            \tl_set:Ne \l__spintent_sparc_intent_arg_tl
              {
                \tl_if_empty:cF { l__spintent_sp #1 _prefix_tl }
                  { \tl_use:c { l__spintent_sp #1 _prefix_tl } - }
                \l__spintent_sparc_intent_tl :prefix( \l__spintent_sparc_refs_tl )
              }
          }
          {
            \tl_set:Nn \l__spintent_sparc_intent_arg_tl { _- }
          }
      }
```

It is applied with a single native `\MathMLintent` wrapping `\__spintent_sparc_print_arc:n` directly (same shape as `\__spintent_spside_mathml:n`), preceded by `\__spintent_inner_sep:` (a second sibling so the sole `<mrow>` does not collapse into `<math>`, `luamml` issue `#26`). A silent `\operatorname{m}` precedes it for `\spmarc` (always, no `print-m` key here). The sequence is walked a second time, calling `\__spintent_mathmlarg_point:nnn` once per point with its `pt`*n* reference name, looked up under `sp#1`.

```
    \__spintent_invisible_sep: % need by MathCAT (bug in luamml issues #26)
    \MathMLintent { \l__spintent_sparc_intent_arg_tl }
      {
        \__spintent_inner_sep:
        \str_if_eq:eeT {#1} { marc }
```

```
          {
            \__spintent_luafun_mrow_intent_arg:nn
              { } { intent = ":silent" }
            { \operatorname { m } }
          }
        \__spintent_sparc_print_arc:n
          {
            \int_zero:N \l__spintent_sparc_pt_idx_int
            \seq_map_inline:Nn \l__spintent_sparc_pt_seq
              {
                \int_incr:N \l__spintent_sparc_pt_idx_int
                \__spintent_mathmlarg_point:nnn
                  { pt \int_use:N \l__spintent_sparc_pt_idx_int }
                  { sp #1 }
                  { ##1 }
              }
          }
      }
  }
```

(*End of definition for* `\__spintent_sparc_mathml:n`.)

`\__spintent_sparc_exec:nnn`

The function `\__spintent_sparc_exec:nnn` receives the *short name* of the command passed in '`#1`', the optional ⟨*keys*⟩ passed in '`#2`' and the {⟨*argument*⟩} passed in '`#3`', delegating the rendering to `\__spintent_sparc_process:nn`.

```
\cs_new_protected:Nn \__spintent_sparc_exec:nnn
  {
    \keys_set:nn { spintent / sp #1 } { #2 }
    \__spintent_sparc_process:nn { #1 } { #3 }
  }
```

(*End of definition for* `\__spintent_sparc_exec:nnn`.)

`\sparc` `\spmarc`

Both public commands `\sparc` and `\spmarc` enforces *math mode*, opens a local group, and delegates execution to `\__spintent_sparc_exec:nnn`.

```
\NewDocumentCommand \sparc { O{} m }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \attribute \l__spintent_scope_attr_int = 1
          \__spintent_sparc_exec:nnn { arc } { #1 } { #2 }
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \sparc } }
  }
\NewDocumentCommand \spmarc { O{} m }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \attribute \l__spintent_scope_attr_int = 1
          \__spintent_sparc_exec:nnn { marc } { #1 } { #2 }
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spmarc } }
  }
```

(*End of definition for* `\sparc` *and* `\spmarc`. *These functions are documented on page 21.*)

11.43 The command `\spPoly`

The user command `\spPoly` typesets a polygon defined by its vertices. The figure type and its visual symbol are inferred from the vertex count: three points produce a triangle, four a square, five or more a generic polygon with no symbol. It never sends a boolean to Lua: `\l__spintent_geo_luaset_intent_str` (the full `intent`, with the figure name).

11.43.1 Definition of variables for `\spPoly`

`\l__spintent_spPoly_mode_str` `\l__spintent_spPoly_prefix_tl`

The variable `\l__spintent_spPoly_mode_str` stores the value set by the key `read-cmd`; the variable `\l__spintent_spPoly_prefix_tl` stores the value set by the key `prefix`.

```
\str_new:N \l__spintent_spPoly_mode_str
\tl_new:N  \l__spintent_spPoly_prefix_tl
```

(*End of definition for* `\l__spintent_spPoly_mode_str` *and* `\l__spintent_spPoly_prefix_tl`.)

11.43.2 Definition of error messages for \spPoly

spPoly-invalid-points Raised when the argument is not a comma-separated list of at least three uppercase letters.

```
\msg_new:nnnn { spintent } { spPoly-invalid-points }
  {
    Invalid~point~format~in~\token_to_str:N \spPoly:~'#1'~\msg_line_context:.
  }
  {
    See~the~documentation~of~\token_to_str:N \spPoly~for~the~accepted~point~formats.
  }
```

(*End of definition for* spPoly-invalid-points.)

11.43.3 Definition of keys for \spPoly

read-cmd, prefix, print-sym, silent-cmd, concept, unknown

We define the ⟨*keys*⟩ read-cmd, prefix, print-sym, silent-cmd and concept.

```
\keys_define:nn { spintent / spPoly }
  {
    read-cmd            .choices:nn = { school , mathcat }
      {
        \int_case:nn { \l_keys_choice_int }
          {
            {1} { \str_set:Nn \l__spintent_spPoly_mode_str { school  } }
            {2} { \str_set:Nn \l__spintent_spPoly_mode_str { mathcat } }
          }
      },
    read-cmd            .initial:n  = school,
    read-cmd            .value_required:n = true,
    read-cmd / unknown  .code:n =
      \msg_error:nneee { spintent } { unknown-choice }
        { read-cmd } { school,~mathcat } { \exp_not:n {#1} },
    prefix              .code:n =
      { \__spintent_sanitize_affix:Nn \l__spintent_spPoly_prefix_tl {#1} },
    prefix              .initial:n  = ,
    prefix              .value_required:n = true,
    print-sym           .bool_set:N = \l__spintent_spPoly_print_sym_bool,
    print-sym           .initial:n  = true,
    print-sym           .value_required:n = true,
    silent-cmd          .bool_set:N = \l__spintent_spPoly_silent_cmd_bool,
    silent-cmd          .initial:n  = false,
    silent-cmd          .value_required:n = true,
    concept             .bool_set:N = \l__spintent_spPoly_concept_bool,
    concept             .initial:n  = true,
    concept             .value_required:n = true,
    unknown             .code:n =
      {
        \str_set:Nn \l__spintent_command_name_str { spPoly }
        \__spintent_unknown_keys_cmd:n {#1}
      },
  }
```

(*End of definition for* read-cmd *and others.*)

11.43.4 Internal functions for \spPoly

\l__spintent_spPoly_pt_seq, \l__spintent_spPoly_refs_tl, \l__spintent_spPoly_intent_arg_tl, \l__spintent_spPoly_pt_idx_int

Scratch variables to split \l__spintent_geo_luaset_points_clist and build the $pt1,$pt2,... reference list, and to hold the final concept intent once built.

```
\seq_new:N \l__spintent_spPoly_pt_seq
\tl_new:N  \l__spintent_spPoly_refs_tl
\tl_new:N  \l__spintent_spPoly_intent_arg_tl
\int_new:N \l__spintent_spPoly_pt_idx_int
```

(*End of definition for* \l__spintent_spPoly_pt_seq *and others.*)

__spintent_spPoly_sym: Unchanged: reads \l__spintent_geo_luaset_poly_sym_str and typesets the corresponding symbol.

```
\cs_new_protected:Nn \__spintent_spPoly_sym:
  {
    \str_case:Vn \l__spintent_geo_luaset_poly_sym_str
      {
        { triangle } { \triangle      }
        { square   } { \mdlgwhtsquare }
      }
  }
```

(*End of definition for* `\__spintent_spPoly_sym:`.)

`\__spintent_spPoly_build_refs:` Splits `\l__spintent_geo_luaset_points_clist` into `\l__spintent_spPoly_pt_seq` and walks it once to build `\l__spintent_spPoly_refs_tl`, the comma-separated `$pt1,$pt2,...` reference list, naming each vertex by position.

```
\cs_new_protected:Nn \__spintent_spPoly_build_refs:
  {
    \seq_set_from_clist:NN \l__spintent_spPoly_pt_seq \l__spintent_geo_luaset_points_clist
    \tl_clear:N \l__spintent_spPoly_refs_tl
    \int_zero:N \l__spintent_spPoly_pt_idx_int
    \seq_map_inline:Nn \l__spintent_spPoly_pt_seq
      {
        \int_incr:N \l__spintent_spPoly_pt_idx_int
        \tl_if_empty:NF \l__spintent_spPoly_refs_tl
          { \tl_put_right:Nn \l__spintent_spPoly_refs_tl { , } }
        \tl_put_right:Ne \l__spintent_spPoly_refs_tl { $ pt \int_use:N \l__spintent_spPoly_pt_idx
      }
  }
```

(*End of definition for* `\__spintent_spPoly_build_refs:`.)

`\__spintent_spPoly_build_concept:` Builds `\l__spintent_spPoly_intent_arg_tl`, independently of `print-sym`: `_(refs)` when `silent-cmd` is `true` or `concept` is `false`; otherwise `\l__spintent_geo_luaset_concept_str:prefix(refs)`, prefixed with `prefix` when given. Must be called after `\__spintent_spPoly_build_refs:`.

```
\cs_new_protected:Nn \__spintent_spPoly_build_concept:
  {
    \bool_if:NTF \l__spintent_spPoly_silent_cmd_bool
      {
        \tl_set:Ne \l__spintent_spPoly_intent_arg_tl { _( \l__spintent_spPoly_refs_tl ) }
      }
      {
        \bool_if:NTF \l__spintent_spPoly_concept_bool
          {
            \tl_set:Ne \l__spintent_spPoly_intent_arg_tl
              {
                \tl_if_empty:NF \l__spintent_spPoly_prefix_tl
                  { \tl_use:N \l__spintent_spPoly_prefix_tl - }
                \l__spintent_geo_luaset_concept_str :prefix( \l__spintent_spPoly_refs_tl )
              }
          }
          {
            \tl_set:Ne \l__spintent_spPoly_intent_arg_tl { _( \l__spintent_spPoly_refs_tl ) }
          }
      }
  }
```

(*End of definition for* `\__spintent_spPoly_build_concept:`.)

`\__spintent_spPoly_points_loop:` Walks `\l__spintent_spPoly_pt_seq` a second time, calling `\__spintent_mathmlarg_point:nnn` once per vertex with its `pt`*n* reference name, looked up under `spPoly`: it alone decides `arg`/literal `intent` per vertex — replacing the single combined `\MathMLarg{pts}{...}` that lost `arg` on any vertex with a subscript or prime (the same `annotate_attribute`/sub-sup limitation already found and worked around for `\spPoint`).

```
\cs_new_protected:Nn \__spintent_spPoly_points_loop:
  {
    \int_zero:N \l__spintent_spPoly_pt_idx_int
    \seq_map_inline:Nn \l__spintent_spPoly_pt_seq
      {
        \int_incr:N \l__spintent_spPoly_pt_idx_int
        \__spintent_mathmlarg_point:nnn
          { pt \int_use:N \l__spintent_spPoly_pt_idx_int }
          { spPoly }
          { ##1 }
      }
  }
```

(*End of definition for* `\__spintent_spPoly_points_loop:`.)

`\__spintent_spPoly_process:n` Calls `\__spintent_luafun_geo_poly_parse_and_set:n`, reports `spPoly-invalid-points` on failure, then delegates to `\__spintent_spPoly_mathml:`. No `intent` is built here: it needs the reference list, which `\__spintent_spPoly_mathml:` builds after splitting the vertices.

```
\cs_new_protected:Nn \__spintent_spPoly_process:n
```

```
{
  \__spintent_luafun_geo_poly_parse_and_set:n { \exp_not:n {#1} }
  \str_if_eq:VnT \l__spintent_geo_luaset_error_str { true }
    {
      \msg_error:nne { spintent } { spPoly-invalid-points }
        { \exp_not:n {#1} }
    }
  \__spintent_spPoly_mathml:
}
```

(*End of definition for* `\__spintent_spPoly_process:n`.)

`\__spintent_spPoly_mathml:` Builds the reference list and the concept `intent` (`\__spintent_spPoly_build_refs:`, `\__spintent_spPoly_buil` then wraps everything in a single native `\MathMLintent`, preceded by `\__spintent_inner_sep:` (a second sibling so the sole `<mrow>` does not collapse into `<math>`, `luamml` issue `#26`).
`print-sym` and `concept/silent-cmd` are independent: the former only decides whether the glyph (`\__spintent_spPoly_sym:`, silenced with `\__spintent_luafun_mrow_intent_arg:nn`) is attempted — it is skipped anyway when `\l__spintent_geo_luaset_poly_sym_str` is empty (no symbol defined for that vertex count, e.g. pentagon and above) — and never affects whether the concept word itself is spoken.

```
\cs_new_protected:Nn \__spintent_spPoly_mathml:
  {
    \__spintent_invisible_sep: % (luamml issues #26)
    \__spintent_spPoly_build_refs:
    \__spintent_spPoly_build_concept:
    \MathMLintent { \l__spintent_spPoly_intent_arg_tl }
      {
        \__spintent_inner_sep:
        \bool_if:NT \l__spintent_spPoly_print_sym_bool
          {
            \str_if_empty:NF \l__spintent_geo_luaset_poly_sym_str
              {
                \__spintent_luafun_mrow_intent_arg:nn
                  { } { intent = ":silent" }
                { \__spintent_spPoly_sym: }
              }
          }
        \__spintent_spPoly_points_loop:
      }
  }
```

(*End of definition for* `\__spintent_spPoly_mathml:`.)

`\__spintent_spPoly_exec:nn` The function `\__spintent_spPoly_exec:nn` receives the optional keys in `#1` and the argument in `#2`, delegating the rendering to `\__spintent_spPoly_process:n`.

```
\cs_new_protected:Nn \__spintent_spPoly_exec:nn
  {
    \keys_set:nn { spintent / spPoly } { #1 }
    \__spintent_spPoly_process:n { #2 }
  }
```

(*End of definition for* `\__spintent_spPoly_exec:nn`.)

`\spPoly` The public command `\spPoly` enforces *math mode*, opens a local group, and delegates execution to `\__spintent_spPoly_exec:nn`.

```
\NewDocumentCommand \spPoly { O{} m }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \attribute \l__spintent_scope_attr_int = 1
          \__spintent_spPoly_exec:nn { #1 } { #2 }
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spPoly } }
  }
```

(*End of definition for* `\spPoly`. *This function is documented on page* 22.)

11.44 The commands \spAfig and \spPfig

The user commands \spAfig and \spPfig typeset the area and the perimeter of a polygonal figure, using the delimited-optional argument syntax d() already established by \spmcd/\spmcm. The key print-sym is never sent to Lua: the symbol type is returned unconditionally in \l__spintent_geo_luaset_poly_sym_str, and expl3 decides whether to wrap it in \MathMLintent{⟨*:silent*⟩} — the same mechanism already used by \spPoly.

11.44.1 Definition of variables for \spAfig and \spPfig

\l__spintent_geo_final_print_tl

Scratch variables used by __spintent_spfig_process:nn to assemble the visual and the intent. \l__spintent_geo_final_print_tl holds the final visual placed under \mathcal's subscript. The final literal intent and the concept forwarded when read-cmd is mathcat.

```
\tl_new:N  \l__spintent_geo_final_print_tl
```

(*End of definition for* \l__spintent_geo_final_print_tl.)

\l__spintent_spAfig_mode_str
\l__spintent_spPfig_mode_str
\l__spintent_spAfig_print_sym_str
\l__spintent_spPfig_print_sym_str
\l__spintent_spAfig_prefix_tl
\l__spintent_spPfig_prefix_tl
\l__spintent_spAfig_read_arg_tl
\l__spintent_spPfig_read_arg_tl

These variables store the value set by each corresponding key: read-cmd, print-sym, prefix and read-arg, respectively. The value set by read-arg overrides the figure name inferred from the vertex count, independently of print-sym.

```
\clist_map_inline:nn { Afig , Pfig }
  {
    \str_new:c { l__spintent_sp #1 _mode_str      }
    \str_new:c { l__spintent_sp #1 _print_sym_str }
    \tl_new:c  { l__spintent_sp #1 _prefix_tl     }
    \tl_new:c  { l__spintent_sp #1 _read_arg_tl   }
  }
```

(*End of definition for* \l__spintent_spAfig_mode_str *and others.*)

11.44.2 Definition of error messages for \spAfig and \spPfig

spAfig-invalid-points
spPfig-invalid-points
spAfig-invalid-arg
spPfig-invalid-arg
spAfig-sym-letra-conflict
spPfig-sym-letra-conflict
spAfig-sym-cardinality-conflict
spPfig-sym-cardinality-conflict
spAfig-read-arg-key
spPfig-read-arg-key

All eight messages are generated by the same map, using the same ##1/##2 doubling convention as \spside/\spmside. sp #1-invalid-points is currently unused by __spintent_spfig_process:nn, which raises only sp #1-invalid-arg for any parse failure, regardless of whether a comma was present. sp #1-sym-letra-conflict and sp #1-sym-cardinality-conflict are raised when print-sym is forced to a value incompatible with the actual argument.

