spanish-mathml-intent

MATHML INTENT EN ESPAÑOL

v1.3 2026-11-01*

©2026 by Pablo González L†

CTAN: https://www.ctan.org/pkg/spanish-mathml-intent

https://github.com/pablgonz/spanish-mathml-intent

Resumen

El paquete spanish-mathml-intent agrega el atributo *intent* de MathML a las matemáticas escritas en español con LuaLaTeX. El atributo le indica a un lector de pantalla qué concepto es una expresión y cómo se debe leer, de modo que NVDA con MathCAT lea correctamente las fórmulas en documentos PDF accesibles (*tagged* PDF). Sus comandos escriben y etiquetan números, unidades, dinero, fechas y horas, geometría, álgebra y estadística, cada uno con la lectura en español que le corresponde. Está pensado para docentes de educación primaria y secundaria.

Índice

1 Descripción . . . 2
- 1.1 Carga del paquete . . . 2
- 1.2 El paquete babel . . . 3
- 1.3 Fuentes tipográficas . . . 3
  - 1.3.1 La llave mathcal . . . 3
- 1.4 El paquete hyperref . . . 3
- 1.5 Interfaz de usuario . . . 4
  - 1.5.1 Lo que ofrece el paquete . . . 4
  - 1.5.2 Lo que ofrece cada eje . . . 4
  - 1.5.3 Reglas para argumentos . . . 5
  - 1.5.4 Consulta rápida . . . 5
- 1.6 Un primer documento . . . 5
- 1.7 Cómo comprobar el resultado . . . 6
  - 1.7.1 Escuchar la lectura . . . 6
  - 1.7.2 Validar la estructura . . . 6
  - 1.7.3 Instalar rnv-wrapp y veraPDF . . . 7
- 1.8 Si algo no suena como esperaba . . . 7

2 El comando \setspintent . . . 7

3 The big four . . . 8
- 3.1 El comando \spnum . . . 8
  - 3.1.1 Llaves para \spnum . . . 8
- 3.2 El comando \spunit . . . 9
  - 3.2.1 Llaves para \spunit . . . 9
  - 3.2.2 Argumentos para \spunit . . . 9
  - 3.2.3 El comando \spnewunit . . . 10
  - 3.2.4 El comando \spunitalias . . . 11
- 3.3 El comando \spmoney . . . 11
  - 3.3.1 Llaves para \spmoney . . . 11
  - 3.3.2 Monedas y alias para \spmoney . . . 12
- 3.4 El comando \spshort . . . 13
  - 3.4.1 Llaves para \spshort . . . 13
  - 3.4.2 Argumentos para \spshort . . . 13

4 Utilidades generales . . . 15

5 Utilidades para símbolos . . . 16

6 Utilidades para números . . . 18

7 Utilidades para geometría . . . 24

8 Utilidades para álgebra . . . 39

9 Utilidades para estadística . . . 43

10 Modo mathml-AF y spmmli-tool . . . 49
- 10.1 Cuál usar y cuándo . . . 49
- 10.2 De mathml-SE a mathml-AF . . . 49
- 10.3 La herramienta spmmli-tool . . . 49

11 Ejemplos de PDF accesible y etiquetado . . . 50

12 Referencias . . . 52

13 Change history . . . 54

14 Índice de la Documentación . . . 55

15 Implementation . . . 59

16 Índice de la Implementación . . . 249

Motivación y agradecimientos

Desde que desarrollé scontents[1] en 2019 y, más tarde, enumext[2] 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[3] ha realizado un trabajo extraordinario en este campo. Hoy es posible generar documentos etiquetados, validarlos con herramientas como veraPDF[4] o inspeccionar su estructura con ngpdf[5]. Sin embargo, siempre me quedó una duda: cuando un estudiante utiliza NVDA[6] junto con MathCAT[7], ¿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.

*Este archivo describe la documentación para la versión v1.3 revisada por última vez el 2026-11-01.

†E-mail: «pablgonz@educarchile.cl».

El objetivo de spanish-mathml-intent 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[8, 9, 10], las especificaciones de WTPDF[11] y la guía más reciente de la PDF Association, «Best Practice Guide: Math in PDF»[12], 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.

Debo agradecer a todo el equipo de LaTeX que desarrolló el proyecto de *tagged* PDF y a los diversos miembros de la comunidad TeX-SX que han respondido mis muchas dudas en el chat sobre este tema; destaco entre ellos a David Carlisle, Ulrike Fischer, Clea F. Rees y Max Chernoff, quien me mostró el camino de Lua.

Un agradecimiento especial a mi esposa Javiera, quien me acompañó incontables horas en las escuchas de NVDA/MathCAT durante el desarrollo de spanish-mathml-intent.

Licencia y requerimientos

El paquete spanish-mathml-intent carga y requiere el paquete unicode-math[13] 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 la clase estándar article proporcionada por LaTeX en tamaño 10pt, usando NVDA 2026.2 y Adobe Acrobat Reader[14] actualizado 2026; además se utilizó MathCATDemo[15]. No se ha probado con otros lectores de pantalla.

Para profundizar en el etiquetado de documentos, es decir, \DocumentMetadata, tagging y tagging-setup, consulte la documentación de The LaTeX Tagged PDF Project[3, 16], que está en inglés. Quien da sus primeros pasos en LaTeX puede apoyarse en Learn LaTeX[17], un sitio respaldado por el equipo de LaTeX con lecciones breves y ejemplos que se pueden probar en línea; como referencia general sirve el manual no oficial de LaTeX 2_ε[18].

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[19] release 2026-11-01.

Este paquete solo se puede utilizar con LuaLaTeX y *tagged* PDF activo. Está presente en TeX Live, para instalarlo utilice el gestor de paquetes con `tlmgr install spanish-mathml-intent`. Si desea probar la versión de desarrollo, que puede contener cambios aún no publicados, clone el repositorio y ejecute `l3build install`:

```
git clone https://github.com/pablgonz/spanish-mathml-intent.git
cd spanish-mathml-intent
l3build install
```

1 Descripción

El nombre spanish-mathml-intent resume lo que hace el paquete: *«Spanish»* por el idioma, *«MathML»* por el lenguaje en que se escriben las fórmulas accesibles e *«intent»* por el atributo del mismo nombre que indica cómo se lee cada fórmula. El nombre corto original, *«Spanish Parse Intent»*, sobrevive en el prefijo `sp` de los comandos y en el módulo interno `spintent`. El paquete está pensado para docentes de educación primaria y secundaria.

Un documento *tagged* PDF, en español *«PDF etiquetado»*, es un PDF que, además de componer la página, guarda su estructura: qué es un título, un párrafo, una lista, una tabla o una fórmula, y en qué orden se leen. Los lectores de pantalla, como NVDA, usan esa información para leer el documento a estudiantes ciegos o con baja visión; sin ella solo disponen del texto suelto, sin saber qué es cada parte. Cómo activarlo se explica en la §1.1 y un documento completo en la §1.6.

El paquete está pensado para crear *«apuntes»*, *«hojas de ejercicios»* y *«clases escritas»* enfocadas en el estudiante. Solo un docente sabe cuál 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 spanish-mathml-intent usa y abusa del atributo `:intent` de MathML por medio del comando \MathMLintent definido en latex-lab-mathintent[20], un encapsulado de \luamml_attribute:een provista por luamml[21]. ¡Gracias Marcel!

1.1 Carga del paquete

El paquete spanish-mathml-intent 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 tagpdf[22] y documentmetadata-support[20]; es deber del usuario revisar la documentación asociada a ellos.

[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.

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

1.2 El paquete `babel`

El paquete babel-spanish[23] 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» provistas 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 usan. El paquete spanish-mathml-intent provee el comando \spshort (§3.4) que cubre estos casos, de esta manera podemos cargar babel[24] con todas sus funcionalidades sin problemas.

1.3 Fuentes tipográficas

Como spanish-mathml-intent 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[25], si no se especifica 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 la fuente Libertinus para el *modo texto* por medio del paquete libertinus-otf[26] y la fuente Erewhon-Math para el modo matemático cargada por el paquete fourier-otf[27]. Adicionalmente cargo la fuente Source Code Pro para el texto tipo «máquina de escribir» por medio del paquete sourcecodepro[28].

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

1.3.1 La llave `mathcal`

Algunos comandos, como \spAfig y \spPfig, usan \mathcal en su salida. Por defecto, con mathcal=true, el paquete spanish-mathml-intent busca la fuente NewCMMath-Regular.otf, parte del paquete newcomputermodern[29], para imprimir esas letras caligráficas con el estilo clásico; se recomienda tenerla instalada. Si no la encuentra, avisa con un "warning" y usa el \mathcal de la fuente matemática cargada. Con mathcal=false se usa siempre el \mathcal de la fuente matemática cargada, y la apariencia de la letra cambia. La misma fuente se usa para el símbolo del ángulo recto de \spangle (v. pág. 28). La llave se pasa al cargar el paquete:

```
\usepackage[mathcal=false]{spanish-mathml-intent}
```

1.4 El paquete `hyperref`

El paquete hyperref[30] 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{spanish-mathml-intent, hyperref}
\hypersetup
  {
    colorlinks         = true,
    pdfstartview       = FitH,
    pdfdisplaydoctitle = true,
    pdftitle           = {Test para el paquete spanish-mathml-intent},
    pdfinfo            =
      {
        Title   = {Test},
        Subject = {spanish-mathml-intent},
      },
  }
```

1.5 Interfaz de usuario

La interfaz de usuario de spanish-mathml-intent está formada solo por *comandos*: el comando de configuración `\setspintent` (§2) y 73 utilidades para escribir y verbalizar matemática escolar. El paquete no define entornos, contadores ni dimensiones públicas.

1.5.1 Lo que ofrece el paquete

Cada utilidad hace dos cosas a la vez: imprime la expresión en la salida PDF con el formato correcto y establece cómo la verbalizará NVDA/MathCAT, para que la lea como la diría un docente y no según la heurística general de NVDA/MathCAT. Las utilidades que reciben llaves `[`⟨*key* = *val*⟩`]` las toman de tres niveles, de menor a mayor prioridad: los valores por defecto del paquete, los establecidos de forma global con `\setspintent` y los del argumento opcional de cada comando. Un argumento que no cumple la forma documentada devuelve un "error" y se detiene la compilación del documento.

Los cuatro grandes (§3.1, §3.2, §3.3 y §3.4) son la base del paquete: `\spnum` escribe números, `\spunit` unidades de medida, `\spmoney` cifras con moneda y `\spshort` abreviaturas, acrónimos y ordinales. Los comandos `\spnewunit` y `\spunitalias` amplían el diccionario de `\spunit`. Muchas utilidades de los demás ejes procesan sus argumentos numéricos bajo `\spnum` y `\spunit`, por ejemplo `\spdiv`, `\spmul`, `\spcoord` y `\spang`.

1.5.2 Lo que ofrece cada eje

El resto de las utilidades se agrupan por eje temático, en el mismo orden en que se describen en este documento:

- *Generales* (§4): `\spsiglo` escribe siglos en números arábigos o romanos, `\sptime` horas y `\spdate` fechas.
- *Símbolos* (§5): `\spnewread` y `\spread` registran y usan la lectura de un símbolo, `\spusep` escribe paréntesis, corchetes y llaves sueltos, y `\spdsym` y `\spmsym` fijan cómo se imprime y se lee el símbolo de división y el de multiplicación.
- *Números* (§6): operaciones con `\spdiv` y `\spmul`, divisores y múltiplos con `\spnD` y `\spnM`, máximo común divisor y mínimo común múltiplo con `\spmcd` y `\spmcm`, números enteros con `\spnZ`, números mixtos con `\spmQ`, razones y proporciones con `\sprazon` y `\spProp`.
- *Geometría* (§7): puntos y coordenadas con `\spPoint` y `\spcoord`, vectores con `\spvec`, rectas, rayos y lados con `\sprecta`, `\sprayo`, `\spside` y `\spmside`, ángulos con `\spangle`, `\spmangle` y `\spang`, arcos con `\sparc` y `\spmarc`, polígonos con `\spPoly`, áreas y perímetros de figuras con `\spAfig` y `\spPfig`, circunferencias con `\spcirc`, `\spAcirc` y `\spPcirc`, radio y diámetro con `\sprad` y `\spdiam`, y volumen con `\spVol`.
- *Álgebra* (§8): intervalos con `\spinterval`, funciones con `\spfun`, dominio, recorrido y codominio con `\spdom`, `\sprec` y `\spcod`, composición de funciones con `\spfuncomp` y valor absoluto con `\spabs`.
- *Estadística* (§9): promedio con `\spProm`, medidas de tendencia central y de dispersión con `\spmedia`, `\spmediana`, `\spmoda`, `\spdesv`, `\spvarz`, `\sprango`, `\spriq`, `\spmin` y `\spmax`, cuartiles, deciles y percentiles con `\spcuartil`, `\spdecil` y `\spPerc`, frecuencias con `\spFrec`, probabilidad con `\spcase`, `\spOmega`, `\spPval`, `\spProb` y `\spcomp`, y conteo con `\spcomb` y `\spfact`.

©2026 by Pablo González L

1.5.3 Reglas para argumentos

Casi todas las utilidades envían su argumento a `spintent.lua`, que lo lee como *texto plano* y reconoce números, letras, signos y la sintaxis propia de cada comando. Por eso, salvo lo que se documente en cada comando, el {⟨*argumento*⟩} NO puede contener comandos de formato o de estilo, con o sin argumentos, como `\textstyle`, `\mathrm{#1}` o `\mathbf{#1}`, tal como se advierte en `\spnum`, `\spunit` y `\spmoney`. Un argumento que no cumple devuelve un "error" y se detiene la compilación del documento.

Hacen excepción las utilidades que tipografían su argumento directamente, sin procesamiento semántico adicional, y por eso aceptan cualquier expresión matemática: `\sprazon`, `\spProp`, `\spPval`, `\spfact`, `\spProm` y `\spabs`. En `\spinterval`, los extremos que no son numéricos, como una letra, un subíndice, una fracción o una raíz, también se imprimen tal cual.

1.5.4 Consulta rápida

La tabla 1 reúne, para lo que un docente escribe con más frecuencia, el comando que corresponde y la página donde se describe, con su sintaxis completa y sus llaves.

Tabla 1: Consulta rápida: qué quiero escribir y qué comando usar

Quiero escribir	Uso	Página
Generales		
un siglo	`\spsiglo{XXI}`	15
una hora	`$\sptime{23:45}$`	16
una fecha	`$\spdate{1981-01-02}$`	16
Los cuatro grandes		
un número	`$\spnum{3,14}$`	8
una medida	`$\spunit{23 m}$`	9
una cantidad de dinero	`$\spmoney{500}$`	11
una abreviatura, sigla u ordinal	`\spshort{prof.a}`	13
Símbolos		
la lectura de un símbolo	`$\spread{\cong}$`	16
Números		
una división	`$\spdiv{20}{26}$`	18
un máximo común divisor	`$\spmcd(2, 3, 6)$`	20
un número entero	`$\spnZ{-23}$`	22
una razón	`$\sprazon{3}{4}$`	23
una proporción	`$\spProp{3}{4}{6}{8}$`	23
un número mixto	`$\spmQ{2}{5}{7}$`	22
Geometría		
un punto	`$\spPoint{A}$`	24
unas coordenadas	`$\spcoord{-3, 4}$`	24
un vector	`$\spvec{AB}$`	25
una recta	`$\sprecta{AB}$`	25
un ángulo	`$\spangle{A}$`	28
un arco	`$\sparc{AB}$`	30
un polígono	`$\spPoly{ABC}$`	31
Álgebra		
un intervalo	`$\spinterval{(1,5)}$`	39
una función	`$\spfun{f(x)}$`	40
un dominio	`$\spdom{f}$`	41
un valor absoluto	`$\spabs{-5}$`	42
Estadística		
un promedio	`$\spProm{x}$`	43
un cuartil	`$\spcuartil{1}$`	44
una probabilidad	`$\spProb{A \cap B}$`	47
una combinatoria	`$\spcomb{5,2}$`	48
un factorial	`$\spfact{5}$`	48

1.6 Un primer documento

El siguiente documento está completo: se puede copiar, guardar con extensión `.tex` y compilar con LuaLaTeX. Usa comandos de varios ejes temáticos de este documento y los metadatos mínimos que necesita un *tagged* PDF.

```
\DocumentMetadata
  {
    lang=es-CL,
    pdfversion=2.0,
    pdfstandard=ua-2,
    tagging=on,
    tagging-setup={math/setup={mathml-SE}},
  }
\documentclass{article}
\usepackage[spanish]{babel}
\usepackage{spanish-mathml-intent, hyperref}
\hypersetup
  {
    pdfdisplaydoctitle = true,
    pdftitle           = {Guía de ejercicios},
  }

\begin{document}

\section*{Guía de ejercicios}

El número $\spnum{3,14}$ es una aproximación de $\pi$. El pasillo del
colegio mide $\spunit{23 m}$ de largo y un cuaderno cuesta $\spmoney{500}$.

Dibuje el triángulo $\spPoly{ABC}$ y marque el ángulo $\spangle{A}$. Sea
$\spfun{f(x)}$ una función y sean $A$ y $B$ dos sucesos: calcule
$\spProb{A \cap B}$.

Material preparado por la \spshort{prof.a} Pérez.

\end{document}
```

La tabla 2 muestra qué se imprime en el PDF y qué lee NVDA/MathCAT en cada caso; los valores salen de los ejemplos documentados en las secciones de cada comando.

Tabla 2: Lo que escribe el docente y lo que escucha el estudiante

Entrada	Salida PDF	Lectura NVDA
`$\spnum{3,14}$`	3,14	*«tres coma catorce»*
`$\spunit{23 m}$`	23 m	*«veintitrés metros»*
`$\spmoney{500}$`	$500	*«quinientos pesos»*
`\spshort{prof.a}`	Prof.ª	*«profesora»*
`$\spPoly{ABC}$`	$\triangle ABC$	*«triángulo A B C»*
`$\spfun{f(x)}$`	$f(x)$	*«f de x»*
`$\spProb{A \cap B}$`	$P(A \cap B)$	*«probabilidad de A y B»*

1.7 Cómo comprobar el resultado

Hay dos formas de comprobar un documento: *escucharlo* y *validar su estructura.* La primera dice si la lectura es la que se quiso; la segunda, si el PDF está bien etiquetado.

1.7.1 Escuchar la lectura

Abra el PDF con NVDA[6] y MathCAT[7] instalados, recorra el texto hasta la fórmula y compare lo que escucha con la columna «Lectura NVDA» de la tabla 2 y de las secciones de cada comando. Si una lectura no es la esperada, vea la §1.8.

1.7.2 Validar la estructura

Las siguientes órdenes, que se escriben en una terminal y no dentro del documento, extraen las etiquetas del PDF y las comprueban con `show-pdf-tags` y `rnv-wrapp`, y validan la norma PDF/UA-2 con veraPDF[4]:

```
show-pdf-tags --xml archivo.pdf | rnv-wrapp
verapdf --flavour ua2 --format text archivo.pdf
```

`show-pdf-tags` está en CTAN; si su distribución no lo incluye, instálelo con el gestor de paquetes.

Si tiene instaladas estas herramientas, la orden `spmmli-tool -c archivo.pdf` ejecuta ambas comprobaciones de una vez (§10).

©2026 by Pablo González L

1.7.3 Instalar `rnv-wrapp` y `veraPDF`

Ambas herramientas se instalan con los scripts del repositorio `rnv-vera-pdf-cli`. `veraPDF` necesita tener `Java` instalado. En `Linux` se ejecuta, en una terminal:

```
curl -sSL https://raw.githubusercontent.com/pablgonz/rnv-vera-pdf-cli/main/install-rnv.sh |
     bash -s -- --unattended
curl -sSL https://raw.githubusercontent.com/pablgonz/rnv-vera-pdf-cli/main/install-verapdf.
    sh | bash -s -- --unattended
```

En `Windows`, en `PowerShell`:

```
Invoke-WebRequest -Uri "https://raw.githubusercontent.com/pablgonz/rnv-vera-pdf-cli/main/
    install-rnv.ps1" -OutFile install-rnv.ps1
.\install-rnv.ps1 --unattended
Invoke-WebRequest -Uri "https://raw.githubusercontent.com/pablgonz/rnv-vera-pdf-cli/main/
    install-verapdf.ps1" -OutFile install-verapdf.ps1
.\install-verapdf.ps1 --unattended
```

La opción `--unattended` agrega las herramientas al `PATH`; después de instalar, abra una terminal nueva para que el cambio tenga efecto. El script de `veraPDF` también sirve en `macOS`; para `rnv-wrapp` el repositorio ofrece solo `Linux` y `Windows`.

1.8 Si algo no suena como esperaba

Revise, en este orden:

1. *El mensaje de error.* Si la compilación se detiene con un "error", el mensaje indica el comando y el argumento que no cumple la forma documentada (§2).
2. *La llave* `read-cmd`. Varios comandos ofrecen más de una lectura; `$\spPoint[read-cmd=school]{A_{1}}$` verbaliza «sub uno». Los valores posibles se describen en la sección de cada comando.
3. *La llave* `read-arg`. Reemplaza la lectura del argumento, `$\spread[read-arg={es-paralela-a}]{\parallel}$`; su valor siempre va entre llaves.
4. *La llave* `silent`. Con `silent=true` se silencia la lectura sin cambiar lo que se imprime; la validación del argumento se mantiene.
5. *La configuración global.* Se puede fijar para todo el documento, `\setspintent{spPoint}{read-cmd=school}`; la llave dada en el argumento opcional de un comando tiene prioridad sobre ella (§2).

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 spanish-mathml-intent. 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 spanish-mathml-intent 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 spanish-mathml-intent se implementa usando el módulo `l3keys` de `expl3`[31, 32]. Por diseño, las llaves que reciben un *valor* después del signo '=' NO aceptan *valores vacíos* y es obligatorio pasar uno válido; las llaves documentadas como ⟨*valor prohibido*⟩ no reciben *ningún valor*. En caso de omitir un *valor* válido o pasar un *valor vacío* a una llave que acepte *valor* o pasar un *valor* a una llave del tipo ⟨*valor prohibido*⟩ se devuelve un "error" y se detiene la compilación del documento.

En el modo `mathml-AF` (§10), dos fórmulas con *exactamente la misma fuente* reciben el mismo `MathML` adjunto, aunque una de ellas esté dentro de un grupo con otra configuración de `\setspintent`.

Por ejemplo, tras `{\setspintent{spmedia}{sub-index={x}}$\spmedia$}` la fórmula siguiente `$\spmedia$` se verbaliza igual que la primera. Esto se debe a que el núcleo de LaTeX asocia el `MathML` por el *hash* de la fuente y usa el primer fragmento que coincide; no ocurre en el modo `mathml-SE`. Para evitarlo, haga distinta la fuente de ambas fórmulas, por ejemplo pasando las llaves en el argumento opcional de cada comando.

3 The big four

El paquete spanish-mathml-intent 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.

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

3.1 El comando \spnum

\spnum
\spnum{⟨*número*⟩} \spnum{⟨*número*,*número*⟩} \spnum{⟨*número*.*número*⟩} \spnum{⟨*número*;*número*⟩}
\spnum[⟨*key* = *val*⟩]{⟨*número*,*número*;*número*⟩} \spnum[⟨*key* = *val*⟩]{⟨*número*,*número*;*número* e *exponente*⟩}
\spnum[⟨*key* = *val*⟩]{⟨*número*.*número*;*número*⟩} \spnum[⟨*key* = *val*⟩]{⟨*número*.*número*;*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 '−' seguidos 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 válidos, pero NO puede contener comandos con o sin argumentos como \textstyle, \mathrm{#1} o \mathbf{#1} por ejemplo. Cualquier entrada fuera de esta sintaxis devuelve un "error" y se detiene la compilación del documento.

3.1.1 Llaves para \spnum

`cifras-sep` = {⟨*true* | *false*⟩} por defecto: *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*⟩} por defecto: *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*⟩} por defecto: *coma*

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 ';' de la parte periódica.

`notation-sym` = {⟨*times* | *cdot*⟩} por defecto: *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 potencias de* 10*»*, no se verifica si la entrada cumple o no con la condición de estar escrita en *«notación científica»*.

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

©2026 by Pablo González L

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*...⟩} \spunit[⟨key = val⟩]{⟨número e exponente 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⟩} válido 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⟩], `$\spunit{23 m}$` 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 m s^2' en la salida PDF y NVDA/MathCAT verbalizará *«veintitrés metros segundos 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: `$\spunit{23 m/s^{2}}$` imprimirá '23 m/s^2' en la salida PDF y NVDA/MathCAT verbalizará *«veintitrés metros por segundos 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 válidos, 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 devuelve un "error" y se detiene 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*⟩} por defecto: *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*⟩} por defecto: *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*⟩} por defecto: *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

La tabla 3 resume las {⟨unidades⟩} y los {⟨alias⟩} soportados por \spunit.

Tabla 3: Unidades soportadas

Entrada	Alias / Variantes	Nombre	Salida PDF	Lectura NVDA
m	metro, metros	Metro	m	*«metro»*
g	gramo, gramos	Gramo	g	*«gramo»*
s	segundo, segundos	Segundo (Tiempo)	s	*«segundo»*
l	litro, l-litro	Litro (minúscula)	l	*«litro»*
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»*

Continúa en la siguiente página...

Tabla 3: *(Continuación: Unidades soportadas)*

Entrada	Alias / Variantes	Nombre	Salida PDF	Lectura NVDA
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»*
℧	mho	Mho / Siemens	℧	*«mho / siemens»*
′	minmin	Minuto (Arco)	′	*«minuto»*
″	segseg	Segundo (Arco)	″	*«segundo»*
u	u-masa	Unidad masa atómica	u	*«unidad masa atómica»*
		Unidades directas (sin alias)		
L	—	Litro (Mayúscula)	L	*«litro»*
ℓ	—	Litro (Cursiva)	ℓ	*«litro»*
w	—	Vatio (Minúscula)	w	*«vatio»*
°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»*
lx	—	Lux	lx	*«lux»*
cm	—	Centímetro	cm	*«centímetro»*
min	—	Minuto (Tiempo)	min	*«minuto»*
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»*
y	—	Año (Year)	y	*«año»*
°	—	Grado (Arco)	°	*«grado»*

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 cómo 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 devuelve un "error" y se detiene la compilación del documento.

La declaración de la {⟨*nueva unidad*⟩} es «global» y se almacena en el módulo `spintent.lua` como una {⟨*unidad*⟩} válida 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 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 devuelve un "error" y se detiene la compilación del documento.

La declaración de un {⟨*nuevo alias*⟩} es «global» y se almacena en el módulo `spintent.lua` como un {⟨*alias*⟩} válido 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[⟨key = val⟩]{⟨cifra⟩}`
`\spmoney{⟨cifra símbolo moneda⟩}` `\spmoney[⟨key = val⟩]{⟨cifra símbolo moneda⟩}`
`\spmoney{⟨cifra alias moneda⟩}` `\spmoney[⟨key = val⟩]{⟨cifra alias moneda⟩}`

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

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

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

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

Si se pasa un *«alias moneda»* del tipo `ISO` como `clp` o `usd` escrito en 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 moneda»* como '$', '€', '£' o un *«alias moneda»* como `peso`, `euro` o `libra` produce la salida PDF y verbalización estándar esperada para NVDA/MathCAT. Los *«alias moneda»* se pueden pasar con o sin tilde, por ejemplo `dolares` o `dólares`.

- Los *«alias moneda»* se deben escribir en minúsculas y los códigos `ISO` en minúsculas o todo en mayúsculas. Cualquier otra escritura, por ejemplo `PESO` o `Usd`, devuelve un "error" y se detiene la compilación del documento.
- La posición del *«símbolo 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 moneda»* es tomado de la fuente tipográfica definida para el *modo matemático* al momento de ejecutar el comando, o del *modo texto* si se establece `money-math=false`, se debe tener presente en caso de querer usar un *«símbolo moneda»* poco habitual que esté soportado por spanish-mathml-intent. El {⟨*argumento*⟩} puede contener espacios matemáticos válidos, pero NO puede contener comandos con o sin argumentos como `\textstyle`, `\mathrm{#1}` o `\mathbf{#1}` por ejemplo, con la única excepción de `\$`. La *coma* '`,`' o el *punto* '`.`' como *separador decimal* solo pueden aparecer una «única vez». Cualquier entrada fuera de esta sintaxis devuelve un "error" y se detiene 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*⟩} por defecto: *pesos*

Establece el *símbolo 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 moneda»* o el *«alias moneda»* pasado a esta llave debe ser alguno de los descritos en §3.3.2.

sub-units = {⟨*true* | *false*⟩} por defecto: *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»*.

Si la *«cifra numérica»* no tiene parte decimal, la llave sub-units no tiene efecto. Si la moneda en curso no admite *subunidades* se emite una advertencia y la llave tampoco tiene efecto.

money-math = {⟨*true* | *false*⟩} por defecto: *true*

Establece si el *símbolo moneda* 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 el *símbolo moneda* en la salida PDF; si se establece en false, se usará la fuente tipográfica definida para el *modo texto* para imprimir el *símbolo moneda* en la salida PDF.

3.3.2 Monedas y alias para \spmoney

La tabla 4 resume los *«símbolos moneda»* y los *«alias moneda»* soportados en el {⟨*argumento*⟩} de \spmoney.

Tabla 4: Monedas y alias soportados

Entrada	Alias / Variantes	Nombre	Salida PDF	Lectura NVDA
$	peso, pesos	peso	$	*«peso / pesos»*
dolar	dolares, dólar, dólares	dólar	$	*«dólar / dólares»*
€	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	—	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»*

©2026 by Pablo González L

3.4 El comando \spshort

\spshort

\spshort{⟨*abreviaturas* | *acrónimos* | *números ordinales*⟩}
\spshort[⟨*key* = *val*⟩]{⟨*abreviaturas* | *acrónimos* | *números ordinales*⟩}
\spshort[⟨*read-arg*={⟨*verbalización* 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 una nueva {⟨*abreviatura*⟩}, {⟨*acrónimo*⟩} o {⟨*número ordinal*⟩} que no se encuentre registrada por spanish-mathml-intent 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 volada»* '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 volada»*, 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 spanish-mathml-intent que trabajan *solo en modo texto*. Internamente utiliza la llave E provista por el paquete tagpdf[22] para el *expansion text*.

3.4.1 Llaves para \spshort

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

Establece la *verbalización del argumento* {⟨*comandos* LaTeX⟩} de la nueva {⟨*abreviatura*⟩}, {⟨*acrónimo*⟩} o {⟨*número ordinal*⟩} para NVDA/MathCAT. 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/MathCAT verbalizará *«octavo»*.

small-caps = {⟨*true* | *false*⟩} por defecto: *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/MathCAT verbalizará *«doctora»*.

3.4.2 Argumentos para \spshort

La tabla 5 resume las abreviaturas, expresiones latinas y títulos registrados en el módulo Lua spintent.lua.

Tabla 5: Abreviaturas soportadas

Entrada	Alias / Variantes	Salida PDF	Lectura NVDA
Abreviaturas con letras voladas			
dr.a	dr.ª	Dr.ª	*«doctora»*
dr.as	—	Dr.ª^{s}	*«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.º^{s}	*«números»*
c.ia	—	C.ª^{ia}	*«compañía»*
Abreviaturas comunes y expresiones latinas			
pág.	pag.	pág.	*«página»*
vol.	—	vol.	*«volumen»*
etc.	—	etc.	*«etcétera»*
et al.	et. al.	et al.	*«y otros»*

Continúa en la siguiente página...

Tabla 5: *(Continuación: Abreviaturas soportadas)*

Entrada	Alias / Variantes	Salida PDF	Lectura NVDA
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»*
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»*

La tabla 6 resume las siglas y acrónimos registrados en el módulo Lua `spintent.lua`.

Tabla 6: 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	—	ONG	*«organización no gubernamental»*
ongs	—	ONGs	*«organizaciones no gubernamentales»*
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»*

Continúa en la siguiente página...

Tabla 6: *(Continuación: Siglas y acrónimos)*

Entrada	Alias / Variantes	Salida PDF	Lectura NVDA
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»*

Según la RAE, las siglas no cambian en plural: se escribe «las ONG». La entrada ongs imprime «ONGs» a propósito, para que NVDA/MathCAT lea el plural *«organizaciones no gubernamentales»*; la entrada ong imprime «ONG» y lee el singular.

La tabla 7 resume ejemplos de números ordinales registrados en el módulo Lua spintent.lua.

Tabla 7: 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»*

Los números ordinales se verbalizan con palabras desde el 1 hasta el 99. Desde el 100 el ordinal no se convierte en palabras y se entrega a NVDA/MathCAT tal como se imprime, por ejemplo 100.o como *«100º»*.

4 Utilidades generales

Además de los *«cuatro grandes»* y sus asociados, el paquete spanish-mathml-intent 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 trabaja en *modo texto* y en *modo matemático*; \sptime y \spdate solo en *modo matemático*. Los tres reciben un *argumento obligatorio*, y un argumento que no cumple alguna de las formas descritas devuelve un "error" y se detiene la compilación del documento.

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 spanish-mathml-intent que trabaja en *modo texto* y en *modo matemático*. Internamente utiliza la llave alt provista por el paquete tagpdf en *modo texto* y usa \MathMLintent para el *modo matemático*.

Llaves para \spsiglo

`print-era` = {⟨*ac* | *dc* | *none*⟩} por defecto: *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 'xxi 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 'xxi 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 'xxi' 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 en *«modo texto»* la salida PDF '23:45' y NVDA/MathCAT verbalizará *«veintitrés cuarenta y cinco minutos»*.

El comando \spdate

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

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 spanish-mathml-intent provee cinco comandos \spnewread, \spread, \spusep, \spdsym y \spmsym para trabajar con *«símbolos matemáticos»* descritos en esta sección. Están pensados desde el punto de vista docente, para situaciones donde la lectura de un *«símbolo»* es relevante.

El comando \spnewread registra la verbalización de un símbolo y trabaja en *modo texto*; \spread usa un símbolo registrado, \spusep escribe paréntesis, corchetes y llaves sueltos, y \spdsym y \spmsym establecen cómo se imprime y se verbaliza el símbolo de división y el de multiplicación. Estos cuatro trabajan en *modo matemático*.

A diferencia de las otras utilidades implementadas por spanish-mathml-intent, 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 \spnewread

\spnewread \spnewread{⟨*símbolo*⟩}{⟨*verbalización* NVDA⟩}

El comando \spnewread trabaja en *modo texto* y recibe dos *argumentos obligatorios*: {⟨*símbolo*⟩} y {⟨*verbalización* NVDA⟩}, los cuales se almacenan de forma interna para ser utilizados como {⟨*argumento*⟩} en \spread. Solo se establece como NVDA/MathCAT verbalizará el {⟨*símbolo*⟩} al utilizarlo con \spread sin modificar la impresión de este.

El argumento {⟨*símbolo*⟩} no debe estar registrado. Si ya está registrado se devuelve un "error" y se detiene la compilación del documento. La declaración es «global» y se almacena en el módulo spintent.lua. Por ejemplo: \spnewread{\cong}{es-congruente-con} registrará '\cong' y cuando se utilice $\spread{\cong}$ se imprimirá '≅' en la salida PDF y NVDA/MathCAT verbalizará *«es congruente con»*.

El {⟨*símbolo*⟩} debe ser un único comando, incluso uno definido por el usuario. Por ejemplo: \newcommand{\xcuad}{x^2}, \spnewread{\xcuad}{x-al-cuadrado} registrará '\xcuad' y cuando se utilice $\spread{\xcuad}$ se imprimirá 'x^2' en la salida PDF y NVDA/MathCAT verbalizará *«x al cuadrado»*.

Los espacios y los guiones bajos de {⟨*verbalización* NVDA⟩} se convierten en guiones '-', por lo que {es congruente con} equivale a {es-congruente-con}. Los guiones repetidos se reducen a uno y se eliminan los del inicio y del final. Si el resultado queda vacío se devuelve un "error" y se detiene la compilación del documento.

El comando \spread

\spread \spread{⟨*símbolo*⟩} \spread[⟨*key* = *val*⟩]{⟨*símbolo*⟩}
\spread[⟨*read-arg*={⟨*verbalización* NVDA⟩}⟩]{⟨*comandos* LaTeX⟩}

El comando \spread trabaja en *modo matemático* y recibe el *argumento obligatorio* de la forma {⟨*símbolo*⟩} previamente registrado por el comando \spnewread o {⟨*comandos* LaTeX⟩} para ser usado junto a la llave read-arg.

©2026 by Pablo González L

Si el argumento es {⟨*símbolo*⟩}, `$\spread{\cong}$` imprimirá '≅' en la salida PDF y el argumento {⟨*verbalización* NVDA⟩} establecido al registrar {⟨*símbolo*⟩} con `\spnewread` se verbalizará en NVDA/MathCAT.

Si el argumento es {⟨*comandos* LaTeX⟩} y no se establece la llave `read-arg` se devuelve un "error" y se detiene la compilación del documento.

Llaves para `\spread`

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

Establece la *verbalización* del argumento {⟨*comandos* LaTeX⟩}. El valor de esta llave *siempre* debe pasarse entre llaves '{ }' y sobrescribe la verbalización del {⟨*símbolo*⟩} si está registrado previamente. Por ejemplo: `$\spread[read-arg={es-paralela-a}]{\parallel}$` imprimirá '∥' en la salida PDF y NVDA/MathCAT verbalizará *«es paralela a»*.

`silent` = {⟨*true* | *false*⟩} por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT del argumento {⟨*símbolo*⟩} o {⟨*comandos* LaTeX⟩} 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.

Al utilizar `silent` no es necesario que el argumento {⟨*símbolo*⟩} esté registrado o usar la llave `read-arg` con {⟨*comandos* LaTeX⟩}. Dependiendo de lo que contenga el {⟨*argumento*⟩}, puede que `silent` no sea suficiente, en ese caso utilice `read-arg={_-}`.

El comando `\spusep`

`\spusep` `\spusep{⟨\{|[|(|)|]|\}⟩}` `\spusep[⟨key = val⟩]{⟨\{|[|(|)|]|\}⟩}`

El comando `\spusep` trabaja en *modo matemático* y recibe el *argumento obligatorio* {⟨`\{|[|(|)|]|\}`⟩}, el cual debe ser *«solo uno»* de los paréntesis soportados: '(', ')', '[', ']', '\{' o '\}'.

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

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.

Hay un segundo motivo para usar `\spusep`. Varios comandos aceptan un *argumento opcional* entre corchetes, [⟨*key* = *val*⟩], o delimitado por paréntesis, (⟨*número, número, ...*⟩), o ambos, escrito justo después del nombre del comando. Si a continuación de un comando necesita un '[' o un '(' explícito que no sea ese argumento, escríbalo con `\spusep`; de lo contrario, LaTeX lo interpretará como el comienzo del argumento opcional. Por ejemplo, `$\spmedia\spusep{[}$` imprime el símbolo de `\spmedia` seguido de '[', mientras que `\spmedia[` haría que LaTeX lea lo que sigue como llaves [⟨*key* = *val*⟩].

Llaves para `\spusep`

`size` = {⟨*none* | *big* | *Big* | *bigg* | *Bigg*⟩} por defecto: *none*

Establece el *tamaño del paréntesis* pasado como {⟨*argumento*⟩} que se imprimirá en la salida PDF al ejecutar el comando sin modificar la verbalización de este. Si se establece en `none`, se imprime el {⟨*argumento*⟩} en el tamaño por defecto del modo matemático en curso; si se establece en `big`, se usa `\bigl`/`\bigr` antes del {⟨*argumento*⟩}; si se establece en `Big`, se usa `\Bigl`/`\Bigr` antes del {⟨*argumento*⟩}; si se establece en `bigg`, se usa `\biggl`/`\biggr` antes del {⟨*argumento*⟩}; y si se establece en `Bigg`, se usa `\Biggl`/`\Biggr` antes del {⟨*argumento*⟩}.

`silent` = {⟨*true* | *false*⟩} por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT 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`, sí verbalizará la salida del comando.

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 cómo 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*⟩} por defecto: *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 '{ }'.

©2026 by Pablo González L

`suffix` = {⟨*sufijo*⟩} por defecto: *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*⟩} por defecto: *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 en `slash`, imprimirá '/'.

`read-sym` = {⟨*dividido* | *dividido-por* | *dividido-entre*⟩} por defecto: *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 cómo 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*⟩} por defecto: *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*⟩} por defecto: *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*⟩} por defecto: *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*⟩} por defecto: *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»*.

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

Establece si se *imprime* el *símbolo de multiplicación*. Si se establece en `false`, 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 spanish-mathml-intent provee diez comandos para trabajar con números, descritos en esta sección: `\spdiv` y `\spmul` para la división y la multiplicación, `\spnD` y `\spnM` para los divisores y los múltiplos de un número, `\spmcd` y `\spmcm` para el máximo común divisor y el mínimo común múltiplo, `\spnZ` para los números enteros, `\spmQ` para los números mixtos y `\sprazon` y `\spProp` para las razones y las proporciones. Todos trabajan en *modo matemático*.

Los comandos `\spnD`, `\spnM`, `\spmcd` y `\spmcm` reciben un *argumento opcional delimitado por paréntesis*; los demás reciben *argumentos obligatorios*. Los argumentos numéricos de varios de ellos se procesan internamente bajo `\spnum`, de modo que siguen sus reglas de escritura (§3.1).

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 veintiséis»*.

©2026 by Pablo González L

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 (v. pág. 17).

wrap-parens = {⟨*true* | *false*⟩} por defecto: *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*⟩} por defecto: *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 '$20 \cdot 26$' en la salida PDF y NVDA/MathCAT verbalizará *«veinte por veintiséis»*.

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 (v. pág. 18).

wrap-parens = {⟨*true* | *false*⟩} por defecto: *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*⟩} por defecto: *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*⟩) \spnD[⟨*key* = *val*⟩](⟨*número*⟩) \spnD(⟨*letra*⟩) \spnD[⟨*key* = *val*⟩](⟨*letra*⟩)

El comando \spnD trabaja en *modo matemático* y recibe el *argumento opcional delimitado por paréntesis* de la forma (⟨*número*⟩) que se procesa internamente por \spnum o (⟨*letra*⟩) que se ejecuta de manera directa.

Si no se proporciona el *argumento opcional* [⟨*key* = *val*⟩] ni el *argumento opcional delimitado por paréntesis*, $\spnD$ imprimirá '$\mathrm{D}(n)$' en la salida PDF y NVDA/MathCAT verbalizará *«divisores de ene»*.

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

Si se usa el argumento (⟨*letra*⟩), $\spnD(m)$ imprimirá '$\mathrm{D}(m)$' en la salida PDF y NVDA/MathCAT verbalizará *«divisores de eme»*.

El número 0 no es válido: $\spnD(0)$ devuelve un "error" y se detiene la compilación del documento.

Llaves para \spnD

concept = {⟨*true* | *false*⟩} por defecto: *true*

Establece si se *verbaliza* en NVDA/MathCAT el concepto *«divisores de»* al ejecutar el comando sin modificar la impresión de este. Si se establece en true, NVDA/MathCAT verbalizará *«divisores de»* y el (⟨*argumento*⟩); si se establece en false, NVDA/MathCAT solo verbalizará el (⟨*argumento*⟩).

prefix = {⟨*prefijo*⟩} por defecto: *no usado*

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

result = {⟨*true* | *false*⟩} por defecto: *false*

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

result-num = {⟨*número natural*⟩} por defecto: *todos*

©2026 by Pablo González L

Establece la *cantidad* de divisores que se mostrarán en la salida PDF al establecer `result=true`. Si se establece en `2`, solo se imprimirán dos valores y luego se añadirá '...' en la salida PDF; si no se establece, se imprimirán todos los divisores, porque el conjunto es finito.

`silent` = {⟨*true* | *false*⟩} por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT del concepto *«divisores de»* 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 `\spnM`

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

El comando `\spnM` trabaja en *modo matemático* y recibe el *argumento opcional delimitado por paréntesis* de la forma (⟨*número*⟩) que se procesa internamente por `\spnum` o (⟨*letra*⟩) que se ejecuta de manera directa.

Si no se proporciona el *argumento opcional* [⟨*key* = *val*⟩] ni el *argumento opcional delimitado por paréntesis*, `$\spnM$` imprimirá 'M(n)' en la salida PDF y NVDA/MathCAT verbalizará *«múltiplos de ene»*.

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

Si se usa el argumento (⟨*letra*⟩), `$\spnM(m)$` imprimirá 'M(m)' en la salida PDF y NVDA/MathCAT verbalizará *«múltiplos de eme»*.

El número `0` no es válido: `$\spnM(0)$` devuelve un "error" y se detiene la compilación del documento.

Llaves para `\spnM`

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

Establece si se *verbaliza* en NVDA/MathCAT el concepto *«múltiplos de»* al ejecutar el comando sin modificar la impresión de este. Si se establece en `true`, NVDA/MathCAT verbalizará *«múltiplos de»* y el (⟨*argumento*⟩); si se establece en `false`, NVDA/MathCAT solo verbalizará el (⟨*argumento*⟩).

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

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

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

Establece si se *imprime* el resultado de calcular los *múltiplos* del argumento (⟨*número*⟩). 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 *«múltiplos de seis son seis, doce, dieciocho, veinticuatro, treinta puntos suspensivos»*; si se establece en `false`, no se imprimirá ni verbalizará el resultado de calcular los *múltiplos*.

`result-num` = {⟨*número natural*⟩} por defecto: *5*

Establece la *cantidad* de números que se mostrarán en la salida PDF al establecer `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.

`silent` = {⟨*true* | *false*⟩} por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT del concepto *«múltiplos de»* 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 `\spmcd`

`\spmcd` `\spmcd` `\spmcd(`⟨*número, número, ...*⟩`)` `\spmcd(`⟨*letra, letra, ...*⟩`)` `\spmcd[`⟨*key* = *val*⟩`](`⟨*número, número, ...*⟩`)`

El comando `\spmcd` trabaja en *modo matemático* y recibe el *argumento opcional delimitado por paréntesis* de la forma (⟨*número, número, ...*⟩) que se procesa internamente por `\spnum` o (⟨*letra, letra, ...*⟩) que se ejecuta de manera directa.

Si no se proporciona el *argumento opcional* [⟨*key* = *val*⟩] ni el *argumento opcional delimitado por paréntesis*, `$\spmcd$` imprimirá 'mcd' en la salida PDF y NVDA/MathCAT verbalizará *«máximo común divisor»*.

Si se usa el argumento (⟨*número, número, ...*⟩), `$\spmcd(2, 3, 6)$` imprimirá 'mcd(2, 3, 6)' en la salida PDF y NVDA/MathCAT verbalizará *«máximo común divisor entre dos, tres y seis»*.

Si se usa el argumento (⟨*letra, letra, ...*⟩), `$\spmcd(m, n)$` imprimirá 'mcd(m, n)' en la salida PDF y NVDA/MathCAT verbalizará *«máximo común divisor entre eme y ene»*.

Los argumentos (⟨*número, número, ...*⟩) o (⟨*letra, letra, ...*⟩) requieren al menos dos elementos.

©2026 by Pablo González L

Llaves para \spmcd

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

Establece si se *verbaliza* en NVDA/MathCAT el 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 *«máximo común divisor …»* o *«m c d …»* y el (⟨*argumento*⟩); si se establece en `false`, NVDA/MathCAT solo verbalizará el (⟨*argumento*⟩).

`read-sty` = {⟨*terse* | *clear*⟩} por defecto: *clear*

Establece el *concepto* que verbalizará NVDA/MathCAT al ejecutar el comando sin modificar la impresión de este. Si se establece en `terse`, NVDA/MathCAT verbalizará *«m c d»*; si se establece en `clear`, NVDA/MathCAT verbalizará *«máximo común divisor»*.

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

Establece si se usarán *mayúsculas* en la salida PDF al ejecutar el comando sin modificar la verbalización de este. Si se establece en `true`, se imprimirá 'MCD' en la salida PDF; si se establece en `false`, se imprimirá 'mcd' en la salida PDF.

`prefix` = {⟨*prefijo*⟩} por defecto: *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 '{ }'.

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

Establece si se *imprime* el resultado de calcular el *«máximo común divisor»* del argumento (⟨*número, número, …*⟩). 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 ni verbalizará el resultado de calcular el *«máximo común divisor»*.

`silent` = {⟨*true* | *false*⟩} por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT del concepto *«máximo común divisor …»* 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 \spmcm

`\spmcm` `\spmcm` `\spmcm(⟨número, número, …⟩)` `\spmcm(⟨letra, letra, …⟩)` `\spmcm[⟨key = val⟩](⟨número, número, …⟩)`

El comando `\spmcm` trabaja en *modo matemático* y recibe el *argumento opcional delimitado por paréntesis* de la forma (⟨*número, número, …*⟩) que se procesa internamente por `\spnum` o (⟨*letra, letra, …*⟩) que se ejecuta de manera directa.

Si no se proporciona el *argumento opcional* [⟨*key = val*⟩] ni el *argumento opcional delimitado por paréntesis*, `$\spmcm$` imprimirá 'mcm' en la salida PDF y NVDA/MathCAT verbalizará *«mínimo común múltiplo»*.

Si se usa el argumento (⟨*número, número, …*⟩), `$\spmcm(2, 3, 6)$` imprimirá 'mcm$(2,3,6)$' en la salida PDF y NVDA/MathCAT verbalizará *«mínimo común múltiplo entre dos, tres y seis»*.

Si se usa el argumento (⟨*letra, letra, …*⟩), `$\spmcm(m, n)$` imprimirá 'mcm(m,n)' en la salida PDF y NVDA/MathCAT verbalizará *«mínimo común múltiplo entre eme y ene»*.

Los argumentos (⟨*número, número, …*⟩) o (⟨*letra, letra, …*⟩) requieren al menos dos elementos.

Llaves para \spmcm

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

Establece si se *verbaliza* en NVDA/MathCAT el 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 *«mínimo común múltiplo …»* o *«m c m …»* y el (⟨*argumento*⟩); si se establece en `false`, NVDA/MathCAT solo verbalizará el (⟨*argumento*⟩).

`read-sty` = {⟨*terse* | *clear*⟩} por defecto: *clear*

Establece el *concepto* que verbalizará NVDA/MathCAT al ejecutar el comando sin modificar la impresión de este. Si se establece en `terse`, NVDA/MathCAT verbalizará *«m c m»*; si se establece en `clear`, NVDA/MathCAT verbalizará *«mínimo común múltiplo»*.

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

Establece si se usarán *mayúsculas* en la salida PDF al ejecutar el comando sin modificar la verbalización de este. Si se establece en `true`, se imprimirá 'MCM' en la salida PDF; si se establece en `false`, se imprimirá 'mcm' en la salida PDF.

`prefix` = {⟨*prefijo*⟩} por defecto: *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 '{ }'.

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

©2026 by Pablo González L

Establece si se *imprime* el resultado de calcular el *«mínimo común múltiplo»* del argumento ({⟨*número*, *número*, ...⟩}). Si se establece en `true`: `$\spmcm[result=true](2, 3, 6)$` imprimirá '$\mathrm{mcm}(2,3,6) = 6$' 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 ni verbalizará el resultado de calcular el *«mínimo común múltiplo»*.

`silent` = {⟨*true* | *false*⟩} por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT del concepto *«mínimo común múltiplo ...»* 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 \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 la demarcación e interacción con NVDA/MathCAT.

Si no se proporciona el *argumento opcional* [⟨*key = val*⟩], `$\spnZ{23}$` imprimirá '23' en la salida PDF y NVDA/MathCAT verbalizará *«veintitrés»*; `$\spnZ{-23}$` imprimirá '−23' y verbalizará *«menos veintitrés»*. El argumento debe ser un entero: con un decimal o una unidad, el comando devuelve un "error" y se detiene la compilación del documento.

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*⟩} por defecto: *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. Por ejemplo: `$\spnZ[wrap-parens=true]{-23}$` imprimirá '(−23)' en la salida PDF.

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

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 de los paréntesis '(...)'; si se establece en `false`, NVDA/MathCAT
no los verbalizará: `$\spnZ[wrap-parens=true,read-parens=false]{-23}$` verbalizará *«menos veintitrés»*.

El comando \spmQ

`\spmQ` `\spmQ{⟨entero⟩}{⟨numerador⟩}{⟨denominador⟩}` `\spmQ[⟨key = val⟩]{⟨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»*. Los tres deben ser *enteros*; el {⟨*numerador*⟩} y el {⟨*denominador*⟩} no pueden llevar signo y el {⟨*denominador*⟩} no puede ser cero. Cualquier otro valor devuelve un "error" y se detiene la compilación del documento.

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 spanish-mathml-intent. 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*⟩} por defecto: *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`, `$\spmQ[join-word=entero]{1}{5}{7}$` imprimirá '$1\frac{5}{7}$' en la salida PDF y NVDA/MathCAT verbalizará *«un entero cinco séptimos»*; si se establece en `enteros`, `$\spmQ[join-word=enteros]{2}{5}{7}$` imprimirá '$2\frac{5}{7}$' en la salida PDF y NVDA/MathCAT verbalizará *«dos enteros cinco séptimos»*; si se establece en `y`, `$\spmQ[join-word=y]{2}{5}{7}$` imprimirá '$2\frac{5}{7}$' en la salida PDF y NVDA/MathCAT verbalizará *«dos y cinco séptimos»*.

`use-spnZ` = {⟨*true* | *false*⟩} por defecto: *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, por ejemplo, `$\spmQ{2}{5}{7}$` se verbalizará *«dos y cinco séptimos»*; si se establece en `false`, los argumentos se tipografían directamente sin procesamiento semántico adicional y NVDA/MathCAT aplica sus propias reglas a la

©2026 by Pablo González L

fracción: `$\spmQ[use-spnZ=false]{2}{5}{7}$` también se verbalizará *«dos y cinco séptimos»*, de modo que la diferencia está solo en la estructura `MathML` de la salida `PDF`.

`use-dfrac` = {⟨*true* | *false*⟩} por defecto: *false*

Establece si la fracción del *número mixto* se imprime usando `\dfrac` en lugar de `\frac`. Si se establece en `true`, `$\spmQ[use-dfrac=true]{2}{5}{7}$` imprimirá la fracción en tamaño *display* '$2\frac{5}{7}$' 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*⟩} por defecto: *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*⟩} por defecto: *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.

`silent` = {⟨*true* | *false*⟩} por defecto: *false*

Establece si se *silencia* la verbalización en `NVDA/MathCAT` del *número mixto* sin modificar la impresión de este. Si se establece en `true`, `NVDA/MathCAT` NO verbalizará la salida del comando; si se establece en `false`, sí verbalizará la salida del comando. Un argumento inválido devuelve siempre un "error" y se detiene la compilación del documento, también con `silent=true`.

El comando `\sprazon`

`\sprazon` `\sprazon{`⟨*antecedente*⟩`}{`⟨*consecuente*⟩`}` `\sprazon[`⟨*key* = *val*⟩`]{`⟨*antecedente*⟩`}{`⟨*consecuente*⟩`}`

El comando `\sprazon` trabaja en *modo matemático* y recibe *dos argumentos obligatorios*: {⟨*antecedente*⟩} y {⟨*consecuente*⟩}, los tipografía como una fracción y devuelve una *«razón»*.

Si no se proporciona el *argumento opcional* [⟨*key* = *val*⟩], `$\sprazon{3}{4}$` imprimirá '$\frac{3}{4}$' en la salida `PDF` y `NVDA/MathCAT` verbalizará *«tres es a cuatro»*.

El {⟨*antecedente*⟩} y el {⟨*consecuente*⟩} se tipografían directamente, sin procesamiento semántico adicional.

Llaves para `\sprazon`

`silent` = {⟨*true* | *false*⟩} por defecto: *false*

Establece si se *silencia* la verbalización en `NVDA/MathCAT` de la *razón* pasada al comando sin modificar la impresión de esta. Si se establece en `true`, `NVDA/MathCAT` NO verbalizará la salida del comando; si se establece en `false`, sí verbalizará la salida del comando.

`use-dfrac` = {⟨*true* | *false*⟩} por defecto: *false*

Establece si la fracción de la *razón* se imprime usando `\dfrac` en lugar de `\frac`. Si se establece en `true`, `$\sprazon[use-dfrac=true]{3}{4}$` imprimirá la fracción en tamaño *display* '$\frac{3}{4}$' en la salida `PDF`; si se establece en `false`, la fracción se imprime con `\frac` siguiendo el tamaño matemático del contexto en curso.

`inline` = {⟨*true* | *false*⟩} por defecto: *false*

Establece si la *razón* se imprime en línea con dos puntos ':' en lugar de una fracción. Si se establece en `true`, `$\sprazon[inline=true]{3}{4}$` imprimirá '3 : 4' en la salida `PDF` y `NVDA/MathCAT` verbalizará *«tres es a cuatro»*; si se establece en `false`, la razón se imprime como una fracción.

La llave `use-dfrac` no tiene efecto cuando la llave `inline` se establece en `true`, porque en ese caso no se imprime una fracción.

El comando `\spProp`

`\spProp` `\spProp{`⟨*a*⟩`}{`⟨*b*⟩`}{`⟨*c*⟩`}{`⟨*d*⟩`}` `\spProp[`⟨*key* = *val*⟩`]{`⟨*a*⟩`}{`⟨*b*⟩`}{`⟨*c*⟩`}{`⟨*d*⟩`}`

El comando `\spProp` trabaja en *modo matemático* y recibe *cuatro argumentos obligatorios*: {⟨*a*⟩} y {⟨*b*⟩} forman la primera *razón*, {⟨*c*⟩} y {⟨*d*⟩} forman la segunda, y devuelve una *«proporción»* con las dos razones unidas por el signo igual.

Si no se proporciona el *argumento opcional* [⟨*key* = *val*⟩], `$\spProp{3}{4}{6}{8}$` imprimirá '$\frac{3}{4} = \frac{6}{8}$' en la salida `PDF` y `NVDA/MathCAT` verbalizará *«tres es a cuatro como seis es a ocho»*.

Los cuatro argumentos se tipografían directamente, sin procesamiento semántico adicional.

Llaves para `\spProp`

`silent` = {⟨*true* | *false*⟩} por defecto: *false*

Establece si se *silencia* la verbalización en `NVDA/MathCAT` de la *proporción* pasada al comando sin modificar la impresión de esta. Si se establece en `true`, `NVDA/MathCAT` NO verbalizará la salida del comando; si se establece en `false`, sí verbalizará la salida del comando.

©2026 by Pablo González L

use-dfrac = {⟨*true* | *false*⟩} por defecto: *false*

Establece si las fracciones de la *proporción* se imprimen usando `\dfrac` en lugar de `\frac`. Si se establece en `true`, `$\spProp[use-dfrac=true]{3}{4}{6}{8}$` imprimirá las fracciones en tamaño *display* '$\frac{3}{4} = \frac{6}{8}$, en la salida PDF; si se establece en `false`, las fracciones se imprimen con `\frac` siguiendo el tamaño matemático del contexto en curso.

inline = {⟨*true* | *false*⟩} por defecto: *false*

Establece si las *razones* de la *proporción* se imprimen en línea con dos puntos '`:`' en lugar de fracciones. Si se establece en `true`, `$\spProp[inline=true]{3}{4}{6}{8}$` imprimirá '3 : 4 = 6 : 8' en la salida PDF y NVDA/MathCAT verbalizará *«tres es a cuatro como seis es a ocho»*; si se establece en `false`, las razones se imprimen como fracciones.

La llave `use-dfrac` no tiene efecto cuando la llave `inline` se establece en `true`, porque en ese caso no se imprimen fracciones.

7 Utilidades para geometría

El paquete spanish-mathml-intent provee comandos para trabajar con *geometría* descritos en esta sección. Todos los comandos trabajan en *modo matemático* y reciben como {⟨*argumento*⟩} base una *única* letra, mayúscula en el caso de puntos, con un modificador del tipo subíndice/prima, un número o comando.

Algunos comandos reciben {⟨*argumento*⟩} único, como una "*letra*", o más de uno dentro del mismo, donde opcionalmente se pueden separar por comas '`,`'.

Las primas se deben pasar de forma directa '`'`' y los subíndices siempre deben escribirse usando llaves '`A_{...}`' y nunca sin estas '`A_n`' que funciona solo por coincidencia.

El comando `\spcoord`

`\spcoord`

`\spcoord{`⟨*abscisa, ordenada*⟩`}` `\spcoord[`⟨*key = val*⟩`]{`⟨*abscisa, ordenada*⟩`}`
`\spcoord{`⟨*abscisa; ordenada*⟩`}` `\spcoord[`⟨*key = val*⟩`]{`⟨*abscisa; ordenada*⟩`}`

El comando `\spcoord` trabaja en *modo matemático* y recibe el *argumento obligatorio* {⟨*abscisa, ordenada*⟩} o {⟨*abscisa; ordenada*⟩}, el cual se procesa internamente bajo el comando `\spnum` cuando es numérico no fraccionario.

Si no se proporciona el *argumento opcional* [⟨*key = val*⟩], `$\spcoord{-3, 4}$` imprimirá '$(-3, 4)$' en la salida PDF y NVDA/MathCAT verbalizará *«menos tres coma cuatro»*; `$\spcoord{x_{1},y_{1}}$` imprimirá '(x_1, y_1)' y verbalizará *«x subíndice uno coma y subíndice uno»*. La lectura del subíndice la decide NVDA/MathCAT, porque `\spcoord` no tiene la llave `read-cmd`.

El comando `\spcoord` solo acepta *un separador* '`,`' o '`;`' entre la abscisa y la ordenada, fuera de llaves: `$\spcoord{a_{1,2}, b}$` es válido porque la coma de las llaves no separa. Con '`;`' las comas son decimales, por ejemplo `$\spcoord{1,5; 2,5}$`.

Llaves para `\spcoord`

silent = {⟨*true* | *false*⟩} por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT 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 `\spPoint`

`\spPoint`

`\spPoint{`⟨*punto*⟩`}` `\spPoint[`⟨*key = val*⟩`]{`⟨*punto*⟩`}`
`\spPoint{`⟨*punto*⟩`}(`⟨*abscisa, ordenada*⟩`)` `\spPoint[`⟨*key = val*⟩`]{`⟨*punto*⟩`}(`⟨*abscisa, ordenada*⟩`)`
`\spPoint{`⟨*punto*⟩`}(`⟨*abscisa; ordenada*⟩`)` `\spPoint[`⟨*key = val*⟩`]{`⟨*punto*⟩`}(`⟨*abscisa; ordenada*⟩`)`

El comando `\spPoint` trabaja en *modo matemático* y recibe el *argumento obligatorio* {⟨*punto*⟩}, el cual es una *única letra mayúscula* con prima '`'`' o subíndice '`_{...}`' y el *argumento opcional delimitado por paréntesis* (⟨*abscisa, ordenada*⟩), el cual se procesa internamente bajo el comando `\spcoord`.

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»*.

Si se proporciona el *argumento opcional delimitado por paréntesis*, `$\spPoint{A}(-3,4)$` imprimirá '$A(-3, 4)$' en la salida PDF y NVDA/MathCAT verbalizará *«punto A menos tres coma cuatro»*; `$\spPoint{A_{1}}(-3,4)$` imprimirá '$A_1(-3, 4)$' y verbalizará *«punto A sub uno menos tres coma cuatro»*.

Llaves para `\spPoint`

read-cmd = {⟨*school* | *mathcat*⟩} por defecto: *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*⟩} por defecto: *no usado*

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

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

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

`silent` = {⟨*true* | *false*⟩} por defecto: *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 `\spvec`

`\spvec`

`\spvec`{⟨*letra*⟩}
`\spvec`{⟨*letra*⟩}(⟨*abscisa*, *ordenada*⟩)
`\spvec`[⟨*key* = *val*⟩]{⟨*letra*⟩}(⟨*abscisa*, *ordenada*⟩)
`\spvec`{⟨*punto*, *punto*⟩}
`\spvec`{⟨*punto*, *punto*⟩}(⟨*abscisa*, *ordenada*⟩)
`\spvec`[⟨*key* = *val*⟩]{⟨*punto*, *punto*⟩}(⟨*abscisa*, *ordenada*⟩)

El comando `\spvec` trabaja en *modo matemático* y recibe un *argumento obligatorio* de la forma {⟨*letra*⟩} minúscula o {⟨*punto*, *punto*⟩} junto al *argumento opcional delimitado por paréntesis* (⟨*abscisa*, *ordenada*⟩), el cual se procesa internamente bajo el comando `\spcoord`.

Si el *argumento* es de la forma {⟨*punto*, *punto*⟩} con prima(s) '`'`' o subíndice(s) '`_{...}`' opcionales se aplica `\overrightarrow` en la impresión; si el *argumento* es de la forma {⟨*letra*⟩} minúscula con subíndice '`_{...}`' opcional se aplica `\vec` en la impresión.

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

Si se proporciona el *argumento opcional delimitado por paréntesis*, `$\spvec{a}(-3,4)$` imprimirá '$\vec{a} = (-3, 4)$' en la salida `PDF` y `NVDA/MathCAT` verbalizará *«vector a es igual a menos tres coma cuatro»*; `$\spvec{AB}(-3,4)$` imprimirá '$\overrightarrow{AB} = (-3, 4)$' y verbalizará *«vector A B es igual a menos tres coma cuatro»*.

Llaves para `\spvec`

`read-cmd` = {⟨*school* | *mathcat*⟩} por defecto: *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»: `$\spvec{AB}$` se verbaliza *«vector 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*⟩} por defecto: *no usado*

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

`autofit` = {⟨*true* | *false*⟩} por defecto: *false*

Establece si se *utiliza* `\vphantom{b}` al ejecutar el comando cuando se utiliza el argumento {⟨*letra*⟩}. Si se establece en `true`, se aplicará `\vphantom{b}` al inicio del argumento {⟨*letra*⟩}; si se establece en `false`, no se aplicará `\vphantom{b}`.

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

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

`silent` = {⟨*true* | *false*⟩} por defecto: *false*

Establece si se *silencia* la verbalización en `NVDA/MathCAT` del concepto *«vector»* 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* de la forma {⟨*letra*⟩} o {⟨*punto*, *punto*⟩} con prima(s) '`'`' o subíndice(s) '`_{...}`' opcionales. Si el *argumento* es de la forma {⟨*punto*, *punto*⟩} se aplica `\overleftrightarrow` en la impresión; si el *argumento* es de la forma {⟨*letra*⟩}, mayúscula o minúscula, se imprimirá solo la {⟨*letra*⟩} sin aplicar `\overleftrightarrow`.

©2026 by Pablo González L

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*⟩} por defecto: *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*⟩} por defecto: *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*⟩} por defecto: *true*

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

`silent` = {⟨*true* | *false*⟩} por defecto: *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*⟩} con prima(s) '`'`' o subíndice(s) '`_{...}`' opcionales. 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*⟩} por defecto: *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*⟩} por defecto: *rayo*

Establece el *concepto* que verbalizará NVDA/MathCAT al 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*⟩} por defecto: *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*⟩} por defecto: *true*

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

`silent` = {⟨*true* | *false*⟩} por defecto: *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».

©2026 by Pablo González L