```
\clist_map_inline:nn { Afig , Pfig }
  {
    \msg_new:nnnn { spintent } { sp #1 -invalid-points }
      {
        Invalid~point~format~in~##1:~'##2'~\msg_line_context:.
      }
      {
        See~the~documentation~of~##1~for~the~accepted~point~formats.
      }
    \msg_new:nnnn { spintent } { sp #1 -invalid-arg }
      {
        Invalid~argument~format~in~##1:~'##2'~\msg_line_context:.
      }
      {
        See~the~documentation~of~##1~for~the~accepted~argument~forms.
      }
    \msg_new:nnnn { spintent } { sp #1 -sym-letra-conflict }
      {
        print-sym~cannot~be~combined~with~a~lettered~argument~in~'##1'~\msg_line_context:.
      }
      {
        See~the~documentation~of~##1~for~details~on~print-sym.
      }
    \msg_new:nnnn { spintent } { sp #1 -sym-cardinality-conflict }
      {
        print-sym~does~not~match~the~vertex~count~in~'##1'~\msg_line_context:.
      }
      {
        See~the~documentation~of~##1~for~details~on~print-sym.
      }
    \msg_new:nnnn { spintent } { sp #1 -read-arg-key }
      {
        read-arg~has~no~effect~with~a~single~argument~in~'##1'~\msg_line_context:.
      }
      {
        See~the~documentation~of~##1~for~details~on~read-arg.
```

```
      }
  }
```

(*End of definition for* `spAfig-invalid-points` *and others.*)

11.44.3 Definition of keys for \spAfig and \spPfig

read-cmd
prefix
print-sym
read-arg
read-sub
silent-cmd
concept
unknown

The key `print-sym` chooses the visual symbol: `auto` (default) infers it from the vertex count (three points give `\triangle`, four give `\mdlgwhtsquare`, five or more give none); `triangle`/`square` force one, requiring the vertex count to match; `none` omits it, even with vertices. With a bare number/letter argument, `auto` and `none` behave the same (no symbol); `triangle`/`square` nest the argument under the forced symbol instead, and cannot be combined with a table letter (`F`/`P`/`B`/`L`). The key `read-arg` overrides the spoken figure name for real vertices, independently of `print-sym`. The key `read-sub` controls whether the bare number/letter argument is read with the word «sub» or as a plain number.

```
\clist_map_inline:nn { Afig , Pfig }
  {
    \keys_define:nn { spintent / sp #1 }
      {
        read-cmd            .choices:nn = { school , mathcat }
          {
            \int_case:nn { \l_keys_choice_int }
              {
                {1} { \str_set:cn { l__spintent_sp #1 _mode_str } { school  } }
                {2} { \str_set:cn { l__spintent_sp #1 _mode_str } { mathcat } }
              }
          },
        read-cmd            .initial:n  = school,
        read-cmd            .value_required:n = true,
        read-cmd / unknown  .code:n =
          \msg_error:nneee { spintent } { unknown-choice }
            { read-cmd } { school,~mathcat } { \exp_not:n {##1} },
        prefix              .code:n =
          { \__spintent_sanitize_affix:cn { l__spintent_sp #1 _prefix_tl } {##1} },
        prefix              .initial:n  = ,
        prefix              .value_required:n = true,
        print-sym           .choices:nn = { auto , triangle , square , none }
          {
            \int_case:nn { \l_keys_choice_int }
              {
                {1} { \str_set:cn { l__spintent_sp #1 _print_sym_str } { auto     } }
                {2} { \str_set:cn { l__spintent_sp #1 _print_sym_str } { triangle } }
                {3} { \str_set:cn { l__spintent_sp #1 _print_sym_str } { square   } }
                {4} { \str_set:cn { l__spintent_sp #1 _print_sym_str } { none     } }
              }
          },
        print-sym           .initial:n  = auto,
        print-sym           .value_required:n = true,
        print-sym / unknown  .code:n =
          \msg_error:nneee { spintent } { unknown-choice }
            { print-sym } { auto,~triangle,~square,~none } { \exp_not:n {##1} },
        read-arg            .code:n =
          { \__spintent_sanitize_affix:cn { l__spintent_sp #1 _read_arg_tl } {##1} },
        read-arg            .initial:n  = ,
        read-arg            .value_required:n = true,
        read-sub            .bool_set:c = { l__spintent_sp #1 _read_sub_bool },
        read-sub            .initial:n  = false,
        read-sub            .value_required:n = true,
        silent-cmd          .bool_set:c = { l__spintent_sp #1 _silent_cmd_bool },
        silent-cmd          .initial:n  = false,
        silent-cmd          .value_required:n = true,
        concept             .bool_set:c = { l__spintent_sp #1 _concept_bool },
        concept             .initial:n  = true,
        concept             .value_required:n = true,
        unknown             .code:n =
          {
            \str_set:Nn \l__spintent_command_name_str { sp #1 }
            \__spintent_unknown_keys_cmd:n {##1}
          },
      }
  }
```

(*End of definition for* `read-cmd` *and others.*)

11.44.4 Internal functions for \spAfig and \spPfig

__spintent_spfig_sym:

The function __spintent_spfig_sym: reads \l__spintent_geo_print_sym_str and typesets the corresponding symbol: \triangle for the value triangle, \mdlgwhtsquare for the value square.

```
\cs_new_protected:Nn \__spintent_spfig_sym:
  {
    \str_case:Vn \l__spintent_geo_print_sym_str
      {
        { triangle } { \triangle      }
        { square   } { \mdlgwhtsquare }
      }
  }
```

(*End of definition for* __spintent_spfig_sym:.)

\l__spintent_spfig_op_tl
\l__spintent_spfig_intent_tl
\l__spintent_spfig_symbol_tl
\l__spintent_spfig_pt_seq
\l__spintent_spfig_refs_tl
\l__spintent_spfig_intent_arg_tl
\l__spintent_spfig_pt_idx_int
\l__spintent_geo_print_sym_str

\l__spintent_spfig_op_tl holds área or perímetro. \l__spintent_spfig_intent_tl holds the concept word for the vertices case (read-arg or the auto-detected figure name). The rest are scratch variables to split \l__spintent_geo_luaset_points_clist and build the $pt1,$pt2,... reference list.

```
\tl_new:N  \l__spintent_spfig_op_tl
\tl_new:N  \l__spintent_spfig_intent_tl
\tl_new:N  \l__spintent_spfig_desc_tl
\tl_new:N  \l__spintent_spfig_symbol_tl
\seq_new:N \l__spintent_spfig_pt_seq
\tl_new:N  \l__spintent_spfig_refs_tl
\tl_new:N  \l__spintent_spfig_intent_arg_tl
\int_new:N \l__spintent_spfig_pt_idx_int
\str_new:N \l__spintent_geo_print_sym_str
```

(*End of definition for* \l__spintent_spfig_op_tl *and others.*)

__spintent_spfig_process:nn

Receives the short command name (Afig or Pfig) in '#1' and the {⟨*argument*⟩} in '#2'. Calls __spintent_luafun_ge or __spintent_luafun_geo_pfig_parse_and_set:n, reports sp#1-invalid-arg on failure, and sp#1-read-ar when read-arg is given for a bare number/letter argument (no vertices), where it has no effect. Sets \l__spintent_spfig_op_tl to área/perímetro, then delegates to __spintent_spfig_mathml:n.

```
\cs_new_protected:Nn \__spintent_spfig_process:nn
  {
    \str_if_eq:eeTF {#1} { Afig }
      { \__spintent_luafun_geo_afig_parse_and_set:n { \exp_not:n {#2} } }
      { \__spintent_luafun_geo_pfig_parse_and_set:n { \exp_not:n {#2} } }
    \str_if_eq:VnT \l__spintent_geo_luaset_error_str { true }
      {
        \msg_error:nnee { spintent } { sp #1 -invalid-arg }
          { \c_backslash_str sp #1 } { \exp_not:n {#2} }
      }
    \clist_if_empty:NT \l__spintent_geo_luaset_points_clist
      {
        \tl_if_empty:cF { l__spintent_sp #1 _read_arg_tl }
          {
            \msg_error:nne { spintent } { sp #1 -read-arg-key }
              { \c_backslash_str sp #1 }
          }
      }
    \str_if_eq:eeTF {#1} { Afig }
      { \tl_set:Nn \l__spintent_spfig_op_tl { área } }
      { \tl_set:Nn \l__spintent_spfig_op_tl { perímetro } }
    \str_set:Ne \l__spintent_geo_print_sym_str { \str_use:c { l__spintent_sp #1 _print_sym_str } }
    \__spintent_spfig_mathml:n {#1}
  }
```

(*End of definition for* __spintent_spfig_process:nn.)

__spintent_spfig_build_simple:n

Handles the argument-less, table-letter and bare number/letter cases (no vertices). Builds \l__spintent_geo_final_pr (the visual placed under \mathcal's subscript) and \l__spintent_spfig_desc_tl (the descriptive part, without área/perímetro). Receives the short command name in '#1'.
With an empty argument, both are cleared for print-sym auto/ none; a forced print-sym instead places the symbol alone, with its own name as the description.

```
\cs_new_protected:Nn \__spintent_spfig_build_simple:n
  {
    \tl_if_empty:nTF { \l__spintent_geo_luaset_print_tl }
      {
        \str_case:VnF \l__spintent_geo_print_sym_str
```

```
          {
            { auto }
              {
                \tl_clear:N \l__spintent_geo_final_print_tl
                \tl_clear:N \l__spintent_spfig_desc_tl
              }
            { none }
              {
                \tl_clear:N \l__spintent_geo_final_print_tl
                \tl_clear:N \l__spintent_spfig_desc_tl
              }
          }
          {
            \tl_set:Nn \l__spintent_geo_final_print_tl { \__spintent_spfig_sym: }
            \tl_set:Ne \l__spintent_spfig_desc_tl
              {
                \str_if_eq:VnTF \l__spintent_geo_print_sym_str { triangle }
                  { triángulo } { cuadrado }
              }
          }
      }
```

With a bare number/letter (empty `\l__spintent_geo_luaset_letra_word_str`), the value itself becomes the visual for `print-sym` `auto`/`none`, or its own subscript under a forced symbol otherwise; the description is the spoken value, prefixed with «sub-» when `read-sub` is `true` (further prefixed with the symbol's own name for a forced `print-sym`).

```
      {
        \str_if_empty:NTF \l__spintent_geo_luaset_letra_word_str
          {
            \str_case:VnF \l__spintent_geo_print_sym_str
              {
                { auto } { \tl_set:Nn \l__spintent_geo_final_print_tl { \l__spintent_geo_luaset_p
                { none } { \tl_set:Nn \l__spintent_geo_final_print_tl { \l__spintent_geo_luaset_p
              }
              {
                \tl_set:Nn \l__spintent_geo_final_print_tl
                  {
                    \__spintent_spfig_sym:
                    \c_math_subscript_token { \l__spintent_geo_luaset_print_tl }
                  }
              }
            \str_case:VnF \l__spintent_geo_print_sym_str
              {
                { auto }
                  {
                    \tl_set:Ne \l__spintent_spfig_desc_tl
                      {
                        \bool_if:cT { l__spintent_sp #1 _read_sub_bool } { sub - }
                        \l__spintent_geo_luaset_sub_spoken_str
                      }
                  }
                { none }
                  {
                    \tl_set:Ne \l__spintent_spfig_desc_tl
                      {
                        \bool_if:cT { l__spintent_sp #1 _read_sub_bool } { sub - }
                        \l__spintent_geo_luaset_sub_spoken_str
                      }
                  }
              }
              {
                \tl_set:Ne \l__spintent_spfig_desc_tl
                  {
                    \str_if_eq:VnTF \l__spintent_geo_print_sym_str { triangle }
                      { triángulo } { cuadrado }
                    -
                    \bool_if:cT { l__spintent_sp #1 _read_sub_bool } { sub - }
                    \l__spintent_geo_luaset_sub_spoken_str
                  }
              }
          }
```

With a table letter (F/P/B/L/T, non-empty \l__spintent_geo_luaset_letra_word_str), a forced print-sym raises sp#1-sym-letra-conflict — a table letter is never nested under a symbol. The visual is the value as-is; the description is its own word, connected as \l_@@_geo_luaset_letra_conector_str requires, with the same spoken subscript as above when present.

```
          {
            \str_case:VnF \l__spintent_geo_print_sym_str
              {
                { auto } { }
                { none } { }
              }
              {
                \msg_error:nne { spintent } { sp #1 -sym-letra-conflict }
                  { \c_backslash_str sp #1 }
              }
            \tl_set:Nn \l__spintent_geo_final_print_tl { \l__spintent_geo_luaset_print_tl }
            \tl_set:Ne \l__spintent_spfig_desc_tl
              {
                \str_if_empty:NF \l__spintent_geo_luaset_letra_conector_str
                  { \l__spintent_geo_luaset_letra_conector_str - }
                \l__spintent_geo_luaset_letra_word_str
                \str_if_empty:NF \l__spintent_geo_luaset_sub_spoken_str
                  {
                    -
                    \bool_if:cT { l__spintent_sp #1 _read_sub_bool } { sub - }
                    \l__spintent_geo_luaset_sub_spoken_str
                  }
              }
          }
      }
```

\l__spintent_spfig_intent_tl is always the _word- shape: \l_@@_spfig_op_tl alone when \l__spintent_sp is empty, both joined by a dash otherwise. silent-cmd overrides \l__spintent_spfig_intent_arg_tl with _-; with silent-cmd=false and concept false, it becomes _\l__spintent_spfig_desc_tl- (the descriptive part alone, without área/perímetro); with concept true it is \l__spintent_spfig_intent_tl.

```
    \tl_set:Ne \l__spintent_spfig_intent_tl
      {
        \tl_if_empty:NTF \l__spintent_spfig_desc_tl
          { _ \l__spintent_spfig_op_tl - }
          { _ \l__spintent_spfig_op_tl - \l__spintent_spfig_desc_tl - }
      }
    \bool_if:cTF { l__spintent_sp #1 _silent_cmd_bool }
      { \tl_set:Nn \l__spintent_spfig_intent_arg_tl { _- } }
      {
        \bool_if:cTF { l__spintent_sp #1 _concept_bool }
          { \tl_set:Ne \l__spintent_spfig_intent_arg_tl { \l__spintent_spfig_intent_tl } }
          { \tl_set:Ne \l__spintent_spfig_intent_arg_tl { _ \l__spintent_spfig_desc_tl - } }
      }
  }
```

(*End of definition for* __spintent_spfig_build_simple:n.)

__spintent_spfig_build_refs: Splits \l__spintent_geo_luaset_points_clist into \l__spintent_spfig_pt_seq and walks it once to build \l__spintent_spfig_refs_tl, the comma-separated $pt1,$pt2,... reference list, naming each vertex by position.

```
\cs_new_protected:Nn \__spintent_spfig_build_refs:
  {
    \seq_set_from_clist:NN \l__spintent_spfig_pt_seq \l__spintent_geo_luaset_points_clist
    \tl_clear:N \l__spintent_spfig_refs_tl
    \int_zero:N \l__spintent_spfig_pt_idx_int
    \seq_map_inline:Nn \l__spintent_spfig_pt_seq
      {
        \int_incr:N \l__spintent_spfig_pt_idx_int
        \tl_if_empty:NF \l__spintent_spfig_refs_tl
          { \tl_put_right:Nn \l__spintent_spfig_refs_tl { , } }
        \tl_put_right:Ne \l__spintent_spfig_refs_tl { $ pt \int_use:N \l__spintent_spfig_pt_idx_i
      }
  }
```

(*End of definition for* __spintent_spfig_build_refs:.)

__spintent_spfig_build_concept_vertices:n

Handles the vertices case. Receives the short command name in '#1'. Must be called after __spintent_spfig_build_r read-arg, when given, overrides the auto-detected figure name in \l__spintent_spfig_intent_tl: with concept true it is still prefixed with área-del-/perímetro-del-, with concept false it is used verbatim. With read-arg empty, \l__spintent_spfig_intent_tl is simply \l__spintent_geo_luaset_concept_str.

```
\cs_new_protected:Nn \__spintent_spfig_build_concept_vertices:n
  {
    \tl_if_empty:cTF { l__spintent_sp #1 _read_arg_tl }
      { \tl_set:Ne \l__spintent_spfig_intent_tl { \l__spintent_geo_luaset_concept_str } }
      {
        \bool_if:cTF { l__spintent_sp #1 _concept_bool }
          {
            \tl_set:Ne \l__spintent_spfig_intent_tl
              {
                \l__spintent_spfig_op_tl -del-
                \tl_use:c { l__spintent_sp #1 _read_arg_tl }
              }
          }
          {
            \tl_set:Ne \l__spintent_spfig_intent_tl { \tl_use:c { l__spintent_sp #1 _read_arg_tl
          }
      }
```

Determines \l__spintent_spfig_symbol_tl (empty when none applies): for print-sym=auto, the symbol matching \l__spintent_geo_luaset_poly_sym_str (\triangle/ \mdlgwhtsquare), or none when that is empty; print-sym=none always clears it; a forced print-sym requires it to match the actual vertex count, reporting sp#1-sym-cardinality-conflict otherwise.

```
    \str_case:VnF \l__spintent_geo_print_sym_str
      {
        { auto }
          {
            \str_case:Vn \l__spintent_geo_luaset_poly_sym_str
              {
                { triangle } { \tl_set:Nn \l__spintent_spfig_symbol_tl { \triangle      } }
                { square   } { \tl_set:Nn \l__spintent_spfig_symbol_tl { \mdlgwhtsquare } }
              }
            \str_if_empty:NT \l__spintent_geo_luaset_poly_sym_str
              { \tl_clear:N \l__spintent_spfig_symbol_tl }
          }
        { none }
          { \tl_clear:N \l__spintent_spfig_symbol_tl }
      }
      {
        \str_if_eq:VVTF \l__spintent_geo_print_sym_str \l__spintent_geo_luaset_poly_sym_str
          { \tl_set:Nn \l__spintent_spfig_symbol_tl { \__spintent_spfig_sym: } }
          {
            \msg_error:nne { spintent } { sp #1 -sym-cardinality-conflict }
              { \c_backslash_str sp #1 }
          }
      }
```

silent-cmd sets \l__spintent_spfig_intent_arg_tl to _(refs). Otherwise, with read-arg empty, concept chooses between \l__spintent_spfig_intent_tl:prefix(refs) (prefixed with prefix when given) and _(refs) alone; with read-arg given, it is always \l__spintent_spfig_intent_tl:prefix(refs) — read-arg is never silenced by concept alone, only by silent-cmd (same convention as __spintent_spside_proc

```
    \bool_if:cTF { l__spintent_sp #1 _silent_cmd_bool }
      {
        \tl_set:Ne \l__spintent_spfig_intent_arg_tl { _( \l__spintent_spfig_refs_tl ) }
      }
      {
        \tl_if_empty:cTF { l__spintent_sp #1 _read_arg_tl }
          {
            \bool_if:cTF { l__spintent_sp #1 _concept_bool }
              {
                \tl_set:Ne \l__spintent_spfig_intent_arg_tl
                  {
                    \tl_if_empty:cF { l__spintent_sp #1 _prefix_tl }
                      { \tl_use:c { l__spintent_sp #1 _prefix_tl } - }
                    \l__spintent_spfig_intent_tl :prefix( \l__spintent_spfig_refs_tl )
                  }
              }
              {
```

```
                \tl_set:Ne \l__spintent_spfig_intent_arg_tl { _( \l__spintent_spfig_refs_tl ) }
              }
          }
          {
            \tl_set:Ne \l__spintent_spfig_intent_arg_tl
              {
                \tl_if_empty:cF { l__spintent_sp #1 _prefix_tl }
                  { \tl_use:c { l__spintent_sp #1 _prefix_tl } - }
                \l__spintent_spfig_intent_tl :prefix( \l__spintent_spfig_refs_tl )
              }
          }
      }
  }
```

(*End of definition for* `\__spintent_spfig_build_concept_vertices:n`.)

`\__spintent_spfig_points_loop:n` Walks `\l__spintent_spfig_pt_seq` a second time, calling `\__spintent_mathmlarg_point:nnn` once per vertex with its `pt`*n* reference name, looked up under `sp#1`. Receives the short command name in '`#1`'.

```
\cs_new_protected:Nn \__spintent_spfig_points_loop:n
  {
    \int_zero:N \l__spintent_spfig_pt_idx_int
    \seq_map_inline:Nn \l__spintent_spfig_pt_seq
      {
        \int_incr:N \l__spintent_spfig_pt_idx_int
        \__spintent_mathmlarg_point:nnn
          { pt \int_use:N \l__spintent_spfig_pt_idx_int }
          { sp #1 }
          { ##1 }
      }
  }
```

(*End of definition for* `\__spintent_spfig_points_loop:n`.)

`\__spintent_spfig_mathml:n` Receives the short command name in '`#1`'. Branches on whether `\l__spintent_geo_luaset_points_clist` is empty.

Without vertices, `\__spintent_spfig_build_simple:n` builds the content; a `\MathMLintent` wraps it only when `concept` is `true` — with `concept=false` the content is typeset directly, with no `intent` at all.

With vertices, `\__spintent_spfig_build_refs:` then `\__spintent_spfig_build_concept_vertices:n` build the reference list and the concept `intent`, applied with a single native `\MathMLintent`. Inside, `\__spintent_mathcal:n{`$\langle A \mid P \rangle$`}` is silenced with `\__spintent_luafun_mrow_intent_arg:nn` (it would otherwise be claimed by the queue meant for the symbol/vertices, the same ordering issue already found and fixed for `\spangle`'s own symbol); `\l__spintent_spfig_symbol_tl` is silenced the same way when non-empty; the sequence is then walked by `\__spintent_spfig_points_loop:n`. All of this sits nested inside a subscript (`\c_math_subscript_token`), reached by the shared `mathml_filter` callback's recursive lookup through a noad's `sub` field, not just its `nucleus`.

`\__spintent_invisible_sep:` precedes both branches (a second sibling so the sole `<mrow>` does not collapse into `<math>`.

```
\cs_new_protected:Nn \__spintent_spfig_mathml:n
  {
    \__spintent_invisible_sep: % (luamml issues #26)
    \clist_if_empty:NTF \l__spintent_geo_luaset_points_clist
      {
        \__spintent_spfig_build_simple:n {#1}
        \MathMLintent { \l__spintent_spfig_intent_arg_tl }
          {
            \__spintent_inner_sep:
            \str_if_eq:eeTF {#1} { Afig } { \__spintent_mathcal:n { A } } { \__spintent_mathcal:n
            \tl_if_empty:NF \l__spintent_geo_final_print_tl
              {
                \c_math_subscript_token { \l__spintent_geo_final_print_tl }
              }
          }
      }
      {
        \__spintent_spfig_build_refs:
        \__spintent_spfig_build_concept_vertices:n {#1}
        \MathMLintent { \l__spintent_spfig_intent_arg_tl }
          {
            \__spintent_inner_sep:
            \__spintent_luafun_mrow_intent_arg:nn
```

```
                { } { intent = ":silent" }
              {
                \str_if_eq:eeTF {#1} { Afig } { \__spintent_mathcal:n { A } } { \__spintent_mathcal
              }
              \c_math_subscript_token
                {
                  \tl_if_empty:NF \l__spintent_spfig_symbol_tl
                    {
                      \__spintent_luafun_mrow_intent_arg:nn
                        { } { intent = ":silent" }
                      { \tl_use:N \l__spintent_spfig_symbol_tl }
                    }
                  \__spintent_spfig_points_loop:n {#1}
                }
            }
        }
    }
```

(*End of definition for* `\__spintent_spfig_mathml:n`.)

`\__spintent_spfig_exec:nnn` The function `\__spintent_spfig_exec:nnn` receives the short name of the command in `#1`, the optional keys in `#2`, and the argument in `#3`, delegating the rendering to `\__spintent_spfig_process:nn`.

```
\cs_new_protected:Nn \__spintent_spfig_exec:nnn
  {
    \keys_set:nn { spintent / sp #1 } { #2 }
    \__spintent_spfig_process:nn { #1 } { #3 }
  }
```

(*End of definition for* `\__spintent_spfig_exec:nnn`.)

`\spAfig` `\spPfig` Both public commands `\spAfig` and `\spPfig` enforce math mode, open a local group, and delegate execution to `\__spintent_spfig_exec:nnn`.

```
\NewDocumentCommand \spAfig { O{} d() }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \attribute \l__spintent_scope_attr_int = 1
          \tl_if_novalue:nTF {#2}
            { \__spintent_spfig_exec:nnn { Afig } { #1 } { } }
            { \__spintent_spfig_exec:nnn { Afig } { #1 } { #2 } }
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spAfig } }
  }
\NewDocumentCommand \spPfig { O{} d() }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \attribute \l__spintent_scope_attr_int = 1
          \tl_if_novalue:nTF {#2}
            { \__spintent_spfig_exec:nnn { Pfig } { #1 } { } }
            { \__spintent_spfig_exec:nnn { Pfig } { #1 } { #2 } }
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spPfig } }
  }
```

(*End of definition for* `\spAfig` *and* `\spPfig`. *These functions are documented on page 23.*)

11.45 The commands `\sprad` and `\spdiam`

The user-level commands `\sprad` and `\spdiam` provide a semantic interface for the radius and diameter of a circle. They delegate argument validation to the Lua module `spintent.lua`, thereby constructing a valid intent for `MathCAT`.

11.45.1 Definition of variables for \sprad and \spdiam

\l__spintent_sprad_mode_str
\l__spintent_spdiam_mode_str
\l__spintent_sprad_upper_case_bool
\l__spintent_spdiam_upper_case_bool
\l__spintent_sprad_sub_index_str
\l__spintent_spdiam_sub_index_str

Each pair of variables is declared once for both \sprad and \spdiam via \clist_map_inline:nn.

```
\clist_map_inline:nn { rad , diam }
  {
    \str_new:c  { l__spintent_sp #1 _mode_str        }
    \str_new:c  { l__spintent_sp #1 _sub_index_str   }
    \tl_new:c   { l__spintent_sp #1 _read_arg_tl     }
  }
```

(*End of definition for* \l__spintent_sprad_mode_str *and others.*)

\l__spintent_sp_rad_diam_letra_tl
\l__spintent_sp_rad_diam_word_tl

Scratch variables used by the function __spintent_sp_rad_diam_print:n.

```
\tl_new:N \l__spintent_sp_rad_diam_letra_tl
\tl_new:N \l__spintent_sp_rad_diam_word_tl
```

(*End of definition for* \l__spintent_sp_rad_diam_letra_tl *and* \l__spintent_sp_rad_diam_word_tl.)

11.45.2 Definition of keys for \sprad and \spdiam

read-cmd
upper-case
sub-index
read-sub
read-arg
concept
silent-cmd
print-sym
unknown

We define the shared ⟨*keys*⟩ read-cmd, upper-case, sub-index, read-sub, read-arg, concept, silent-cmd and print-sym.

```
\clist_map_inline:nn { rad , diam }
  {
    \keys_define:nn { spintent / sp #1 }
      {
        read-cmd    .choices:nn = { school , mathcat }
          {
            \int_case:nn { \l_keys_choice_int }
              {
                {1} { \str_set:cn { l__spintent_sp #1 _mode_str } { school  } }
                {2} { \str_set:cn { l__spintent_sp #1 _mode_str } { mathcat } }
              }
          },
        read-cmd    .initial:n  = school,
        read-cmd    .value_required:n = true,
        read-cmd / unknown .code:n =
          \msg_error:nneee { spintent } { unknown-choice }
            { read-cmd } { school,~mathcat } { \exp_not:n {##1} },
        upper-case  .bool_set:c = { l__spintent_sp #1 _upper_case_bool },
        upper-case  .initial:n  = false,
        upper-case  .value_required:n = true,
        sub-index   .code:n =
          { \str_set:cn { l__spintent_sp #1 _sub_index_str } {##1} },
        sub-index   .initial:n  = ,
        sub-index   .value_required:n = true,
        read-sub    .bool_set:c = { l__spintent_sp #1 _read_sub_bool },
        read-sub    .initial:n  = false,
        read-sub    .value_required:n = true,
        read-arg    .code:n =
          { \__spintent_sanitize_affix:cn { l__spintent_sp #1 _read_arg_tl } {##1} },
        read-arg    .initial:n  = ,
        read-arg    .value_required:n = true,
        concept     .bool_set:c = { l__spintent_sp #1 _concept_bool },
        concept     .initial:n  = true,
        concept     .value_required:n = true,
        silent-cmd  .bool_set:c = { l__spintent_sp #1 _silent_cmd_bool },
        silent-cmd  .initial:n  = false,
        silent-cmd  .value_required:n = true,
        print-sym   .bool_set:c = { l__spintent_sp #1 _print_sym_bool },
        print-sym   .initial:n  = false,
        print-sym   .value_required:n = true,
        unknown             .code:n =
          {
            \str_set:Nn \l__spintent_command_name_str { sp #1 }
            \__spintent_unknown_keys_cmd:n {##1}
          },
      }
  }
```

The keys print-sym are defined for both commands for simplicity, but has no effect on \sprad.

(*End of definition for* read-cmd *and others.*)

11.45.3 Internal functions for \sprad and \spdiam

__spintent_sp_rad_diam_print:n

The function __spintent_sp_rad_diam_print:n sets \l__spintent_sp_rad_diam_letra_tl to the bare letter — r/R for \sprad, d/D for \spdiam — per upper-case, and, for \spdiam with print-sym true, silently typesets \diameter first. Receives the short command name in '#1' (rad/diam).

```
\cs_new_protected:Nn \__spintent_sp_rad_diam_print:n
  {
    \bool_if:cTF { l__spintent_sp #1 _upper_case_bool }
      {
        \str_if_eq:eeTF {#1} { rad }
          { \tl_set:Nn \l__spintent_sp_rad_diam_letra_tl { R } }
          { \tl_set:Nn \l__spintent_sp_rad_diam_letra_tl { D } }
      }
      {
        \str_if_eq:eeTF {#1} { rad }
          { \tl_set:Nn \l__spintent_sp_rad_diam_letra_tl { r } }
          { \tl_set:Nn \l__spintent_sp_rad_diam_letra_tl { d } }
      }
    \__spintent_invisible_sep:
    \str_if_eq:eeT {#1} { diam }
      {
        \bool_if:NT \l__spintent_spdiam_print_sym_bool
          { \MathMLintent { :silent } { \diameter } }
      }
```

With silent-cmd true, the intent is _- — the letter alone through the shared queue when there is no sub-index, or letter and subscript together in one native \MathMLintent (subscript silenced) when there is.

```
    \bool_if:cTF { l__spintent_sp #1 _silent_cmd_bool }
      {
        \str_if_empty:cTF { l__spintent_sp #1 _sub_index_str }
          {
            \__spintent_luafun_mrow_intent_arg:nn
              { } { intent = "_-" }
            { \l__spintent_sp_rad_diam_letra_tl }
          }
          {
            \MathMLintent { _- }
              {
                \__spintent_inner_sep:
                \l__spintent_sp_rad_diam_letra_tl
                \c_math_subscript_token
                  {
                    \MathMLintent { :silent }
                      { \str_use:c { l__spintent_sp #1 _sub_index_str } }
                  }
              }
          }
      }
```

Otherwise, with concept true (the default), it verbalizes radio/diámetro (read-arg replaces this word when given), with sub-index, if present, appended right after (read-sub choosing sub-⟨*value*⟩ or ⟨*value*⟩ alone) — the letter itself is never read here. With no sub-index, the shared queue tags the letter alone; with one, letter and subscript are wrapped together in one native \MathMLintent for read-cmd=school (subscript silenced, already spoken), or the shared queue tags the letter alone with the subscript left bare for read-cmd=mathcat.

```
      {
        \bool_if:cTF { l__spintent_sp #1 _concept_bool }
          {
            \tl_if_empty:cTF { l__spintent_sp #1 _read_arg_tl }
              {
                \str_if_eq:eeTF {#1} { rad }
                  { \tl_set:Nn \l__spintent_sp_rad_diam_word_tl { radio } }
                  { \tl_set:Nn \l__spintent_sp_rad_diam_word_tl { diámetro } }
              }
              { \tl_set:Ne \l__spintent_sp_rad_diam_word_tl { \tl_use:c { l__spintent_sp #1 _read_
            \str_if_empty:cTF { l__spintent_sp #1 _sub_index_str }
              {
                \__spintent_luafun_mrow_intent_arg:nn
                  { } { intent = "_ \l__spintent_sp_rad_diam_word_tl - :prefix" }
                { \l__spintent_sp_rad_diam_letra_tl }
              }
              {
```

```
                \str_if_eq:eeTF
                  { \str_use:c { l__spintent_sp #1 _mode_str } } { school }
                  {
                    \MathMLintent
                      {
                        _ \l__spintent_sp_rad_diam_word_tl -
                        \bool_if:cT { l__spintent_sp #1 _read_sub_bool } { sub- }
                        \str_use:c { l__spintent_sp #1 _sub_index_str }
                        :prefix
                      }
                      {
                        \__spintent_inner_sep:
                        \l__spintent_sp_rad_diam_letra_tl
                        \c_math_subscript_token
                          {
                            \MathMLintent { :silent }
                              { \str_use:c { l__spintent_sp #1 _sub_index_str } }
                          }
                      }
                  }
                  {
                    \__spintent_luafun_mrow_intent_arg:nn
                      { } { intent = "_ \l__spintent_sp_rad_diam_word_tl - :prefix" }
                    { \l__spintent_sp_rad_diam_letra_tl }
                    \c_math_subscript_token
                      { \str_use:c { l__spintent_sp #1 _sub_index_str } }
                  }
              }
          }
```

With `concept false`, the letter itself is read instead — but only when `read-cmd` is `school`, built the same way from the letter (with `sub-index`, letter and subscript wrapped together, subscript silenced). With `read-cmd mathcat`, the intent is left empty: `MathCAT` applies its own rules to the letter and its subscript, unmarked by us.