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⟩} con prima(s) '' o subíndice(s) '_{...}' opcionales. 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⟩} por defecto: *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»: $\spside{AB}$ 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⟩} por defecto: *lado*

Establece el *concepto* que verbalizará NVDA/MathCAT al 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⟩} por defecto: *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⟩} por defecto: *true*

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

read-arg = {⟨verbalización NVDA⟩} por defecto: *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 = {⟨true | false⟩} por defecto: *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⟩} con prima(s) '' o subíndice(s) '_{...}' opcionales. 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*, que es 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⟩} por defecto: *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»: $\spmside{AB}$ 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⟩} por defecto: *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»*.

©2026 by Pablo González L

`prefix` = {⟨*prefijo*⟩} por defecto: *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*⟩} por defecto: *true*

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

`read-arg` = {⟨*verbalización* NVDA⟩} por defecto: *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 'AB' en la salida PDF y NVDA/MathCAT verbalizará *«medida de la cuerda A B»*.

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

Establece *si se imprime* la notación visual 'm' antes de aplicar `\overline` al {⟨*argumento*⟩}. Si se establece en `true`, `$\spmside[print-sym=true]{AB}$` 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` = {⟨*true* | *false*⟩} por defecto: *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* de la forma {⟨*punto* | *letra griega* | *número*⟩} o {⟨*punto, punto, punto*⟩} aplicando `\angle` o `\rightanglesqr` en la impresión. Solo los argumentos de la forma {⟨*punto*⟩} o {⟨*punto, punto, punto*⟩} admiten prima(s) '' o subíndice(s) '_{...}' opcionales.

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»*; `$\spangle{1}$` imprimirá '$\angle 1$' en la salida PDF y NVDA/MathCAT verbalizará *«ángulo uno»* y `$\spangle{AOB}$` imprimirá '$\angle AOB$' en la salida PDF y NVDA/MathCAT verbalizará *«ángulo A O B»*.

Si no se tiene instalada la fuente matemática `NewCMMath-Regular.otf`, parte del paquete `newcomputermodern`[29], en el sistema o no se provee una fuente matemática con soporte para `\rightanglesqr`, se usará `\rightangle` en su reemplazo.

Llaves para `\spangle`

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

Establece la *forma* en la que NVDA/MathCAT verbalizará el argumento {⟨*punto*⟩} o {⟨*punto, punto, punto*⟩}. Si se establece en `school`, NVDA/MathCAT verbaliza las letras del argumento sin el prefijo «mayúscula»: `$\spangle{AOB}$` 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*⟩} por defecto: *no usado*

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

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

Establece si se *usará* `\rightanglesqr` en la impresión añadiendo el término *«recto»* a la verbalización en NVDA/MathCAT. Si se establece en `true`, `$\spangle[recto=true]{A}$` imprimirá '$\llcorner A$' en la salida PDF y NVDA/MathCAT verbalizará *«ángulo recto en A»*, `$\spangle[recto=true]{AOB}$` imprimirá '$\llcorner AOB$' 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»*.

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

Establece si se *verbaliza* en NVDA/MathCAT el concepto *«ángulo»* al ejecutar el comando sin modificar la impresión de este. Si se establece en `true`, NVDA/MathCAT verbalizará el concepto *«ángulo»* y el {⟨*argumento*⟩};

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

`silent` = {⟨*true* | *false*⟩} por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT del concepto *«ángulo»* 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 `\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* de la forma {⟨*punto* | *letra griega* | *número*⟩} o {⟨*punto*, *punto*, *punto*⟩} aplicando `\measuredangle` o `\measuredrightangle` en la impresión. Solo los argumentos de la forma {⟨*punto*⟩} o {⟨*punto*, *punto*, *punto*⟩} admiten prima(s) '' o subíndice(s) '_{...}' opcionales.

Si no se proporciona el *argumento opcional* [⟨*key* = *val*⟩], `$\spmangle{A}$` imprimirá '$\measuredangle A$' en la salida PDF y NVDA/MathCAT verbalizará *«medida del ángulo en A»*; `$\spmangle{\alpha}$` imprimirá '$\measuredangle\alpha$' en la salida PDF y NVDA/MathCAT verbalizará *«medida del ángulo alfa»*; `$\spmangle{1}$` imprimirá '$\measuredangle 1$' en la salida PDF y NVDA/MathCAT verbalizará *«medida del ángulo uno»* y `$\spmangle{AOB}$` imprimirá '$\measuredangle AOB$' en la salida PDF y NVDA/MathCAT verbalizará *«medida del ángulo A O B»*.

La distinción entre `\spangle` y `\spmangle` es semántica: `\spangle` representa el *objeto geométrico* '$\angle AOB$'; `\spmangle` representa su *medida*, que es un valor en grados.

Llaves para `\spmangle`

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

Establece la *forma* en la que NVDA/MathCAT verbalizará el argumento {⟨*punto*⟩} o {⟨*punto*, *punto*, *punto*⟩}. Si se establece en `school`, NVDA/MathCAT verbaliza las letras del argumento sin el prefijo «mayúscula»: `$\spmangle{AOB}$` 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*⟩} por defecto: *no usado*

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

`recto` = {⟨*true* | *false*⟩} por defecto: *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 '$\measuredrightangle A$' en la salida PDF y NVDA/MathCAT verbalizará *«medida del ángulo recto en A»*, `$\spmangle[recto=true]{AOB}$` imprimirá '$\measuredrightangle AOB$' 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-sym` = {⟨*true* | *false*⟩} por defecto: *false*

Establece *si se imprime* la notación visual 'm' antes del argumento pasado al comando sin modificar la verbalización de este. Si se establece en `true`, `$\spmangle[print-sym=true]{AOB}$` imprimirá 'm$\angle AOB$' en la salida PDF; si se establece en `false` imprimirá '$\angle AOB$' en la salida PDF. En ambos casos NVDA/MathCAT verbalizará *«medida del ángulo A O B»*.

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

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

`silent` = {⟨*true* | *false*⟩} por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT del término *«medida del»*, del concepto *«ángulo»* 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 `\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.

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»*.

©2026 by Pablo González L

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* de la forma {⟨punto, punto⟩} o {⟨punto, punto, punto⟩} con prima(s) '' o subíndice(s) '_{...}' opcionales aplicando \widearc en la impresión.

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

El acento \widearc es proporcionado por algunos paquetes como fourier-otf o kpfonts-otf, en caso de no estar disponible se utilizará una versión interna para este.

Llaves para \sparc

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

Establece la *forma* en la que NVDA/MathCAT verbalizará el argumento {⟨punto, punto⟩} o {⟨punto, punto, punto⟩}. Si se establece en school, NVDA/MathCAT verbaliza las letras del argumento sin el prefijo «mayúscula»: $\sparc{AB}$ 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⟩} por defecto: *no usado*

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

concept = {⟨true | false⟩} por defecto: *true*

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

silent = {⟨true | false⟩} por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT del concepto *«arco»* 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 \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* de la forma {⟨punto, punto⟩} o {⟨punto, punto, punto⟩} con prima(s) '' o subíndice(s) '_{...}' opcionales colocando 'm' antes de aplicar \widearc en la impresión.

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

La distinción entre \sparc y \spmarc es semántica: \sparc representa el *objeto geométrico* '$\overset{\frown}{AB}$'; \spmarc representa su *medida*, que es un valor en grados.

Llaves para \spmarc

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

Establece la *forma* en la que NVDA/MathCAT verbalizará el argumento {⟨punto, punto⟩} o {⟨punto, punto, punto⟩}. Si se establece en school, NVDA/MathCAT verbaliza las letras del argumento sin el prefijo «mayúscula»: $\spmarc{AB}$ 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⟩} por defecto: *no usado*

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

concept = {⟨true | false⟩} por defecto: *true*

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

silent = {⟨true | false⟩} por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT del término *«medida del»*, el concepto *«arco»* y el {⟨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.

©2026 by Pablo González L

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⟩} con prima(s) '' ' o subíndice(s) '_{...}' opcionales. El *símbolo* usado en la impresión se maneja por medio de la llave print-sym y la verbalización del concepto asociado se determina acorde al número de puntos en el argumento y la llave concept.

Si son tres puntos se aplica \triangle y añade el concepto *«triángulo»* a la verbalización; si son cuatro puntos se aplica \mdlgwhtsquare y añade el concepto *«cuadrado»* a la verbalización; si son cinco o más puntos no se aplica ningún símbolo y se añade a la verbalización el nombre del polígono: *«pentágono»*, *«hexágono»*, *«heptágono»* u *«octágono»* de cinco a ocho puntos, y *«polígono»* desde nueve puntos.

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

Llaves para \spPoly

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

Establece la *forma* en la que NVDA/MathCAT verbalizará el argumento {⟨punto, punto, punto, ...⟩}. Si se establece en school, NVDA/MathCAT verbaliza las letras del argumento sin el prefijo «mayúscula»: $\spPoly{ABC}$ 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⟩} por defecto: *no usado*

Establece el *prefijo* que será verbalizado por NVDA/MathCAT *antes* de verbalizar el {⟨argumento⟩} o el concepto *«triángulo»*, *«cuadrado»*, *«pentágono»*, *«hexágono»*, *«heptágono»*, *«octágono»* o *«polígono»* establecido por la llave concept. El valor de esta llave *siempre* debe pasarse entre llaves '{ }'.

concept = {⟨true | false⟩} por defecto: *true*

Establece si se *verbaliza* en NVDA/MathCAT el concepto, que es *«triángulo»*, *«cuadrado»*, *«pentágono»*, *«hexágono»*, *«heptágono»*, *«octágono»* o *«polígono»*, acorde al número de puntos en el {⟨argumento⟩} al ejecutar el comando sin modificar la impresión de este. Si se establece en true, NVDA/MathCAT verbalizará el concepto y el {⟨argumento⟩}; si se establece en false, NVDA/MathCAT solo verbalizará el {⟨argumento⟩} acorde al valor establecido por la llave read-cmd.

print-sym = {⟨true | false⟩} por defecto: *true*

Establece si se *usará* \triangle o \mdlgwhtsquare en la impresión sin modificar la verbalización de este. Si se establece en true, $\spPoly[print-sym=true]{ABC}$ imprimirá '$\triangle ABC$' en la salida PDF y NVDA/MathCAT verbalizará *«triángulo A B C»*; $\spPoly[print-sym=true]{ABCD}$ imprimirá '$\square ABCD$' en la salida PDF y NVDA/MathCAT verbalizará *«cuadrado A B C D»*; si se establece en false, solo se imprimirá el {⟨argumento⟩} en la salida PDF y la verbalización será la misma.

Esta llave no tiene efecto con cinco o más puntos, porque no hay un símbolo definido para ellos.

silent = {⟨true | false⟩} por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT del concepto establecido por la llave concept 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 \spAfig

\spAfig
`\spAfig` `\spAfig(⟨punto, punto, punto, ...⟩)`
`\spAfig[⟨key = val⟩]` `\spAfig[⟨key = val⟩](⟨punto, punto, punto, ...⟩)`
`\spAfig(⟨1 | 2 | 3 | ... | n⟩)` `\spAfig(⟨F | P | B | L | T⟩)`
`\spAfig[⟨key = val⟩](⟨1 | 2 | 3 | ... | n⟩)` `\spAfig[⟨key = val⟩](⟨F | P | B | L | T⟩)`

El comando \spAfig trabaja en *modo matemático* y recibe el *argumento opcional delimitado por paréntesis* de la forma (⟨1 | 2 | 3 | ...| n⟩), (⟨punto, punto, punto, ...⟩) o (⟨F | P | B | L | T⟩) aplicando \mathcal{A} a la impresión. El *símbolo* usado en la impresión se maneja por medio de la llave print-sym y la verbalización del concepto asociado se determina acorde al (⟨argumento⟩) y la llave concept.

Si se proporciona el *argumento* (⟨1 | 2 | 3 | ...| n⟩), $\spAfig(1)$ imprimirá '$\mathcal{A}_1$' en la salida PDF y NVDA/MathCAT verbalizará *«área uno»*; $\spAfig(n)$ imprimirá '$\mathcal{A}_n$' en la salida PDF y NVDA/MathCAT verbalizará *«área ene»*.

Si se proporciona el *argumento* (⟨punto, punto, punto, ...⟩), $\spAfig(ABC)$ imprimirá '$\mathcal{A}_{\triangle ABC}$' en la salida PDF y NVDA/MathCAT verbalizará *«área del triángulo A B C»*; $\spAfig(ABCD)$ imprimirá '$\mathcal{A}_{\square ABCD}$' en la salida PDF y NVDA/MathCAT verbalizará *«área del cuadrado A B C D»*; $\spAfig(ABCDE)$ imprimirá '$\mathcal{A}_{ABCDE}$' en la salida PDF y NVDA/MathCAT verbalizará *«área del pentágono A B C D E»*. Con cinco o más puntos no

se imprime ningún símbolo y se añade a la verbalización el nombre del polígono: *«pentágono»*, *«hexágono»*, *«heptágono»* u *«octágono»* de cinco a ocho puntos, y *«polígono»* desde nueve puntos.

Si se proporciona el *argumento* (⟨F | P | B | L | T⟩), `$\spAfig(F)$` imprimirá '$\mathcal{A}_F$' en la salida PDF y NVDA/MathCAT verbalizará *«área de la figura»*; `$\spAfig(F_{1})$` imprimirá '$\mathcal{A}_{F_1}$' en la salida PDF y NVDA/MathCAT verbalizará *«área de la figura uno»*.

Si no se proporciona el *argumento opcional* [⟨*key* = *val*⟩] ni el *argumento opcional delimitado por paréntesis*, `$\spAfig$` imprimirá '$\mathcal{A}$' en la salida PDF y NVDA/MathCAT verbalizará *«área»*.

El *argumento* (⟨*punto*, *punto*, *punto*, ...⟩) acepta prima(s) '′' o subíndice(s) '_{...}'. El *argumento* (⟨F | P | B | L | T⟩) solo acepta subíndices '_{...}' y tiene una impresión y verbalización fija para cada letra: F *«figura»*, P *«polígono»*, B *«basal»*, L *«lateral»* y T *«total»*.

Llaves para \spAfig

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

Establece la *forma* en la que NVDA/MathCAT verbalizará el argumento. Si se establece en `school`, NVDA/MathCAT verbaliza las letras del argumento sin el prefijo «mayúscula»: `$\spAfig(ABC)$` 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*⟩} por defecto: *no usado*

Establece el *prefijo* que será verbalizado por NVDA/MathCAT *antes* de verbalizar el término *«área del»* y el concepto *«nombre de la figura»* acorde al (⟨*argumento*⟩) pasado al comando. El valor de esta llave *siempre* debe pasarse entre llaves '{ }'.

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

Establece si se *verbaliza* en NVDA/MathCAT el término *«área del»* y el concepto *«nombre de la figura»* acorde al (⟨*argumento*⟩) pasado al comando sin modificar la impresión de este. Si se establece en `true`, NVDA/MathCAT verbalizará el término *«área del»*, el concepto *«nombre de la figura»* y el (⟨*argumento*⟩); si se establece en `false`, NVDA/MathCAT solo verbalizará el (⟨*argumento*⟩) acorde al valor establecido por la llave `read-cmd`.

Utilizar `concept=false` sin el (⟨*argumento*⟩) es equivalente a usar `silent=true`.

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

Sobrescribe el *concepto «nombre de la figura»* que verbalizará NVDA/MathCAT al ejecutar el comando sin modificar la impresión de este, cuando se utiliza el argumento (⟨*punto*, *punto*, *punto*, ...⟩). Con `concept=true` se mantiene el término *«área del»* antes del valor; con `concept=false` se verbaliza solo el valor de esta llave. El valor de esta llave *siempre* debe pasarse entre llaves '{ }' y se debe usar en conjunto con la llave `concept`. Por ejemplo: `$\spAfig[read-arg={trapecio}](ABCD)$` imprimirá '$\mathcal{A}_{\square ABCD}$' en la salida PDF y NVDA/MathCAT verbalizará *«área del trapecio A B C D»*.

Esta llave no tiene efecto y emite un aviso con los argumentos (⟨F | P | B | L | T⟩) y (⟨*1* | *2* | *3* | ... | *n*⟩).

`concept*` ⟨*valor prohibido*⟩ por defecto: *no usado*

Establece si se *silencia* la verbalización en NVDA/MathCAT del concepto *«nombre de la figura»* acorde al (⟨*argumento*⟩) pasado al comando sin modificar la impresión de este. Si se establece, NVDA/MathCAT verbalizará el término *«área»* y el (⟨*argumento*⟩) acorde al valor establecido por la llave `read-cmd`.

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

Establece si se *antepone* la palabra «sub» al verbalizar los argumentos (⟨*1* | *2* | *3* | ... | *n*⟩) y (⟨F | P | B | L | T⟩) para NVDA/MathCAT. Si se utiliza (⟨*1* | *2* | *3* | ... | *n*⟩) y se establece en `true`, `$\spAfig[read-sub=true](1)$` imprimirá '$\mathcal{A}_1$' en la salida PDF y NVDA/MathCAT verbalizará *«área sub uno»*; si se establece en `false`, la impresión será la misma y se verbaliza *«área uno»*.

Si se utiliza (⟨F | P | B | L | T⟩) y se establece en `true`, `$\spAfig[read-sub=true](F_{1})$` imprimirá '$\mathcal{A}_{F_1}$' en la salida PDF y NVDA/MathCAT verbalizará *«área de la figura sub uno»*; si se establece en `false`, la impresión será la misma y se verbaliza *«área de la figura uno»*.

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

Establece si se *usará* `\triangle` o `\mdlgwhtsquare` en la impresión para los argumentos (⟨*punto*, *punto*, *punto*, ...⟩) y (⟨*1* | *2* | *3* | ... | *n*⟩) pasado al comando sin modificar la verbalización de este. Si se utiliza (⟨*punto*, *punto*, *punto*, ...⟩) se infiere el *símbolo* acorde al número de puntos; si se establece en `auto` se imprime y si se establece en `none` se omite: `$\spAfig[print-sym=auto](ABC)$` y `$\spAfig[print-sym=none](ABC)$` imprimirán '$\mathcal{A}_{\triangle ABC}$' y '$\mathcal{A}_{ABC}$' en la salida PDF y en ambos casos NVDA/MathCAT verbalizará *«área del triángulo A B C»*.

Si se utiliza (⟨*1* | *2* | *3* | ... | *n*⟩) los valores `auto` y `none` quedan sin efecto; si se establece en `triangle` o `square` anida el argumento bajo el *símbolo*: `$\spAfig[print-sym=triangle](1)$` imprimirá '$\mathcal{A}_{\triangle_1}$' en la salida PDF y NVDA/MathCAT verbalizará *«área del triángulo uno»*; `$\spAfig[print-sym=square](1)$` imprimirá '$\mathcal{A}_{\square_1}$' en la salida PDF y NVDA/MathCAT verbalizará *«área del cuadrado uno»*.

Esta llave no se puede utilizar con el argumento (⟨F | P | B | L | T⟩).

©2026 by Pablo González L

`silent` = {⟨*true* | *false*⟩} por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT del término *«área del»*, el concepto *«nombre de la figura»* y el {⟨*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 `\spPfig`

`\spPfig`

`\spPfig` `\spPfig`
`\spPfig[`⟨*key* = *val*⟩`]`
`\spPfig(`⟨*1* | *2* | *3* | ... | *n*⟩`)`
`\spPfig[`⟨*key* = *val*⟩`](`⟨*1* | *2* | *3* | ... | *n*⟩`)`
`\spPfig(`⟨*punto*, *punto*, *punto*, ...⟩`)`
`\spPfig[`⟨*key* = *val*⟩`](`⟨*punto*, *punto*, *punto*, ...⟩`)`
`\spPfig(`⟨F | P | B | L | T⟩`)`
`\spPfig[`⟨*key* = *val*⟩`](`⟨F | P | B | L | T⟩`)`

El comando `\spPfig` trabaja en *modo matemático* y recibe el *argumento opcional delimitado por paréntesis* de la forma (⟨*1* | *2* | *3* | ... | *n*⟩), (⟨*punto*, *punto*, *punto*, ...⟩) o (⟨F | P | B | L | T⟩) aplicando `\mathcal{P}` a la impresión. El *símbolo* usado en la impresión se maneja por medio de la llave `print-sym` y la verbalización del concepto asociado se determina acorde al (⟨*argumento*⟩) y la llave `concept`.

Si se proporciona el *argumento* (⟨*1* | *2* | *3* | ... | *n*⟩), `$\spPfig(1)$` imprimirá '$\mathcal{P}_1$' en la salida PDF y NVDA/MathCAT verbalizará *«perímetro uno»*; `$\spPfig(n)$` imprimirá '$\mathcal{P}_n$' en la salida PDF y NVDA/MathCAT verbalizará *«perímetro ene»*.

Si se proporciona el *argumento* (⟨*punto*, *punto*, *punto*, ...⟩), `$\spPfig(ABC)$` imprimirá '$\mathcal{P}_{\triangle ABC}$' en la salida PDF y NVDA/MathCAT verbalizará *«perímetro del triángulo A B C»*; `$\spPfig(ABCD)$` imprimirá '$\mathcal{P}_{\square ABCD}$' en la salida PDF y NVDA/MathCAT verbalizará *«perímetro del cuadrado A B C D»*; `$\spPfig(ABCDE)$` imprimirá '$\mathcal{P}_{ABCDE}$' en la salida PDF y NVDA/MathCAT verbalizará *«perímetro del pentágono A B C D E»*. Con cinco o más puntos no se imprime ningún símbolo y se añade a la verbalización el nombre del polígono: *«pentágono»*, *«hexágono»*, *«heptágono»* u *«octágono»* de cinco a ocho puntos, y *«polígono»* desde nueve puntos.

Si se proporciona el *argumento* (⟨F | P | B | L | T⟩), `$\spPfig(F)$` imprimirá '$\mathcal{P}_F$' en la salida PDF y NVDA/MathCAT verbalizará *«perímetro de la figura»*; `$\spPfig(F_{1})$` imprimirá '$\mathcal{P}_{F_1}$' en la salida PDF y NVDA/MathCAT verbalizará *«perímetro de la figura uno»*.

Si no se proporciona el *argumento opcional* [⟨*key* = *val*⟩] ni el *argumento opcional delimitado por paréntesis*, `$\spPfig$` imprimirá '$\mathcal{P}$' en la salida PDF y NVDA/MathCAT verbalizará *«perímetro»*.

El *argumento* (⟨*punto*, *punto*, *punto*, ...⟩) acepta prima(s) '′' o subíndice(s) '`_{...}`'. El *argumento* (⟨F | P | B | L | T⟩) solo acepta subíndices '`_{...}`' y tiene una impresión y verbalización fija para cada letra: F *«figura»*, P *«polígono»*, B *«basal»*, L *«lateral»* y T *«total»*.

Llaves para `\spPfig`

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

Establece la *forma* en la que NVDA/MathCAT verbalizará el argumento. Si se establece en `school`, NVDA/MathCAT verbaliza las letras del argumento sin el prefijo «mayúscula»: `$\spPfig(ABC)$` 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*⟩} por defecto: *no usado*

Establece el *prefijo* que será verbalizado por NVDA/MathCAT *antes* de verbalizar el término *«perímetro del»* y el concepto *«nombre de la figura»* acorde al (⟨*argumento*⟩) pasado al comando. El valor de esta llave *siempre* debe pasarse entre llaves '`{ }`'.

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

Establece si se *verbaliza* en NVDA/MathCAT el término *«perímetro del»* y el concepto *«nombre de la figura»* acorde al (⟨*argumento*⟩) pasado al comando sin modificar la impresión de este. Si se establece en `true`, NVDA/MathCAT verbalizará el término *«perímetro del»*, el concepto *«nombre de la figura»* y el (⟨*argumento*⟩); si se establece en `false`, NVDA/MathCAT solo verbalizará el (⟨*argumento*⟩) acorde al valor establecido por la llave `read-cmd`.

Utilizar `concept=false` sin el (⟨*argumento*⟩) es equivalente a usar `silent=true`.

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

Sobrescribe el *concepto «nombre de la figura»* que verbalizará NVDA/MathCAT al ejecutar el comando sin modificar la impresión de este, cuando se utiliza el argumento (⟨*punto*, *punto*, *punto*, ...⟩). Con `concept=true` se mantiene el término *«perímetro del»* antes del valor; con `concept=false` se verbaliza solo el valor de esta llave. El valor de esta llave *siempre* debe pasarse entre llaves '`{ }`' y se debe usar en conjunto con la llave `concept`. Por ejemplo: `$\spPfig[read-arg={trapecio}](ABCD)$` imprimirá '$\mathcal{P}_{\square ABCD}$' en la salida PDF y NVDA/MathCAT verbalizará *«perímetro del trapecio A B C D»*.

Esta llave no tiene efecto y emite un aviso con los argumentos (⟨F | P | B | L | T⟩) y (⟨*1* | *2* | *3* | ... | *n*⟩).

`concept*` ⟨*valor prohibido*⟩ por defecto: *no usado*

Establece si se *silencia* la verbalización en NVDA/MathCAT del concepto *«nombre de la figura»* acorde al (⟨*argumento*⟩) pasado al comando sin modificar la impresión de este. Si se establece, NVDA/MathCAT verbalizará el término *«perímetro»* y el (⟨*argumento*⟩) acorde al valor establecido por la llave `read-cmd`.

©2026 by Pablo González L

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

Establece si se *antepone* la palabra «sub» al verbalizar los argumentos (⟨*1* | *2* | *3* | ...| *n*⟩) y (⟨F | P | B | L | T⟩) para NVDA/MathCAT. Si se utiliza (⟨*1* | *2* | *3* | ...| *n*⟩) y se establece en `true`, `$\spPfig[read-sub=true](1)$` imprimirá '$\mathcal{P}_1$' en la salida PDF y NVDA/MathCAT verbalizará *«perímetro sub uno»*; si se establece en `false`, la impresión será la misma y se verbaliza *«perímetro uno»*.

Si se utiliza (⟨F | P | B | L | T⟩) y se establece en `true`, `$\spPfig[read-sub=true](F_{1})$` imprimirá '$\mathcal{P}_{F_1}$' en la salida PDF y NVDA/MathCAT verbalizará *«perímetro de la figura sub uno»*; si se establece en `false`, la impresión será la misma y se verbaliza *«perímetro de la figura uno»*.

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

Establece si se *usará* `\triangle` o `\mdlgwhtsquare` en la impresión para los argumentos (⟨*punto*, *punto*, *punto*, ...⟩) y (⟨*1* | *2* | *3* | ...| *n*⟩) pasado al comando sin modificar la verbalización de este. Si se utiliza (⟨*punto*, *punto*, *punto*, ...⟩) se infiere el *símbolo* acorde al número de puntos; si se establece en `auto` se imprime y si se establece en `none` se omite: `$\spPfig[print-sym=auto](ABC)$` y `$\spPfig[print-sym=none](ABC)$` imprimirán '$\mathcal{P}_{\triangle ABC}$' y '$\mathcal{P}_{ABC}$' en la salida PDF y en ambos casos NVDA/MathCAT verbalizará *«perímetro del triángulo A B C»*.

Si se utiliza (⟨*1* | *2* | *3* | ...| *n*⟩) los valores `auto` y `none` quedan sin efecto; si se establece en `triangle` o `square` anida el argumento bajo el *símbolo*: `$\spPfig[print-sym=triangle](1)$` imprimirá '$\mathcal{P}_{\triangle_1}$' en la salida PDF y NVDA/MathCAT verbalizará *«perímetro del triángulo uno»*; `$\spPfig[print-sym=square](1)$` imprimirá '$\mathcal{P}_{\square_1}$' en la salida PDF y NVDA/MathCAT verbalizará *«perímetro del cuadrado uno»*.

Esta llave no se puede utilizar con el argumento (⟨F | P | B | L | T⟩).

`silent` = {⟨*true* | *false*⟩} por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT del término *«perímetro del»*, el concepto *«nombre de la figura»* y el {⟨*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.

Los comandos `\sprad` y `\spdiam`

`\sprad`	`\sprad`	`\sprad`[⟨*key* = *val*⟩]
`\spdiam`	`\spdiam`	`\spdiam`[⟨*key* = *val*⟩]

Los comandos `\sprad` y `\spdiam` trabajan en *modo matemático* y reciben el *argumento opcional* [⟨*key* = *val*⟩] mediante el cual se establece la *«forma»* en la que se imprimirán en la salida PDF las letras '*r*' y '*d*' y cómo se verbalizarán en NVDA/MathCAT.

Si no se proporciona el *argumento opcional* [⟨*key* = *val*⟩], `$\sprad$` imprimirá '*r*' en la salida PDF y NVDA/MathCAT verbalizará *«radio»*; `$\spdiam$` imprimirá '*d*' en la salida PDF y NVDA/MathCAT verbalizará *«diámetro»*.

Llaves para `\sprad` y `\spdiam`

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

Establece si se *verbalizan* en NVDA/MathCAT los conceptos *«radio»* o *«diámetro»* al ejecutar el comando sin modificar la impresión de este. Si se establece en `true`, `$\sprad$` y `$\spdiam$` se verbalizan como *«radio»* y *«diámetro»* en NVDA/MathCAT; si se establece en `false`, se verbalizarán las letras '*r*', '*R*', '*d*' o '*D*' acorde al valor establecido por la llave `read-cmd`.

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

Establece la *forma* en la que NVDA/MathCAT verbalizará las letras '*R*' o '*D*' cuando `concept=false` y `upper-case=true`. Si se establece en `school`, NVDA/MathCAT verbalizará las letras '*R*' o '*D*' sin el prefijo «mayúscula»; si se establece en `mathcat`, NVDA/MathCAT aplica sus propias reglas, que pueden incluir el prefijo «mayúscula» dependiendo de la configuración del usuario.

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

Establece si se *imprime* usando mayúsculas. Si se establece en `true`, `$\sprad[upper-case=true]$` y `$\spdiam[upper-case=true]$` imprimirán '*R*' y '*D*' en la salida PDF; si se establece en `false` se imprimirá usando minúsculas.

La verbalización al establecer esta llave es controlada por la llave `read-cmd`.

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

Establece si se *usará* `\diameter` antes de la letra '*d*' o '*D*' en la impresión del comando `\spdiam`. Si se establece en `true`, `$\spdiam[print-sym=true]$` imprimirá '⌀*d*' en la salida PDF y NVDA/MathCAT verbalizará *«diámetro»*.

La llave `print-sym` no tiene ningún efecto en `\sprad`.

`sub-index` = {⟨*subíndice*⟩} por defecto: *no usado*

Establece un *subíndice* sobre las letras '*r*', '*R*', '*d*' o '*D*'. El valor de esta llave *siempre* debe pasarse entre llaves '{ }'. Por ejemplo: `$\sprad[sub-index={1}]$` imprimirá 'r_1' en la salida PDF y NVDA/MathCAT verbalizará *«radio uno»*.

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

Establece si se *antepone* la palabra *«sub»* al verbalizar el subíndice establecido por la llave `sub-index` al ejecutar el comando sin modificar la impresión de este. Si se establece en `true`, `$\sprad[sub-index={1},read-sub=true]$` se verbalizará *«radio sub uno»* en NVDA/MathCAT; si se establece en `false`, NVDA/MathCAT verbalizará *«radio uno»*.

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

Sobrescribe el *concepto* establecido por la llave `concept` 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: `$\sprad[read-arg={medida exacta}]$` imprimirá '*r*' en la salida PDF y NVDA/MathCAT verbalizará *«radio medida exacta»*; `$\sprad[concept=false,read-arg={medida exacta}]$` imprimirá '*r*' y NVDA/MathCAT verbalizará *«medida exacta»*.

Las llaves `sub-index` y `read-sub` no tienen efecto al usar `read-arg`.

`silent` = {⟨*true* | *false*⟩} por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT del *concepto* establecido por la llave `concept` y de las letras '*r*', '*R*', '*d*' o '*D*' al ejecutar el 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 `\spcirc`

`\spcirc`

`\spcirc`	`\spcirc[`⟨*key* = *val*⟩`]`
`\spcirc(`⟨*centro*⟩`)`	`\spcirc[`⟨*key* = *val*⟩`](`⟨*centro*⟩`)`
`\spcirc(`⟨*centro*, *radio*⟩`)`	`\spcirc[`⟨*key* = *val*⟩`](`⟨*centro*, *radio*⟩`)`

El comando `\spcirc` trabaja en *modo matemático* y recibe el *argumento opcional delimitado por paréntesis* de la forma (⟨*centro*⟩) o (⟨*centro*, *radio*⟩), donde ⟨*radio*⟩ es opcional y debe pasarse junto al argumento (⟨*centro*⟩) separado por una coma ',' obligatoria.

La forma en la que se procesa el ⟨*radio*⟩ junto a su impresión y verbalización se establece de manera interna: si es un ⟨*número*⟩ entero o decimal se usa `\spnum`; si es una ⟨*medida*⟩ se usa `\spunit`; si es un segmento ⟨*punto punto*⟩ se usa `\spside` o `\spmside` acorde al valor establecido por la llave `radio-sty`; y si es una letra se imprime tal cual. Cualquier otra forma devuelve un "error" y se detiene la compilación del documento.

Si no se proporciona el *argumento opcional* [⟨*key* = *val*⟩] ni el *argumento opcional delimitado por paréntesis*, `$\spcirc$` imprimirá '⊙' en la salida PDF y NVDA/MathCAT verbalizará *«circunferencia»*.

Si se proporciona el *argumento* (⟨*centro*⟩), `$\spcirc(O)$` imprimirá '⊙(*O*)' en la salida PDF y NVDA/MathCAT verbalizará *«circunferencia de centro en O»*.

Si se proporciona el *argumento* (⟨*centro*, *radio*⟩), `$\spcirc(O,3)$` imprimirá '⊙(*O*, 3)' en la salida PDF y NVDA/MathCAT verbalizará *«circunferencia de centro en O y radio tres»*.

El argumento (⟨*centro*⟩) y el argumento ⟨*radio*⟩ cuando tiene la forma ⟨*punto punto*⟩ aceptan prima '' ' o subíndice '_{...}'.

Llaves para `\spcirc`

El comando `\spcirc` posee las ⟨*meta-keys*⟩ `spside` pasada al comando `\spside` (v. pág. 27) y `spmside` pasada al comando `\spmside` (v. pág. 27).

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

Establece la *forma* en la que NVDA/MathCAT verbalizará el argumento (⟨*centro*⟩). Si se establece en `school`, NVDA/MathCAT verbaliza el argumento (⟨*centro*⟩) sin el prefijo «mayúscula»; si se establece en `mathcat`, NVDA/MathCAT aplica sus propias reglas, que pueden incluir el prefijo «mayúscula» dependiendo de la configuración del usuario.

Esta llave tiene efecto sobre el argumento ⟨*radio*⟩ cuando tiene la forma ⟨*punto punto*⟩.

`prefix` = {⟨*prefijo*⟩} por defecto: *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 '{ }'.

`print-sym` = {⟨*usual* | *odot*⟩} por defecto: *odot*

Establece si se *usará* `\bigodot` o la letra '*C*' en la impresión del comando sin modificar la verbalización de este. Si se establece en `usual`, `$\spcirc[print-sym=usual](O)$` imprimirá '*C*(*O*)' en la salida PDF; si se establece en `odot`, `$\spcirc[print-sym=odot](O)$` imprimirá '⊙(*O*)' en la salida PDF.

`read-sty` = {⟨*circun* | *circulo*⟩} por defecto: *circun*

©2026 by Pablo González L

Establece el *concepto* que verbalizará NVDA/MathCAT al ejecutar el comando sin modificar la impresión de este. Si se establece en `circun`, `$\spcirc[read-sty=circun](O)$` se verbalizará *«circunferencia de centro en O»* en NVDA/MathCAT; si se establece en `circulo`, `$\spcirc[read-sty=circulo](O)$` se verbalizará *«círculo de centro en O»* en NVDA/MathCAT.

`diameter` = {⟨*true* | *false*⟩} por defecto: *false*

Establece *cómo* se verbaliza el argumento ⟨*radio*⟩ al ejecutar el comando sin modificar la impresión de este. Si se establece en `true`, `$\spcirc[diameter=true](O, 3)$` se verbalizará *«circunferencia de centro en O y diámetro tres»* en NVDA/MathCAT; si se establece en `false`, `$\spcirc[diameter=false](O, 3)$` se verbalizará *«circunferencia de centro en O y radio tres»* en NVDA/MathCAT.

`radio-sty` = {⟨*side* | *mside*⟩} por defecto: *mside*

Establece el *comando* que procesa internamente el argumento ⟨*radio*⟩ cuando tiene la forma ⟨*punto punto*⟩. Si se establece en `side` se usará `\spside{`⟨*punto punto*⟩`}`, `$\spcirc[radio-sty=side](O, AB)$` imprimirá '$\odot(O, \overline{AB})$' en la salida PDF y NVDA/MathCAT verbalizará *«circunferencia de centro en O y radio lado A B»*; si se establece en `mside` se usará `\spmside{`⟨*punto punto*⟩`}`, `$\spcirc[radio-sty=mside](O, AB)$` imprimirá '$\odot(O, AB)$' en la salida PDF y NVDA/MathCAT verbalizará *«circunferencia de centro en O y radio medida del lado A B»*.

Las ⟨*meta-keys*⟩ `spside` y `spmside` permiten modificar la impresión y verbalización del argumento ⟨*radio*⟩ cuando tiene la forma ⟨*punto punto*⟩.

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

Establece si se *verbaliza* en NVDA/MathCAT el *«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»* y el argumento (⟨*centro*⟩) o (⟨*centro*, *radio*⟩); si se establece en `false`, NVDA/MathCAT solo verbalizará el argumento (⟨*centro*⟩) o (⟨*centro*, *radio*⟩) acorde al valor establecido por la llave `read-cmd`.

Utilizar `concept=false` sin el argumento (⟨*centro*⟩) es equivalente a usar `silent=true`.

`sub-index` = {⟨*subíndice*⟩} por defecto: *no usado*

Establece un *subíndice* sobre `\bigodot` o la letra '*C*' establecida por la llave `print-sym`. El valor de esta llave *siempre* debe pasarse entre llaves '`{ }`'.
Por ejemplo: `$\spcirc[sub-index={1},radio-sty=mside](O, AB)$` imprimirá '$\odot_1(O, AB)$' en la salida PDF y NVDA/MathCAT verbalizará *«circunferencia uno de centro en O y radio medida del lado A B»*.

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

Establece si se *antepone* la palabra *«sub»* al verbalizar el subíndice establecido por la llave `sub-index` al ejecutar el comando sin modificar la impresión de este. Si se establece en `true`,
`$\spcirc[sub-index={1},read-sub=true](O)$` se verbalizará *«circunferencia sub uno de centro en O»* en NVDA/MathCAT; si se establece en `false`, NVDA/MathCAT verbalizará *«circunferencia uno de centro en O»*.

`read-arg` = {⟨*verbalización* NVDA⟩} por defecto: *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: `$\spcirc[read-arg={disco}](O)$` imprimirá '$\odot(O)$' en la salida PDF y NVDA/MathCAT verbalizará *«disco de centro en O»*.

`silent` = {⟨*true* | *false*⟩} por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT del *«concepto»* establecido por la llave `read-sty`, del argumento (⟨*centro*⟩) o (⟨*centro*, *radio*⟩) pasados 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 `\spAcirc`

`\spAcirc`

`\spAcirc` `\spAcirc[`⟨*key* = *val*⟩`]`
`\spAcirc(`⟨*centro*⟩`)` `\spAcirc[`⟨*key* = *val*⟩`](`⟨*centro*⟩`)`
`\spAcirc(`⟨*1* | *2* | *3* | ... | *n*⟩`)` `\spAcirc[`⟨*key* = *val*⟩`](`⟨*1* | *2* | *3* | ... | *n*⟩`)`

El comando `\spAcirc` trabaja en *modo matemático* y recibe el *argumento opcional delimitado por paréntesis* de la forma (⟨*centro*⟩) o (⟨*1* | *2* | *3* | ...| *n*⟩), aplicando `\mathcal{A}` a la impresión.

Si no se proporciona el *argumento opcional* [⟨*key* = *val*⟩] ni el *argumento opcional delimitado por paréntesis*, `$\spAcirc$` imprimirá '$\mathcal{A}$' en la salida PDF y NVDA/MathCAT verbalizará *«área de la circunferencia»*.

Si se proporciona el *argumento* (⟨*centro*⟩), `$\spAcirc(O)$` imprimirá '$\mathcal{A}_{\odot(O)}$' en la salida PDF y NVDA/MathCAT verbalizará *«área de la circunferencia con centro en O»*.

Si se proporciona el *argumento* (⟨*1* | *2* | *3* | ...| *n*⟩), `$\spAcirc(1)$` imprimirá '$\mathcal{A}_{\odot_1}$' en la salida PDF y NVDA/MathCAT verbalizará *«área de la circunferencia uno»*.

El *argumento* (⟨*centro*⟩) acepta prima '`'`' o subíndice '`_{...}`' y no acepta un ⟨*radio*⟩. Una coma '`,`' en el argumento devuelve siempre un "error" y se detiene la compilación del documento.

©2026 by Pablo González L

Llaves para \spAcirc

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

Establece la *forma* en la que NVDA/MathCAT verbalizará el argumento (⟨centro⟩). Si se establece en school, NVDA/MathCAT verbaliza el argumento (⟨centro⟩) sin el prefijo «mayúscula»; 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⟩} por defecto: *no usado*

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

read-sty = {⟨circun | circulo⟩} por defecto: *circun*

Establece el *concepto* que verbalizará NVDA/MathCAT después de verbalizar el término *«área de ...»* al ejecutar el comando sin modificar la impresión de este. Si se establece en circun, $\spAcirc[read-sty=circun](0)$ se verbalizará *«área de la circunferencia con centro en O»* en NVDA/MathCAT; si se establece en circulo, $\spAcirc[read-sty=circulo](0)$ se verbalizará *«área del círculo con centro en O»* en NVDA/MathCAT.

concept = {⟨true | false⟩} por defecto: *true*

Establece si se *verbaliza* en NVDA/MathCAT el término *«área de ...»* y el *«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, el concepto y el argumento (⟨centro⟩) o (⟨1 | 2 | 3 | ... | n⟩); si se establece en false, NVDA/MathCAT solo verbalizará el argumento (⟨centro⟩) acorde al valor establecido por la llave read-cmd.

Utilizar concept=false sin el argumento (⟨centro⟩) o con el argumento (⟨1 | 2 | 3 | ... | n⟩) es equivalente a usar silent=true.

read-arg = {⟨verbalización NVDA⟩} por defecto: *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: $\spAcirc[read-arg={esfera}](0)$ se verbalizará *«área de la esfera con centro en O»* en NVDA/MathCAT; $\spAcirc[concept=false,read-arg={esfera}](0)$ se verbalizará *«esfera con centro en O»* en NVDA/MathCAT.

Utilizar esta llave con concept=false y el argumento (⟨1 | 2 | 3 | ... | n⟩) es equivalente a usar silent=true.

print-sym = {⟨usual | odot⟩} por defecto: *odot*

Establece si se *usará* \bigodot o la letra '*C*' en la impresión del comando sin modificar la verbalización de este. Si se establece en usual, $\spAcirc[print-sym=usual](0)$ imprimirá '$\mathcal{A}_{C(O)}$' en la salida PDF; si se establece en odot, $\spAcirc[print-sym=odot](0)$ imprimirá '$\mathcal{A}_{\odot(O)}$' en la salida PDF.

sub-index = {⟨subíndice⟩} por defecto: *no usado*

Establece un *subíndice* sobre \bigodot o la letra '*C*' establecida por la llave print-sym cuando se utiliza el argumento (⟨centro⟩). El valor de esta llave *siempre* debe pasarse entre llaves '{ }'. Por ejemplo: $\spAcirc[sub-index={1}](0)$ imprimirá '$\mathcal{A}_{\odot_1(O)}$' en la salida PDF y NVDA/MathCAT verbalizará *«área de la circunferencia uno con centro en O»*.

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

Establece si se *antepone* la palabra *«sub»* al verbalizar el subíndice establecido por la llave sub-index o al usar el argumento (⟨1 | 2 | 3 | ... | n⟩) al ejecutar el comando sin modificar la impresión de este. Si se establece en true, $\spAcirc[read-sub=true](1)$ se verbalizará *«área de la circunferencia sub uno»* en NVDA/MathCAT; si se establece en false, NVDA/MathCAT verbalizará *«área de la circunferencia uno»*.

silent = {⟨true | false⟩} por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT del término *«área de ...»*, del concepto establecido por la llave read-sty y del argumento (⟨centro⟩) o (⟨1 | 2 | 3 | ... | n⟩) 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 \spPcirc

\spPcirc

\spPcirc \spPcirc[⟨key = val⟩]
\spPcirc(⟨centro⟩) \spPcirc[⟨key = val⟩](⟨centro⟩)
\spPcirc(⟨1 | 2 | 3 | ... | n⟩) \spPcirc[⟨key = val⟩](⟨1 | 2 | 3 | ... | n⟩)

El comando \spPcirc trabaja en *modo matemático* y recibe el *argumento opcional delimitado por paréntesis* de la forma (⟨centro⟩) o (⟨1 | 2 | 3 | ... | n⟩), aplicando \mathcal{P} a la impresión.

Si no se proporciona el *argumento opcional* [⟨key = val⟩] ni el *argumento opcional delimitado por paréntesis*, $\spPcirc$ imprimirá '$\mathcal{P}$' en la salida PDF y NVDA/MathCAT verbalizará *«perímetro de la circunferencia»*.

©2026 by Pablo González L

Si se proporciona el *argumento* (⟨*centro*⟩), `$\spPcirc(O)$` imprimirá '$\mathcal{P}_{\odot(O)}$' en la salida PDF y NVDA/MathCAT verbalizará *«perímetro de la circunferencia con centro en O»*.

Si se proporciona el *argumento* (⟨*1* | *2* | *3* | *...*| *n*⟩), `$\spPcirc(1)$` imprimirá '$\mathcal{P}_{\odot_1}$' en la salida PDF y NVDA/MathCAT verbalizará *«perímetro de la circunferencia uno»*.

El *argumento* (⟨*centro*⟩) acepta prima '`'`' o subíndice '`_{...}`' y no acepta un ⟨*radio*⟩. Una coma '`,`' en el argumento devuelve siempre un "error" y se detiene la compilación del documento.

Llaves para `\spPcirc`

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

Establece la *forma* en la que NVDA/MathCAT verbalizará el argumento (⟨*centro*⟩). Si se establece en `school`, NVDA/MathCAT verbaliza el argumento (⟨*centro*⟩) sin el prefijo «mayúscula»; 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*⟩} por defecto: *no usado*

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

`read-sty` = {⟨*circun* | *circulo*⟩} por defecto: *circun*

Establece el *concepto* que verbalizará NVDA/MathCAT después de verbalizar el término *«perímetro de ...»* al ejecutar el comando sin modificar la impresión de este. Si se establece en `circun`, `$\spPcirc[read-sty=circun](O)$` se verbalizará *«perímetro de la circunferencia con centro en O»* en NVDA/MathCAT; si se establece en `circulo`, `$\spPcirc[read-sty=circulo](O)$` se verbalizará *«perímetro del círculo con centro en O»* en NVDA/MathCAT.

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

Establece si se *verbaliza* en NVDA/MathCAT el término *«perímetro de ...»* y el *«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, el concepto y el argumento (⟨*centro*⟩) o (⟨*1* | *2* | *3* | *...*| *n*⟩); si se establece en `false`, NVDA/MathCAT solo verbalizará el argumento (⟨*centro*⟩) acorde al valor establecido por la llave `read-cmd`.

Utilizar `concept=false` sin el argumento (⟨*centro*⟩) o con el argumento (⟨*1* | *2* | *3* | *...*| *n*⟩) es equivalente a usar `silent=true`.

`read-arg` = {⟨*verbalización* NVDA⟩} por defecto: *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: `$\spPcirc[read-sty=circulo,read-arg={disco}](O)$` se verbalizará *«perímetro del disco con centro en O»* en NVDA/MathCAT; `$\spPcirc[concept=false,read-arg={disco}](O)$` se verbalizará *«disco con centro en O»* en NVDA/MathCAT.

Utilizar esta llave con `concept=false` y el argumento (⟨*1* | *2* | *3* | *...*| *n*⟩) es equivalente a usar `silent=true`.

`print-sym` = {⟨*usual* | *odot*⟩} por defecto: *odot*

Establece si se *usará* `\bigodot` o la letra '*C*' en la impresión del comando sin modificar la verbalización de este. Si se establece en `usual`, `$\spPcirc[print-sym=usual](O)$` imprimirá '$\mathcal{P}_{C(O)}$' en la salida PDF; si se establece en `odot`, `$\spPcirc[print-sym=odot](O)$` imprimirá '$\mathcal{P}_{\odot(O)}$' en la salida PDF.

`sub-index` = {⟨*subíndice*⟩} por defecto: *no usado*

Establece un *subíndice* sobre `\bigodot` o la letra '*C*' establecida por la llave `print-sym` cuando se utiliza el argumento (⟨*centro*⟩). El valor de esta llave *siempre* debe pasarse entre llaves '`{ }`'. Por ejemplo: `$\spPcirc[sub-index={1}](O)$` imprimirá '$\mathcal{P}_{\odot_1(O)}$' en la salida PDF y NVDA/MathCAT verbalizará *«perímetro de la circunferencia uno con centro en O»*.

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

Establece si se *antepone* la palabra *«sub»* al verbalizar el subíndice establecido por la llave `sub-index` o al usar el argumento (⟨*1* | *2* | *3* | *...*| *n*⟩) al ejecutar el comando sin modificar la impresión de este. Si se establece en `true`, `$\spPcirc[read-sub=true](1)$` se verbalizará *«perímetro de la circunferencia sub uno»* en NVDA/MathCAT; si se establece en `false`, NVDA/MathCAT verbalizará *«perímetro de la circunferencia uno»*.

`silent` = {⟨*true* | *false*⟩} por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT del término *«perímetro de ...»*, del concepto establecido por la llave `read-sty` y del argumento (⟨*centro*⟩) o (⟨*1* | *2* | *3* | *...*| *n*⟩) 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 \spVol

\spVol `\spVol` `\spVol[⟨key = val⟩]`

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

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

Llaves para \spVol

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

Establece si se *verbaliza* en NVDA/MathCAT el concepto *«volumen»* al ejecutar el comando sin modificar la impresión de este. Si se establece en `true`, $\spVol$ se verbaliza como *«volumen»* en NVDA/MathCAT; si se establece en `false`, se verbalizará la letra '$\mathcal{V}$' acorde al valor establecido por la llave `read-cmd`.

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