```
          {
            \str_if_empty:cTF { l__spintent_sp #1 _sub_index_str }
              {
                \str_if_eq:eeTF
                  { \str_use:c { l__spintent_sp #1 _mode_str } } { school }
                  {
                    \__spintent_luafun_mrow_intent_arg:nn
                      { } { intent = "_ \l__spintent_sp_rad_diam_letra_tl - :prefix" }
                    { \l__spintent_sp_rad_diam_letra_tl }
                  }
                  { \l__spintent_sp_rad_diam_letra_tl }
              }
              {
                \str_if_eq:eeTF
                  { \str_use:c { l__spintent_sp #1 _mode_str } } { school }
                  {
                    \MathMLintent
                      {
                        _ \l__spintent_sp_rad_diam_letra_tl -
                        \bool_if:cT { l__spintent_sp #1 _read_sub_bool } { sub- }
                        \str_use:c { l__spintent_sp #1 _sub_index_str }
                        :prefix
                      }
                      {
                        \__spintent_inner_sep:
                        \l__spintent_sp_rad_diam_letra_tl
                        \c_math_subscript_token
                          {
                            \MathMLintent { :silent }
                              { \str_use:c { l__spintent_sp #1 _sub_index_str } }
                          }
                      }
                  }
                  {
                    \l__spintent_sp_rad_diam_letra_tl
                    \c_math_subscript_token
                      { \str_use:c { l__spintent_sp #1 _sub_index_str } }
```

```
                    }
                }
            }
        }
    }
```

(*End of definition for* `\__spintent_sp_rad_diam_print:n`.)

`\__spintent_sp_rad_diam_exec:nn`

The function `\__spintent_sp_rad_diam_exec:nn` receives the *short name* of the command passed in '`#1`' and the optional ⟨*keys*⟩ passed in '`#2`' then delegating the rendering to `\__spintent_sp_rad_diam_print:n`.

```
\cs_new_protected:Nn \__spintent_sp_rad_diam_exec:nn
  {
    \keys_set:nn { spintent / sp #1 } {#2}
    \__spintent_sp_rad_diam_print:n {#1}
  }
```

(*End of definition for* `\__spintent_sp_rad_diam_exec:nn`.)

`\sprad` `\spdiam`

Both public commands `\sprad` and `\spdiam` enforces *math mode*, opens a local group, and delegates execution to `\__spintent_sp_rad_diam_exec:nn`.

```
\NewDocumentCommand \sprad { O{} }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \__spintent_sp_rad_diam_exec:nn { rad } {#1}
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \sprad } }
  }
\NewDocumentCommand \spdiam { O{} }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \__spintent_sp_rad_diam_exec:nn { diam } {#1}
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spdiam } }
  }
```

(*End of definition for* `\sprad` *and* `\spdiam`. *These functions are documented on page* ??.)

11.46 The command `\spcirc`

The user command `\spcirc` provides a semantic interface for a circle or a circumference, using the delimited-optional argument syntax `d()` already established by `\spmcd`/`\spmcm` and `\spAfig`/`\spPfig`. The argument holds a required center point and an optional radius, separated by a comma. The radius is never required to declare its own kind via a key: its shape alone decides whether it is a pure number, a measure (number + unit), a bare letter, or a two-letter segment — reusing, respectively, `\spnum`, `\spunit`, and `\spside`/`\spmside` directly as sub-commands, rather than reimplementing any of their parsing.

11.46.1 Definition of variables for `\spcirc`

`\l__spintent_geo_luaset_circ_radio_type_str`
`\l__spintent_geo_luaset_circ_radio_raw_str`

Shared outputs of `\__spintent_luafun_geo_circ_parse_and_set:n`. `\l__spintent_geo_luaset_circ_radi` holds one of `number`, `measure`, `label`, `segment`, or an empty string when no radius was given. `\l__spintent_geo_lu` holds the corresponding raw text, ready to be forwarded to the sub-command that renders it.

```
\str_new:N \l__spintent_geo_luaset_circ_radio_type_str
\str_new:N \l__spintent_geo_luaset_circ_radio_raw_str
```

(*End of definition for* `\l__spintent_geo_luaset_circ_radio_type_str` *and* `\l__spintent_geo_luaset_circ_radio_raw_str`.)

`\l__spintent_spcirc_mode_str`
`\l__spintent_spcirc_prefix_tl`
`\l__spintent_spcirc_print_sym_str`
`\l__spintent_spcirc_read_sty_str`
`\l__spintent_spcirc_radio_sty_str`
`\l__spintent_spcirc_sub_index_str`
`\l__spintent_spcirc_articulo_str`

The variable `\l__spintent_spcirc_mode_str` stores the value set by the key `read-cmd`; the variable `\l__spintent_spcirc_prefix_tl` stores the value set by the key `prefix`. The variable `\l__spintent_spcirc_pr` stores the value set by the key `print-sym` (`usual` or `odot`). The variable `\l__spintent_spcirc_read_sty_str` stores the value set by the key `read-sty` (`circun` or `circle`). The variable `\l__spintent_spcirc_radio_sty_str` stores the value set by the key `radio-sty`, only meaningful when the radius is a segment. The variable `\l__spintent_spcirc_sub_index_str` stores the value set by the key `sub-index`. The variable `\l__spintent_spcirc_articulo_str` is set internally by `\__spintent_spcirc_word_and_article:` to `la` or `el`, matching the grammatical gender of the chosen word.

```
\str_new:N \l__spintent_spcirc_mode_str
\tl_new:N  \l__spintent_spcirc_prefix_tl
```

```
\str_new:N \l__spintent_spcirc_print_sym_str
\str_new:N \l__spintent_spcirc_read_sty_str
\str_new:N \l__spintent_spcirc_radio_sty_str
\str_new:N \l__spintent_spcirc_sub_index_str
\str_new:N \l__spintent_spcirc_articulo_str
\tl_new:N  \l__spintent_spcirc_read_arg_tl
```

(*End of definition for* `\l__spintent_spcirc_mode_str` *and others.*)

`\l__spintent_spcirc_word_tl` Scratch variables used by `\__spintent_spcirc_process:n` to assemble the chosen word (`circunferencia`/`círculo` or the `read-arg` override) and the final literal `intent` string.

```
\tl_new:N  \l__spintent_spcirc_word_tl
```

(*End of definition for* `\l__spintent_spcirc_word_tl`.)

11.46.2 Definition of error messages for `\spcirc`

`spcirc-invalid-arg` Raised when the argument does not match a single center point, optionally followed by a comma and a valid radius shape.

```
\msg_new:nnnn { spintent } { spcirc-invalid-arg }
  {
    Invalid~argument~format~in~\token_to_str:N \spcirc:~'#1'~\msg_line_context:.
  }
  {
    See~the~documentation~of~\token_to_str:N \spcirc~for~the~accepted~argument~forms.
  }
```

(*End of definition for* `spcirc-invalid-arg`.)

11.46.3 Definition of keys for `\spcirc`

`read-cmd` `prefix` `print-sym` `read-sty` `diameter` `radio-sty` `sub-index` `read-sub` `read-arg` `silent-cmd` `concept` `spside` `spmside` `unknown`

The key `print-sym` chooses the visual symbol: `usual` for a plain `C`, `odot` (default) for `\odot`. The key `read-sty` chooses the verbalized word and its article: `circun` (default) for *«la circunferencia»*, `circle` for *«el círculo»*. The key `radio-sty` chooses, only when the radius is a two-letter segment, whether it is rendered as the *object* (`side`, via `\spside`) or its *measure* (`mside`, default, via `\spmside`). The key `sub-index` sets a subscript on the symbol; the key `read-sub` controls whether it is read with the word «sub» or as a plain number, matching the same convention as `\spAfig`.

```
\keys_define:nn { spintent / spcirc }
  {
    read-cmd     .choices:nn = { school , mathcat }
      {
        \int_case:nn { \l_keys_choice_int }
          {
            {1} { \str_set:Nn \l__spintent_spcirc_mode_str { school  } }
            {2} { \str_set:Nn \l__spintent_spcirc_mode_str { mathcat } }
          }
      },
    read-cmd     .initial:n = school,
    read-cmd     .value_required:n = true,
    prefix       .code:n =
      { \__spintent_sanitize_affix:Nn \l__spintent_spcirc_prefix_tl {#1} },
    prefix       .initial:n = ,
    prefix       .value_required:n = true,
    print-sym    .choices:nn = { usual , odot }
      {
        \int_case:nn { \l_keys_choice_int }
          {
            {1} { \str_set:Nn \l__spintent_spcirc_print_sym_str { usual } }
            {2} { \str_set:Nn \l__spintent_spcirc_print_sym_str { odot  } }
          }
      },
    print-sym    .initial:n = odot,
    print-sym    .value_required:n = true,
    read-sty     .choices:nn = { circun , circle }
      {
        \int_case:nn { \l_keys_choice_int }
          {
            {1} { \str_set:Nn \l__spintent_spcirc_read_sty_str { circun } }
            {2} { \str_set:Nn \l__spintent_spcirc_read_sty_str { circle } }
          }
      },
    read-sty     .initial:n = circun,
    read-sty     .value_required:n = true,
```

```
    diameter      .bool_set:N = \l__spintent_spcirc_diameter_bool,
    diameter      .initial:n  = false,
    diameter      .value_required:n = true,
    radio-sty     .choices:nn = { side , mside }
      {
        \int_case:nn { \l_keys_choice_int }
          {
            {1} { \str_set:Nn \l__spintent_spcirc_radio_sty_str { side  } }
            {2} { \str_set:Nn \l__spintent_spcirc_radio_sty_str { mside } }
          }
      },
    radio-sty     .initial:n = mside,
    radio-sty     .value_required:n = true,
    sub-index     .code:n =
      { \str_set:Nn \l__spintent_spcirc_sub_index_str {#1} },
    sub-index     .initial:n = ,
    sub-index     .value_required:n = true,
    read-sub      .bool_set:N = \l__spintent_spcirc_read_sub_bool,
    read-sub      .initial:n = false,
    read-sub      .value_required:n = true,
    read-arg      .code:n =
      { \__spintent_sanitize_affix:Nn \l__spintent_spcirc_read_arg_tl {#1} },
    read-arg      .initial:n = ,
    read-arg      .value_required:n = true,
    silent-cmd    .bool_set:N = \l__spintent_spcirc_silent_cmd_bool,
    silent-cmd    .initial:n = false,
    silent-cmd    .value_required:n = true,
    concept       .bool_set:N = \l__spintent_spcirc_concept_bool,
    concept       .initial:n  = true,
    concept       .value_required:n = true,
    spside        .meta:nn = { spintent / spside  } { #1 },
    spside        .value_required:n = true,
    spmside       .meta:nn = { spintent / spmside } { #1 },
    spmside       .value_required:n = true,
    unknown       .code:n =
      {
        \str_set:Nn \l__spintent_command_name_str { spcirc }
        \__spintent_unknown_keys_cmd:n {#1}
      },
  }
```

(*End of definition for* `read-cmd` *and others.*)

11.46.4 Internal functions for `\spcirc`

`\l__spintent_spcirc_has_arg_bool`
`\l__spintent_spcirc_intent_tl`
`\l__spintent_spcirc_intent_arg_tl`
`\l__spintent_spcirc_connector_tl`
`\l__spintent_spcirc_connector_arg_tl`
`\l__spintent_spcirc_radio_arg_tl`

`\l__spintent_spcirc_has_arg_bool` records whether the `d()` argument was given at all (`\spcirc` with no `(...)` at all has neither a center nor a radius). `\l__spintent_spcirc_intent_tl` holds the word (`circunferencia` or `círculo`, with an optional `sub-index` suffix) paired with the symbol. `\l__spintent_spcirc_connector_tl` holds `de-centro-en`, paired with the `(center)` group; the center is read through `\__spintent_mathmlarg_point:nnn`, referenced as `$pt`, not concatenated into either word — it can be subscripted or primed (e.g. `O_{1}`, as in the Euler circle) without breaking anything. The `_arg_tl` variables hold each word's final wrapped form; a third, `\l__spintent_spcirc_radio_arg_tl`, is spoken directly on the `,` delimiter after the center, replacing its own reading rather than pairing with a group.

```
\bool_new:N \l__spintent_spcirc_has_arg_bool
\tl_new:N   \l__spintent_spcirc_intent_tl
\tl_new:N   \l__spintent_spcirc_intent_arg_tl
\tl_new:N   \l__spintent_spcirc_connector_tl
\tl_new:N   \l__spintent_spcirc_connector_arg_tl
\tl_new:N   \l__spintent_spcirc_radio_arg_tl
```

(*End of definition for* `\l__spintent_spcirc_has_arg_bool` *and others.*)

`\__spintent_spcirc_sym:`

Typesets `\bigodot` or the plain letter `C`, according to `print-sym`.

```
\cs_new_protected:Nn \__spintent_spcirc_sym:
  {
    \str_if_eq:VnTF \l__spintent_spcirc_print_sym_str { odot }
      {
        \group_begin:
          \textstyle\bigodot
        \group_end:
      }
      { C }
  }
```

(*End of definition for* `\__spintent_spcirc_sym:`.)

`\__spintent_spcirc_word_and_article:` Sets `\l__spintent_spcirc_word_tl` to `read-arg` or `circunferencia`/`círculo` from `read-sty`, and `\l__spintent_spcirc_articulo_str` to the matching article, which always follows `read-sty` even when `read-arg` overrides the word.

```
\cs_new_protected:Nn \__spintent_spcirc_word_and_article:
  {
    \tl_if_empty:NTF \l__spintent_spcirc_read_arg_tl
      {
        \str_if_eq:VnTF \l__spintent_spcirc_read_sty_str { circun }
          {
            \tl_set:Nn \l__spintent_spcirc_word_tl { circunferencia }
            \str_set:Nn \l__spintent_spcirc_articulo_str { la }
          }
          {
            \tl_set:Nn \l__spintent_spcirc_word_tl { círculo }
            \str_set:Nn \l__spintent_spcirc_articulo_str { el }
          }
      }
      {
        \tl_set:Nn \l__spintent_spcirc_word_tl { \l__spintent_spcirc_read_arg_tl }
        \str_if_eq:VnTF \l__spintent_spcirc_read_sty_str { circun }
          { \str_set:Nn \l__spintent_spcirc_articulo_str { la } }
          { \str_set:Nn \l__spintent_spcirc_articulo_str { el } }
      }
  }
```

(*End of definition for* `\__spintent_spcirc_word_and_article:`.)

`\__spintent_spcirc_radio_print:` Dispatches on `\l__spintent_geo_luaset_circ_radio_type_str`, reusing `\spnum`/`\spunit`/`\spside`/`\spmside` directly rather than reimplementing any parsing.

```
\cs_new_protected:Nn \__spintent_spcirc_radio_print:
  {
    \str_case:Vn \l__spintent_geo_luaset_circ_radio_type_str
      {
        { number  } { \exp_args:Ne \spnum { \l__spintent_geo_luaset_circ_radio_raw_str } }
        { measure } { \exp_args:Ne \spunit { \l__spintent_geo_luaset_circ_radio_raw_str } }
        { label   } { \l__spintent_geo_luaset_circ_radio_raw_str }
        { segment }
          {
            \str_if_eq:VnTF \l__spintent_spcirc_radio_sty_str { side }
              { \exp_args:Ne \spside  { \l__spintent_geo_luaset_circ_radio_raw_str } }
              { \exp_args:Ne \spmside { \l__spintent_geo_luaset_circ_radio_raw_str } }
          }
      }
  }
```

(*End of definition for* `\__spintent_spcirc_radio_print:`.)

`\__spintent_spcirc_build_concept:` Builds two words: `\l__spintent_spcirc_intent_tl` (via `\__spintent_spcirc_word_and_article:`, plus an optional `sub-index` suffix), paired with the symbol, and `\l__spintent_spcirc_connector_tl` (`de-centro-en`), paired with the `(center)` group — built even when `\l__spintent_spcirc_has_arg_bool` is `false`, though `\__spintent_spcirc_mathml:` never uses it in that case. Neither carries the center's own text: it is read separately, by reference, through `\__spintent_mathmlarg_point:nnn` — so a subscripted or primed center (e.g. `O_{1}`, as in the Euler circle) is never an issue.

```
\cs_new_protected:Nn \__spintent_spcirc_build_concept:
  {
    \__spintent_spcirc_word_and_article:
    \tl_set:Ne \l__spintent_spcirc_intent_tl { \l__spintent_spcirc_word_tl }
    \str_if_empty:NF \l__spintent_spcirc_sub_index_str
      {
        \tl_put_right:Ne \l__spintent_spcirc_intent_tl
          {
            -
            \bool_if:NT \l__spintent_spcirc_read_sub_bool { sub - }
            \l__spintent_spcirc_sub_index_str
          }
      }
    \tl_set:Nn \l__spintent_spcirc_connector_tl { de-centro-en }
```

silent-cmd true overrides \l__spintent_spcirc_intent_arg_tl/\l__spintent_spcirc_radio_arg_tl with _- and \l__spintent_spcirc_connector_arg_tl with _($pt) (the center is always read), regardless of read-arg.

```
\bool_if:NTF \l__spintent_spcirc_silent_cmd_bool
  {
    \tl_set:Nn \l__spintent_spcirc_intent_arg_tl     { _- }
    \tl_set:Ne \l__spintent_spcirc_connector_arg_tl  { _( $ pt ) }
    \tl_set:Nn \l__spintent_spcirc_radio_arg_tl      { _- }
  }
```

With concept true, \l__spintent_spcirc_intent_arg_tl is \l__spintent_spcirc_intent_tl:prefix (bare — the group it wraps has nothing to reference, only a silenced symbol), prefixed with prefix when given; \l__spintent_spcirc_connector_arg_tl is \l__spintent_spcirc_connector_tl:prefix($pt); and \l__spintent_spcirc_radio_arg_tl — spoken directly on the , delimiter, replacing its own reading rather than pairing with any group — is the literal _y-radio-, or _y-diámetro- when diameter is true, read as its own phrase right before __spintent_spcirc_radio_print:'s independent reading follows as the next sibling.