Establece la *forma* en la que NVDA/MathCAT verbalizará la letra '$\mathcal{V}$' cuando `concept=false`. Si se establece en `school`, NVDA/MathCAT verbalizará únicamente la letra 'V'; si se establece en `mathcat`, NVDA/MathCAT aplica sus propias reglas sobre '$\mathcal{V}$'.

`sub-index` = {⟨*subíndice*⟩} por defecto: *no usado*

Establece un *subíndice* sobre '$\mathcal{V}$'. El valor de esta llave *siempre* debe pasarse entre llaves '`{ }`'. Por ejemplo: `$\spVol[sub-index={1}]$` imprimirá '$\mathcal{V}_1$' en la salida PDF y NVDA/MathCAT verbalizará *«volumen uno»*.

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

Establece si se *antepone* la palabra *«sub»* al verbalizar el subíndice establecido por la llave `sub-index` al ejecutar el comando sin modificar la impresión de este. Si se establece en `true`, `$\spVol[sub-index={1},read-sub=true]$` se verbalizará *«volumen sub uno»* en NVDA/MathCAT; si se establece en `false`, NVDA/MathCAT verbalizará *«volumen uno»*.

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

Sobrescribe el *concepto* establecido por la llave `concept` 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: `$\spVol[read-arg={medida exacta}]$` imprimirá '$\mathcal{V}$' en la salida PDF y NVDA/MathCAT verbalizará *«volumen medida exacta»*; `$\spVol[concept=false,read-arg={medida exacta}]$` imprimirá '$\mathcal{V}$' y NVDA/MathCAT verbalizará *«medida exacta»*.

Las llaves `sub-index` y `read-sub` no tienen efecto al usar `read-arg`.

`silent` = {⟨*true* | *false*⟩} por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT del *concepto* establecido por la llave `concept` y de '$\mathcal{V}$' al ejecutar el 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.

8 Utilidades para álgebra

El paquete spanish-mathml-intent provee utilidades para trabajar con álgebra, descritas en esta sección: \spinterval para los intervalos, \spfun para las funciones, \spdom, \sprec y \spcod para el dominio, el recorrido y el codominio de una función, \spfuncomp para la composición de funciones y \spabs para el valor absoluto. Todos trabajan en *modo matemático* y reciben un *argumento obligatorio*.

El nombre de una función sigue la misma regla en \spfun, \spdom, \sprec, \spcod y \spfuncomp (v. pág. 40). Un argumento que no cumple la forma documentada devuelve un "error" y se detiene la compilación del documento.

El comando \spinterval

\spinterval `\spinterval{⟨intervalo⟩}` `\spinterval[⟨key = val⟩]{⟨intervalo⟩}`

El comando \spinterval trabaja en *modo matemático* y recibe el *argumento* obligatorio {⟨*intervalo*⟩} de la forma {⟨*(a; b)*⟩}, donde '*a*' y '*b*' son los *extremos* del intervalo separados por un punto y coma ';' o, si no hay punto y coma, por una única coma ','. El delimitador izquierdo puede ser '(', '[' o ']' y el derecho ')', ']' o '[', en cualquier combinación. Los extremos numéricos se procesan internamente bajo el comando \spnum; cualquier otro extremo, como una letra, un subíndice, una fracción o una raíz, se imprime tal cual y lo verbaliza NVDA/MathCAT.

Un delimitador '(' o ']' a la izquierda, y ')' o '[' a la derecha, *excluye* su extremo y el intervalo es *abierto* en él; los demás delimitadores lo *incluyen* y el intervalo es *cerrado* en él.

©2026 by Pablo González L

Si no se proporciona el *argumento opcional* [⟨*key* = *val*⟩] y el intervalo es abierto en sus dos extremos, `$\spinterval{(1,5)}$` imprimirá '(1, 5)' en la salida PDF y NVDA/MathCAT verbalizará *«intervalo abierto de uno a cinco»*; la notación chilena `$\spinterval{]1,5[}$` imprimirá ']1, 5[' y se verbalizará igual.

Si el intervalo es cerrado en sus dos extremos, `$\spinterval{[1,5]}$` imprimirá '[1, 5]' y NVDA/MathCAT verbalizará *«intervalo cerrado de uno a cinco»*.

Si el intervalo es abierto en un extremo y cerrado en el otro, `$\spinterval{]1,5]}$` imprimirá ']1, 5]' y NVDA/MathCAT verbalizará *«intervalo abierto en uno y cerrado en cinco»*; `$\spinterval{[1,5[}$` imprimirá '[1, 5[' y verbalizará *«intervalo cerrado en uno y abierto en cinco»*.

Si un extremo es infinito, `$\spinterval{]-\infty,3]}$` imprimirá '] − ∞, 3]' y NVDA/MathCAT verbalizará *«intervalo abierto en menos infinito y cerrado en tres»*.

Si los extremos son números decimales con coma, el separador debe ser el punto y coma: `$\spinterval{[3,5;4,2]}$` imprimirá '[3,5; 4,2]' y NVDA/MathCAT verbalizará *«intervalo cerrado de tres coma cinco a cuatro coma dos»*.

El comando `\spinterval` devuelve un "error" y se detiene la compilación del documento cuando el {⟨*intervalo*⟩} no comienza con un delimitador izquierdo y termina con uno derecho de los indicados, cuando un extremo está vacío, cuando no hay separador, cuando hay más de una coma sin punto y coma, cuando las llaves no están balanceadas o cuando el infinito está mal ubicado o se incluye en el intervalo.

El argumento {⟨*intervalo*⟩} siempre debe pasarse entre llaves '{ }', incluso cuando comienza con un corchete: sin ellas, `$\spinterval[1,5]$` interpreta los corchetes como el argumento opcional [⟨*key* = *val*⟩].

El infinito '∞' nunca se incluye en el intervalo: '−∞' solo puede ir en el extremo izquierdo y '∞' en el derecho, ambos junto a un delimitador que los excluya.

Llaves para `\spinterval`

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

Establece la *forma* en la que NVDA/MathCAT verbalizará los extremos del intervalo. Si se establece en `school`, `$\spinterval[read-cmd=school]{]-2,+4[}$` se verbalizará *«intervalo abierto de menos dos a más cuatro»*; si se establece en `mathcat`, NVDA/MathCAT aplica sus propias reglas y `$\spinterval[read-cmd=mathcat]{]-2,+4[}$` se verbalizará *«intervalo abierto de menos dos a positivo cuatro»*.

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

Establece el *prefijo* que será verbalizado por NVDA/MathCAT *antes* de verbalizar el *«concepto»* intervalo. El valor de esta llave *siempre* debe pasarse entre llaves '{ }'. Por ejemplo: `$\spinterval[prefix={el}]{(1,5)}$` imprimirá '(1, 5)' en la salida PDF y NVDA/MathCAT verbalizará *«el intervalo abierto de uno a cinco»*.

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

Sobrescribe la verbalización del *intervalo* completo que será verbalizada 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: `$\spinterval[read-arg={rango}]{[1,5]}$` imprimirá '[1, 5]' en la salida PDF y NVDA/MathCAT verbalizará *«rango»*.

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

Establece si se *verbaliza* en NVDA/MathCAT el *«concepto»* intervalo y el prefijo establecido por la llave `prefix` al ejecutar el comando sin modificar la impresión de este. Si se establece en `true`, NVDA/MathCAT verbalizará el *«concepto»* y los extremos; si se establece en `false`, `$\spinterval[concept=false]{(1,5)}$` se verbalizará *«abierto de uno a cinco»*.

`size` = {⟨*auto* | *none* | *big* | *Big* | *bigg* | *Bigg*⟩} por defecto: *auto*

Establece el *tamaño* de los delimitadores del intervalo sin modificar la verbalización de este. Si se establece en `auto`, los delimitadores conservan su tamaño normal salvo que algún extremo contenga una fracción `\frac` o `\dfrac`, en cualquier posición del extremo: en ese caso se usan `\bigl` y `\bigr` en estilo texto, `\Bigl` y `\Bigr` en estilo display, y siempre `\Bigl` y `\Bigr` cuando la fracción es `\dfrac`. Si se establece en `none`, los delimitadores nunca cambian de tamaño; si se establece en `big`, `Big`, `bigg` o `Bigg`, se usan siempre `\bigl` y `\bigr`, `\Bigl` y `\Bigr`, `\biggl` y `\biggr`, o `\Biggl` y `\Biggr`, respectivamente.

`silent` = {⟨*true* | *false*⟩} por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT del {⟨*intervalo*⟩} 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 `\spfun`

`\spfun` `\spfun{⟨función⟩}` `\spfun[⟨key = val⟩]{⟨función⟩}`

El comando `\spfun` trabaja en *modo matemático* y recibe el *argumento* obligatorio {⟨*función*⟩} de la forma {⟨*f(x)*⟩}, donde '*f*' es el *nombre* de la función y lo que va entre paréntesis son uno o más *argumentos* separados por coma ','. El nombre es una letra, mayúscula o minúscula, con prima(s) '' ' o subíndice '_{...}' opcionales; el subíndice solo admite letras y dígitos. Los argumentos pueden ser letras, números o expresiones.

©2026 by Pablo González L

Si no se proporciona el *argumento opcional* [⟨*key* = *val*⟩], `$\spfun{f(x)}$` imprimirá '$f(x)$' en la salida PDF y NVDA/MathCAT verbalizará *«f de x»*; `$\spfun{f_{1}(x)}$` imprimirá '$f_1(x)$' y verbalizará *«f sub uno de x»*; `$\spfun{f'(x)}$` imprimirá '$f'(x)$' y verbalizará *«f prima de x»*; `$\spfun{f''(x)}$` imprimirá '$f''(x)$' y verbalizará *«f doble prima de x»*.

Con dos o más argumentos, `$\spfun{f(x,y)}$` imprimirá '$f(x,y)$' y NVDA/MathCAT verbalizará *«f de x coma y»*. Un argumento puede ser una expresión: `$\spfun{f(x+1)}$` imprimirá '$f(x+1)$' y verbalizará *«f de x más 1»*.

Un argumento puede ser a su vez una llamada, tantas veces como se quiera: `$\spfun{f(g(x))}$` imprimirá '$f(g(x))$' y verbalizará *«f de g de x»*; `$\spfun{f(f^{-1}(y))}$` imprimirá '$f(f^{-1}(y))$' y verbalizará *«f de f inversa de y»*; `$\spfun{f(g(x),h(y))}$` imprimirá '$f(g(x),h(y))$'. Una llamada anidada se valida igual que la exterior. Si el argumento es otra cosa, como `g(x)+1`, se imprime y se verbaliza con normalidad pero sin anotar la llamada que contiene. Las llaves `read-cmd` y `silent` valen para todos los niveles; `concept` y `prefix` solo para la llamada exterior.

Las comas que quedan dentro de llaves, de paréntesis o de corchetes no separan argumentos. Por eso, un número decimal con coma debe escribirse entre llaves: `$\spfun{f(1{,}5)}$` tiene un solo argumento.

El comando `\spfun` devuelve un "error" y se detiene la compilación del documento cuando el {⟨*función*⟩} no comienza con un nombre válido, cuando el subíndice contiene algo distinto de letras o dígitos, cuando el nombre no va seguido de paréntesis, cuando los paréntesis, las llaves o los corchetes no están balanceados, cuando un argumento está vacío o cuando hay algo después del paréntesis que cierra. Lo mismo vale para una llamada anidada en un argumento.

Llaves para `\spfun`

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

Establece la *forma* en la que NVDA/MathCAT verbalizará el nombre de la función. Si se establece en `school`, NVDA/MathCAT verbaliza el nombre sin el prefijo «mayúscula»: `$\spfun{F''(x)}$` se verbaliza *«F doble prima de x»*; 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*⟩} por defecto: *no usado*

Establece el *prefijo* que será verbalizado por NVDA/MathCAT *antes* de verbalizar el concepto *«función»*. El valor de esta llave *siempre* debe pasarse entre llaves '{ }'. Solo se verbaliza cuando `concept` es `true`.

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

Establece si se verbaliza el concepto *«función»* antes de la función. Si se establece en `true`, NVDA/MathCAT verbalizará el concepto, el prefijo establecido por la llave `prefix` y la función; si se establece en `false`, solo verbalizará la función.

`silent` = {⟨*true* | *false*⟩} por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT de la {⟨*función*⟩} pasada 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`, sí verbalizará la salida del comando. El argumento se valida igual en ambos casos.

Los comandos `\spdom`, `\sprec` y `\spcod`

`\spdom`	`\spdom{`⟨*función*⟩`}`	`\spdom[`⟨*key* = *val*⟩`]{`⟨*función*⟩`}`
`\sprec`	`\sprec{`⟨*función*⟩`}`	`\sprec[`⟨*key* = *val*⟩`]{`⟨*función*⟩`}`
`\spcod`	`\spcod{`⟨*función*⟩`}`	`\spcod[`⟨*key* = *val*⟩`]{`⟨*función*⟩`}`

Los comandos `\spdom`, `\sprec` y `\spcod` trabajan en *modo matemático* y reciben el *argumento* obligatorio {⟨*función*⟩}: el nombre de una función como en `\spfun`, esto es, una letra con prima(s) o subíndice opcionales y, si se quiere, el exponente '`^{-1}`', sin argumentos. Imprimen respectivamente Dom, Rec y Cod seguidos de la función entre paréntesis.

Si no se proporciona el *argumento opcional* [⟨*key* = *val*⟩], `$\spdom{f}$` imprimirá 'Dom(f)' en la salida PDF y NVDA/MathCAT verbalizará *«dominio de f»*; `$\sprec{f}$` imprimirá 'Rec(f)' y verbalizará *«recorrido de f»*; `$\spcod{f}$` imprimirá 'Cod(f)' y verbalizará *«codominio de f»*.

El nombre de la función admite prima(s) y subíndice, y se verbaliza sin anunciar la mayúscula: `$\spdom{f'}$` imprimirá 'Dom(f')' y verbalizará *«dominio de f prima»*; `$\sprec{f_{1}}$` imprimirá 'Rec(f_1)' y verbalizará *«recorrido de f sub uno»*; `$\spcod{F''}$` imprimirá 'Cod(F'')' y verbalizará *«codominio de F doble prima»*.

Con la inversa, `$\spdom{f^{-1}}$` imprimirá 'Dom(f^{-1})' y verbalizará *«dominio de f inversa»*; `$\sprec{f^{-1}}$` verbalizará *«recorrido de f inversa»*.

Los comandos devuelven un "error" y se detiene la compilación del documento cuando el argumento no es el nombre de una función.

Llaves para `\spdom`, `\sprec` y `\spcod`

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

Establece el *prefijo* que será verbalizado por NVDA/MathCAT *antes* de la palabra del concepto. El valor de esta llave *siempre* debe pasarse entre llaves '{ }'. Con `$\spdom[prefix={el}]{f}$`, NVDA/MathCAT verbalizará *«el dominio de f»*; con `$\sprec[prefix={el}]{f}$`, *«el recorrido de f»*; con `$\spcod[prefix={el}]{f}$`, *«el codominio de f»*. La impresión no cambia.

`silent` = {⟨*true* | *false*⟩} por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT de la {⟨*función*⟩} pasada al comando sin modificar la impresión de este. Si se establece en `true`, NVDA/MathCAT NO verbalizará la salida del comando: `$\spdom[silent=true]{f}$` imprimirá 'Dom(f)' y no se verbalizará nada; si se establece en `false`, sí verbalizará la salida del comando. El argumento se valida igual en ambos casos.

El comando \spfuncomp

`\spfuncomp` `\spfuncomp{⟨composición⟩}` `\spfuncomp[⟨key = val⟩]{⟨composición⟩}`

El comando `\spfuncomp` trabaja en *modo matemático* y recibe el *argumento* obligatorio {⟨*composición*⟩}: dos o más *funciones* separadas por `\circ`, como en {⟨$f\circ g$⟩} o {⟨$f\circ g\circ h$⟩}. Cada función es un nombre como en `\spfun`: una letra, mayúscula o minúscula, con prima(s) o subíndice opcionales, y el exponente '`^{-1}`' de la función inversa. La composición puede ir entre paréntesis, {⟨$(f\circ g)$⟩}, y puede aplicarse a uno o más argumentos entre paréntesis separados por coma, {⟨$(f\circ g)(x)$⟩}. Los argumentos siguen las reglas de `\spfun`: pueden ser letras, números, expresiones o llamadas.

Si no se proporciona el *argumento opcional* [⟨*key = val*⟩], `$\spfuncomp{f\circ g}$` imprimirá '$f\circ g$' en la salida PDF y NVDA/MathCAT verbalizará *«f compuesta con g»*; `$\spfuncomp{f\circ g\circ h}$` imprimirá '$f\circ g\circ h$' y verbalizará *«f compuesta con g compuesta con h»*; `$\spfuncomp{(f\circ g)(x)}$` imprimirá '$(f\circ g)(x)$' y verbalizará *«f compuesta con g de x»*; `$\spfuncomp{(f^{-1}\circ g)(x)}$` imprimirá '$(f^{-1}\circ g)(x)$' y verbalizará *«f inversa compuesta con g de x»*. Un argumento puede ser una llamada: `$\spfuncomp{(f\circ g)(h(x))}$` verbalizará *«f compuesta con g de h de x»*.

El comando `\spfuncomp` devuelve un "error" y se detiene la compilación del documento cuando hay menos de dos funciones, cuando algo entre los `\circ` no es el nombre de una función, cuando los paréntesis no están balanceados, cuando lo que sigue a la composición no es una lista de argumentos entre paréntesis o cuando un argumento está vacío. La forma `f\circ g(x)`, sin paréntesis alrededor de la composición, no es válida porque es ambigua.

Llaves para \spfuncomp

`silent` = {⟨*true* | *false*⟩} por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT de la {⟨*composición*⟩} pasada 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`, sí verbalizará la salida del comando. El argumento se valida igual en ambos casos.

La composición no admite `prefix` ni `read-cmd`: no lleva palabra de concepto y su lectura es fija. Los nombres se verbalizan sin anunciar la mayúscula.

El comando \spabs

`\spabs` `\spabs{⟨x⟩}` `\spabs[⟨key = val⟩]{⟨x⟩}`

El comando `\spabs` trabaja en *modo matemático* y recibe el *argumento obligatorio* {⟨x⟩}: cualquier expresión matemática. Imprime el valor absoluto de {⟨x⟩} entre las barras `\lvert` y `\rvert`.

Si no se proporciona el *argumento opcional* [⟨*key = val*⟩], `$\spabs{x}$` imprimirá '$|x|$' en la salida PDF y NVDA/MathCAT verbalizará *«valor absoluto de x»*; `$\spabs{-5}$` imprimirá '$|-5|$' y verbalizará *«valor absoluto de menos 5»*; `$\spabs{x-3}$` imprimirá '$|x-3|$' y verbalizará *«valor absoluto de x menos 3 fin valor absoluto»*.

Cuando el argumento tiene más de un símbolo, sin contar un signo '+' o '-' inicial, NVDA/MathCAT anuncia el final del valor absoluto con *«fin valor absoluto»*. Así se sabe dónde termina cuando la expresión sigue: `$\spabs{x-3} - 1$` verbalizará *«valor absoluto de x menos 3 fin valor absoluto menos 1»*. La llave `close` controla este cierre.

El comando `\spabs` devuelve un "error" y se detiene la compilación del documento cuando el argumento está vacío.

Llaves para \spabs

`close` = {⟨*auto* | *true* | *false*⟩} por defecto: *auto*

Establece si NVDA/MathCAT verbaliza el cierre *«fin valor absoluto»* después del argumento. Con `auto` el cierre se verbaliza solo cuando {⟨x⟩} tiene más de un símbolo, sin contar un signo + o - inicial: `$\spabs{x}$` y `$\spabs{-5}$` no lo llevan y `$\spabs{x-3}$` sí. Con `true` siempre se verbaliza: `$\spabs[close=true]{x}$` verbalizará *«valor absoluto de x fin valor absoluto»*. Con `false` nunca se verbaliza: `$\spabs[close=false]{x-3}$` verbalizará *«valor absoluto de x menos 3»*. La llave no tiene efecto con `silent` ni con `read-arg`.

©2026 by Pablo González L

`size` = {⟨*none* | *big* | *Big* | *bigg* | *Bigg*⟩} por defecto: *none*

Establece el *tamaño* de las barras. Con `none` no se cambia el tamaño; con `big`, `Big`, `bigg` y `Bigg` se fija con `\bigl`, `\Bigl`, `\biggl` y `\Biggl`, y sus pares de cierre. Por ejemplo: `$\spabs[size=Big]{\dfrac{x}{2}}$` imprime las barras más grandes que el texto y no cambia la verbalización.

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

Sobrescribe lo que verbalizará NVDA/MathCAT al ejecutar el comando sin modificar la impresión de este. El valor de esta llave *siempre* debe pasarse entre llaves '{ }'. Por ejemplo: `$\spabs[read-arg={módulo de x}]{x}$` imprimirá '$|x|$' en la salida PDF y NVDA/MathCAT verbalizará *«módulo de x»*.

`silent` = {⟨*true* | *false*⟩} por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT del valor absoluto al ejecutar el 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`, sí verbalizará la salida del comando.

9 Utilidades para estadística

El paquete spanish-mathml-intent provee utilidades para trabajar con estadística y probabilidad, descritas en esta sección. Todos los comandos trabajan en *modo matemático*.

Los comandos `\spmedia`, `\spmediana`, `\spmoda`, `\spdesv`, `\spvarz`, `\sprango`, `\spriq`, `\spmin`, `\spmax` y `\spOmega` escriben un símbolo y no reciben argumento obligatorio, solo el *argumento opcional* [⟨*key* = *val*⟩] que establece cómo se verbaliza. Los demás, que son `\spProm`, `\spcuartil`, `\spdecil`, `\spPerc`, `\spFrec`, `\spcase`, `\spPval`, `\spProb`, `\spcomp`, `\spcomb` y `\spfact`, reciben *argumentos obligatorios*; si no cumplen la forma documentada devuelven un "error" y se detiene la compilación del documento.

El comando `\spProm`

`\spProm` `\spProm{⟨argumento⟩}` `\spProm[⟨key = val⟩]{⟨argumento⟩}`

El comando `\spProm` trabaja en *modo matemático* y recibe el *argumento* obligatorio {⟨*argumento*⟩}: cualquier expresión matemática, como una letra, una letra con subíndice o una expresión. Lo tipografía con una barra encima, la notación usual del *«promedio»*.

Si no se proporciona el *argumento opcional* [⟨*key* = *val*⟩], `$\spProm{x}$` imprimirá '$\bar{x}$' en la salida PDF y NVDA/MathCAT verbalizará *«promedio de x»*; `$\spProm{x_{1}}$` imprimirá '$\bar{x_1}$' y verbalizará *«promedio de x subíndice uno»*.

El {⟨*argumento*⟩} se tipografía directamente, sin procesamiento semántico adicional, y NVDA/MathCAT lo verbaliza con sus propias reglas.

Llaves para `\spProm`

`silent` = {⟨*true* | *false*⟩} por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT 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`, sí verbalizará la salida del comando.

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

Sobrescribe las palabras *«promedio de»* que verbaliza NVDA/MathCAT antes del {⟨*argumento*⟩}, sin modificar la impresión del comando. El valor de esta llave *siempre* debe pasarse entre llaves '{ }'. Por ejemplo: `$\spProm[read-arg={media de}]{x}$` imprimirá '$\bar{x}$' en la salida PDF y NVDA/MathCAT verbalizará *«media de x»*.

Los comandos `\spmedia`, `\spmediana`, `\spmoda`, `\spdesv` y `\spvarz`

`\spmedia`	`\spmedia`	`\spmedia[⟨key = val⟩]`
`\spmediana`	`\spmediana`	`\spmediana[⟨key = val⟩]`
`\spmoda`	`\spmoda`	`\spmoda[⟨key = val⟩]`
`\spdesv`	`\spdesv`	`\spdesv[⟨key = val⟩]`
`\spvarz`	`\spvarz`	`\spvarz[⟨key = val⟩]`

Los comandos `\spmedia`, `\spmediana`, `\spmoda`, `\spdesv` y `\spvarz` trabajan en *modo matemático* y reciben el *argumento opcional* [⟨*key* = *val*⟩] mediante el cual se establece la *«forma»* en la que se verbalizarán en NVDA/MathCAT los símbolos de la estadística descriptiva.

Si no se proporciona el *argumento opcional*, `$\spmedia$` imprimirá 'MA' en la salida PDF y NVDA/MathCAT verbalizará *«media aritmética»*; `$\spmediana$` imprimirá 'MD' y verbalizará *«mediana»*; `$\spmoda$` imprimirá 'MO' y verbalizará *«moda»*; `$\spdesv$` imprimirá 's' y verbalizará *«desviación estándar s»*; `$\spvarz$` imprimirá 's^2' y verbalizará *«varianza s al cuadrado»*.

©2026 by Pablo González L

Llaves para `\spmedia`, `\spmediana`, `\spmoda`, `\spdesv` **y** `\spvarz`

`sigma` = {⟨*true* | *false*⟩} por defecto: *false*

Establece si se *imprime* la letra 'σ' en lugar de la letra '*s*'. Si se establece en `true`, `$\spdesv[sigma=true]$` imprimirá 'σ' y `$\spvarz[sigma=true]$` imprimirá 'σ^2' en la salida `PDF`, y `NVDA/MathCAT` verbalizará *«desviación estándar sigma»* y *«varianza sigma al cuadrado»*. Esta llave solo existe para `\spdesv` y `\spvarz`.

`sub-index` = {⟨*subíndice*⟩} por defecto: *no usado*

Establece un *subíndice* sobre el símbolo del comando, que `NVDA/MathCAT` verbaliza después del concepto con la palabra *«de»*, o *«sub»* en `\spdesv` y `\spvarz` según la llave `read-sub`. El valor de esta llave *siempre* debe pasarse entre llaves '`{ }`'. Por ejemplo: `$\spmedia[sub-index={x}]$` imprimirá 'MA_x' en la salida `PDF` y `NVDA/MathCAT` verbalizará *«media aritmética de x»*; `$\spdesv[sub-index={x}]$` imprimirá 's_x' y verbalizará *«desviación estándar s sub x»*; `$\spvarz[sub-index={x}]$` imprimirá 's_x^2' y verbalizará *«varianza s sub x al cuadrado»*.

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

Establece si se *usa* la palabra *«sub»* en lugar de *«de»* al verbalizar el subíndice establecido por la llave `sub-index`, sin modificar la impresión del comando. Si se establece en `true`, `$\spmedia[sub-index={x},read-sub=true]$` se verbalizará *«media aritmética sub x»* en `NVDA/MathCAT`; si se establece en `false`, `NVDA/MathCAT` verbalizará *«media aritmética de x»*. El valor por defecto es `false`, salvo en `\spdesv` y `\spvarz`, donde es `true` porque estos comandos leen su letra y «s sub x» es más claro que «s de x»: en *«varianza s de x al cuadrado»*, la «x» podría entenderse como parte del exponente. Así, `$\spvarz[sub-index={x},read-sub=false]$` verbalizará *«varianza s de x al cuadrado»*.

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

Sobrescribe el *concepto* que verbalizará `NVDA/MathCAT` al ejecutar el comando sin modificar la impresión de este. El valor de esta llave *siempre* debe pasarse entre llaves '`{ }`'.

Por ejemplo: `$\spdesv[read-arg={desviación estándar poblacional}]$` imprimirá '*s*' en la salida `PDF` y `NVDA/MathCAT` verbalizará *«desviación estándar poblacional»*.

Las llaves `sub-index` y `read-sub` no tienen efecto al usar `read-arg`.

`silent` = {⟨*true* | *false*⟩} por defecto: *false*

Establece si se *silencia* la verbalización en `NVDA/MathCAT` del símbolo al ejecutar el 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.

Los comandos `\sprango`, `\spriq`, `\spmin` y `\spmax`

`\sprango`	`\sprango`	`\sprango[`⟨*key* = *val*⟩`]`
`\spriq`	`\spriq`	`\spriq[`⟨*key* = *val*⟩`]`
`\spmin`	`\spmin`	`\spmin[`⟨*key* = *val*⟩`]`
`\spmax`	`\spmax`	`\spmax[`⟨*key* = *val*⟩`]`

Los comandos `\sprango`, `\spriq`, `\spmin` y `\spmax` trabajan en *modo matemático* y reciben el *argumento opcional* `[`⟨*key* = *val*⟩`]` mediante el cual se establece la *«forma»* en la que se verbalizarán en `NVDA/MathCAT` estas medidas de la estadística descriptiva.

Si no se proporciona el *argumento opcional*, `$\sprango$` imprimirá '*R*' en la salida `PDF` y `NVDA/MathCAT` verbalizará *«rango»*; `$\spriq$` imprimirá 'RIQ' y verbalizará *«rango intercuartil»*; `$\spmin$` imprimirá '$x_{\min}$' y verbalizará *«mínimo»*; `$\spmax$` imprimirá '$x_{\max}$' y verbalizará *«máximo»*.

Llaves para `\sprango`, `\spriq`, `\spmin` **y** `\spmax`

Las llaves `silent`, `read-arg`, `sub-index` y `read-sub` de estos comandos funcionan como las de `\spmedia` (v. pág. 43), con una excepción: `\spmin` y `\spmax` solo tienen `silent` y `read-arg`, porque su símbolo ya lleva un subíndice.

Los comandos `\spcuartil`, `\spdecil` y `\spPerc`

`\spcuartil`	`\spcuartil{`⟨*k*⟩`}`	`\spcuartil[`⟨*key* = *val*⟩`]{`⟨*k*⟩`}`
`\spdecil`	`\spdecil{`⟨*k*⟩`}`	`\spdecil[`⟨*key* = *val*⟩`]{`⟨*k*⟩`}`
`\spPerc`	`\spPerc{`⟨*k*⟩`}`	`\spPerc[`⟨*key* = *val*⟩`]{`⟨*k*⟩`}`

Los comandos `\spcuartil`, `\spdecil` y `\spPerc` trabajan en *modo matemático* y reciben el *argumento obligatorio* `{`⟨*k*⟩`}`: un número entero sin signo, de 1 a 3 en `\spcuartil`, de 1 a 9 en `\spdecil` y de 1 a 99 en `\spPerc`. Cualquier otro valor devuelve un "error" y se detiene la compilación del documento.

Si no se proporciona el *argumento opcional* `[`⟨*key* = *val*⟩`]`, `$\spcuartil{1}$` imprimirá 'Q_1' en la salida `PDF` y `NVDA/MathCAT` verbalizará *«cuartil 1»*; `$\spdecil{3}$` imprimirá 'D_3' y verbalizará *«decil 3»*; `$\spPerc{25}$` imprimirá 'P_{25}' y verbalizará *«percentil 25»*.

©2026 by Pablo González L

Llaves para `\spcuartil`, `\spdecil` **y** `\spPerc`

`silent` = {⟨*true* | *false*⟩} por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT del símbolo al ejecutar el 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`, sí verbalizará la salida del comando. Un índice inválido devuelve siempre un "error" y se detiene la compilación del documento, también con `silent=true`.

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

Sobrescribe lo que verbalizará NVDA/MathCAT al ejecutar el comando sin modificar la impresión de este. El valor de esta llave *siempre* debe pasarse entre llaves '`{ }`'.
Por ejemplo: `$\spcuartil[read-arg={primer cuartil}]{1}$` imprimirá 'Q_1' en la salida PDF y NVDA/MathCAT verbalizará *«primer cuartil»*.

El comando `\spFrec`

`\spFrec` `\spFrec{`⟨*símbolo*⟩`}` `\spFrec[`⟨*key* = *val*⟩`]{`⟨*símbolo*⟩`}`

El comando `\spFrec` trabaja en *modo matemático* y recibe el *argumento obligatorio* {⟨*símbolo*⟩}: el símbolo de una frecuencia, escrito tal como se imprime. Los símbolos válidos son `f`, `F`, `f_{r}`, `F_{r}`, `f_{r}\%` y `F_{r}\%`; cualquier otro devuelve un "error" y se detiene la compilación del documento, salvo que se use la llave `read-arg`.

Si no se proporciona el *argumento opcional* [⟨*key* = *val*⟩], `$\spFrec{f}$` imprimirá 'f' en la salida PDF y NVDA/MathCAT verbalizará *«frecuencia absoluta»*; `$\spFrec{F}$` imprimirá 'F' y verbalizará *«frecuencia absoluta acumulada»*; `$\spFrec{f_{r}}$` imprimirá 'f_r' y verbalizará *«frecuencia relativa»*; `$\spFrec{F_{r}}$` imprimirá 'F_r' y verbalizará *«frecuencia relativa acumulada»*; `$\spFrec{f_{r}\%}$` imprimirá 'f_r%' y verbalizará *«frecuencia porcentual»*; `$\spFrec{F_{r}\%}$` imprimirá 'F_r%' y verbalizará *«frecuencia porcentual acumulada»*.

Llaves para `\spFrec`

`sub-index` = {⟨*subíndice*⟩} por defecto: *no usado*

Establece un *subíndice* sobre el símbolo, que NVDA/MathCAT verbaliza después del nombre de la frecuencia con la palabra *«sub»*. Solo es válida con los símbolos `f` y `F`. El valor de esta llave *siempre* debe pasarse entre llaves '`{ }`'. Por ejemplo: `$\spFrec[sub-index={i}]{f}$` imprimirá 'f_i' en la salida PDF y NVDA/MathCAT verbalizará *«frecuencia absoluta sub i»*.

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

Establece si se *usa* la palabra *«sub»* en lugar de *«de»* al verbalizar el subíndice establecido por la llave `sub-index`, sin modificar la impresión del comando. Con `$\spFrec[sub-index={i},read-sub=false]{f}$`, NVDA/MathCAT verbalizará *«frecuencia absoluta de i»*.

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

Sobrescribe el *nombre de la frecuencia* que verbalizará NVDA/MathCAT al ejecutar el comando sin modificar la impresión de este. Es la única forma de usar un símbolo que no está en la tabla anterior: `$\spFrec[read-arg={frecuencia absoluta}]{n_{i}}$` imprimirá 'n_i' y verbalizará *«frecuencia absoluta»*. El valor de esta llave *siempre* debe pasarse entre llaves '`{ }`'.

Las llaves `sub-index` y `read-sub` no tienen efecto al usar `read-arg`.

`silent` = {⟨*true* | *false*⟩} por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT del símbolo al ejecutar el 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`, sí verbalizará la salida del comando. Un símbolo inválido devuelve siempre un "error" y se detiene la compilación del documento, también con `silent=true`.

El comando `\spcase`

`\spcase` `\spcase{`⟨*letra*⟩`}` `\spcase[`⟨*key* = *val*⟩`]{`⟨*letra*⟩`}`

El comando `\spcase` trabaja en *modo matemático* y recibe el *argumento obligatorio* {⟨*letra*⟩}: la letra que acompaña a los casos de una probabilidad. Las letras válidas son `f`, `p` y `t`; cualquier otra devuelve un "error" y se detiene la compilación del documento.

Si no se proporciona el *argumento opcional* [⟨*key* = *val*⟩], `$\spcase{f}$` imprimirá 'C_f' en la salida PDF y NVDA/MathCAT verbalizará *«casos favorables»*; `$\spcase{p}$` imprimirá 'C_p' y verbalizará *«casos posibles»*; `$\spcase{t}$` imprimirá 'C_t' y verbalizará *«casos totales»*.

Llaves para `\spcase`

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

©2026 by Pablo González L

Sobrescribe el *nombre de los casos* que verbalizará NVDA/MathCAT al ejecutar el comando sin modificar la impresión de este. El valor de esta llave *siempre* debe pasarse entre llaves '{ }'. Por ejemplo: `$\spcase[read-arg={casos que cumplen}]{f}$` imprimirá 'C_f' en la salida PDF y NVDA/MathCAT verbalizará *«casos que cumplen»*.

`silent` = {⟨*true* | *false*⟩} por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT del símbolo al ejecutar el 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`, sí verbalizará la salida del comando. Una letra inválida devuelve siempre un "error" y se detiene la compilación del documento, también con `silent=true`.

El comando \spOmega

`\spOmega` `\spOmega` `\spOmega[⟨key = val⟩]`

El comando `\spOmega` trabaja en *modo matemático* y recibe el *argumento opcional* `[⟨key = val⟩]` mediante el cual se establece la *«forma»* en la que se verbalizará en NVDA/MathCAT el espacio muestral.

Si no se proporciona el *argumento opcional*, `$\spOmega$` imprimirá 'Ω' en la salida PDF y NVDA/MathCAT verbalizará *«espacio muestral»*.

Llaves para \spOmega

`card` = {⟨*true* | *false*⟩} por defecto: *false*

Establece si se *imprime* el símbolo '#' antes de la letra, para representar la cardinalidad del espacio muestral. Si se establece en `true`, `$\spOmega[card=true]$` imprimirá '$\#\Omega$' en la salida PDF y NVDA/MathCAT verbalizará *«cardinalidad del espacio muestral»*.

`sub-index` = {⟨*subíndice*⟩} por defecto: *no usado*

Establece un *subíndice* sobre la letra, que NVDA/MathCAT verbaliza después del nombre con la palabra *«sub»*. El valor de esta llave *siempre* debe pasarse entre llaves '{ }'.

Por ejemplo: `$\spOmega[sub-index={1}]$` imprimirá 'Ω_1' en la salida PDF y NVDA/MathCAT verbalizará *«espacio muestral sub 1»*; `$\spOmega[card=true,sub-index={1}]$` imprimirá '$\#\Omega_1$' y verbalizará *«cardinalidad del espacio muestral sub 1»*.

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

Sobrescribe el *nombre* que verbalizará NVDA/MathCAT al ejecutar el comando sin modificar la impresión de este. El valor de esta llave *siempre* debe pasarse entre llaves '{ }'.
Por ejemplo: `$\spOmega[read-arg={universo}]$` imprimirá 'Ω' en la salida PDF y NVDA/MathCAT verbalizará *«universo»*.

La llave `sub-index` no tiene efecto al usar `read-arg`.

`silent` = {⟨*true* | *false*⟩} por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT del símbolo al ejecutar el 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`, sí verbalizará la salida del comando.

El comando \spPval

`\spPval` `\spPval{⟨casos⟩}{⟨total⟩}` `\spPval[⟨key = val⟩]{⟨casos⟩}{⟨total⟩}`

El comando `\spPval` trabaja en *modo matemático* y recibe *dos argumentos obligatorios*: {⟨*casos*⟩} y {⟨*total*⟩}, los tipografía como una fracción y devuelve el *«valor de una probabilidad»*.

Si no se proporciona el *argumento opcional* `[⟨key = val⟩]`, `$\spPval{1}{5}$` imprimirá '$\frac{1}{5}$' en la salida PDF y NVDA/MathCAT verbalizará *«1 de 5»*.

Los argumentos {⟨*casos*⟩} y {⟨*total*⟩} se tipografían directamente, sin procesamiento semántico adicional.

Llaves para \spPval

`silent` = {⟨*true* | *false*⟩} por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT del *valor de la probabilidad* 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`, sí verbalizará la salida del comando.

`use-dfrac` = {⟨*true* | *false*⟩} por defecto: *false*

Establece si la fracción del *valor de la probabilidad* se imprime usando `\dfrac` en lugar de `\frac`. Si se establece en `true`, `$\spPval[use-dfrac=true]{1}{5}$` imprimirá la fracción en tamaño *display* '$\dfrac{1}{5}$' en la salida PDF; si se establece en `false`, la fracción se imprime con `\frac` siguiendo el tamaño matemático del contexto en curso.

El comando \spProb

\spProb `\spProb{⟨evento⟩}` `\spProb[⟨key = val⟩]{⟨evento⟩}`

El comando \spProb trabaja en *modo matemático* y recibe el *argumento obligatorio* {⟨evento⟩}: un evento, o dos eventos unidos por *exactamente uno* de los operadores \cap, \cup, \mid o el carácter '|', escritos tal como se imprimen. El comando imprime la probabilidad entre paréntesis precedida de la letra 'P'. Cualquier otro argumento, como uno vacío, con llaves o paréntesis sin cerrar, con un evento vacío o con más de un operador a la vez, devuelve un "error" y se detiene la compilación del documento. Lo que va entre llaves, paréntesis o corchetes no se examina, de modo que `$\spProb{A \cap (B \cup C)}$` es válido.

Si no se proporciona el *argumento opcional* [⟨key = val⟩], `$\spProb{A}$` imprimirá '$P(A)$' en la salida PDF y NVDA/MathCAT verbalizará *«probabilidad de A»*; `$\spProb{A \cap B}$` imprimirá '$P(A \cap B)$' y verbalizará *«probabilidad de A y B»*; `$\spProb{A \cup B}$` imprimirá '$P(A \cup B)$' y verbalizará *«probabilidad de A o B»*; `$\spProb{A \mid B}$`, o `$\spProb{A | B}$`, imprimirá '$P(A \mid B)$' y verbalizará *«probabilidad de A dado B»*.

Llaves para \spProb

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

Sobrescribe lo que verbalizará NVDA/MathCAT al ejecutar el comando sin modificar la impresión de este. El valor de esta llave *siempre* debe pasarse entre llaves '{ }'.

Por ejemplo: `$\spProb[read-arg={probabilidad del suceso}]{A}$` imprimirá '$P(A)$' en la salida PDF y NVDA/MathCAT verbalizará *«probabilidad del suceso»*.

silent = {⟨*true* | *false*⟩} por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT de la probabilidad al ejecutar el 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, sí verbalizará la salida del comando. Un argumento inválido devuelve siempre un "error" y se detiene la compilación del documento, también con silent=true.

Dentro del argumento de \spProb, los operadores \cap, \cup y \mid que quedan entre llaves o paréntesis, por ejemplo en `$\spProb{A \cap (B \cup C)}$`, también se verbalizan como «y», «o» y «dado», respectivamente.

El comando \spcomp

\spcomp `\spcomp{⟨evento⟩}` `\spcomp[⟨key = val⟩]{⟨evento⟩}`

El comando \spcomp trabaja en *modo matemático* y recibe el *argumento obligatorio* {⟨evento⟩}: el evento del que se quiere el complemento, escrito tal como se imprime. El comando imprime el complemento con el exponente 'c' y se puede usar solo o dentro del argumento de \spProb. Si el evento contiene una operación entre conjuntos, que es \cap, \cup, \setminus o \mid, se imprime entre paréntesis y se verbaliza entre paréntesis para no confundir el complemento de toda la operación con el de su primer evento.

Si no se proporciona el *argumento opcional* [⟨key = val⟩], `$\spcomp{A}$` imprimirá 'A^c' en la salida PDF y NVDA/MathCAT verbalizará *«complemento de A»*; `$\spProb{A \cap \spcomp{B}}$` imprimirá '$P(A \cap B^c)$' y verbalizará *«probabilidad de A y complemento de B»*; `$\spcomp{A \cup B}$` imprimirá '$(A \cup B)^c$' y verbalizará *«complemento de se abren paréntesis A o B se cierran paréntesis»*.

Llaves para \spcomp

bar = {⟨*true* | *false*⟩} por defecto: *false*

Establece si el complemento se imprime con una *barra horizontal* sobre el evento en lugar del exponente 'c'. Si se establece en true, `$\spcomp[bar=true]{A}$` imprimirá '$\overline{A}$' en la salida PDF; la verbalización es la misma en ambos casos.

wrap-parens = {⟨*auto* | *true* | *false*⟩} por defecto: *auto*

Establece si se *imprimen* paréntesis redondos '(...)' alrededor del evento cuando se usa el exponente 'c'. Esta llave admite true, false y auto. Con auto se agregan solo si el evento contiene \cap, \cup, \setminus o \mid; con true siempre, y con false nunca. Con bar=true la barra cubre el evento y no se agregan paréntesis. Como NVDA/MathCAT no verbaliza la barra, cuando el evento es compuesto y se usa bar=true el complemento se verbaliza *«complemento de la expresión»*; por ejemplo, `$\spcomp[bar=true]{A \cup B}$` se verbaliza *«complemento de la expresión A o B»*, para dejar claro que abarca todo el evento.

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

Sobrescribe lo que verbalizará NVDA/MathCAT al ejecutar el comando sin modificar la impresión de este. El valor de esta llave *siempre* debe pasarse entre llaves '{ }'.
Por ejemplo: `$\spcomp[read-arg={no ocurre A}]{A}$` imprimirá 'A^c' en la salida PDF y NVDA/MathCAT verbalizará *«no ocurre A»*.

silent = {⟨*true* | *false*⟩} por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT del complemento al ejecutar el 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, sí verbalizará la salida del comando.

©2026 by Pablo González L

El comando \spcomb

\spcomb `\spcomb{⟨n, k⟩}` `\spcomb[⟨key = val⟩]{⟨n, k⟩}`

El comando `\spcomb` trabaja en *modo matemático* y escribe los símbolos de conteo: combinaciones, variaciones, arreglos y permutaciones, con o sin repetición, y las permutaciones circulares. Recibe un único *argumento obligatorio* con elementos separados por comas: `{⟨n⟩}`, el total de elementos, y, después de la primera coma, `{⟨k⟩}`, los elementos que se toman. Según el símbolo, lo que sigue a la primera coma puede ser *la lista de repeticiones*, en las permutaciones con repetición, o puede no escribirse, en las permutaciones sin repetición y circulares, que solo llevan `{⟨n⟩}`.

Los argumentos aceptan números o símbolos; si ambos son números enteros, el comando comprueba que $k \leq n$, salvo cuando hay repetición, y que las repeticiones no sumen más de n.

Si no se proporciona el *argumento opcional* `[⟨key = val⟩]`, `$\spcomb{5,2}$` imprimirá '$\binom{5}{2}$' en la salida PDF y NVDA/MathCAT verbalizará *«5 sobre 2»*; `$\spcomb[kind=variation]{5,2}$` imprimirá 'V_2^5' y verbalizará *«variación de 5 elementos tomados de a 2»*; `$\spcomb[kind=permutation]{n}$` imprimirá 'P_n' y verbalizará *«permutación de n elementos»*.

Llaves para \spcomb

`kind` = {⟨*combination* | *variation* | *arrangement* | *permutation*⟩} por defecto: *combination*

Establece el tipo de símbolo: combinaciones, que son C o $\binom{n}{k}$, variaciones, que son V, arreglos, que son A, o permutaciones, que son P. Las variaciones y los arreglos se escriben distinto pero se verbalizan según su nombre, *«variación»* y *«arreglo»*. Con `kind=permutation` el `{⟨k⟩}` es opcional: sin él, con `$\spcomb[kind=permutation]{n}$`, se imprime P_n y se verbaliza *«permutación de n elementos»*, y con él, con `$\spcomb[kind=permutation]{n,k}$`, se imprime P_k^n y se verbaliza *«permutación de n elementos tomados de a k»*.

`repeat` = {⟨*true* | *false*⟩} por defecto: *false*

Establece si los elementos pueden *repetirse*. Si se establece en `true`, se agrega una R a la letra y *«con repetición»* a la verbalización: `$\spcomb[kind=variation,repeat=true]{5,2}$` imprimirá 'VR_2^5' y verbalizará *«variación con repetición de 5 elementos tomados de a 2»*; si se establece en `false`, no se agrega nada. Con `kind=permutation` y `repeat=true` lo que sigue a la primera coma es la *lista de repeticiones*, que también va separada por comas y tiene al menos dos elementos.
Por ejemplo: `$\spcomb[kind=permutation,repeat=true]{10,3,3,4}$` imprimirá '$PR_{10}^{3,3,4}$' y verbalizará *«permutación con repetición de 10 elementos con repeticiones 3 3 y 4»*.

`circular` = {⟨*true* | *false*⟩} por defecto: *false*

Establece si se trata de una *permutación circular*. Si se establece en `true`, `$\spcomb[kind=permutation,circular=true]{n}$` imprimirá 'PC_n' en la salida PDF y NVDA/MathCAT verbalizará *«permutación circular de n elementos»*; si se establece en `false`, la permutación es la usual. Solo es válida con `kind=permutation` y `repeat=false`, y entonces el comando solo recibe `{⟨n⟩}`.

`print-sty` = {⟨*auto* | *binom* | *letter*⟩} por defecto: *auto*

Establece cómo se imprimen las combinaciones sin repetición. Si se establece en `auto` o `binom`, se imprime el coeficiente binomial $\binom{n}{k}$ y NVDA/MathCAT lo verbaliza *«n sobre k»*; si se establece en `letter`, `$\spcomb[print-sty=letter]{5,2}$` imprimirá 'C_2^5' y NVDA/MathCAT verbalizará *«combinación de 5 elementos tomados de a 2»*. El valor `binom` solo es válido para combinaciones sin repetición.

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

Sobrescribe lo que verbalizará NVDA/MathCAT al ejecutar el comando sin modificar la impresión de este. El valor de esta llave *siempre* debe pasarse entre llaves '`{ }`'.
Por ejemplo: `$\spcomb[read-arg={cinco sobre dos}]{5,2}$` imprimirá '$\binom{5}{2}$' en la salida PDF y NVDA/MathCAT verbalizará *«cinco sobre dos»*.

`silent` = {⟨*true* | *false*⟩} por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT del símbolo al ejecutar el 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`, sí verbalizará la salida del comando.

Un argumento inválido devuelve siempre un "error" y se detiene la compilación del documento, también con `silent=true`: argumentos vacíos, una lista de repeticiones con menos de dos elementos o con un elemento vacío, $k > n$ sin repetición, repeticiones que suman más de n, `{⟨k⟩}` ausente cuando el símbolo lo necesita o presente cuando no lo admite, y combinaciones de llaves que no existen: `circular` sin `kind=permutation` o con `repeat`, o `print-sty=binom` fuera de las combinaciones sin repetición.

El comando \spfact

\spfact `\spfact{⟨n⟩}` `\spfact[⟨key = val⟩]{⟨n⟩}`

El comando `\spfact` trabaja en *modo matemático* y recibe el *argumento obligatorio* `{⟨n⟩}`. Imprime el factorial de `{⟨n⟩}`. Si `{⟨n⟩}` es una expresión, los paréntesis se escriben en el argumento: `$\spfact{(n-1)}$` imprimirá '$(n-1)!$'.

©2026 by Pablo González L

Si no se proporciona el *argumento opcional* [⟨*key* = *val*⟩], $\spfact{5}$ imprimirá '5!' en la salida PDF y NVDA/MathCAT verbalizará *«5 factorial»*.

El comando \spfact devuelve un "error" y se detiene la compilación del documento cuando el argumento está vacío.

Llaves para \spfact

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

Sobrescribe lo que verbalizará NVDA/MathCAT al ejecutar el comando sin modificar la impresión de este. El valor de esta llave *siempre* debe pasarse entre llaves '{ }'.
Por ejemplo: $\spfact[read-arg={cinco factorial}]{5}$ imprimirá '5!' en la salida PDF y NVDA/MathCAT verbalizará *«cinco factorial»*.

silent = {⟨*true* | *false*⟩} por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT del factorial al ejecutar el 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, sí verbalizará la salida del comando.

10 Modo mathml-AF y spmmli-tool

Las matemáticas de un documento *tagged* PDF se configuran con la llave math/setup de \DocumentMetadata. Todos los ejemplos de este documento usan mathml-SE, que significa *Structure Elements*: el MathML de cada fórmula se inserta en el árbol de etiquetas del PDF. Existe además el modo mathml-AF, que significa *Associated Files* o *«archivos asociados»*, en el que las fórmulas se leen desde un archivo externo que contiene un MathML por cada fórmula, en lugar de generarse al compilar.

10.1 Cuál usar y cuándo

Como mathml-AF lee las fórmulas desde un archivo fijo, los cambios que haga en las matemáticas *no se reflejan* en él mientras no lo genere de nuevo: si agrega, borra o modifica fórmulas con frecuencia, el PDF seguirá leyendo el contenido anterior. Por eso se recomienda:

- usar mathml-SE mientras el documento está en desarrollo, y
- usar mathml-AF solo en documentos terminados.

La ventaja de mathml-AF es que el archivo asociado contiene el MathML de cada fórmula y puede editarse: cambiar su contenido cambia la lectura, e incluso se puede escribir MathML directamente.

Editar el archivo asociado es una opción para quien sabe lo que hace: spanish-mathml-intent no revisa lo que se escriba en él.

10.2 De mathml-SE a mathml-AF

1. Escriba y revise el documento con mathml-SE.
2. Cuando esté terminado, ejecute en una terminal spmmli-tool archivo.tex. Se genera el archivo archivo-mathml.html.
3. En \DocumentMetadata, cambie mathml-SE por mathml-AF y compile de nuevo.
4. Si después modifica alguna fórmula, repita desde el segundo paso con la opción -o, que reemplaza el archivo existente.

La línea del tercer paso queda así:

```
\DocumentMetadata
  {
    lang=es-CL,
    pdfversion=2.0,
    pdfstandard=ua-2,
    tagging=on,
    tagging-setup={math/setup={mathml-AF}},
  }
```

10.3 La herramienta spmmli-tool

El script spmmli-tool se instala junto con el paquete. Prepara los archivos de MathML que genera luamml[21] para el modo mathml-AF: elimina de ellos unos caracteres invisibles, U+2063, los *«fantasmas»*, y escribe el resultado en archivo-mathml.html. También valida el PDF, siempre que las herramientas de la §1.7 estén instaladas. Según el archivo que reciba, hace lo siguiente:

Tabla 8: Modos de spmmli-tool

Archivo	Qué hace
archivo.tex	Compila una vez en un directorio temporal con AFghost=true, obtiene el archivo de MathML de luamml, lo limpia y escribe archivo-mathml.html. El PDF temporal se descarta y no se valida.
archivo-luamml-mathml.html	Solo limpia un archivo ya generado.
archivo.pdf	Solo valida el PDF con rnv-wrapp y veraPDF para PDF/UA-2, si están en el PATH.

Las opciones son:

Tabla 9: Opciones de spmmli-tool

Opción	Efecto
-c, --check-only	Solo valida; el archivo debe ser un PDF.
-o, --overwrite	Reemplaza los archivos -mathml.html existentes.
-p, --path=(⟨*carpeta*⟩)	Carpeta donde se escriben los archivos, que se crea si no existe.
-e, --engine=(⟨*motor*⟩)	Motor con que se compila; por defecto lualatex-dev si existe y, si no, lualatex.
-s, --strict	Se detiene en la primera falla; sin ella procesa todos los archivos.
-h, --help	Muestra la ayuda.
-v, --version	Muestra la versión.

Si algún paso falla, el código de salida de spmmli-tool es distinto de cero. Pueden indicarse varios archivos a la vez y la extensión puede omitirse. Algunos ejemplos:

```
spmmli-tool -c ejemplo.pdf
spmmli-tool -c -s ejemplo
spmmli-tool ejemplo-luamml-mathml.html
spmmli-tool -o -p build/ ejemplo.tex
```

11 Ejemplos de PDF accesible y etiquetado

Esta sección muestra el uso de spanish-mathml-intent en documentos *tagged* PDF con LuaLaTeX. Los archivos están en el directorio sources/test-pkg/: los cinco primeros prueban *todos* los comandos del paquete, agrupados por eje, y los tres últimos son ejemplos reales, del tipo de documento que haría un docente de aula. Cada uno se compila con lualatex-dev en los dos modos, mathml-SE y mathml-AF (§10), y los PDF resultantes se validan con rnv-wrapp y veraPDF (§1.7).

This section is just to show the compatibility of spanish-mathml-intent with tagged PDF using LuaLaTeX. The attached files are intended as examples and, in a way, to save time for Matthew Bertucci (@mbertucci), who runs the tests on the LaTeX side, when he sees this package and adds it to The LaTeX Tagged PDF repository.

Para compilar los ejemplos con lualatex-dev se requieren los paquetes babel con la opción spanish, spintent y hyperref, junto con la línea de \DocumentMetadata (§1.1). Las versiones mathml-AF se obtienen cambiando mathml-SE por mathml-AF después de ejecutar spmmli-tool.

Los archivos .tex van adjuntos a este PDF, junto con los PDF ya compilados en cada modo: al final de cada descripción están el archivo fuente y los PDF en mathml-SE y en mathml-AF.

- El archivo spintent-01.tex contiene las pruebas de \spnum, \spunit, \spmoney y \spshort, junto con \spnewunit y \spunitalias. Archivo fuente, *tagged* PDF mathml-SE y mathml-AF.
- El archivo spintent-02.tex contiene las pruebas de las utilidades generales, que son \spsiglo, \sptime y \spdate, y de las utilidades para símbolos, que son \spnewread, \spread, \spusep, \spdsym y \spmsym. Archivo fuente, *tagged* PDF mathml-SE y mathml-AF.
- El archivo spintent-03.tex contiene las pruebas de las utilidades para números: \spdiv, \spmul, \spnD, \spnM, \spmcd, \spmcm, \spnZ, \spmQ, \sprazon y \spProp. Archivo fuente, *tagged* PDF mathml-SE y mathml-AF.
- El archivo spintent-04.tex contiene las pruebas de las utilidades para geometría: \spcoord, \spPoint, \spvec, \sprecta, \sprayo, \spside, \spmside, \spangle, \spmangle, \spang, \sparc, \spmarc, \spPoly, \spAfig, \spPfig, \sprad, \spdiam, \spcirc, \spAcirc, \spPcirc y \spVol. Archivo fuente, *tagged* PDF mathml-SE y mathml-AF.

- El archivo `spintent-05.tex` contiene las pruebas de las utilidades para álgebra: `\spinterval`, `\spfun`, `\spdom`, `\sprec`, `\spcod`, `\spfuncomp` y `\spabs`; y para estadística: `\spProm`, `\spmedia`, `\spmediana`, `\spmoda`, `\spdesv`, `\spvarz`, `\sprango`, `\spriq`, `\spmin`, `\spmax`, `\spcuartil`, `\spdecil`, `\spPerc`, `\spFrec`, `\spcase`, `\spOmega`, `\spPval`, `\spProb`, `\spcomp`, `\spcomb` y `\spfact`. Archivo fuente, *tagged* PDF `mathml-SE` y `mathml-AF`.
- El archivo `spintent-06.tex` es un ejemplo real: una guía de ejercicios de divisibilidad, fracciones, razones y enteros. Archivo fuente, *tagged* PDF `mathml-SE` y `mathml-AF`.
- El archivo `spintent-07.tex` es un ejemplo real: una guía de ejercicios de geometría con ángulos, áreas y perímetros. Archivo fuente, *tagged* PDF `mathml-SE` y `mathml-AF`.
- El archivo `spintent-08.tex` es un ejemplo real: una guía de ejercicios de estadística y probabilidad. Archivo fuente, *tagged* PDF `mathml-SE` y `mathml-AF`.

©2026 by Pablo González L

A Work in Progress

Todo comenzó cuando salió la versión de desarrollo de LaTeX release 2025-11-01 [33], la cual implementó `\MathMLintent` y `\MathMLarg`. Mi idea inicial era modesta: un paquete llamado `spunit`, pensado para las unidades de medida. Pero una unidad casi nunca viene sola, y de ahí nació `\spnum`; luego vinieron `\spmoney` y `\spshort` en reemplazo de `\sptext`, hasta formar *«los cuatro grandes»*, que cubren lo esencial de un documento en español con `babel`.

Durante un tiempo fui desarrollando pequeñas utilidades, una a una, pero pronto quedaron cortas para lo que necesita un docente de enseñanza básica y media. Lo que partió como un conjunto de utilidades sueltas se transformó en un trabajo por ejes: números, geometría, álgebra y estadística. El nombre también cambió. Después de leer mucho sobre el tema, vi una y otra vez la palabra *intent*, y lo que yo *intentaba* hacer sonaba mejor en mi cabeza como spanish-mathml-intent que como `spunit`.

En el camino aprendí bastante más que Lua y LaTeX: a validar un PDF etiquetado con `rnv` y `veraPDF`[4], a construir baterías de tests de regresión, a comprender los modos `mathml-SE` y `mathml-AF`, a escuchar con `NVDA`/`MathCAT`[6, 7] una y otra vez, a revisar los *intents* del núcleo antes de crear uno propio, a definir una interfaz de usuario amigable y, sobre todo, a documentar, y documentar otra vez.

Esta sección es para mí, el de hoy y el de mañana. Hoy el proyecto está terminado: el conjunto de comandos está cerrado y no se agregarán más. Lo que sí seguiré haciendo es seguirle el paso al proyecto de etiquetado de LaTeX[3] y a `luamml`[21], para que el paquete siga funcionando con ellos. El código y la documentación están pensados para quien necesite un comando que aquí no existe: los comandos actuales son las plantillas para copiar. Por eso esta sección se llama *A Work in Progress*: porque un trabajo en progreso es el corazón de spanish-mathml-intent.

Queda además una idea abierta. El atributo *intent* no nació en este paquete: viene de una discusión más amplia sobre cómo eliminar la ambigüedad de `MathML` dejando que sea quien escribe el documento quien diga lo que quiso expresar[34]. Lo que hace spanish-mathml-intent es poner esa idea al servicio de la matemática y demás en el ámbito escolar en español, pero nada impide que otros idiomas tengan su propio paquete. Me gustaría ver algún día algo como un hipotético `ukenglish-mathml-intent`, con una interfaz pensada para su lengua y su escuela. Si alguien se anima, esta sección es una invitación, y la §15.2, en inglés, dice por dónde empezar.

12 Referencias

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

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

[3] The LaTeX Project. “LaTeX Tagged PDF Project - Documentation and resources”. Available from LaTeX project, https://latex3.github.io/tagging-project/documentation/, 2026.

[4] veraPDF Consortium. “veraPDF - Industry supported PDF/A and PDF/UA validator”. Available from veraPDF, https://verapdf.org/, 2026.

[5] Dual Lab. “ngPDF - Next Generation PDF demonstrator and validator”. Available from ngPDF, https://ngpdf.com/, 2026.

[6] NV Access. “NVDA - NonVisual Desktop Access screen reader”. Available from NV Access, https://www.nvaccess.org/, 2026.

[7] Soiffer, Neil. “MathCAT - Math to speech and braille translation”. Available from GitHub, https://github.com/daisy/MathCAT, 2026.

[8] ISO. “ISO 32000-2:2020 - Portable Document Format (PDF 2.0)”. Available from PDF Association, https://pdfa.org/resource/iso-32000-2/, 2020.

[9] 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.

[10] PDF Association. “PDF 2.0 Errata Collection 3”. Available from PDFA, https://pdfa.org/PDF-2-0-errata-collection-3-now-available/, 2026.

[11] PDF Association. “WTPDF - Well-Tagged PDF specification”. Available from PDFA, https://pdfa.org/wtpdf/, 2024.

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

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

©2026 by Pablo González L

[14] Adobe Inc. “Adobe Acrobat Reader - Reliable PDF viewer”. Available from adobe, https://get.adobe.com/reader/, 2026.

[15] Soiffer, Neil. “MathCAT Demo - Online demonstration of MathCAT”. Available from github, https://daisy.github.io/MathCATDemo/, 2026.

[16] The LaTeX Project. “LaTeX for authors current version”. Available from ctan, https://ctan.org/pkg/latex-base, 2026.

[17] learnlatex.org. “Learn LaTeX - Your first steps in LaTeX, in short lessons”. Available from learnlatex.org, https://www.learnlatex.org/es/, 2026.

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

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

[20] The LaTeX Project. “latex-lab – LaTeX laboratory”. Available from ctan, https://www.ctan.org/pkg/latex-lab, 2026.

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

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

[23] Bezos, Javier. “babel-spanish - Babel support for Spanish”. Available from ctan, https://www.ctan.org/pkg/babel-spanish, 2026.

[24] 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.

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

[26] Voß, Herbert. “libertinus-otf - Support for Libertinus OpenType”. Available from ctan, https://www.ctan.org/pkg/libertinus-otf, 2025.

[27] Flipo, Daniel. “erewhon-math - Utopia based OpenType Math font”. Available from ctan, https://ctan.org/pkg/erewhon-math, 2026.

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

[29] Tsolomitis, Antonis. “newcomputermodern - The New Computer Modern family of fonts”. Available from ctan, https://ctan.org/pkg/newcomputermodern, 2026.

[30] The LaTeX Project. “Extensive support for hypertext in LaTeX”. Available from ctan, https://www.ctan.org/pkg/hyperref, 2026.

[31] The LaTeX Project. “The expl3 package”. Available from ctan, https://www.ctan.org/pkg/l3kernel, 2026.

[32] The LaTeX Project. “The LaTeX3 Interfaces”. Available from ctan, https://www.ctan.org/pkg/l3kernel, 2026.

[33] The LaTeX Project. “LaTeX News, Issue 42, November 2025”. Available from ctan, https://ctan.org/tex-archive/macros/latex/base, 2025.

[34] Carlisle, David and Libbrecht, Paul and Schubotz, Moritz and Soiffer, Neil. “Author Intent: Eliminating Ambiguity in MathML”. Available from arXiv, https://arxiv.org/abs/2407.06720, 2025.

©2026 by Pablo González L

13 Change history

v1.3 (ctan), 2026-11-01
- Documentation: the variables of the implementation are described in groups, by purpose.
- Documentation: new note on the plural of acronyms, which explains why the entry `ongs` prints "ONGs".
- The Spanish text of the manual follows the RAE rules.

v1.2 (ctan), 2026-10-24
- New subsection "Other languages" in the implementation, linked from "A Work in Progress".
- The title and the `PDF` metadata of the manual and of the example use the name "Spanish MathML intent".
- `\spfact` prints the exclamation mark with `\mathexclam`, as the other commands do with their symbols.
- Internal: the function that builds the intent of the statistics commands is split from the one that typesets them, and all function bodies follow the same layout.
- Internal: the Lua module has no unused code, sends its variables to TEX through helper functions, and is organized in indexed sections that keep their local variables to themselves.

v1.1 (ctan), 2026-10-22
- The command-line tool `spitool` is renamed `spmmli-tool` (script and man page).
- The message `pos-invalid-index` is renamed `stat-pos-invalid-index`.
- New abstract of the manual, in line with the description of the `README`.

v1.0 (ctan), 2026-10-20
- First public release.

©2026 by Pablo González L

14 Índice de la Documentación

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

C

Clases de documento:

article 1

Comandos provistos por spanish-mathml-intent:

\setspintent 6
\spAcirc 35
\spAfig 30
\spFrec 44
\spOmega 45
\spPcirc 37
\spPerc 43
\spPfig 32
\spPoint 23
\spPoly 30
\spProb 46
\spProm 42
\spProp 22
\spPval 45
\spVol 38
\spabs 41
\spangle 27
\spang 29
\sparc 29
\spcase 44
\spcirc 34
\spcod 40
\spcomb 47
\spcomp 46
\spcoord 23
\spcuartil 43
\spdate 15
\spdecil 43
\spdesv 42
\spdiam 33
\spdiv 18
\spdom 40
\spdsym 17
\spfact 47
\spfuncomp 41
\spfun 40
\spinterval 39
\spmQ 21
\spmangle 28
\spmarc 29
\spmax 43
\spmcd 19
\spmcm 20
\spmediana 42
\spmedia 42
\spmin 43
\spmoda 42
\spmoney 10
\spmside 26
\spmsym 17
\spmul 18
\spnD 18
\spnM 19
\spnZ 21
\spnewread 15
\spnewunit 10
\spnum 7
\sprad 33
\sprango 43
\sprayo 25
\sprazon 22
\spread 16
\sprecta 25
\sprec 40
\spriq 43
\spshort 12
\spside 26
\spsiglo 15
\sptime 15
\spunitalias 10
\spunit 8
\spusep 16
\spvarz 42
\spvec 24

K

Llaves de \spabs provistas por spanish-mathml-intent:

close 42
read-arg 42
silent 42
size 42

Llaves de \spAcirc provistas por spanish-mathml-intent:

concept 36
prefix 36
print-sym 36
read-arg 36
read-cmd 36
read-sty 36
read-sub 36
silent 36
sub-index 36

Llaves de \spAfig provistas por spanish-mathml-intent:

concept* 31
concept 31
prefix 31
print-sym 31
read-arg 31
read-cmd 31
read-sub 31
silent 32

Llaves de \spangle provistas por spanish-mathml-intent:

concept 28
prefix 27
read-cmd 27
recto 27
silent 28

Llaves de \sparc provistas por spanish-mathml-intent:

concept 29
prefix 29
read-cmd 29
silent 29

Llaves de \spcase provistas por spanish-mathml-intent:

read-arg 45
silent 45

Llaves de \spcirc provistas por spanish-mathml-intent:

concept 35
diameter 35

©2026 by Pablo González L

prefix . . . 35
print-sym . . . 35
radio-sty . . . 35
read-arg . . . 35
read-cmd . . . 34
read-sty . . . 35
read-sub . . . 35
silent . . . 35
spmside . . . 34
spside . . . 34
sub-index . . . 35

Llaves de \spcomb provistas por spanish-mathml-intent:
circular . . . 47
kind . . . 47
print-sty . . . 47
read-arg . . . 47
repeat . . . 47
silent . . . 47

Llaves de \spcomp provistas por spanish-mathml-intent:
bar . . . 46
read-arg . . . 46
silent . . . 46
wrap-parens . . . 46

Llaves de \spcoord provistas por spanish-mathml-intent:
silent . . . 23

Llaves de \spcuartil provistas por spanish-mathml-intent:
read-arg . . . 44
silent . . . 44

Llaves de \spdecil provistas por spanish-mathml-intent:
read-arg . . . 44
silent . . . 44

Llaves de \spdesv provistas por spanish-mathml-intent:
read-arg . . . 43
read-sub . . . 43
sigma . . . 43
silent . . . 43
sub-index . . . 43

Llaves de \spdiam provistas por spanish-mathml-intent:
concept . . . 33
print-sym . . . 34
read-arg . . . 34
read-cmd . . . 33
read-sub . . . 34
silent . . . 34
sub-index . . . 34
upper-case . . . 33

Llaves de \spdiv provistas por spanish-mathml-intent:
cifras-sep . . . 18
notation-sym . . . 18
read-parens . . . 18
read-period-sep . . . 18
read-sign . . . 18
read-sym . . . 18
set-sym . . . 18
wrap-parens . . . 18

Llaves de \spdom provistas por spanish-mathml-intent:
prefix . . . 41
silent . . . 41

Llaves de \spdsym provistas por spanish-mathml-intent:
prefix . . . 17
read-sym . . . 17
set-sym . . . 17
suffix . . . 17

Llaves de \spfact provistas por spanish-mathml-intent:
read-arg . . . 48
silent . . . 48

Llaves de \spFrec provistas por spanish-mathml-intent:
read-arg . . . 44
read-sub . . . 44
silent . . . 44
sub-index . . . 44

Llaves de \spfuncomp provistas por spanish-mathml-intent:
silent . . . 41

Llaves de \spfun provistas por spanish-mathml-intent:
concept . . . 40
prefix . . . 40
read-cmd . . . 40
silent . . . 40

Llaves de \spinterval provistas por spanish-mathml-intent:
concept . . . 39
prefix . . . 39
read-arg . . . 39
read-cmd . . . 39
silent . . . 40
size . . . 39

Llaves de \spmangle provistas por spanish-mathml-intent:
concept . . . 28
prefix . . . 28
print-sym . . . 28
read-cmd . . . 28
recto . . . 28
silent . . . 28

Llaves de \spmarc provistas por spanish-mathml-intent:
concept . . . 29
prefix . . . 29
read-cmd . . . 29
silent . . . 30

Llaves de \spmcd provistas por spanish-mathml-intent:
concept . . . 20
prefix . . . 20
read-sty . . . 20
result . . . 20
silent . . . 20
upper-case . . . 20

Llaves de \spmcm provistas por spanish-mathml-intent:
concept . . . 20
prefix . . . 21
read-sty . . . 20
result . . . 21
silent . . . 21
upper-case . . . 20

Llaves de \spmediana provistas por spanish-mathml-intent:
read-arg . . . 43
read-sub . . . 43
silent . . . 43
sub-index . . . 43

Llaves de \spmedia provistas por spanish-mathml-intent:
read-arg . . . 43
read-sub . . . 43
silent . . . 43
sub-index . . . 43

Llaves de \spmoda provistas por spanish-mathml-intent:
read-arg . . . 43
read-sub . . . 43
silent . . . 43
sub-index . . . 43

Llaves de \spmoney provistas por spanish-mathml-intent:
cifras-sep . . . 11
currency . . . 11

©2026 by Pablo González L

money-math . . . 11
sub-units . . . 11
Llaves de \spmQ provistas por spanish-mathml-intent:
join-word . . . 21
read-parens . . . 22
silent . . . 22
use-dfrac . . . 22
use-spnZ . . . 21
wrap-parens . . . 22
Llaves de \spmside provistas por spanish-mathml-intent:
concept . . . 27
prefix . . . 27
print-sym . . . 27
read-arg . . . 27
read-cmd . . . 26
read-sty . . . 27
silent . . . 27
Llaves de \spmsym provistas por spanish-mathml-intent:
prefix . . . 17
print-sym . . . 17
read-sym . . . 17
set-sym . . . 17
suffix . . . 17
Llaves de \spmul provistas por spanish-mathml-intent:
cifras-sep . . . 18
notation-sym . . . 18
read-parens . . . 18
read-period-sep . . . 18
read-sign . . . 18
read-sym . . . 18
set-sym . . . 18
wrap-parens . . . 18
Llaves de \spnD provistas por spanish-mathml-intent:
concept . . . 18
prefix . . . 18
result-num . . . 19
result . . . 19
silent . . . 19
Llaves de \spnM provistas por spanish-mathml-intent:
concept . . . 19
prefix . . . 19
result-num . . . 19
result . . . 19
silent . . . 19
Llaves de \spnum provistas por spanish-mathml-intent:
cifras-sep . . . 7
notation-sym . . . 8
read-period-sep . . . 7
read-sign . . . 7
Llaves de \spnZ provistas por spanish-mathml-intent:
cifras-sep . . . 21
read-parens . . . 21
read-sign . . . 21
wrap-parens . . . 21
Llaves de \spOmega provistas por spanish-mathml-intent:
card . . . 45
read-arg . . . 45
silent . . . 45
sub-index . . . 45
Llaves de \spPcirc provistas por spanish-mathml-intent:
concept . . . 37
prefix . . . 37
print-sym . . . 37
read-arg . . . 37
read-cmd . . . 37
read-sty . . . 37
read-sub . . . 37
silent . . . 38
sub-index . . . 37
Llaves de \spPerc provistas por spanish-mathml-intent:
read-arg . . . 44
silent . . . 44
Llaves de \spPfig provistas por spanish-mathml-intent:
concept* . . . 33
concept . . . 32
prefix . . . 32
print-sym . . . 33
read-arg . . . 32
read-cmd . . . 32
read-sub . . . 33
silent . . . 33
Llaves de \spPoint provistas por spanish-mathml-intent:
concept . . . 24
prefix . . . 24
read-cmd . . . 24
silent . . . 24
Llaves de \spPoly provistas por spanish-mathml-intent:
concept . . . 30
prefix . . . 30
print-sym . . . 30
read-cmd . . . 30
silent . . . 30
Llaves de \spProb provistas por spanish-mathml-intent:
read-arg . . . 46
silent . . . 46
Llaves de \spProm provistas por spanish-mathml-intent:
read-arg . . . 42
silent . . . 42
Llaves de \spProp provistas por spanish-mathml-intent:
inline . . . 23
silent . . . 23
use-dfrac . . . 23
Llaves de \spPval provistas por spanish-mathml-intent:
silent . . . 45
use-dfrac . . . 45
Llaves de \sprad provistas por spanish-mathml-intent:
concept . . . 33
read-arg . . . 34
read-cmd . . . 33
read-sub . . . 34
silent . . . 34
sub-index . . . 34
upper-case . . . 33
Llaves de \sprayo provistas por spanish-mathml-intent:
concept . . . 25
prefix . . . 25
read-cmd . . . 25
read-sty . . . 25
silent . . . 25
Llaves de \sprazon provistas por spanish-mathml-intent:
inline . . . 22
silent . . . 22
use-dfrac . . . 22
Llaves de \spread provistas por spanish-mathml-intent:
read-arg . . . 16
silent . . . 16
Llaves de \sprecta provistas por spanish-mathml-intent:
concept . . . 25
prefix . . . 25
read-cmd . . . 25

©2026 by Pablo González L

silent 25
Llaves de \spshort provistas por spanish-mathml-intent:
read-arg 12
small-caps 12
Llaves de \spside provistas por spanish-mathml-intent:
concept 26
prefix 26
read-arg 26
read-cmd 26
read-sty 26
silent 26
Llaves de \spsiglo provistas por spanish-mathml-intent:
print-era 15
Llaves de \spunit provistas por spanish-mathml-intent:
cifras-sep 8
notation-sym 8
print-cdot 8
read-cdot 8
read-period-sep 8
read-slash 8
Llaves de \spusep provistas por spanish-mathml-intent:
silent 16
size 16
Llaves de \spvarz provistas por spanish-mathml-intent:
read-arg 43
read-sub 43
sigma 43
silent 43
sub-index 43
Llaves de \spvec provistas por spanish-mathml-intent:
autofit 24
concept 24
prefix 24
read-cmd 24
silent 24
Llaves de \spVol provistas por spanish-mathml-intent:
concept 38
read-arg 38
read-cmd 38
read-sub 38
silent 38
sub-index 38
Llaves de \usepackage provistas por spanish-mathml-intent:
mathcal 2

M

Módulo Lua provisto por spanish-mathml-intent:
spintent.lua 4, 10–15

P

Paquetes :
babel-spanish 2
babel 2, 12, 49, 51
documentmetadata-support 2
enumext 1
expl3 7
fontspec 2
fourier-otf 2
hyperref 3, 49
l3keys 7
latex-lab-mathintent 2
libertinus-otf 2
luamml 2, 48, 49, 51
scontents 1
sourcecodepro 2
spintent 49
tagpdf 2, 12, 15
unicode-math 1

T

Herramienta provista por spanish-mathml-intent:
spmmli-tool 48

©2026 by Pablo González L

15 Implementation

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

15.1 General conventions

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

In the same way, variables of the form `\l__spintent_luaset_some_tl`, `\l__spintent_luaset_some_str` or `\l__spintent_luaset_some_clist`, also with the prefix of a command, such as `\l__spintent_spcomb_luaset_` are only declared in `expl3`; their values are set from Lua.

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

15.2 Other languages

The *intent* attribute is not tied to Spanish[34]. What belongs to this package is the Spanish reading of each concept, written in the names of the intents, such as `_valor-absoluto(`$x`)`, and the set of commands for the school curriculum it targets. A package for another language would keep the same idea, a command that typesets an expression and annotates it with an intent, but would choose its own intent names and its own commands.

Every command in this implementation follows the same order: variables, error messages, keys and internal functions, with the public command at the end. The code written in Lua only parses the argument and sets variables that `expl3` reads. That order is meant to be reused, under the terms of the LPPL. Anyone who wants to start a package for another language is welcome to do so, as a separate package with its own name.

15.3 Initial set up

Start the DocStrip guards.