```
  {
    \bool_if:NTF \l__spintent_spcirc_concept_bool
      {
        \tl_set:Ne \l__spintent_spcirc_intent_arg_tl
          {
            \tl_if_empty:NF \l__spintent_spcirc_prefix_tl
              { \tl_use:N \l__spintent_spcirc_prefix_tl - }
            \l__spintent_spcirc_intent_tl :prefix
          }
        \tl_set:Ne \l__spintent_spcirc_connector_arg_tl
          { \l__spintent_spcirc_connector_tl :prefix( $ pt ) }
        \bool_if:NTF \l__spintent_spcirc_diameter_bool
          { \tl_set:Nn \l__spintent_spcirc_radio_arg_tl { _y-diámetro- } }
          { \tl_set:Nn \l__spintent_spcirc_radio_arg_tl { _y-radio- } }
      }
```

With concept false and read-arg empty, the same _-/_($pt) silencing applies as silent-cmd; with read-arg given, none of the three is silenced instead — the connector and y-radio/y-diámetro words are fixed structural pieces, not customized by read-arg, but once a real word is being spoken there is nothing left to silence around it (same convention as __spintent_spside_process:nn).

```
      {
        \tl_if_empty:NTF \l__spintent_spcirc_read_arg_tl
          {
            \tl_set:Nn \l__spintent_spcirc_intent_arg_tl    { _- }
            \tl_set:Ne \l__spintent_spcirc_connector_arg_tl { _( $ pt ) }
            \tl_set:Nn \l__spintent_spcirc_radio_arg_tl     { _- }
          }
          {
            \tl_set:Ne \l__spintent_spcirc_intent_arg_tl { \l__spintent_spcirc_intent_tl }
            \tl_set:Ne \l__spintent_spcirc_connector_arg_tl
              { \l__spintent_spcirc_connector_tl :prefix( $ pt ) }
            \bool_if:NTF \l__spintent_spcirc_diameter_bool
              { \tl_set:Nn \l__spintent_spcirc_radio_arg_tl { _y-diámetro- } }
              { \tl_set:Nn \l__spintent_spcirc_radio_arg_tl { _y-radio- } }
          }
      }
  }
}
```

(*End of definition for* __spintent_spcirc_build_concept:.)

__spintent_spcirc_process:n

Receives the d() argument in '#1', an explicit empty group when omitted (\spcirc itself converts -NoValue- to { } before calling __spintent_spcirc_exec:nn) — \l__spintent_spcirc_has_arg_bool set to false in that case, and __spintent_luafun_geo_circ_parse_and_set:n never called at all — there is nothing to parse. Otherwise calls it, reports spcirc-invalid-arg on failure, then delegates to __spintent_spcirc_mathml:, which calls __spintent_spcirc_build_concept: itself.

```
\cs_new_protected:Nn \__spintent_spcirc_process:n
  {
    \tl_if_empty:nTF {#1}
      {
        \bool_set_false:N \l__spintent_spcirc_has_arg_bool
      }
      {
        \bool_set_true:N \l__spintent_spcirc_has_arg_bool
```

```
        \__spintent_luafun_geo_circ_parse_and_set:n { \exp_not:n {#1} }
        \str_if_eq:VnT \l__spintent_geo_luaset_error_str { true }
          {
            \msg_error:nne { spintent } { spcirc-invalid-arg } { \exp_not:n {#1} }
          }
      }
    \__spintent_spcirc_mathml:
  }
```

(*End of definition for* `\__spintent_spcirc_process:n`.)

`\__spintent_spcirc_mathml:` Calls `\__spintent_spcirc_build_concept:`, then wraps the symbol (`\l__spintent_spcirc_intent_arg_tl`) in its own native `\MathMLintent` — this alone is the whole result when `\l__spintent_spcirc_has_arg_bool` is `false` (`\spcirc` with no `(...)` at all: no center, no radius).
Otherwise, the `(`center`)` group (`\l__spintent_spcirc_connector_arg_tl`) follows as a separate sibling, never nested inside the symbol's own `\MathMLintent` — or `MathCAT` falls back to its own structural reading of the underlying `<mover>`/`<msub>` instead, the same problem already found for `\spangle`'s symbol and `\sparc`'s accent. When a radius is present, the `,` delimiter after the center carries `\l__spintent_spcirc_radio_arg_` directly (replacing its own reading, not paired with any group), and `\__spintent_spcirc_radio_print:` follows as an ordinary sibling right after — wrapping it in another `\MathMLintent` would lose the reading of whatever it produces (itself a native `\MathMLintent`, from `\spnum`/`\spunit`, nested inside another one does not compose).
The symbol and, when `sub-index` is given, its subscript number are each silenced with `\__spintent_luafun_mrow_in` *separately* — one queue entry per leaf. The two are reached differently by the shared `mathml_filter` callback's recursive lookup (the symbol via `nucleus`, the subscript via `sub`), so a single entry covering both leaves nothing registered for the second one, which the recursion then assigns whatever comes next in the queue instead — shifting every claim after it by one. The `(` delimiter is silenced the same way, but *without* the usual `{...}` around the character: `luamml` only keeps a delimiter's own math class (giving `<mo>` instead of `<mi>`, needed for `MathCAT` to read the group at all) when it is not itself sitting inside a brace group.
The final `)` — following `\__spintent_spcirc_radio_print:`, whose own output the shared queue does not reliably skip over — is typeset as `\rparen` inside its own native `\MathMLintent{:silent}` instead of a bare `)`: `\rparen` keeps its own math class through the wrapping, unlike a bare delimiter character, and this sits outside the shared queue entirely, so nothing in between can claim an entry meant for it.
The center itself is read through `\__spintent_mathmlarg_point:nnn`, referenced as `$pt`.

```
\cs_new_protected:Nn \__spintent_spcirc_mathml:
  {
    \__spintent_invisible_sep: % (luamml issues #26)
    \__spintent_spcirc_build_concept:
    \MathMLintent { \l__spintent_spcirc_intent_arg_tl }
      {
        \__spintent_inner_sep:
        \__spintent_luafun_mrow_intent_arg:nn
          { } { intent = ":silent" }
        { \__spintent_spcirc_sym: }
        \str_if_empty:NF \l__spintent_spcirc_sub_index_str
          {
            \c_math_subscript_token
              {
                \__spintent_luafun_mrow_intent_arg:nn
                  { } { intent = ":silent" }
                { \l__spintent_spcirc_sub_index_str }
              }
          }
      }
    \bool_if:NT \l__spintent_spcirc_has_arg_bool
      {
        \str_if_empty:NTF \l__spintent_geo_luaset_circ_radio_type_str
          {
            \MathMLintent { \l__spintent_spcirc_connector_arg_tl }
              {
                \__spintent_inner_sep:
                \__spintent_luafun_mrow_intent_arg:nn { } { intent = ":silent" } (
                \__spintent_mathmlarg_point:nnV { pt } { spcirc } \l__spintent_geo_luaset_points_
                \__spintent_luafun_mrow_intent_arg:nn { } { intent = ":silent" } )
              }
          }
          {
            \MathMLintent { \l__spintent_spcirc_connector_arg_tl }
              {
```

```
                \__spintent_inner_sep:
                \__spintent_luafun_mrow_intent_arg:nn { } { intent = ":silent" } (
                \__spintent_mathmlarg_point:nnV { pt } { spcirc } \l__spintent_geo_luaset_points_c
              }
            \MathMLintent { \l__spintent_spcirc_radio_arg_tl } { , }
            \__spintent_spcirc_radio_print:
            \MathMLintent { :silent } { \rparen }
          }
      }
  }
```

(*End of definition for* `\__spintent_spcirc_mathml:`.)

`\__spintent_spcirc_exec:nn`

The function `\__spintent_spcirc_exec:nn` receives the optional keys in `#1` and the argument in `#2`, delegating the rendering to `\__spintent_spcirc_print:n`.

```
\cs_new_protected:Nn \__spintent_spcirc_exec:nn
  {
    \keys_set:nn { spintent / spcirc } { #1 }
    \__spintent_spcirc_process:n { #2 }
  }
```

(*End of definition for* `\__spintent_spcirc_exec:nn`.)

`\spcirc`

The public command `\spcirc` enforces math mode, opens a local group, and delegates execution to `\__spintent_spcirc_exec:nn`. It follows the same `d()`-argument pattern as `\spmcd`/`\spmcm` and `\spAfig`/`\spPfig`; an omitted radius forwards an explicit empty group rather than the `-NoValue-` marker.

```
\NewDocumentCommand \spcirc { O{} d() }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \attribute \l__spintent_scope_attr_int = 1
          \tl_if_novalue:nTF {#2}
            { \__spintent_spcirc_exec:nn { #1 } { } }
            { \__spintent_spcirc_exec:nn { #1 } { #2 } }
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spcirc } }
  }
```

(*End of definition for* `\spcirc`. *This function is documented on page* ??.)

11.47 The commands `\spAcirc` and `\spPcirc`

The user commands `\spAcirc` and `\spPcirc` typeset the area and the perimeter of a circle or a circumference. The argument follows `\spcirc`'s own point-and-radius grammar for a real center (with an optional radius); when no comma is present and no valid center is found, it falls back to a bare number/letter shortcut, reusing the same Lua entry point as `\spAfig`/`\spPfig`. A table letter matched by that fallback is rejected, since it has no meaning for a circle.

11.47.1 Definition of variables for `\spAcirc` and `\spPcirc`

`\l__spintent_spAcirc_mode_str`
`\l__spintent_spPcirc_mode_str`
`\l__spintent_spAcirc_print_sym_str`
`\l__spintent_spPcirc_print_sym_str`
`\l__spintent_spAcirc_read_sty_str`
`\l__spintent_spPcirc_read_sty_str`
`\l__spintent_spAcirc_radio_sty_str`
`\l__spintent_spPcirc_radio_sty_str`
`\l__spintent_spAcirc_sub_index_str`
`\l__spintent_spPcirc_sub_index_str`
`\l__spintent_spAcirc_prefix_tl`
`\l__spintent_spPcirc_prefix_tl`
`\l__spintent_spAcirc_read_arg_tl`
`\l__spintent_spPcirc_read_arg_tl`

These variables store the value set by each corresponding key: `read-cmd`, `print-sym`, `read-sty`, `radio-sty`, `sub-index`, `prefix` and `read-arg`, respectively.

```
\clist_map_inline:nn { Acirc , Pcirc }
  {
    \str_new:c { l__spintent_sp #1 _mode_str      }
    \str_new:c { l__spintent_sp #1 _print_sym_str }
    \str_new:c { l__spintent_sp #1 _read_sty_str  }
    \str_new:c { l__spintent_sp #1 _radio_sty_str }
    \str_new:c { l__spintent_sp #1 _sub_index_str }
    \tl_new:c  { l__spintent_sp #1 _prefix_tl     }
    \tl_new:c  { l__spintent_sp #1 _read_arg_tl   }
  }
```

(*End of definition for* `\l__spintent_spAcirc_mode_str` *and others.*)

`\l__spintent_geo_circfig_mode_str`

The variable `\l__spintent_geo_circfig_mode_str` stores whether the argument was resolved as a real circle (`circle`) or as the bare number/letter shortcut (`bare`).

```
\str_new:N \l__spintent_geo_circfig_mode_str
```

(*End of definition for* `\l__spintent_geo_circfig_mode_str`.)

11.47.2 Definition of error messages for \spAcirc and \spPcirc

spAcirc-invalid-arg
spPcirc-invalid-arg

The error message shown when the argument does not match any of the accepted forms: empty, a center point (optionally followed by a comma and a radius), or a bare number/letter.

```
\clist_map_inline:nn { Acirc , Pcirc }
  {
    \msg_new:nnnn { spintent } { sp #1 -invalid-arg }
      {
        Invalid~argument~in~##1:~'##2'~\msg_line_context:.
      }
      {
        See~the~documentation~of~##1~for~the~accepted~argument.
      }
```

(*End of definition for* spAcirc-invalid-arg *and* spPcirc-invalid-arg.)

11.47.3 Definition of keys for \spAcirc and \spPcirc

read-cmd
prefix
print-sym
read-sty
concept
sub-index
read-sub
read-arg
silent-cmd
unknown

We define the shared ⟨*keys*⟩ read-cmd, prefix, print-sym, read-sty, concept, sub-index, read-sub, read-arg and silent-cmd.

```
    \keys_define:nn { spintent / sp #1 }
      {
        read-cmd            .choices:nn = { school , mathcat }
          {
            \int_case:nn { \l_keys_choice_int }
              {
                {1} { \str_set:cn { l__spintent_sp #1 _mode_str } { school  } }
                {2} { \str_set:cn { l__spintent_sp #1 _mode_str } { mathcat } }
              }
          },
        read-cmd            .initial:n  = school,
        read-cmd            .value_required:n = true,
        read-cmd / unknown  .code:n =
          \msg_error:nneee { spintent } { unknown-choice }
            { read-cmd } { school,~mathcat } { \exp_not:n {##1} },
        prefix              .code:n =
          { \__spintent_sanitize_affix:cn { l__spintent_sp #1 _prefix_tl } {##1} },
        prefix              .initial:n  = ,
        prefix              .value_required:n = true,
        print-sym           .choices:nn = { usual , odot }
          {
            \int_case:nn { \l_keys_choice_int }
              {
                {1} { \str_set:cn { l__spintent_sp #1 _print_sym_str } { usual } }
                {2} { \str_set:cn { l__spintent_sp #1 _print_sym_str } { odot  } }
              }
          },
        print-sym           .initial:n  = odot,
        print-sym           .value_required:n = true,
        print-sym / unknown .code:n =
          \msg_error:nneee { spintent } { unknown-choice }
            { print-sym } { usual,~odot } { \exp_not:n {##1} },
        read-sty            .choices:nn = { circun , circle }
          {
            \int_case:nn { \l_keys_choice_int }
              {
                {1} { \str_set:cn { l__spintent_sp #1 _read_sty_str } { circun } }
                {2} { \str_set:cn { l__spintent_sp #1 _read_sty_str } { circle } }
              }
          },
        read-sty            .initial:n  = circun,
        read-sty            .value_required:n = true,
        concept             .bool_set:c = { l__spintent_sp #1 _concept_bool },
        concept             .initial:n  = true,
        concept             .value_required:n = true,
        sub-index           .code:n =
          { \str_set:cn { l__spintent_sp #1 _sub_index_str } {##1} },
        sub-index           .initial:n  = ,
        sub-index           .value_required:n = true,
        read-sub            .bool_set:c = { l__spintent_sp #1 _read_sub_bool },
        read-sub            .initial:n  = false,
        read-sub            .value_required:n = true,
```

```
      read-arg            .code:n =
        { \__spintent_sanitize_affix:cn { l__spintent_sp #1 _read_arg_tl } {##1} },
      read-arg            .initial:n  = ,
      read-arg            .value_required:n = true,
      silent-cmd          .bool_set:c = { l__spintent_sp #1 _silent_cmd_bool },
      silent-cmd          .initial:n  = false,
      silent-cmd          .value_required:n = true,
      unknown             .code:n =
        {
          \str_set:Nn \l__spintent_command_name_str { sp #1 }
          \__spintent_unknown_keys_cmd:n {##1}
        },
    }
  }
```

(*End of definition for* `read-cmd` *and others.*)

`\l__spintent_spcircfig_short_bool`
`\l__spintent_spcircfig_op_tl`
`\l__spintent_spcircfig_intent_tl`
`\l__spintent_spcircfig_intent_arg_tl`

`\l__spintent_spcircfig_short_bool` records whether the argument resolved as the short number/letter form (via `\__spintent_luafun_geo_afig_parse_and_set:n`/`\__spintent_luafun_geo_pfig_parse_and_se` the same entry point `\spAfig`/`\spPfig` use) rather than a real circle. `\l__spintent_spcircfig_op_tl` holds `área` or `perímetro`. `\l__spintent_spcircfig_intent_tl` holds the full spoken phrase: `\l__spintent_spc` (via `\__spintent_spcircfig_del_word:n`: the word, contracted with `del-`/`de-la-`, or `read-arg`), an optional `sub-index` suffix, and, when a center is given, `-con-centro-en` — the center itself is never concatenated into it: it is read separately, by reference, through `\__spintent_mathmlarg_point:nnn`, so a subscripted or primed center (e.g. `O_{1}`, as in the Euler circle) is never an issue.

```
\bool_new:N \l__spintent_spcircfig_short_bool
\tl_new:N   \l__spintent_spcircfig_op_tl
\tl_new:N   \l__spintent_spcircfig_del_word_tl
\tl_new:N   \l__spintent_spcircfig_intent_tl
\tl_new:N   \l__spintent_spcircfig_intent_arg_tl
```

(*End of definition for* `\l__spintent_spcircfig_short_bool` *and others.*)

`\__spintent_spcircfig_sym:n`

Typesets `\bigodot` or the plain letter `C`, according to `print-sym`. Receives the short command name in '`#1`'.

```
\cs_new_protected:Nn \__spintent_spcircfig_sym:n
  {
    \str_if_eq:vnTF { l__spintent_sp #1 _print_sym_str } { odot }
      { \bigodot }
      { C }
  }
```

(*End of definition for* `\__spintent_spcircfig_sym:n`.)

`\__spintent_spcircfig_word_and_article:n`

Sets `\l__spintent_spcirc_word_tl` to `read-arg` or `circunferencia`/`círculo` from `read-sty`, and `\l__spintent_spcirc_articulo_str` to the matching article, which always follows `read-sty` even when `read-arg` overrides the word. Receives the short command name in '`#1`'.

```
\cs_new_protected:Nn \__spintent_spcircfig_word_and_article:n
  {
    \str_if_eq:vnTF { l__spintent_sp #1 _read_sty_str } { circun }
      { \str_set:Nn \l__spintent_spcirc_articulo_str { la } }
      { \str_set:Nn \l__spintent_spcirc_articulo_str { el } }
    \tl_if_empty:cTF { l__spintent_sp #1 _read_arg_tl }
      {
        \str_if_eq:vnTF { l__spintent_sp #1 _read_sty_str } { circun }
          { \tl_set:Nn \l__spintent_spcirc_word_tl { circunferencia } }
          { \tl_set:Nn \l__spintent_spcirc_word_tl { círculo } }
      }
      {
        \tl_set:Ne \l__spintent_spcirc_word_tl
          {
            \tl_use:c { l__spintent_sp #1 _read_arg_tl }
          }
      }
  }
```

(*End of definition for* `\__spintent_spcircfig_word_and_article:n`.)

`\__spintent_spcircfig_del_word:n`

Sets `\l__spintent_spcircfig_del_word_tl` to `\__spintent_spcircfig_word_and_article:n`'s word, prefixed with `del-` or `de-la-` — contracted, not `de-el-`, since a screen reader does not contract that on its own. Receives the short command name in '`#1`'.

```
\cs_new_protected:Nn \__spintent_spcircfig_del_word:n
```

```
{
  \__spintent_spcircfig_word_and_article:n {#1}
  \str_if_eq:VnTF \l__spintent_spcirc_articulo_str { el }
    {
      \tl_set:Ne \l__spintent_spcircfig_del_word_tl { del- \l__spintent_spcirc_word_tl }
    }
    {
      \tl_set:Ne \l__spintent_spcircfig_del_word_tl { de-la- \l__spintent_spcirc_word_tl }
    }
}
```

(*End of definition for* `\__spintent_spcircfig_del_word:n`.)

`\__spintent_spcircfig_build_short:n` Handles the short number/letter case: builds `\l__spintent_spcircfig_intent_arg_tl` as `\l__spintent_spcirc` *ber word*⟩, wrapped in bare `:prefix` (no reference — there is nothing to point to), or `_-` when `silent-cmd` is `true` or `concept` is `false`. Receives the short command name in '`#1`'.

```
\cs_new_protected:Nn \__spintent_spcircfig_build_short:n
  {
    \bool_if:cTF { l__spintent_sp #1 _silent_cmd_bool }
      {
        \tl_set:Nn \l__spintent_spcircfig_intent_arg_tl { _- }
      }
      {
        \bool_if:cTF { l__spintent_sp #1 _concept_bool }
          {
            \__spintent_spcircfig_del_word:n {#1}
            \tl_set:Ne \l__spintent_spcircfig_intent_arg_tl
              {
                \l__spintent_spcircfig_op_tl - \l__spintent_spcircfig_del_word_tl -
                \bool_if:cT { l__spintent_sp #1 _read_sub_bool } { sub- }
                \l__spintent_geo_luaset_sub_spoken_str :prefix
              }
          }
          {
            \tl_set:Nn \l__spintent_spcircfig_intent_arg_tl { _- }
          }
      }
  }
```

(*End of definition for* `\__spintent_spcircfig_build_short:n`.)

`\__spintent_spcircfig_finish_intent:nn` Appends an optional `sub-index` suffix and, when `#2` is non-empty, `-con-centro-en` to `\l__spintent_spcircfig_i` — shared by both branches of `\__spintent_spcircfig_build_concept:nn`, after either one has set the word itself. Receives the short command name in '`#1`' and the argument in '`#2`'.

```
\cs_new_protected:Nn \__spintent_spcircfig_finish_intent:nn
  {
    \str_if_empty:cF { l__spintent_sp #1 _sub_index_str }
      {
        \tl_put_right:Ne \l__spintent_spcircfig_intent_tl
          {
            -
            \bool_if:cT { l__spintent_sp #1 _read_sub_bool } { sub- }
            \str_use:c { l__spintent_sp #1 _sub_index_str }
          }
      }
    \tl_if_empty:nF {#2}
      {
        \tl_put_right:Nn \l__spintent_spcircfig_intent_tl { -con-centro-en }
      }
  }
```

(*End of definition for* `\__spintent_spcircfig_finish_intent:nn`.)

`\__spintent_spcircfig_build_concept:nn` Builds `\l__spintent_spcircfig_intent_arg_tl`. `_-` when `silent-cmd` is `true`, or when `false` and `concept` is `false` with `read-arg` empty (`_($pt)` instead when `#2` is non-empty — the center is still read either way). Otherwise `\l__spintent_spcircfig_intent_tl:prefix` — with `($pt)` appended when `#2` is non-empty — prefixed with `prefix` when given, where `\l__spintent_spcircfig_intent_tl` is: with `concept true`, `\l__spintent_spcircfig_op_tl- \l__spintent_spcircfig_del_word_tl` (via `\__spintent_spcircfig_del_word:n`: the word, contracted with `del-`/`de-la-`, or `read-arg` the same way); with `false` and `read-arg` given, `read-arg` verbatim instead, with `\l__spintent_spcircfig_op_tl` dropped entirely — `área`/ `perímetro` is itself the concept `concept` toggles, not the `-con-centro-en` that

follows, which always stays (it identifies which figure, not the concept). No automatic article either — the word (and any article it needs) is the user's to write, same convention as __spintent_spcirc_process:n. Either way, an optional sub-index suffix and, when #2 is non-empty, -con-centro-en follow. Receives the short command name in '#1' and the argument in '#2'.

```
\cs_new_protected:Nn \__spintent_spcircfig_build_concept:nn
  {
    \bool_if:cTF { l__spintent_sp #1 _silent_cmd_bool }
      {
        \tl_if_empty:nTF {#2}
          { \tl_set:Nn \l__spintent_spcircfig_intent_arg_tl { _- } }
          { \tl_set:Ne \l__spintent_spcircfig_intent_arg_tl { _( $ pt ) } }
      }
      {
        \bool_if:cTF { l__spintent_sp #1 _concept_bool }
          {
            \__spintent_spcircfig_del_word:n {#1}
            \tl_set:Ne \l__spintent_spcircfig_intent_tl
              {
                \l__spintent_spcircfig_op_tl - \l__spintent_spcircfig_del_word_tl
              }
            \__spintent_spcircfig_finish_intent:nn {#1} {#2}
            \tl_set:Ne \l__spintent_spcircfig_intent_arg_tl
              {
                \tl_if_empty:cF { l__spintent_sp #1 _prefix_tl }
                  {
                    \tl_use:c { l__spintent_sp #1 _prefix_tl } -
                  }
                \l__spintent_spcircfig_intent_tl :prefix
                \tl_if_empty:nF {#2} { ( $ pt ) }
              }
          }
          {
            \tl_if_empty:cTF { l__spintent_sp #1 _read_arg_tl }
              {
                \tl_if_empty:nTF {#2}
                  { \tl_set:Nn \l__spintent_spcircfig_intent_arg_tl { _- } }
                  { \tl_set:Ne \l__spintent_spcircfig_intent_arg_tl { _( $ pt ) } }
              }
              {
                \tl_set:Ne \l__spintent_spcircfig_intent_tl
                  { \tl_use:c { l__spintent_sp #1 _read_arg_tl } }
                \__spintent_spcircfig_finish_intent:nn {#1} {#2}
                \tl_set:Ne \l__spintent_spcircfig_intent_arg_tl
                  {
                    \l__spintent_spcircfig_intent_tl :prefix
                    \tl_if_empty:nF {#2} { ( $ pt ) }
                  }
              }
          }
      }
  }
```

(*End of definition for* __spintent_spcircfig_build_concept:nn.)

__spintent_spcircfig_process:nn

Receives the short command name in '#1' and the argument in '#2'. Sets \l__spintent_spcircfig_op_tl to área/perímetro. With #2 empty, \l__spintent_spcircfig_short_bool is set false and nothing else is parsed. With a comma in #2, reports sp#1-invalid-arg immediately — \@@_luafun_geo_circ_parse_and_set: (shared with \spcirc) still accepts a second, comma-separated part as a generic label-type radius, matching almost any text, so it would never itself report an error for a comma-containing argument here, where a comma is never meaningful at all. Without a comma, calls __spintent_luafun_geo_circ_parse_and_set:n; on error, retries through __spintent_luafun_geo_afig_parse_and_set:n/ __spintent_luafun_geo_pfig_par (the same entry point \spAfig/\spPfig use) for the short number/letter form, rejecting a table-letter match (not meaningful for a circle) back to an error. Reports sp#1-invalid-arg on failure either way, then delegates to __spintent_spcircfig_build_short:n/ __spintent_spcircfig_build_concept:nn and __spintent_spcircfig_mathml:nn.

Unlike \spcirc, there is no radius/diameter here at all: a formula's left-hand side (A_0) names the figure, it does not carry the values needed to compute it — those belong on the right-hand side, elsewhere in the document.

```
\cs_new_protected:Nn \__spintent_spcircfig_process:nn
  {
    \str_if_eq:eeTF {#1} { Acirc }
```

```
      { \tl_set:Nn \l__spintent_spcircfig_op_tl { área } }
      { \tl_set:Nn \l__spintent_spcircfig_op_tl { perímetro } }
    \tl_if_empty:nTF {#2}
      {
        \bool_set_false:N \l__spintent_spcircfig_short_bool
      }
      {
        \bool_set_false:N \l__spintent_spcircfig_short_bool
        \tl_if_in:nnTF {#2} { , }
          {
            \msg_error:nnee { spintent } { sp #1 -invalid-arg }
              { \c_backslash_str sp #1 } { \exp_not:n {#2} }
          }
          {
            \__spintent_luafun_geo_circ_parse_and_set:n { \exp_not:n {#2} }
            \str_if_eq:VnT \l__spintent_geo_luaset_error_str { true }
              {
                \str_if_eq:eeTF {#1} { Acirc }
                  { \__spintent_luafun_geo_afig_parse_and_set:n { \exp_not:n {#2} } }
                  { \__spintent_luafun_geo_pfig_parse_and_set:n { \exp_not:n {#2} } }
                \str_if_eq:VnF \l__spintent_geo_luaset_error_str { true }
                  {
                    \str_if_empty:NTF \l__spintent_geo_luaset_letra_word_str
                      { \bool_set_true:N \l__spintent_spcircfig_short_bool }
                      { \str_set:Nn \l__spintent_geo_luaset_error_str { true } }
                  }
              }
            \bool_if:NF \l__spintent_spcircfig_short_bool
              {
                \str_if_eq:VnT \l__spintent_geo_luaset_error_str { true }
                  {
                    \msg_error:nnee { spintent } { sp #1 -invalid-arg }
                      { \c_backslash_str sp #1 } { \exp_not:n {#2} }
                  }
              }
          }
      }
    \bool_if:NTF \l__spintent_spcircfig_short_bool
      { \__spintent_spcircfig_build_short:n {#1} }
      { \__spintent_spcircfig_build_concept:nn {#1} {#2} }
    \__spintent_spcircfig_mathml:nn { #1 } { #2 }
  }
```

(*End of definition for* `\__spintent_spcircfig_process:nn`.)

`\__spintent_spcircfig_mathml:nn`

Renders `\l__spintent_spcircfig_intent_arg_tl`, already built by `\__spintent_spcircfig_process:nn`. Three cases: the short case prints the symbol with the short value as its own subscript; an empty `#2` prints the symbol alone; a real center prints the symbol (plus optional `sub-index` subscript) and the center between parentheses. In every case a single native `\MathMLintent` wraps `\__spintent_mathcal:n`$\{\langle A \mid P\rangle\}$ and the whole subscript together — splitting the symbol and the center into separate `\MathMLintent` siblings inside the same subscript was tried and reads as `<msub>`'s own native structural narration instead. The center is read through `\__spintent_mathmlarg_point:nnn`, referenced as `$pt` (same convention as `\spcirc`). Receives the short command name in '`#1`' and the argument in '`#2`'.

```
\cs_new_protected:Nn \__spintent_spcircfig_mathml:nn
  {
    \__spintent_invisible_sep: % (luamml issues #26)
    \MathMLintent { \l__spintent_spcircfig_intent_arg_tl }
      {
        \__spintent_inner_sep:
        \__spintent_luafun_mrow_intent_arg:nn
          { } { intent = ":silent" }
        {
          \str_if_eq:eeTF {#1} { Acirc } { \__spintent_mathcal:n { A } } { \__spintent_mathcal:n
        }
        \bool_if:NTF \l__spintent_spcircfig_short_bool
          {
            \c_math_subscript_token
              {
                \__spintent_spcircfig_sym:n {#1}
                \c_math_subscript_token { \l__spintent_geo_luaset_print_tl }
              }
```

```
            }
            {
              \c_math_subscript_token
                {
                  \MathMLintent { :silent } { \__spintent_spcircfig_sym:n {#1} }
                  \str_if_empty:cF { l__spintent_sp #1 _sub_index_str }
                    {
                      \c_math_subscript_token
                        {
                          \MathMLintent { :silent }
                            { \str_use:c { l__spintent_sp #1 _sub_index_str } }
                        }
                    }
                  \tl_if_empty:nF {#2}
                    {
                      \MathMLintent { :silent } { \lparen }
                      \__spintent_mathmlarg_point:nnV { pt } { sp #1 } \l__spintent_geo_luaset_poin
                      \MathMLintent { :silent } { \rparen }
                    }
                }
            }
        }
    }
```

(*End of definition for* `\__spintent_spcircfig_mathml:nn`.)

`\__spintent_spcircfig_exec:nnn`

The function `\__spintent_spcircfig_exec:nnn` receives the short name of the command in `#1`, the optional keys in `#2`, and the argument in `#3`, delegating the rendering to `\__spintent_spcircfig_process:nn`.

```
\cs_new_protected:Nn \__spintent_spcircfig_exec:nnn
  {
    \keys_set:nn { spintent / sp #1 } { #2 }
    \__spintent_spcircfig_process:nn { #1 } { #3 }
  }
```

(*End of definition for* `\__spintent_spcircfig_exec:nnn`.)

`\spAcirc` `\spPcirc`

Both public commands `\spAcirc` and `\spPcirc` enforce math mode, open a local group, and delegate execution to `\__spintent_spcircfig_exec:nnn`.

```
\NewDocumentCommand \spAcirc { O{} d() }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \attribute \l__spintent_scope_attr_int = 1
          \tl_if_novalue:nTF {#2}
            { \__spintent_spcircfig_exec:nnn { Acirc } { #1 } { } }
            { \__spintent_spcircfig_exec:nnn { Acirc } { #1 } { #2 } }
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spAcirc } }
  }
\NewDocumentCommand \spPcirc { O{} d() }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \attribute \l__spintent_scope_attr_int = 1
          \tl_if_novalue:nTF {#2}
            { \__spintent_spcircfig_exec:nnn { Pcirc } { #1 } { } }
            { \__spintent_spcircfig_exec:nnn { Pcirc } { #1 } { #2 } }
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spPcirc } }
  }
```

(*End of definition for* `\spAcirc` *and* `\spPcirc`. *These functions are documented on page* ??.)

11.48 Finish package

Finish package implementation.