```
⟨*package⟩
```

Identify the internal prefix, following the LaTeX3 DocStrip convention, for `l3doc` class.

```
⟨@@=spintent⟩
```

15.4 Declaration of the package

First we will make sure we have a minimum release version of LaTeX to work correctly, otherwise, we will return an “error”.

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

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

```
\ProvidesExplPackage {spanish-mathml-intent} {2026-11-01} {1.3} {Spanish MathML intent}
\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 ]
```

©2026 by Pablo González L

We check if the requirements for running the spanish-mathml-intent 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,~spanish-mathml-intent~will~make~the~intent!
          }
        \msg_info:nn { spintent } { env-success }
      }
      {
        \msg_new:nnnn { spintent } { fatal-engine-requires-luatex }
          {
            The~spanish-mathml-intent~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'~is~missing.
      }
      {
        The~spanish-mathml-intent~package~requires~tagged~PDF~active.\\
        You~must~declare~'\token_to_str:N \DocumentMetadata { tagging=on }'.
      }
    \msg_fatal:nn { spintent } { fatal-metadata-missing }
  }
```

15.5 Some kernel variants

\str_lowercase:V
\seq_set_from_clist:Ne
\int_set:NV

Non-standard kernel variants.

```
\cs_generate_variant:Nn \str_lowercase:n       { V }
\cs_generate_variant:Nn \seq_set_from_clist:Nn { Ne }
\cs_generate_variant:Nn \int_set:Nn            { NV }
```

(*End of definition for* \str_lowercase:V, \seq_set_from_clist:Ne, *and* \int_set:NV.)

15.6 Variables for the key mathcal, \rightanglesqr and \attribute

\l__spintent_luaset_cal_fam_ok_str
\l__spintent_luaset_sym_fam_ok_str
\l__spintent_luaset_scope_attr_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, the string variable \l__spintent_luaset_sym_fam_ok_str used by the function __spintent_luafun_right_angle_sqr: to copy \rightanglesqr alongside the string variable \l__spintent_luaset_scope_attr_str used by \attribute 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
\str_new:N \l__spintent_luaset_scope_attr_str
```

©2026 by Pablo González L

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

15.7 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. If it is `true`, the two ghost characters are marked with `sptmp="outer"` or `sptmp="inner"` in the file `-luamml-mathml.html`, so that the Lua script `spmmli-tool` can remove them before generating the `Associated~Files` structure of the *tagged* PDF for `mathml-AF` mode.

```
\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

A LuaTEX attribute number allocated from Lua with `luatexbase.new_attribute` and exposed as an integer. Every command that uses the function `\__spintent_luafun_mrow_intent_arg:nn` sets `\attribute \l__spintent_scope_attr_int = 1` inside its own group, which marks the nodes typeset within it as eligible for the queue.

```
\int_new:N  \l__spintent_scope_attr_int
\int_set:NV \l__spintent_scope_attr_int \l__spintent_luaset_scope_attr_str
```

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

_spintent_luafun_spunit_parse_product:nV
__spintent_luafun_spmoney_lookup_data:V
__spintent_luafun_spmQ_scale_int:Vn
__spintent_luafun_spfun_parse_and_set:v

Variants generated for `\__spintent_luafun_spunit_parse_product:nn`, used by `\spunit`; `\__spintent_luafun_spmoney_lookup_data:n`, used by `\spmoney`; `\__spintent_luafun_spmQ_scale_int:n` used by `\spmQ`; and `\__spintent_luafun_spfun_parse_and_set:n`, used by `\spfun`.

```
\cs_generate_variant:Nn \__spintent_luafun_spunit_parse_product:nn { nV }
\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_spfun_parse_and_set:n   { v }
```

(*End of definition for* `\__spintent_luafun_spunit_parse_product:nV` *and others.*)

15.8 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 define two warnings in case our fallback fails: the "warning" `mathcal-fallback` is raised when no suitable font or free math family is available, and the "warning" `right-angle-sqr-fallback` is raised when the right-angle glyph is replaced by `\rightangle`.

```
\hook_gput_code:nnn { begindocument } { spintent }
  {
    \__spintent_luafun_init_math_fallbacks:
  }
\msg_new:nnn { spintent } { mathcal-fallback }
  {
    Using~classic~'\token_to_str:N \mathcal'~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'~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 return a warning message and use \mathcal. Otherwise, that is, mathcal=false we 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 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:.)

15.9 Shared annotation mechanisms: the queue and the ghosts

Almost all commands in the spanish-mathml-intent 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 is called directly, not through an expl3 function. It pushes onto a queue one entry formed by a concept intent in '#1', which may be empty, and a Lua table literal arg= or intent= in '#2'. A pre_mlist_to_hlist_filter callback, installed before the one of luamml[21], walks the top-level noads of the formula in order. Each noad whose nucleus or subscript was typeset inside the group of a spanish-mathml-intent command, marked with the LuaTEX \attribute activated by every command that uses this queue, takes the next pending entry: arg or the literal intent is assigned to that leaf and, if the concept intent is not empty, the leaf is wrapped in a native <mrow> carrying it. The entries are consumed in order and those left when the formula ends are discarded. The queue does not see the claims of a native \MathMLarg or \MathMLintent.

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

The other two are the "ghost" characters, an invisible U+2063 tagged :silent through their own mathml_filter, which the queue never claims. A ghost is added as an extra sibling of the <mrow> so that it is not the only top-level content of the formula; otherwise luamml would promote it to <math intent=...>, as in issue #26.

__spintent_invisible_sep:

Places the ghost before the outer \MathMLintent of the formula. Only the first call in the document has effect; the following ones, in any command, do nothing.

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

Executing this function writes the following line 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 mathml-AF mode, you can remove the line generated by AFghost=true using the Lua script spmmli-tool.

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

`\__spintent_inner_sep:` Places the ghost just inside a native `\MathMLintent` group. Every call has effect, one for each such group of the formula, nested or sibling.

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

Executing this function writes the following line 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 mathml-AF mode, you can remove the line generated by AFghost=true using the Lua script spmmli-tool.

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

`\c__spintent_invisible_sep_int` A raw, untagged copy of the character U+2063 defined with the LuaTEX primitive `\Umathchardef`. It is used as content inside an explicit `\MathMLintent{`⟨*intent*⟩`}`, typically `:silent`.

```
\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." We follow this recommendation.

(*End of definition for* `\c__spintent_invisible_sep_int`.)

15.10 Normalize argument and affixes

We normalize the {⟨*argument*⟩} to support prefixes and 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 function 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, it converts the {⟨*argument*⟩} into a string with every character in category code 12, so it no longer matters how an underscore '_' was typed, trims the bounds, converts spaces '␣' and underscores '_' to hyphens '-', collapses multiple hyphens, and strips leading/trailing hyphens. Because underscores are converted *before* collapsing, any run mixing hyphens and underscores is reduced to a single hyphen, and the stored value never starts or ends with '-' or '_': the command that uses the value is the one that places the leading '_' of the concept name. 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. In the same way, `{ __foo___bar__ }` is stored as `foo-bar`.

The only exception is the exact value '_-', once the bounds are trimmed, which is stored as a single hyphen '-'. After the command prepends its own leading '_', this yields the *silent* literal '_-' documented for `\spread`. Any other value made only of hyphens, underscores and spaces is stored as *empty*.

```
\cs_new_protected:Nn \__spintent_sanitize_affix:Nn
  {
    \tl_if_blank:nTF {#2}
      { \tl_clear:N #1 }
      {
        \tl_set:Ne #1 { \tl_to_str:e {#2} }
        \tl_trim_spaces:N #1
        \str_if_eq:VnTF #1 { _- }
          { \tl_set:Nn #1 { - } }
          {
            \tl_replace_all:Nnn #1 { ~ } { - }
            \tl_replace_all:Nee #1 { \tl_to_str:n { _ } } { - }
            \__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`.)

15.11 The command `\setspintent`

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

15.11.1 Definition of 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
```

The "error" `unknown-command-setup` is raised when the {⟨*argument*⟩} '`#1`' passed to `\setspintent` is a *"non-existent"* command.

```
\msg_new:nnnn { spintent } { unknown-command-setup }
  {
    Cannot~configure~unknown~command~'\c_backslash_str #1'~\msg_line_context:.
  }
  {
    The~command~'\c_backslash_str #1'~is~not~defined.~Check~the~spelling~of~the~command~
    name.
  }
```

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

15.11.2 Internal functions for `\setspintent`

`\__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 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* 7.)

15.12 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:nnnn { spintent } { unknown-choice }
  {
    Invalid~value~'#3'~for~key~'#1';~use~one~of:~#2~\msg_line_context:.
  }
  {
    The~key~'#1'~only~accepts~one~of:~#2.
  }
\msg_new:nnnn { spintent } { cmd-key-unknown }
  {
    The~key~'#1'~is~unknown~by~command~
      '\c_backslash_str \l__spintent_command_name_str'~and~is~being~ignored~
      \msg_line_context:.
  }
  {
    Check~the~spelling~of~the~key~name.
  }
\msg_new:nnnn { spintent } { cmd-key-value-unknown }
  {
    The~key~'#1=#2'~is~unknown~by~command~
```

```
      '\c_backslash_str \l__spintent_command_name_str'~and~is~being~ignored~
      \msg_line_context:.
  }
  {
    Check~the~spelling~of~the~key~name.
  }
```

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

`\__spintent_unknown_keys_cmd:nn` `\__spintent_unknown_keys_cmd:Vn` `\__spintent_unknown_keys_cmd:n`

The internal function `\__spintent_unknown_keys_cmd:nn` raises the "error" for an unknown ⟨*key*⟩. Its first {⟨*argument*⟩} is the name of the key and the second one its value. If the value is blank it issues the message `cmd-key-unknown`; otherwise it issues `cmd-key-value-unknown`. The variant `\__spintent_unknown_keys_cmd:Vn` takes the name of the key from a variable. The function `\__spintent_unknown_keys_cmd:n` is the one used by the ⟨*key*⟩ `unknown` of each command: it receives the value and takes the name of the key from `\l_keys_key_str`. The name of the command shown in the message is stored in `\l__spintent_command_name_str`.

```
\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_cmd:nn` *and* `\__spintent_unknown_keys_cmd:n`.)

15.13 Definition of core base variables

`\__spintent_luafun_clean_split_and_set:n`

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 part, decimal periodic part, exponents and final physical *"units"*, assigning them 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`

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"*.

```
\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
```

(*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 `\MathMLintent` with the *intent property* `:number` are *stored* in `\l__spintent_luaset_only_part_int_str` for the *"integer part"*, in `\l__spintent_luaset_only_part_dec_str` for the *"finite decimal part"*, and in `\l__spintent_luaset_only_par` 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
```

©2026 by Pablo González L

(*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`
`\l__spintent_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` (§15.15) and `\spunit` (§15.16), 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.*)

15.14 Common messages for text mode and math mode

The "error"s `need-math-mode` and `need-text-mode` are raised when a command that requires math mode or text mode is used in the other one.

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

15.15 The command `\spnum`

The user command `\spnum` is *first* of *"the big four"* provided by the spanish-mathml-intent 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 intents passed to `\MathMLintent` to achieve a valid MathML structure that is accepted by MathCAT.

15.15.1 Definition of 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 speak 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.*)

15.15.2 Definition of error messages for \spnum

spnum-has-units
spnum-no-digits

The "error" spnum-has-units is raised when the user includes *"units"* in the {⟨*argument*⟩} passed in '#1', advising them to use the \spunit command instead. The "error" spnum-no-digits is raised when the provided {⟨*argument*⟩} contains no numerical digits. Since other commands, such as \spdiv and \spmul, delegate to \spnum, these messages do not mention the command name.

```
\msg_new:nnnn { spintent } { spnum-has-units }
  {
    Units~are~not~allowed~in~a~numerical~value~('#1')~\msg_line_context:.
  }
  {
    Use~'\token_to_str:N \spunit'~to~print~numbers~with~units.
  }
\msg_new:nnnn { spintent } { spnum-no-digits }
  {
    Invalid~numerical~value~'#1'~\msg_line_context:.
  }
  {
    A~numerical~value~must~contain~at~least~one~digit.
  }
```

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

15.15.3 Definition of keys for \spnum

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

We define 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.*)

15.15.4 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 property* `: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 property* `: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* _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: sets the variable \l__spintent_spnum_intent_read_ with the function _coma:prefix($dp) for reading a *"coma"* or _punto:prefix($dp) for reading a *"punto"* to \MathMLintent, after parsing the value of \l__spintent_luaset_dec_sep_tl. 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 }
          {
            % keep "dp" in mathml-AF
            \MathMLintent { :silent } { \c__spintent_invisible_sep_int }
            \__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 }
```

```
434           {
435             \__spintent_inner_sep:
436             \__spintent_spnum_print_period_bar:
437           }
438           {
439             \MathMLintent { _con:prefix($pnum) }
440               {
441                 \__spintent_inner_sep:
442                 \MathMLarg { pnum }
443                   {
444                     \__spintent_spnum_print_period_bar:
445                   }
446               }
447           }
448       }
449   }
```

(*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, particularly when the number has a periodic part but lacks a standard decimal part.

```
450 \cs_new_protected:Nn \__spintent_spnum_print_part_dec:
451   {
452     \tl_if_empty:NF \l__spintent_luaset_part_dec_tl
453       {
454         \__spintent_spnum_render_decimal_sep:
455       }
456   }
457 \cs_new_protected:Nn \__spintent_spnum_print_part_period:
458   {
459     \tl_if_empty:NF \l__spintent_luaset_part_period_tl
460       {
461         \str_if_eq:VnTF \l__spintent_luaset_not_decimal_period_str { true }
462           { \__spintent_spnum_render_period_sep: }
463           {
464             \tl_if_empty:NT \l__spintent_luaset_part_dec_tl
465               {
466                 \__spintent_spnum_render_period_sep:
467               }
468           }
469       }
470   }
```

(*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_mathml:`

The function `\__spintent_spnum_print_integer:` checks if the number consists only of an integer. If so, it renders it. If a decimal or periodic part exists, it sets up an 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_mathml:` is the main function. 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:`.

```
471 \cs_new_protected:Nn \__spintent_spnum_print_integer:
472   {
473     \str_if_eq:VnTF \l__spintent_luaset_not_decimal_not_period_str { true }
474       { \__spintent_spnum_render_int: }
475       {
476         \MathMLintent { \l__spintent_luaset_only_part_int_str :number:prefix($next) }
477           {
478             \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_mathml:
    {
      \__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_mathml:`.)

`\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, that is, it has no valid digits. If all checks pass, it calls `\__spintent_spnum_mathml:` 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_mathml: }
          }
      \group_end:
    }
    { \msg_error:nnn { spintent } { need-math-mode } { \spnum } }
  }
```

(*End of definition for* `\spnum`. *This function is documented on page 8.*)

15.16 The command `\spunit`

The user command `\spunit` is the *second* of *"the big four"* provided by the spanish-mathml-intent 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*⟩}.

All the *"units"*, including SÍ fractions, products, and sexagesimal angles, are processed in the same way by the `spintent.lua`.

15.16.1 Definition of variables for `\spunit`

`\l__spintent_spunit_luaset_above_n_str`
`\l__spintent_spunit_luaset_above_found_str`
`\l__spintent_spunit_luaset_above_sexa_str`
`\l__spintent_spunit_luaset_below_n_str`
`\l__spintent_spunit_luaset_below_found_str`

These string variables store the summary returned by the Lua module `spintent.lua` for each of the two products of a *"unit expression"*, the one before the slash, `above`, and the one after it, `below`: the number of *"units"* in the product, whether all of them exist in the dictionary and, for `above`, whether it is a single sexagesimal symbol. For each unit number `<i>` of a product `<p>`, Lua also leaves the canonical symbol, the speech text and the exponent. The canonical symbol is stored in `\l__spintent_spunit_luaset_<p>_<i>_canonical_str`, the speech text in `\l__spintent_spunit_luaset_<p>_<i>_read_str` and the exponent, which is empty when there is none, in `\l__spintent_spunit_luaset_<p>_<i>_exp_tl`.

```
\str_new:N \l__spintent_spunit_luaset_above_n_str
\str_new:N \l__spintent_spunit_luaset_above_found_str
\str_new:N \l__spintent_spunit_luaset_above_sexa_str
\str_new:N \l__spintent_spunit_luaset_below_n_str
\str_new:N \l__spintent_spunit_luaset_below_found_str
```

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

`\l__spintent_spunit_luaset_canonical_str`
`\l__spintent_spunit_luaset_read_str`
`\l__spintent_spunit_luaset_status_str`

The first two hold the data of the *"unit"* being printed, a copy of the one stored by Lua for it: the canonical *"unit symbol"* and the correct speech text for `NVDA/MathCAT`. The last one stores the result of defining a new *"unit"* or alias in Lua.

```
\str_new:N \l__spintent_spunit_luaset_canonical_str
\str_new:N \l__spintent_spunit_luaset_read_str
\str_new:N \l__spintent_spunit_luaset_status_str
```

(*End of definition for* `\l__spintent_spunit_luaset_canonical_str`, `\l__spintent_spunit_luaset_read_str`, *and* `\l__spintent_spunit_luaset_status_str`.)

`\l__spintent_spunit_exponent_tl`
`\l__spintent_spunit_read_slash_str`

These variables are used to format the *"unit"*: its exponent, a copy of the one stored by Lua, and the speech text of the slash.

```
\tl_new:N  \l__spintent_spunit_exponent_tl
\str_new:N \l__spintent_spunit_read_slash_str
```

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

15.16.2 Definition of error messages for `\spunit`

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

The "error" `spunit-invalid-unit` is raised when the *"unit expression"* provided in '`#1`' is unknown or malformed. The "error" `spunit-no-units` is raised when no *"units"* are found in the {⟨*argument*⟩}, advising the user to use `\spnum` for pure numbers. The "error" `spunit-inline-numeric-denominator` is raised when a numeric coefficient is illegally placed in the denominator of an inline fraction.

```
\msg_new:nnnn { spintent } { spunit-invalid-unit }
  {
    Invalid~unit~expression~'#1'~\msg_line_context:.
```

```
  }
  {
    The~expression~contains~an~unknown~or~malformed~unit.~Check~the~spelling~or~verify~
    that~the~unit~is~defined.
  }
\msg_new:nnnn { spintent } { spunit-no-units }
  {
    No~unit~found~in~'#1'~\msg_line_context:.
  }
  {
    At~least~one~unit~is~required;~use~'\token_to_str:N \spnum'~for~pure~
    numbers.
  }
\msg_new:nnnn { spintent } { spunit-inline-numeric-denominator }
  {
    Invalid~numeric~coefficient~in~the~inline~denominator~'#1'~\msg_line_context:.
  }
  {
    Numbers~are~not~allowed~after~the~slash~'/'~in~the~inline~mode~of~units.
  }
```

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

15.16.3 Definition of keys for \spunit

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

We define 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* `print-cdot` *and others.*)

15.16.4 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_load_term:nn`

The function `\__spintent_spunit_load_term:nn` copies the data that Lua stored for the unit number '`#2`' of the product '`#1`', which is `above` or `below`, into the variables used to print it.

```
\cs_new_protected:Nn \__spintent_spunit_load_term:nn
  {
    \str_set_eq:Nc \l__spintent_spunit_luaset_canonical_str
      { l__spintent_spunit_luaset_ #1 _ #2 _canonical_str }
    \str_set_eq:Nc \l__spintent_spunit_luaset_read_str
      { l__spintent_spunit_luaset_ #1 _ #2 _read_str }
    \tl_set_eq:Nc \l__spintent_spunit_exponent_tl
      { l__spintent_spunit_luaset_ #1 _ #2 _exp_tl }
  }
```

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

`\__spintent_spunit_print_term:nn`

The function `\__spintent_spunit_print_term:nn` prints the unit number '`#2`' of the product '`#1`', along with its exponent. Unless this is the first unit of the product, it handles the spacing and multiplication dot: `\cdot` when `print-cdot` is `true`, or a thin space `\,` when it is `false`. In both cases it is read as *"por"* when `read-cdot` is `true`, and it is silent otherwise.

```
\cs_new_protected:Nn \__spintent_spunit_print_term:nn
  {
    \int_compare:nNnT {#2} > { 1 }
      {
        \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:NT \l__spintent_spunit_read_cdot_bool
              { \MathMLintent { _por } { \mathopen { \c__spintent_invisible_sep_int } } }
          }
      }
```

Then it renders the base unit and, if it has one, its exponent.

```
    \__spintent_spunit_load_term:nn {#1} {#2}
    \tl_if_empty:NTF \l__spintent_spunit_exponent_tl
      {
        \MathMLintent { \l__spintent_spunit_luaset_read_str }
          {
            \__spintent_spunit_format_canonical:
          }
      }
      {
        {
          \MathMLintent { \l__spintent_spunit_luaset_read_str }
            {
              \__spintent_spunit_format_canonical:
            }
        }
        ^ { \l__spintent_spunit_exponent_tl }
      }
  }
```

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

`\__spintent_spunit_print_product:n`

The function `\__spintent_spunit_print_product:n` prints all the units of the product '`#1`', `above` or `below`, in the order in which Lua found them.

```
\cs_new_protected:Nn \__spintent_spunit_print_product:n
```

```
  {
    \int_step_inline:nn { \str_use:c { l__spintent_spunit_luaset_ #1 _n_str } }
      { \__spintent_spunit_print_term:nn {#1} {##1} }
  }
```

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

`\__spintent_print_spunit_hierarchy:`

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

```
\cs_new_protected:Nn \__spintent_print_spunit_hierarchy:
  {
    \tl_if_empty:NF \l__spintent_luaset_arg_above_tl
      {
        \__spintent_spunit_print_product:n { above }
        \tl_if_empty:NF \l__spintent_luaset_arg_below_tl
          {
            \MathMLintent { \l__spintent_spunit_read_slash_str } { / }
            \__spintent_spunit_print_product:n { below }
          }
      }
  }
```

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

`\__spintent_spunit_exec:n`

The function `\__spintent_spunit_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 "error"s if any check fails.

Then it asks Lua to split each product of the expression into its units and to look them up; if any of them is unknown, it raises the "error" for an invalid expression.

If the units are valid, it calls `\__spintent_spunit_mathml:`.

```
\cs_new_protected:Nn \__spintent_spunit_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 }
                  { spunit-inline-numeric-denominator } {#1}
              }
              {
                \__spintent_luafun_spunit_parse_product:nV { above }
                  \l__spintent_luaset_arg_above_tl
                \tl_if_empty:NF \l__spintent_luaset_arg_below_tl
                  {
                    \__spintent_luafun_spunit_parse_product:nV { below }
                      \l__spintent_luaset_arg_below_tl
                  }
                \bool_lazy_and:nnTF
                  {
                    \str_if_eq_p:ee { \l__spintent_spunit_luaset_above_found_str }
                      { true }
                  }
                  {
                    \bool_lazy_or_p:nn
                      { \tl_if_empty_p:N \l__spintent_luaset_arg_below_tl }
                      {
                        \str_if_eq_p:ee { \l__spintent_spunit_luaset_below_found_str }
                          { true }
                      }
                  }
                  {
                    \__spintent_spunit_mathml:
                  }
                  { \msg_error:nnn { spintent } { spunit-invalid-unit } {#1} }
```

```
            }
        }
        { \msg_error:nnn { spintent } { spunit-invalid-unit } {#1} }
    }
}
```

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

`\__spintent_spunit_mathml:`

The function `\__spintent_spunit_mathml:` builds the MathML structure of a validated expression. If it is a sexagesimal angle in degrees, minutes, or seconds, it prints the number and the symbol *"without any space in between"*, applying a slight negative space '`\!`' for `arcseconds`. When the number is exactly `1`, with no decimal part, the reading becomes the singular literal `_grado:unit`, `_minuto:unit` or `_segundo:unit`, since neither the fixed plural names nor MathCAT can see the number. For standard units, it prints the number followed by a thin space '`\,`' and then processes the unit sequence.

```
\cs_new_protected:Nn \__spintent_spunit_mathml:
  {
    \str_if_eq:VnTF \l__spintent_spunit_luaset_above_sexa_str { true }
      {
        \__spintent_spunit_load_term:nn { above } { 1 }
        \tl_if_empty:NF \l__spintent_luaset_part_int_tl
          {
            \__spintent_spnum_mathml:
          }
        \bool_lazy_and:nnT
          { \str_if_eq_p:ee { \l__spintent_luaset_part_int_tl } { 1 } }
          { \tl_if_empty_p:N \l__spintent_luaset_part_dec_tl }
          {
            \str_set:Ne \l__spintent_spunit_luaset_read_str
              {
                \str_case:Vn \l__spintent_spunit_luaset_canonical_str
                  {
                    { ° } { _grado:unit   }
                    { ' } { _minuto:unit  }
                    { " } { _segundo:unit }
                  }
              }
          }
        \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_mathml: \,
          }
        \__spintent_print_spunit_hierarchy:
      }
  }
```

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

`\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_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_exec:n {#2}
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spunit } }
  }
```

(*End of definition for* `\spunit`. *This function is documented on page 9.*)

15.17 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_alias:nn`.

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

15.17.1 Definition of error messages for `\spnewunit` and `\spunitalias`

spnewunit-duplicate
spunitalias-duplicate
spunitalias-invalid-expr

The "error"s `spnewunit-duplicate` and `spunitalias-duplicate` are raised when the user attempts to define a *"new unit"* or alias passed in '`#1`' that already exists in the dictionary. The "error" `spunitalias-invalid-expr` is raised by `\spunitalias` when the target expression provided in '`#2`' is invalid, for example, because it contains unknown *"units"* or incorrect syntax.