```
\file_input_stop:
⟨/package⟩
```

12 Índice de la Implementación

The italic numbers denote the pages where the corresponding entry is described, the numbers underlined and all others indicate the line on which they are implemented in the package code.

A

\angle *110*
autofit 3039

C

cifras-sep 284, 569, 842, 1788, 1859, 2380
Document class:
  l3doc *29*
cmd-key-unknown 203
cmd-key-value-unknown 203
Commands provide by spintent:
  \spmoney *47*
  \spnum *37*
  \spsiglo *56, 59*
  \spunit *42*
concept 1950, 2120, 3039, 3638, 3845, 4130, 4283, 4467, 4848, 5066, 5346
currency 842
currency-no-subunits 811

D

diameter 5066
dolar-math 842

I

inline-numeric-denominator 547
invalid-angle-format 2330
invalid-century-format 1231
invalid-currency 811
invalid-date-format 1461
invalid-time-format 1382

J

join-word 2458

K

Keys for \spAcirc provide by spintent:
  concept *139*
  prefix *138, 139*
  print-sym *138–140*
  radio-sty *138*
  read-arg *138, 139*
  read-cmd *138, 139*
  read-sty *138, 139*
  read-sub *139*
  silent-cmd *139*
  sub-index *138, 139*
Keys for \spAfig provide by spintent:
  prefix *121*
  print-sym *121, 122*
  read-arg *121–123*
  read-cmd *121*
  read-sub *122*
Keys for \spangle provide by spintent:
  concept *109*
  prefix *109*
  print-m *109*
  read-cmd *109*
  recto *109*
  silent-cmd *109*
Keys for \sparc provide by spintent:
  concept *115*
  prefix *114, 115*
  read-cmd *114, 115*
  silent-cmd *115*
Keys for \spcirc provide by spintent:
  prefix *132*
  print-sym *132–134*
  radio-sty *132, 133*
  read-arg *133*
  read-cmd *132*
  read-sty *132, 133*
  read-sub *133*
  sub-index *132, 133*
Keys for \spcoord provide by spintent:
  silent-cmd *87*
Keys for \spdiam provide by spintent:
  concept *129*
  read-arg *129*
  read-cmd *129*
  read-sub *129*
  silent-cmd *129*
  sub-index *129*
  upper-case *129*
Keys for \spdiv provide by spintent:
  cifras-sep *67*
  notation-sym *67*
  read-parens *67*
  read-period-sep *67*
  read-sign *67*
  read-sym *67*
  set-sym *67*
  wrap-parens *67*
Keys for \spdsym provide by spintent:
  prefix *65*
  read-sym *65*
  set-sym *65*
  suffix *65*
Keys for \spmangle provide by spintent:
  concept *109*
  prefix *109*
  print-m *109*
  read-cmd *109*
  recto *109*
  silent-cmd *109*
Keys for \spmarc provide by spintent:
  concept *115*
  prefix *114, 115*
  read-cmd *114, 115*
  silent-cmd *115*
Keys for \spmcd provide by spintent:
  concept *71*
  prefix *71*
  read-cmd *71*
  result *71*
  silent-cmd *71*
  upper-case *71*
Keys for \spmcm provide by spintent:
  concept *71*
  prefix *71*

read-cmd 71
result 71
silent-cmd 71
upper-case 71
Keys for \spmoney provide by spintent:
cifras-sep 48
currency 48
dolar-math 48
sub-units 48
Keys for \spmQ provide by spintent:
join-word 81–83
print-dfrac 82–85
read-parens 82, 83
use-spnZ 82, 84, 85
wrap-parens 81–83
Keys for \spmside provide by spintent:
concept 105
prefix 105
print-m 104, 105
print-sym 129
read-arg 105
read-cmd 105, 107
read-sty 105
silent-cmd 105, 107
Keys for \spmsym provide by spintent:
not-print 66
prefix 66
read-sym 66
set-sym 66
suffix 66
Keys for \spmul provide by spintent:
cifras-sep 69
notation-sym 69
read-parens 69
read-period-sep 69
read-sign 69
read-sym 69
set-sym 69
wrap-parens 69
Keys for \spnD provide by spintent:
concept 74
prefix 74
result-num 74
result 74
silent-cmd 74
Keys for \spnM provide by spintent:
concept 74
prefix 74
result-num 74
result 74
silent-cmd 74
Keys for \spnum provide by spintent:
cifras-sep 36–38
notation-sym 37
read-period-sep 37, 39
read-sign 37
Keys for \spnZ provide by spintent:
cifras-sep 80
read-parens 80
read-sign 80
wrap-parens 80
Keys for \spPcirc provide by spintent:
concept 139
prefix 138, 139
print-sym 138–140
radio-sty 138
read-arg 138, 139
read-cmd 138, 139
read-sty 138, 139
read-sub 139
silent-cmd 139
sub-index 138, 139
Keys for \spPfig provide by spintent:
prefix 121
print-sym 121, 122
read-arg 121–123
read-cmd 121
read-sub 122
Keys for \spPoint provide by spintent:
concept 91
prefix 91
read-cmd 91
silent-cmd 91
Keys for \spPoly provide by spintent:
concept 118
prefix 117, 118
print-sym 118
read-cmd 117, 118
silent-cmd 118
Keys for \sprad provide by spintent:
concept 129
read-arg 129
read-cmd 129
read-sub 129
silent-cmd 129
sub-index 129
upper-case 129
Keys for \sprayo provide by spintent:
concept 102
prefix 102
read-cmd 102, 103
read-sty 102
silent-cmd 102, 103
Keys for \sprazon provide by spintent:
silent-cmd 77
use-dfrac 77
Keys for \sprecta provide by spintent:
prefix 98
read-cmd 98
silent-cmd 98
Keys for \spshort provide by spintent:
read-arg 54
small-caps 54
Keys for \spside provide by spintent:
concept 105
prefix 105
read-arg 104, 105
read-cmd 105, 107
read-sty 104, 105
silent-cmd 105, 107
Keys for \spsiglo provide by spintent:
print-era 57, 58
Keys for \spunit provide by spintent:
cifras-sep 43
notation-sym 43
print-cdot 43
read-cdot 43
read-period-sep 43
read-slash 43

©2026 by Pablo González L

Keys for \spusep provide by spintent:
silent 62
size 62
Keys for \spvec provide by spintent:
autofit 94
concept 94
prefix 94
read-cmd 94
silent-cmd 94

M

mathcal-fallback 80
\MathMLintent 112
mc-invalid-arguments 1943
md-invalid-argument 2146
Lua module provide by spintent:
spintent.lua 29, 30, 32, 35–37, 42, 46–49, 53, 54, 56, 58, 59, 61, 70, 74, 78, 81, 86, 91, 128

N

not-print 1714
notation-sym 284, 569, 1788, 1859

P

Packages :
expl3 . . . . . 35, 37
lua-unicode-math 29
luamml 32, 81, 83
tagpdf 53, 58
unicode-math 29, 63
prefix 1644, 1714, 1950, 2120, 2918, 3039, 3281, 3501, 3638, 3845, 4130, 4283, 4467, 5066, 5346
print-dfrac 2458
print-era 1239
print-m 3638, 3845
print-sym 4283, 4467, 4848, 5066, 5346

R

radio-sty 5066
read-arg 1109, 3638, 4467, 4848, 5066, 5346
read-cdot 569
read-cmd . . 1950, 2918, 3039, 3281, 3501, 3638, 3845, 4130, 4283, 4467, 4848, 5066, 5346
read-parens 1788, 1859, 2380, 2458
read-period-sep 284, 569, 1788, 1859
read-sign 284, 1788, 1859, 2380
read-slash 569
read-sty 3501, 3638, 5066, 5346
read-sub 4467, 4848, 5066, 5346
read-sym 1644, 1714, 1788, 1859
recto 3845
result 1950, 2120
result-num 2120
right-angle-sqr-fallback 80
\rightangle 110

S

seq commands:
\seq_set_from_clist:Nn 62
set-sym 1644, 1714, 1788, 1859
\setspintent 3, 33, 191
silent 1509
silent-cmd 1950, 2120, 2260, 2673, 2918, 3039, 3281, 3501, 3638, 3845, 4130, 4283, 4467, 4848, 5066, 5346
size 1509
small-caps 1109
\spAcirc 138, 5628
spAcirc-invalid-arg 5337
\spAfig 23, 121, 4814
spAfig-invalid-arg 4429
spAfig-invalid-points 4429
spAfig-read-arg-key 4429
spAfig-sym-cardinality-conflict 4429
spAfig-sym-letra-conflict 4429
\spang 21, 78, 2370
\spangle 20, 109, 4093
spangle-invalid-arg 3835
\sparc 21, 114, 4252
sparc-invalid-points 4120
\spcirc 132, 5313
spcirc-invalid-arg 5059
\spcoord 86, 2807
spcoord-invalid-arg 2669
\spdate 11, 61, 1485
\spdiam 128, 5028
\spdiv 13, 67, 1849
spdiv-empty-argument 1808
\spdsym 12, 65, 1691
spintent internal commands:
__spintent_collapse_hyphens:N 128
\l__spintent_command_name_str 203
\l__spintent_geo_circfig_mode_str 5336
\l__spintent_geo_final_print_tl 4421
\l__spintent_geo_luaset_angulo_case_-str 2641
\l__spintent_geo_luaset_circ_radio_raw_-str 5048
\l__spintent_geo_luaset_circ_radio_type_-str 5048
\l__spintent_geo_luaset_concept_str . . . 2641
\l__spintent_geo_luaset_error_str 2641
\l__spintent_geo_luaset_intent_str . . . 2641
\l__spintent_geo_luaset_letra_str 2641
\l__spintent_geo_luaset_letra_word_str 2641
\l__spintent_geo_luaset_nombre_es_num_-str 2641
\l__spintent_geo_luaset_points_-clist␣␣␣␣\l__spintent_geo_luaset_is_-name_str 2641
\l__spintent_geo_luaset_poly_sym_str . . 2641
\l__spintent_geo_luaset_print_tl 2641
\l__spintent_geo_luaset_sub_spoken_str 2641
\l__spintent_geo_luaset_sub_str 2641
\l__spintent_geo_luaset_tipo_str 2641
\l__spintent_geo_print_sym_str 4531
__spintent_inner_sep: 120
__spintent_invisible_sep: 116
\c__spintent_invisible_sep_int 124
\c__spintent_invisible_sep_tl 83
__spintent_luafun_clean_split_and_set:n 76, 241
__spintent_luafun_mrow_intent_arg:nn . 116
__spintent_luafun_set_debug_ghost:n . . . 70
__spintent_luafun_spang_parse:n 79
__spintent_luafun_spdate_parse:n 61
__spintent_luafun_spmoney_lookup_data:n 76
__spintent_luafun_spmQ_scale_int:nn . . . 76

__spintent_luafun_sptime_parse:n 60
__spintent_luafun_spunit_lookup_alias:n 76
\l__spintent_luaset_arg_above_tl 260
\l__spintent_luaset_arg_below_tl 260
\l__spintent_luaset_cal_fam_ok_str 67
\l__spintent_luaset_dec_is_all_zeros_-
str 253
\l__spintent_luaset_dec_sep_tl 241
\l__spintent_luaset_decimal_period_str 253
\l__spintent_luaset_denom_has_number_-
str 260
\l__spintent_luaset_exponent_tl 253
\l__spintent_luaset_format_part_dec_str 248
\l__spintent_luaset_format_part_int_str 248
\l__spintent_luaset_has_exponent_str .. 253
\l__spintent_luaset_has_integer_str .. 84, 85,
241
\l__spintent_luaset_has_natural_str ... 241
\l__spintent_luaset_has_units_str 260
\l__spintent_luaset_millons_str 253
\l__spintent_luaset_not_decimal_not_-
period_str 253
\l__spintent_luaset_not_decimal_period_-
str 253
\l__spintent_luaset_only_part_dec_str . 248
\l__spintent_luaset_only_part_int_str . 248
\l__spintent_luaset_only_part_period_-
str 248
\l__spintent_luaset_part_dec_tl 241
\l__spintent_luaset_part_int_tl 84, 241
\l__spintent_luaset_part_period_tl ... 241
\l__spintent_luaset_sign_tl 84, 241
\l__spintent_luaset_status_str 260
\l__spintent_luaset_sym_fam_ok_str 67
__spintent_mathcal:n 94
__spintent_mathmlarg_point:nnn 2822
\l__spintent_mathmlarg_point_kv_tl ... 2818
\l__spintent_mathmlarg_point_mode_str . 2818
\l__spintent_mathmlarg_point_ref_tl ... 2818
\l__spintent_mathmlarg_point_silent_-
bool 2818
\l__spintent_mc_args_seq 1930
\l__spintent_mc_luaset_all_numeric_str 1930
\l__spintent_mc_luaset_types_str 1930
__spintent_mc_process_args:nn 1991
\l__spintent_mc_raw_args_seq 1930
\l__spintent_mc_types_seq␣␣␣\l__spintent_-
spmcd_print_result_bool 1930
\l__spintent_md_out_seq 2112
__spintent_print_spunit_hierarchy: ... 670
__spintent_right_angle_sqr: 107
__spintent_sanitize_affix:Nn 136
\l__spintent_saved_luamml_flag_int .. 83, 84,
2439
\l__spintent_scope_attr_int 73
\l__spintent_scope_luaset_attr_str 73
\l__spintent_setspintent_input_arg_cmd_-
tl 154
__spintent_setspintent_parse_args:nn . 174
__spintent_setspintent_set_keys:nn ... 163
\l__spintent_smparc_prefix_tl 4115
__spintent_sp_rad_diam_exec:nn 5023
\l__spintent_sp_rad_diam_letra_tl 4846
__spintent_sp_rad_diam_print:n 4895
\l__spintent_sp_rad_diam_word_tl 4846
\l__spintent_spAcirc_mode_str 5326
\l__spintent_spAcirc_prefix_tl 5326
\l__spintent_spAcirc_print_sym_str ... 5326
\l__spintent_spAcirc_radio_sty_str ... 5326
\l__spintent_spAcirc_read_arg_tl 5326
\l__spintent_spAcirc_read_sty_str 5326
\l__spintent_spAcirc_sub_index_str ... 5326
\l__spintent_spAfig_mode_str 4422
\l__spintent_spAfig_prefix_tl 4422
\l__spintent_spAfig_print_sym_str 4422
\l__spintent_spAfig_read_arg_tl 4422
\l__spintent_spang_luaset_error_str ... 2326
\l__spintent_spang_luaset_grado_str ... 2326
\l__spintent_spang_luaset_minuto_str .. 2326
\l__spintent_spang_luaset_segundo_str . 2326
__spintent_spang_process:n 2335
__spintent_spangle_exec:nnn 4088
\l__spintent_spangle_intent_arg_tl ... 3904
\l__spintent_spangle_intent_tl 3904
__spintent_spangle_mathml:n 3946
\l__spintent_spangle_mode_str 3830
\l__spintent_spangle_prefix_tl 3830
\l__spintent_spangle_print_m_now_bool . 3904
__spintent_spangle_process:nn 3911
\l__spintent_spangle_pt_idx_int 3904
\l__spintent_spangle_pt_seq 3904
\l__spintent_spangle_refs_tl 3904
__spintent_spangle_sym:n 3885
__spintent_sparc_exec:nnn 4247
\l__spintent_sparc_intent_arg_tl 4174
\l__spintent_sparc_intent_tl 4174
__spintent_sparc_mathml:n 4192
\l__spintent_sparc_mode_str 4115
\l__spintent_sparc_prefix_tl 4115
__spintent_sparc_print_arc:n 4168
__spintent_sparc_process:nn 4179
\l__spintent_sparc_pt_idx_int 4174
\l__spintent_sparc_pt_seq 4174
\l__spintent_sparc_refs_tl 4174
__spintent_sparc_wide_arc:n 4164
\l__spintent_spcirc_articulo_str 5050
__spintent_spcirc_build_concept: 5194
\l__spintent_spcirc_connector_arg_tl .. 5142
\l__spintent_spcirc_connector_tl 5142
__spintent_spcirc_exec:nn 5308
\l__spintent_spcirc_has_arg_bool 5142
\l__spintent_spcirc_intent_arg_tl 5142
\l__spintent_spcirc_intent_tl 5142
__spintent_spcirc_mathml: 5263
\l__spintent_spcirc_mode_str 5050
\l__spintent_spcirc_prefix_tl 5050
\l__spintent_spcirc_print_sym_str 5050
__spintent_spcirc_process:n 5247
\l__spintent_spcirc_radio_arg_tl 5142
__spintent_spcirc_radio_print: 5179
\l__spintent_spcirc_radio_sty_str 5050
\l__spintent_spcirc_read_sty_str 5050
\l__spintent_spcirc_sub_index_str 5050
__spintent_spcirc_sym: 5148

__spintent_spcirc_word_and_article: .. 5158
\l__spintent_spcirc_word_tl 5058
__spintent_spcircfig_build_concept:nn 5490
__spintent_spcircfig_build_short:n ... 5452
__spintent_spcircfig_del_word:n 5441
__spintent_spcircfig_exec:nnn 5623
__spintent_spcircfig_finish_intent:nn 5474
\l__spintent_spcircfig_intent_arg_tl .. 5412
\l__spintent_spcircfig_intent_tl 5412
__spintent_spcircfig_mathml:nn 5582
\l__spintent_spcircfig_op_tl 5412
__spintent_spcircfig_process:nn 5537
\l__spintent_spcircfig_short_bool 5412
__spintent_spcircfig_sym:n 5417
__spintent_spcircfig_word_and_-
article:n 5423
__spintent_spcoord_exec:nn 2789
__spintent_spcoord_intent_infix: 2760
\l__spintent_spcoord_intent_str 2665
\l__spintent_spcoord_luaset_error_str . 2655
\l__spintent_spcoord_luaset_sep_str ... 2655
\l__spintent_spcoord_luaset_x_has_frac_-
str 2655
\l__spintent_spcoord_luaset_x_sign_str 2655
\l__spintent_spcoord_luaset_x_type_str 2655
\l__spintent_spcoord_luaset_x_value_-
str 2655
\l__spintent_spcoord_luaset_y_has_frac_-
str 2655
\l__spintent_spcoord_luaset_y_sign_str 2655
\l__spintent_spcoord_luaset_y_type_str 2655
\l__spintent_spcoord_luaset_y_value_-
str 2655
__spintent_spcoord_mathml: 2772
__spintent_spcoord_print:n 2737
\l__spintent_spcoord_raw_arg_tl 2665
__spintent_spcoord_set_wrap_parens_-
size: 2684
__spintent_spcoord_value:n 2731
\l__spintent_spcoord_wrap_parens_l_tl . 2665
\l__spintent_spcoord_wrap_parens_r_tl . 2665
\l__spintent_spdate_luaset_error_str .. 1459
\l__spintent_spdate_luaset_output_str . 1459
__spintent_spdate_process:n *62*, 1470
\l__spintent_spdiam_mode_str 4840
\l__spintent_spdiam_sub_index_str 4840
\l__spintent_spdiam_upper_case_bool ... 4840
__spintent_spdiv_process:nnn 1834
__spintent_spdiv_render_op:nn 1813
__spintent_spdiv_render_parens:nn ... 1822
\l__spintent_spdsym_prefix_tl 1640
\l__spintent_spdsym_read_sym_tl 1640
\l__spintent_spdsym_suffix_tl 1640
\l__spintent_spdsym_symbol_tl 1640
__spintent_spfig_build_concept_-
vertices:n 4687
__spintent_spfig_build_refs: 4674
__spintent_spfig_build_simple:n 4564
__spintent_spfig_exec:nnn 4809
\l__spintent_spfig_intent_arg_tl 4531
\l__spintent_spfig_intent_tl 4531
__spintent_spfig_mathml:n 4769
\l__spintent_spfig_op_tl 4531
__spintent_spfig_points_loop:n 4757
__spintent_spfig_process:nn 4540
\l__spintent_spfig_pt_idx_int 4531
\l__spintent_spfig_pt_seq 4531
\l__spintent_spfig_refs_tl 4531
__spintent_spfig_sym: 4523
\l__spintent_spfig_symbol_tl 4531
\l__spintent_spmangle_mode_str 3830
\l__spintent_spmangle_prefix_tl 3830
\l__spintent_spmarc_mode_str 4115
__spintent_spmc_process:nnnn 2026
\l__spintent_spmcd_prefix_tl 1930
\l__spintent_spmcd_print_upper_bool ... 1930
\l__spintent_spmcd_read_terse_bool ... 1930
\l__spintent_spmcd_suffix_tl 1930
\l__spintent_spmcm_prefix_tl 1930
\l__spintent_spmcm_print_result_bool .. 1930
\l__spintent_spmcm_print_upper_bool ... 1930
\l__spintent_spmcm_read_terse_bool ... 1930
\l__spintent_spmcm_spmcd_luaset_error_-
str 1930
\l__spintent_spmcm_suffix_tl 1930
__spintent_spmoney_build_tree: 1021
__spintent_spmoney_check_sub_units: .. 974
\l__spintent_spmoney_extracted_curr_str 803
\l__spintent_spmoney_extracted_num_tl . 803
__spintent_spmoney_intent_int: 939
\l__spintent_spmoney_intent_money_str . 803
\l__spintent_spmoney_intent_sign_str .. 803
\l__spintent_spmoney_intent_sub_unit_-
str 803
\l__spintent_spmoney_luaset_conde_str . 793
\l__spintent_spmoney_luaset_currency_arg_-
str 793
\l__spintent_spmoney_luaset_plur_str .. 793
\l__spintent_spmoney_luaset_position_-
str 793
\l__spintent_spmoney_luaset_print_iso_-
str 793
\l__spintent_spmoney_luaset_sing_str .. 793
\l__spintent_spmoney_luaset_status_str 793
\l__spintent_spmoney_luaset_subplur_str 793
\l__spintent_spmoney_luaset_subsing_str 793
\l__spintent_spmoney_luaset_symbol_str 793
__spintent_spmoney_normalize_key:n ... 828
__spintent_spmoney_parse_and_route:n . 1062
__spintent_spmoney_print_dec_sub_-
unit: 1021
__spintent_spmoney_print_int: 910
__spintent_spmoney_print_sep_con: ... 1021
__spintent_spmoney_print_symbol: 876
__spintent_spmoney_render_layout: ... 889
__spintent_spmoney_render_number_node: 901
__spintent_spmoney_render_only_integer_-
node: 901
\l__spintent_spmoney_res_main_voice_str 803
__spintent_spmoney_set_and_print_sub_-
units: 974
__spintent_spmoney_set_currency:n ... 828
\l__spintent_spmoney_sub_units_bool ... 803
\l__spintent_spmoney_subnum_int 803

__spintent_spmoney_validate_money: ... 860
__spintent_spmQ_exec:nnnn 2619
\l__spintent_spmQ_join_word_str ... 83, 2439
__spintent_spmQ_print:nnn 2611
\l__spintent_spmQ_print_dfrac_bool 84
__spintent_spmQ_print_scale_int: 2557
__spintent_spmQ_print_scale_int:n ... 2566
__spintent_spmQ_print_spnZ_dfrac:nn .. 2594
__spintent_spmQ_set_join_word: 2548
__spintent_spmQ_set_wrap_parens_size: 2493
__spintent_spmQ_start_wrap_parens: ... 2528
__spintent_spmQ_stop_wrap_parens: ... 2528
\l__spintent_spmQ_use_spnZ_bool 84
\l__spintent_spmside_mode_str 3621
\l__spintent_spmside_prefix_tl 3621
\l__spintent_spmside_read_arg_tl 3621
\l__spintent_spmside_read_sty_str 3621
\l__spintent_spmsym_prefix_tl 1710
\l__spintent_spmsym_read_sym_tl 1710
\l__spintent_spmsym_suffix_tl 1710
\l__spintent_spmsym_symbol_tl 1710
__spintent_spmul_process:nnn 1905
__spintent_spmul_render_op:nn 1884
__spintent_spmul_render_parens:nn ... 1893
\l__spintent_spnD_luaset_is_truncated_-
str 2112
\l__spintent_spnD_luaset_result_tl ... 2112
\l__spintent_spnD_prefix_tl 2112
\l__spintent_spnD_print_result_num_tl . 2112
\l__spintent_spnM_luaset_is_truncated_-
str 2112
\l__spintent_spnM_luaset_result_tl ... 2112
\l__spintent_spnM_prefix_tl 2112
\l__spintent_spnM_print_result_num_tl . 2112
__spintent_spnM_spnD_exec:nnnn 2192
__spintent_spnM_spnD_intent:nn 2156
__spintent_spnM_spnD_process:nnnnn ... 2151
__spintent_spnM_spnD_result:n 2173
__spintent_spnM_spnD_silent:nn 2165
\l__spintent_spnmd_luaset_is_number_-
str 2112
\l__spintent_spnQ_intent_read_sign_str 2439
\l__spintent_spnQ_read_parens_bool 83
\l__spintent_spnQ_wrap_parens_l_tl .. 82, 83,
2439
\l__spintent_spnQ_wrap_parens_r_tl .. 82, 83,
2439
\l__spintent_spnum__notation_sym_tl 37
\l__spintent_spnum_intent_read_dec_sep_-
str 37, 39, 279
\l__spintent_spnum_intent_read_sign_str 37,
279
\l__spintent_spnum_notation_sym_tl ... 279
__spintent_spnum_print_integer: 446
__spintent_spnum_print_num_start: ... 446
__spintent_spnum_print_number_core: .. 446
__spintent_spnum_print_part_dec: 425
__spintent_spnum_print_part_period: .. 425
__spintent_spnum_print_period_bar: ... 354
\l__spintent_spnum_read_period_sep_str . 37,
279
\l__spintent_spnum_read_terse_bool . 37, 279
__spintent_spnum_render_dec: 337
__spintent_spnum_render_decimal_sep: . 391
__spintent_spnum_render_int: 337
__spintent_spnum_render_only_period: . 354
__spintent_spnum_render_part:n 337
__spintent_spnum_render_period_sep: .. 354
__spintent_spnz_process:nn 2413
__spintent_spnz_render_parens: 2401
\l__spintent_spPcirc_mode_str 5326
\l__spintent_spPcirc_prefix_tl 5326
\l__spintent_spPcirc_print_sym_str ... 5326
\l__spintent_spPcirc_radio_sty_str ... 5326
\l__spintent_spPcirc_read_arg_tl 5326
\l__spintent_spPcirc_read_sty_str 5326
\l__spintent_spPcirc_sub_index_str ... 5326
\l__spintent_spPfig_mode_str 4422
\l__spintent_spPfig_prefix_tl 4422
\l__spintent_spPfig_print_sym_str 4422
\l__spintent_spPfig_read_arg_tl 4422
__spintent_spPoint_exec:nnn 2993
\l__spintent_spPoint_intent_arg_tl ... 2907
__spintent_spPoint_mathml:n 2984
\l__spintent_spPoint_mode_str 2907
\l__spintent_spPoint_prefix_tl 2907
__spintent_spPoint_print:n 2949
__spintent_spPoly_build_concept: 4342
__spintent_spPoly_build_refs: 4329
__spintent_spPoly_exec:nn 4405
\l__spintent_spPoly_intent_arg_tl 4317
__spintent_spPoly_mathml: 4385
\l__spintent_spPoly_mode_str 4274
__spintent_spPoly_points_loop: 4363
\l__spintent_spPoly_prefix_tl 4274
__spintent_spPoly_process:n 4375
\l__spintent_spPoly_pt_idx_int 4317
\l__spintent_spPoly_pt_seq 4317
\l__spintent_spPoly_refs_tl 4317
__spintent_spPoly_sym: 4321
\l__spintent_sprad_mode_str 4840
\l__spintent_sprad_sub_index_str 4840
\l__spintent_sprad_upper_case_bool ... 4840
__spintent_sprayo_exec:nn 3605
\l__spintent_sprayo_intent_arg_tl 3545
__spintent_sprayo_mathml: 3558
\l__spintent_sprayo_mode_str 3491
\l__spintent_sprayo_prefix_tl 3491
__spintent_sprayo_print:n 3549
\l__spintent_sprayo_pt_idx_int 3545
\l__spintent_sprayo_pt_seq 3545
\l__spintent_sprayo_read_sty_str 3491
\l__spintent_sprayo_refs_tl 3545
__spintent_sprazon_exec:nnn 2274
__spintent_spread_exec:nn 1621
\l__spintent_spread_intent_tl 1620
__spintent_sprecta_exec:nn 3475
\l__spintent_sprecta_intent_arg_tl ... 3267
\l__spintent_sprecta_kv_tl 3267
__spintent_sprecta_mathml:n 3321
\l__spintent_sprecta_mode_str 3267
\l__spintent_sprecta_prefix_tl 3267
__spintent_sprecta_print:n 3312

\l__spintent_sprecta_pt_idx_int 3267
\l__spintent_sprecta_pt_seq 3267
\l__spintent_sprecta_refs_tl 3267
__spintent_spshort_execute_layout: ... 1170
\l__spintent_spshort_luaset_base_str .. 1087
\l__spintent_spshort_luaset_expanded_-
str 1087
\l__spintent_spshort_luaset_layout_str 1087
\l__spintent_spshort_luaset_output_str 1087
\l__spintent_spshort_luaset_status_str 1087
\l__spintent_spshort_luaset_suffix_str 1087
__spintent_spshort_print:nn 1123
__spintent_spshort_process_and_-
render:n 1135
__spintent_spshort_render_bypass:n ... 1146
__spintent_spshort_render_error:n ... 1146
__spintent_spshort_render_fallback: .. 1146
__spintent_spshort_render_success: ... 1146
__spintent_spshort_tag_span:nn 1096
__spintent_spside_exec:nnn 3803
\l__spintent_spside_intent_arg_tl 3695
\l__spintent_spside_intent_tl 3695
__spintent_spside_mathml:n 3728
\l__spintent_spside_mode_str 3621
__spintent_spside_points_loop:n 3791
\l__spintent_spside_prefix_tl 3621
__spintent_spside_process:nn 3700
\l__spintent_spside_pt_idx_int 3695
\l__spintent_spside_pt_seq 3695
\l__spintent_spside_read_arg_tl 3621
\l__spintent_spside_read_sty_str 3621
\l__spintent_spside_refs_tl 3695
\l__spintent_spsiglo_luaset_arabic_str 1227
\l__spintent_spsiglo_luaset_error_str . 1227
\l__spintent_spsiglo_luaset_roman_min_-
str 1227
\l__spintent_spsiglo_print_era_str ... 1227
__spintent_spsiglo_print_math:n 1311
__spintent_spsiglo_print_text:n 1311
__spintent_spsiglo_render_simple:nn .. 1262
__spintent_spsiglo_render_with_-
era:nnn 1262
__spintent_spsiglo_tag_span:nn 1298
__spintent_sptime_build_seconds:nn ... 1391
__spintent_sptime_build_tree: *61*, 1409
\l__spintent_sptime_luaset_base_str ... 1377
\l__spintent_sptime_luaset_error_str .. 1377
\l__spintent_sptime_luaset_has_seconds_-
str 1377
\l__spintent_sptime_luaset_is_fraction_-
str 1377
\l__spintent_sptime_luaset_seconds_str 1377
\l__spintent_spunit_exp_seq 540
\l__spintent_spunit_exponent_tl 540
__spintent_spunit_format_canonical: .. 603
\l__spintent_spunit_is_mult_bool 540
\l__spintent_spunit_luaset_canonical_-
str 533
\l__spintent_spunit_luaset_compact_exp_-
str 533
\l__spintent_spunit_luaset_is_compact_-
str 533
\l__spintent_spunit_luaset_is_-
sexagesimal_str 533
\l__spintent_spunit_luaset_lookup_status_-
str 533
\l__spintent_spunit_luaset_read_str ... 533
\l__spintent_spunit_luaset_status_str . 533
\l__spintent_spunit_mult_seq 540
__spintent_spunit_process_base_exp:n . 611
__spintent_spunit_process_mult:n 663
\l__spintent_spunit_read_cdot_bool ... 540
\l__spintent_spunit_read_slash_str ... 540
__spintent_spunit_safe_exec:n 682
\l__spintent_spunit_unit_name_tl 540
\l__spintent_spusep_arg_tl *62*, *63*, 1506
__spintent_spusep_convert:n 1568
\l__spintent_spusep_left_tl *62*, *63*, 1506
\l__spintent_spusep_right_tl *62*, *63*, 1506
__spintent_spusep_set_parens:nn 1557
\l__spintent_spusep_silent_bool *63*
__spintent_spvec_exec:nnn 3231
\l__spintent_spvec_intent_arg_tl 3028
\l__spintent_spvec_kv_tl 3028
__spintent_spvec_mathml:n 3090
\l__spintent_spvec_mode_str 3028
\l__spintent_spvec_prefix_tl 3028
__spintent_spvec_print:n 3073
\l__spintent_spvec_pt_idx_int 3028
\l__spintent_spvec_pt_seq 3028
\l__spintent_spvec_refs_tl 3028
__spintent_unknown_keys:n 230
__spintent_unknown_keys:nn 230
spintent.lua 69
\spmangle *21*, *109*, 4093
spmangle-invalid-arg 3835
\spmarc *22*, *114*, 4252
spmarc-invalid-points 4120
\spmcd *15*, *70*, 2092
\spmcm *15*, *70*, 2092
\spmoney *6*, *47*, 1076
\spmQ *16*, *81*, 2631
spmQ-not-integer 2444
spmQ-zero-denominator 2444
\spmside *19*, *104*, 3808
spmside 5066
spmside-invalid-points 3628
\spmsym *12*, *66*, 1763
\spmul *13*, *69*, 1920
spmul-empty-argument 1879
\spnD *14*, *74*, 2240
\spnewunit *6*, *46*, 761
spnewunit-duplicate 740
\spnM *14*, *74*, 2240
\spnum *3*, *37*, *38*, 515
spnum-has-units 501
spnum-no-digits 501
\spnZ *16*, *79*, 2429
spnz-not-integer 2396
\spPcirc *138*, 5628
spPcirc-invalid-arg 5337
\spPfig *24*, *121*, 4814
spPfig-invalid-arg 4429

spPfig-invalid-points 4429
spPfig-read-arg-key 4429
spPfig-sym-cardinality-conflict 4429
spPfig-sym-letra-conflict 4429
\spPoint *17*, *91*, 3017
spPoint-invalid-point 2910
\spPoly *22*, *117*, 4410
spPoly-invalid-points 4276
\spProp *78*, 2310
\sprad *128*, 5028
\sprayo *18*, *102*, 3610
sprayo-invalid-points 3494
\sprazon *77*, 2300
\spread *11*, *64*, 1630
spread-empty-args 1617
\sprecta *18*, *98*, 3480
sprecta-invalid-points 3274
\spshort *7*, *53*, 1214
\spside *19*, *104*, 3808
spside 5066
spside-invalid-points 3628
\spsiglo *11*, *56*, 1364
\sptime *11*, *59*, 1441
\spunit *4*, *42*, *43*, *45*, 728
spunit-invalid-unit 547
spunit-no-units 547
\spunitalias *6*, *46*, 775
spunitalias-duplicate 740
spunitalias-invalid-expr 740
\spusep *12*, *62*, 1606
spusep-empty-arg 1495
spusep-invalid-arg 1495
\spvec *93*, 3256
spvec-invalid-arg 3035
str commands:
    \str_if_eq:nnTF 62
    \str_lowercase:n 62
    \str_uppercase:n 62
sub-index 4848, 5066, 5346
sub-units 842
suffix 1644, 1714

U

unknown 1509, 1644, 1714, 1950, 2120, 2260, 2458, 2918, 3039, 3281, 3501, 3638, 3845, 4130, 4283, 4467, 4848, 5066, 5346
unknown-abbreviation 1093
unknown-choice 203
unknown-command-setup 154
upper-case 1950, 4848
use-dfrac 2260
use-spnZ 2458

W

wrap-parens 1788, 1859, 2380, 2458