```
\msg_new:nnnn { spintent } { spnewunit-duplicate }
  {
    The~unit~symbol~or~alias~'#1'~is~already~defined~\msg_line_context:.
  }
  {
    Existing~units~and~aliases~cannot~be~redefined~with~
    '\token_to_str:N \spnewunit'.
  }
\msg_new:nnnn { spintent } { spunitalias-duplicate }
  {
    The~alias~'#1'~is~already~defined~or~reserved~\msg_line_context:.
  }
  {
    Existing~units~and~aliases~cannot~be~redefined.
  }
\msg_new:nnnn { spintent } { spunitalias-invalid-expr }
  {
    Invalid~expression~'#2'~for~alias~'#1'~\msg_line_context:.
  }
  {
    The~expression~must~contain~only~defined~units,~exponents~and~one~'/'.
  }
```

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

15.17.2 Internal functions for `\spnewunit` and `\spunitalias`

\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}
```

```
819             }
820         }
821     }
```

(*End of definition for* `\spnewunit`. *This function is documented on page 10.*)

`\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.

```
822 \NewDocumentCommand \spunitalias { m m }
823   {
824     \mode_if_math:TF
825       {
826         \msg_error:nnn { spintent } { need-text-mode } { \spunitalias }
827       }
828       {
829         \__spintent_luafun_define_spunit_alias:nn {#1} {#2}
830         \str_if_eq:VnT \l__spintent_spunit_luaset_status_str { duplicate }
831           {
832             \msg_error:nnn { spintent } { spunitalias-duplicate } {#1}
833           }
834         \str_if_eq:VnT \l__spintent_spunit_luaset_status_str { invalid-expr }
835           {
836             \msg_error:nnnn { spintent } { spunitalias-invalid-expr } {#1} {#2}
837           }
838       }
839   }
```

(*End of definition for* `\spunitalias`. *This function is documented on page 11.*)

15.18 The command \spmoney

The user command `\spmoney` is the *third* of *"the big four"* provided by the spanish-mathml-intent package, designed to format monetary values and currencies. It sends the {⟨*argument*⟩} to the Lua module `spintent.lua` with `\__spintent_luafun_clean_split_and_set:n`, and then gets the *"currency symbol"*, the names of the currency and its layout from the database.

The command accepts *optional arguments* to change the default *"currency symbol"* or to write the *"subunit"*. Finally, it builds the intent for `MathCAT`.

15.18.1 Definition of variables for \spmoney

`\l__spintent_spmoney_luaset_symbol_str` `\l__spintent_spmoney_luaset_position_str` These variables are set by the Lua module `spintent.lua`. The variable `\l__spintent_spmoney_luaset_symbol_st` stores the currency symbol and `\l__spintent_spmoney_luaset_position_str` stores its position, `pre` or `post`.

```
840 \str_new:N \l__spintent_spmoney_luaset_symbol_str
841 \str_new:N \l__spintent_spmoney_luaset_position_str
```

(*End of definition for* `\l__spintent_spmoney_luaset_symbol_str` *and* `\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` The variables `\l__spintent_spmoney_luaset_sing_str`, `\l__spintent_spmoney_luaset_plur_str` and `\l__spintent_spmoney_luaset_conde_str` store the name of the main unit in singular, in plural and in the form with *«de»*. The variables `\l__spintent_spmoney_luaset_subsing_str` and `\l__spintent_spmoney_luaset_subplur_str` store the name of the subunit in singular and in plural, and are empty if the currency has no subunits.

```
842 \str_new:N \l__spintent_spmoney_luaset_sing_str
843 \str_new:N \l__spintent_spmoney_luaset_plur_str
844 \str_new:N \l__spintent_spmoney_luaset_conde_str
845 \str_new:N \l__spintent_spmoney_luaset_subsing_str
846 \str_new:N \l__spintent_spmoney_luaset_subplur_str
```

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

`\l__spintent_spmoney_luaset_print_iso_str` `_spintent_spmoney_luaset_currency_arg_str` `\l__spintent_spmoney_luaset_status_str` The variable `\l__spintent_spmoney_luaset_print_iso_str` stores `true` or `false` according to whether the `ISO` code is printed. The variable `\l__spintent_spmoney_luaset_currency_arg_str` stores the name of the currency, or the text as given if it was not found, and `\l__spintent_spmoney_luaset_status_str` stores `found` or `notfound`.

```
847 \str_new:N \l__spintent_spmoney_luaset_print_iso_str
848 \str_new:N \l__spintent_spmoney_luaset_currency_arg_str
849 \str_new:N \l__spintent_spmoney_luaset_status_str
```

(*End of definition for* `\l__spintent_spmoney_luaset_print_iso_str`, `\l__spintent_spmoney_luaset_currency_arg_str`, *and* `\l__spintent_spmoney_luaset_status_str`.)

`\l__spintent_spmoney_luaset_extracted_curr_str`
`\l__spintent_spmoney_luaset_extracted_num_tl`

The variables `\l__spintent_spmoney_luaset_extracted_curr_str` and `\l__spintent_spmoney_luaset_ext` store the currency and the number that the module extracts from the {⟨*argument*⟩}.

```
\str_new:N \l__spintent_spmoney_luaset_extracted_curr_str
\tl_new:N  \l__spintent_spmoney_luaset_extracted_num_tl
```

(*End of definition for* `\l__spintent_spmoney_luaset_extracted_curr_str` *and* `\l__spintent_spmoney_luaset_extracted_num_tl`.)

`\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`

The variable `\l__spintent_spmoney_sub_units_bool` stores whether the subunits are written; it is created by the key `sub-units`. The variables `\l__spintent_spmoney_intent_money_str`, `\l__spintent_spmoney_intent_sign_str` and `\l__spintent_spmoney_intent_sub_unit_str` store the intent of the main unit, of the sign and of the subunit. The variable `\l__spintent_spmoney_res_main_voice` stores the text read for the main unit. The variable `\l__spintent_spmoney_subnum_int` stores the number of subunits, to choose between singular and plural.

```
\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
```

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

15.18.2 Definition of error messages for \spmoney

`spmoney-no-subunits`
`spmoney-invalid-currency`

The "warning" `spmoney-no-subunits` is raised when subunit rendering is requested for a currency that does not support fractional divisions, such as JPY or CLP, ignoring the flag. The "error" `spmoney-invalid-currency` is raised when the provided currency identifier is not registered in the Lua database or is not written validly: an alias or symbol with any uppercase letter, or an `ISO` code that is neither all lowercase nor all uppercase.

```
\msg_new:nnnn { spintent } { spmoney-no-subunits }
  {
    The~option~'sub-units=true'~is~ignored~for~the~currency~'#1'~\msg_line_context:.
  }
  {
    Currencies~like~'JPY'~or~'CLP'~have~no~subunits~in~the~Lua~module.
  }
\msg_new:nnnn { spintent } { spmoney-invalid-currency }
  {
    Invalid~currency~'#1'~for~'\token_to_str:N \spmoney'~\msg_line_context:.
  }
  {
    The~symbol,~alias~or~ISO~code~does~not~exist~in~the~Lua~module.
  }
```

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

15.18.3 Definition of keys for \spmoney

`currency`
`sub-units`
`money-math`
`cifras-sep`
`unknown`

We define the ⟨*keys*⟩ `currency`, `sub-units`, `money-math` and `cifras-sep`.

```
\keys_define:nn { spintent / spmoney }
  {
    currency    .code:n = { \__spintent_spmoney_set_currency:n {#1} },
    currency    .value_required:n = true,
    sub-units   .bool_set:N = \l__spintent_spmoney_sub_units_bool,
    sub-units   .initial:n = { false },
    sub-units   .value_required:n = true,
    money-math  .bool_set:N = \l__spintent_spmoney_money_math_bool,
    money-math  .initial:n = { true },
    money-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.*)

©2026 by Pablo González L

15.18.4 Internal 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 checks the currency identifier in '#1' against the database. It calls __spintent_spmoney_normalize_key:n, which uses the Lua module spintent.lua. If the module returns notfound, it raises an "error". Otherwise, it stores the identifier in \l__spintent_spmoney_luaset_currency_arg_str.

```
\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 } { spmoney-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 { \exp_not:n {#1} }
  }
\cs_generate_variant:Nn \__spintent_spmoney_normalize_key:n { V }
```

Since the key currency calls __spintent_spmoney_set_currency:n, its default value is set once the function exists.

```
\keys_set:nn { spintent / spmoney } { currency = pesos }
```

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

__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 } { spmoney-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"*, directly in math mode when \l__spintent_spmoney_money_math_bool is true, or wrapped inside \text when it is false. For dollar signs '$', the symbol is typeset through the control sequence \$ in both cases.

```
\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_money_math_bool
              { \$ } { \text { \$ } }
          }
          {
```

```
            \bool_if:NTF \l__spintent_spmoney_money_math_bool
              { \l__spintent_spmoney_luaset_symbol_str }
              { \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:` `\__spintent_spmoney_render_only_integer_node:`

The function `\__spintent_spmoney_render_number_node:` renders the whole number, with the integer and the decimal part. The function `\__spintent_spmoney_render_only_integer_node:` renders only the integer part.

```
\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_mathml:`

The function `\__spintent_spmoney_mathml:` checks if there is an explicit sign in `\l__spintent_luaset_sign_tl`. If there is one, it sets the prefix intent 'más' or 'menos' and puts the sign and the rest inside a `\MathMLintent`. If there is no sign, it continues according to whether the subunits are enabled.

```
\cs_new_protected:Nn \__spintent_spmoney_mathml:
  {
    \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_mathml:`.)

`\__spintent_spmoney_intent_int:`

The function `\__spintent_spmoney_intent_int:` chooses the form of the currency name: singular, plural, or the form with 'de' for exact millions. It looks at the integer and the decimal part to choose it, and sets it as a postfix 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_str
          }
          {
            \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_set:NV \l__spintent_spmoney_intent_money_str
              \l__spintent_spmoney_luaset_conde_str
          }
      }
      {
        \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_conde_str
              }
              {
                \str_set:NV \l__spintent_spmoney_intent_money_str
                  \l__spintent_spmoney_luaset_plur_str
              }
          }
      }
    \__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 disables the *"subunit"* rendering flag. The flag is also disabled, without a warning, when the amount has no decimal part, since there are no *"subunits"* to voice.

The function `\__spintent_spmoney_set_and_print_sub_units:` sets the intents when the *"subunits"* are enabled. It completes a single-digit decimal like '.5' to '50', chooses the singular or the plural for the main *"currency"* and for the *"subunits"*, and calls `\__spintent_spmoney_build_tree:` to build the MathML tree.

```
\cs_new_protected:Nn \__spintent_spmoney_check_sub_units:
  {
    \str_if_empty:NT \l__spintent_spmoney_luaset_subsing_str
      {
        \msg_warning:nne { spintent } { spmoney-no-subunits }
          { \l__spintent_spmoney_luaset_currency_arg_str }
```

```
      \bool_set_false:N \l__spintent_spmoney_sub_units_bool
    }
  \tl_if_empty:NT \l__spintent_luaset_dec_sep_tl
    { \bool_set_false:N \l__spintent_spmoney_sub_units_bool }
}
\cs_new_protected:Nn \__spintent_spmoney_set_and_print_sub_units:
{
  \int_compare:nNnTF { \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:nNnTF { \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:` builds the MathML tree for currencies with *"subunits"*. It places the integer part, the decimal separator and the *"subunit"* according to the position, `pre` or `post`, of the *"currency symbol"*.

`\__spintent_spmoney_print_sep_con:` also 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.

```
\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 in `\l__spintent_spmoney_luaset_extracted_curr_str`, it sets the corresponding variables. It then calls `\__spintent_spmoney_validate_money:` to verify the *"currency"*. If *"subunit"* rendering is enabled in `\l__spintent_spmoney_sub_units_bool`, it calls `\__spintent_spmoney_check_sub_units:`. Finally, it calls `\__spintent_spmoney_mathml:` to format the output.

```
\cs_new_protected:Nn \__spintent_spmoney_parse_and_route:n
  {
    \__spintent_luafun_spmoney_prepare_input:n { \exp_not:n {#1} }
    \__spintent_luafun_clean_split_and_set:n
      { \exp_not:V \l__spintent_spmoney_luaset_extracted_num_tl }
    \str_if_empty:NF \l__spintent_spmoney_luaset_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_luaset_extracted_curr_str
      }
    \__spintent_spmoney_validate_money:
    \bool_if:NT \l__spintent_spmoney_sub_units_bool
      { \__spintent_spmoney_check_sub_units: }
    \__spintent_spmoney_mathml:
  }
```

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

`\spmoney`

The public command `\spmoney` formats monetary values. It accepts an optional argument '`#1`' with the ⟨*keys*⟩ and a mandatory argument '`#2`' with the number and, optionally, the currency. It raises an "error" if it is used outside *math mode*. Then it opens a local group, sets the keys, calls `\__spintent_spmoney_parse_and_route:n` with the {⟨*argument*⟩}, and closes the group.

```
\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 11.*)

15.19 The command \spshort

The user command \spshort is the *fourth* and *final* of *"the big four"*. It writes abbreviations, ordinals and acronyms in *text mode* and makes NVDA read them in full. Unlike the *math mode* commands, which use MathML attributes, this command uses the PDF *tagging tools* provided by the tagpdf[22] package: the key E gives the full form of the abbreviation, ordinal or acronym.

15.19.1 Definition of variables for \spshort

\l__spintent_spshort_luaset_status_str
\l__spintent_spshort_luaset_layout_str
\l__spintent_spshort_luaset_expand_str
\l__spintent_spshort_luaset_output_str
\l__spintent_spshort_luaset_base_str
\l__spintent_spshort_luaset_suffix_str

These variables store the values returned by the Lua module spintent.lua. The variable \l__spintent_spshort_luaset_status_str stores success, fallback or error, and \l__spintent_spshort_luaset_layout_str stores the way the abbreviation is typeset. The variable \l__spintent_spshort_luaset_expand_str stores the text read, which is also the one used by the key E, and \l__spintent_spshort_luaset_output_str stores the text that is printed. For ordinals, \l__spintent_spshort_luaset_base_str stores the number and \l__spintent_spshort_luaset_suffix_str stores the 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_expand_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.*)

15.19.2 Definition of error messages for \spshort

spshort-unknown-abbreviation

The "error" spshort-unknown-abbreviation is raised when the {⟨*argument*⟩} is not an entry of the database in spintent.lua and it cannot be expanded as an ordinal.

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

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

15.19.3 Definition of keys for \spshort

small-caps
read-arg
unknown

We define 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 , read-arg , *and* unknown.)

15.19.4 Internal functions for \spshort

__spintent_spshort_tag_span:nn
__spintent_spshort_tag_span:en

The internal function __spintent_spshort_tag_span:nn prints the {⟨*argument*⟩} in '#2' with the tools of tagpdf. With the keys tag and E it creates the PDF structure /E (#1), so that NVDA reads the text in '#1'.

```
\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 }
```

©2026 by Pablo González L

```
        \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`.)

`\__spintent_spshort_print:nn`

The internal function `\__spintent_spshort_print:nn` prints the abbreviations. It clears `\l__spintent_spshort_read_arg_str` and sets the ⟨*keys*⟩ in '`#1`'. If that string variable is still empty, it calls `\__spintent_spshort_exec:n` with the {⟨*argument*⟩} in '`#2`'. Otherwise, it calls `\__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_exec:n {#2}
      }
      {
        \__spintent_spshort_render_bypass:n {#2}
      }
  }
```

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

`\__spintent_spshort_exec:n`

The internal function `\__spintent_spshort_exec:n` resolves the abbreviation or ordinal in '`#1`' with the Lua function `\__spintent_luafun_spshort_process:n`. Then it reads `\l__spintent_spshort_luaset_status_str` and calls the rendering function for `success`, `fallback` or `error`. Any other status is handled as `error`.

```
\cs_new_protected:Nn \__spintent_spshort_exec: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_exec: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:` uses the key `E` and the layout returned by Lua. The function `\__spintent_spshort_render_fallback:` prints the terms that are not in the database, with SMALL CAPS if the option is active.

The function `\__spintent_spshort_render_bypass:n` uses the reading given by the user for the {⟨*argument*⟩} and does not look in the database. The function `\__spintent_spshort_render_error:n` raises the "error" and prints the {⟨*argument*⟩} as it is, so that the compilation does not stop.

```
\cs_new_protected:Nn \__spintent_spshort_render_success:
  {
    \__spintent_spshort_tag_span:en
      { \l__spintent_spshort_luaset_expand_str }
      { \__spintent_spshort_execute_layout: }
  }
\cs_new_protected:Nn \__spintent_spshort_render_fallback:
  {
    \__spintent_spshort_tag_span:en
      { \l__spintent_spshort_luaset_expand_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 } { spshort-unknown-abbreviation } {#1}
  }
```

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

`\__spintent_spshort_execute_layout:`

The internal function `\__spintent_spshort_execute_layout:` applies the layout. It reads `\l__spintent_spshort_luaset_layout_str`, with a value such as `linear_regular`, `superscript` or `small_caps_pure`, and applies the corresponding formatting. It also combines superscripts with SMALL CAPS according to the 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 command `\spshort` writes abbreviations, acronyms and ordinals in *text mode*. It raises an "error" in *math mode*. Otherwise, it calls `\mode_leave_vertical:`, opens a local group and calls `\__spintent_spshort_print:nn`.

```
\NewDocumentCommand \spshort { O{} m }
  {
    \mode_if_math:TF
      {
        \msg_error:nnn { spintent } { need-text-mode } { \spshort }
      }
```

©2026 by Pablo González L

```
      {
        \mode_leave_vertical:
        \group_begin:
          \__spintent_spshort_print:nn {#1} {#2}
        \group_end:
      }
  }
```

(*End of definition for* `\spshort`. *This function is documented on page 13.*)

15.20 The command `\spsiglo`

The public command `\spsiglo` writes century designations such as siglo XXI. It accepts Arabic and Roman numerals in the {⟨*argument*⟩} and uses the Lua module `spintent.lua` to validate and convert them.

This command works in *text mode* and in *math mode*. In *math mode* it builds a `MathML` `intent` expression. In *text mode* it uses a PDF tagging structure, such as `/Alt (#1)`, with the key `alt`.

15.20.1 Definition of variables for `\spsiglo`

`\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`

The variable `\l__spintent_spsiglo_luaset_error_str` stores `true` or `false` if the Lua module `spintent.lua` could not read the {⟨*argument*⟩}. The variable `\l__spintent_spsiglo_luaset_arabic_str` stores the number in Arabic numerals, used for the speech. The variable `\l__spintent_spsiglo_luaset_roman_min` stores the number in lowercase Roman numerals, which is printed with `\textsc`. The variable `\l__spintent_spsiglo_print_era_str` stores the value set by the key `print-era`.

```
\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.*)

15.20.2 Definition of error messages for `\spsiglo`

`spsiglo-invalid-arg`

The "error" `spsiglo-invalid-arg` is raised 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 } { spsiglo-invalid-arg }
  {
    Invalid~argument~'#2'~for~'\tl_trim_spaces:n {#1}'~\msg_line_context:.
  }
  {
    The~argument~must~be~an~Arabic~or~Roman~numeral~between~'1'~and~'4000'.
  }
```

(*End of definition for* `spsiglo-invalid-arg`.)

15.20.3 Definition of keys for `\spsiglo`

`print-era` `unknown`

We define the ⟨*key*⟩ `print-era`.

```
\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` *and* `unknown`.)

15.20.4 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 build the MathML intent structures in *math mode*, with the Arabic numeral as the speech. __spintent_spsiglo_render_with_era:nnn also adds the era ac or dc, with invisible spacing and postfix intents so that NVDA/MathCAT reads it correctly.

```
\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 writes the century in *text mode*. It uses tagpdf to create the PDF structure /Alt (#1) with the key alt, so that NVDA/MathCAT reads the value correctly without 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_exec:n
__spintent_spsiglo_mathml:
__spintent_spsiglo_text:

The function __spintent_spsiglo_exec:n formats century designations based on the active mode. It first checks the "error" status in the variable \l__spintent_spsiglo_luaset_error_str returned by the Lua module spintent.lua and issues an "error" if validation fails.

If the {⟨*argument*⟩} is valid, in *math mode* it calls __spintent_spsiglo_mathml:, which applies the MathML structures depending on whether an era modifier is present. In *text mode* it leaves vertical mode

first and calls __spintent_spsiglo_text:, which uses the keys tag and alt to expand the value of the print-era option into full text such as “antes de cristo” for NVDA/MathCAT.

```
\cs_new_protected:Nn \__spintent_spsiglo_exec:n
  {
    \mode_if_math:F { \mode_leave_vertical: }
    \str_if_eq:VnTF \l__spintent_spsiglo_luaset_error_str { true }
      {
        \msg_error:nnnn { spintent } { spsiglo-invalid-arg }
          { \spsiglo } {#1}
      }
      {
        \mode_if_math:TF
          { \__spintent_spsiglo_mathml: }
          { \__spintent_spsiglo_text: }
      }
  }
\cs_new_protected:Nn \__spintent_spsiglo_mathml:
  {
    \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_text:
  {
    \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_exec:n, __spintent_spsiglo_mathml:, *and* __spintent_spsiglo_text:.)

\spsiglo The public command \spsiglo opens a local group and sets the ⟨*keys*⟩ in ‘#1’. Then it calls __spintent_luafun_spsiglo_parse:n of the Lua module spintent.lua with the {⟨*argument*⟩} in ‘#2’, and calls __spintent_spsiglo_exec:n for *math mode* or *text mode*.

```
\NewDocumentCommand \spsiglo { O{} m }
  {
    \group_begin:
      \keys_set:nn { spintent / spsiglo } {#1}
      \__spintent_luafun_spsiglo_parse:n {#2}
      \__spintent_spsiglo_exec:n {#2}
    \group_end:
  }
```

(*End of definition for* \spsiglo. *This function is documented on page* 15.)

©2026 by Pablo González L

15.21 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.

15.21.1 Definition of variables for \sptime

\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 variables store the values returned by the Lua module spintent.lua. The variable \l__spintent_sptime_luaset_error_str stores true or false if the {⟨*argument*⟩} is invalid. The variable \l__spintent_sptime_luaset_base_str stores the hours and minutes, and \l__spintent_sptime_luaset_seconds_str stores the seconds, which is empty if there are none. The variables \l__spintent_sptime_luaset_has_seconds_str and \l__spintent_sptime_luaset_is_fraction_str store true or false if the time has seconds and if the seconds have a decimal fraction.

```
\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.*)

15.21.2 Definition of error messages for \sptime

sptime-invalid-arg

The "error" sptime-invalid-arg is raised 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 } { sptime-invalid-arg }
  {
    Invalid~argument~'#2'~for~'\tl_trim_spaces:n {#1}'~\msg_line_context:.
  }
  {
    Allowed~formats:~'HH:MM',~'HH:MM:SS'~or~'HH:MM:SS.ss'.~
    Hours~must~be~'0-23'~and~minutes/seconds~'0-59'.
  }
```

(*End of definition for* sptime-invalid-arg.)

15.21.3 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 a speech intent, 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_mathml:

The internal function __spintent_sptime_mathml: 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:nn with the appropriate prefix, 'con' or 'minutos-con', or appends the 'minutos' intent if no seconds are present.

```
\cs_new_protected:Nn \__spintent_sptime_mathml:
  {
    \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_mathml:.)

\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_st If an invalid format is detected, it issues the sptime-invalid-arg error; otherwise, it calls __spintent_sptime_mathml: 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 } { sptime-invalid-arg }
                { \sptime } {#1}
            }
            {
              \__spintent_sptime_mathml:
            }
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \sptime } }
  }
```

(*End of definition for* \sptime. *This function is documented on page* 16.)

15.22 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.

15.22.1 Definition of variables for \spdate

\l__spintent_spdate_luaset_output_str
\l__spintent_spdate_luaset_error_str

These variables store the values returned by the Lua module spintent.lua. The variable \l__spintent_spdate_luaset_output_str stores the date already formatted, and \l__spintent_spdate_luase stores true or false if the {⟨argument⟩} is invalid.

```
\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.)

15.22.2 Definition of error messages for \spdate

spdate-invalid-arg

The "error" spdate-invalid-arg is raised 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 } { spdate-invalid-arg }
  {
    Invalid~argument~'#2'~for~'\tl_trim_spaces:n {#1}'~\msg_line_context:.
  }
  {
    Use~'DD/MM/YYYY',~'DD-MM-YYYY'~or~'YYYY-MM-DD',~with~a~valid~day~
    and~month.
  }
```

(*End of definition for* spdate-invalid-arg.)

15.22.3 Internal functions for \spdate

__spintent_spdate_exec:n

The internal function __spintent_spdate_exec: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 spdate-invalid-arg error. Otherwise, it calls __spintent_spdate_mathml:.

```
\cs_new_protected:Nn \__spintent_spdate_exec:n
  {
    \__spintent_luafun_spdate_parse:n {#1}
    \str_if_eq:VnTF \l__spintent_spdate_luaset_error_str { true }
      {
        \msg_error:nnnn { spintent } { spdate-invalid-arg }
          { \spdate } {#1}
      }
      { \__spintent_spdate_mathml: }
  }
```

(*End of definition for* __spintent_spdate_exec:n.)

__spintent_spdate_mathml:

The internal function __spintent_spdate_mathml: builds the MathML intent structure using the formatted date stored in \l__spintent_spdate_luaset_output_str.

```
\cs_new_protected:Nn \__spintent_spdate_mathml:
  {
    \MathMLintent { :date }
      {
        \text { \l__spintent_spdate_luaset_output_str }
      }
  }
```

(*End of definition for* __spintent_spdate_mathml:.)

\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_exec:n to process the input.

```
\NewDocumentCommand \spdate { m }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \__spintent_spdate_exec:n {#1}
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spdate } }
  }
```

(*End of definition for* \spdate. *This function is documented on page* 16.)

©2026 by Pablo González L

15.23 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.

15.23.1 Definition of variables for \spusep

\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.)

15.23.2 Definition of error messages for \spusep

spusep-empty-arg
spusep-invalid-arg

The "error" spusep-empty-arg is raised when \spusep receives an empty or blank {⟨*argument*⟩}. The "error" spusep-invalid-arg is raised when the input is invalid.

```
\msg_new:nnnn { spintent } { spusep-empty-arg }
  {
    Empty~argument~for~'\token_to_str:N \spusep'~\msg_line_context:.
  }
  {
    A~non-empty~argument~is~required.
  }
\msg_new:nnnn { spintent } { spusep-invalid-arg }
  {
    Invalid~argument~'#1'~for~'\token_to_str:N \spusep'~\msg_line_context:.
  }
  {
    Only~the~delimiters~listed~in~the~documentation~of~'\token_to_str:N \spusep'~are~
      accepted.
  }
```

(*End of definition for* spusep-empty-arg *and* spusep-invalid-arg.)

15.23.3 Definition of keys for \spusep

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`.)

15.23.4 Internal functions for \spusep

`\__spintent_spusep_mathml:nn`

The function `\__spintent_spusep_mathml: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_mathml: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_mathml:nn`.)

`\__spintent_spusep_exec:n`

The function `\__spintent_spusep_exec: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_mathml:nn` for printing. If the input is not supported or is empty, we return an "error".

```
\cs_new_protected:Nn \__spintent_spusep_exec: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:nn { spintent } { spusep-empty-arg }
      }
    \str_case:VnF \l__spintent_spusep_arg_tl
      {
        { (  }
          {
            \__spintent_spusep_mathml:nn { left } { \lparen }
          }
        { )  }
          {
            \__spintent_spusep_mathml:nn { right } { \rparen }
          }
        { [  }
          {
            \__spintent_spusep_mathml:nn { left } { \lbrack }
          }
        { ]  }
          {
            \__spintent_spusep_mathml:nn { right } { \rbrack }
          }
        { \{ }
```

```
            {
              \__spintent_spusep_mathml:nn { left } { \lbrace }
            }
          { \} }
            {
              \__spintent_spusep_mathml:nn { right } { \rbrace }
            }
        }
        {
          \tl_if_empty:NF \l__spintent_spusep_arg_tl
            { \msg_error:nnn { spintent } { spusep-invalid-arg } {#1} }
        }
    }
```

(*End of definition for* `\__spintent_spusep_exec: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_exec:n`.

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

(*End of definition for* `\spusep`. *This function is documented on page 17.*)

15.24 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.

15.24.1 Definition of variables for `\spread`

`\l__spintent_spread_luaset_status_str` `\l__spintent_spread_luaset_reading_str` `\l__spintent_spread_read_arg_tl` `\l__spintent_spread_read_tmp_tl` The variables `\l__spintent_spread_luaset_status_str` and `\l__spintent_spread_luaset_reading_str` are set by `\__spintent_luafun_spread_new:nn`/`\__spintent_luafun_spread_lookup:n`. The variable `\l__spintent_spread_read_arg_tl` stores `read-arg`.

```
\str_new:N \l__spintent_spread_luaset_status_str
\str_new:N \l__spintent_spread_luaset_reading_str
\tl_new:N  \l__spintent_spread_read_arg_tl
\tl_new:N  \l__spintent_spread_read_tmp_tl
```

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

15.24.2 Definition of error messages for `\spread`

`spnewread-empty-args` `spnewread-duplicate` `spread-empty-sym` `spread-unknown-sym` The "error" `spnewread-empty-args` is raised when `\spnewread` receives a blank symbol or a reading that is empty once normalized. The "error" `spnewread-duplicate` is raised when it registers a symbol already in the dictionary. The "error" `spread-empty-sym` is raised when `\spread` receives a blank symbol. The "error" `spread-unknown-sym` is raised when `\spread` is given a symbol that is neither registered nor accompanied by `read-arg`.

```
\msg_new:nnnn { spintent } { spnewread-empty-args }
  {
    Empty~argument~for~'\token_to_str:N \spnewread'~\msg_line_context:.
  }
  {
    Two~non-empty~arguments~are~required.
  }
\msg_new:nnnn { spintent } { spnewread-duplicate }
  {
    The~symbol~'#1'~is~already~defined~\msg_line_context:.
  }
  {
    Existing~symbols~cannot~be~redefined~with~
    '\token_to_str:N \spnewread'.
  }
```

```
\msg_new:nnnn { spintent } { spread-empty-sym }
  {
    Empty~argument~for~'\token_to_str:N \spread'~\msg_line_context:.
  }
  {
    A~non-empty~symbol~is~required.
  }
\msg_new:nnnn { spintent } { spread-unknown-sym }
  {
    The~symbol~'#1'~is~not~registered~and~no~'read-arg'~was~given~\msg_line_context:.
  }
  {
    Register~the~symbol~first~with~'\token_to_str:N \spnewread'~or~pass~'read-arg'.
  }
```

(*End of definition for* `spnewread-empty-args` *and others.*)

15.24.3 Definition of keys for `\spread`

`read-arg` `silent` `unknown`

We define the ⟨*keys*⟩ `read-arg` and `silent`.

```
\keys_define:nn { spintent / spread }
  {
    read-arg .code:n =
      {
        \__spintent_sanitize_affix:Nn \l__spintent_spread_read_arg_tl {#1}
      },
    read-arg .initial:n = ,
    read-arg .value_required:n = true,
    silent   .bool_set:N = \l__spintent_spread_silent_bool,
    silent   .initial:n = false,
    silent   .value_required:n = true,
    unknown  .code:n =
      {
        \str_set:Nn \l__spintent_command_name_str { spread }
        \__spintent_unknown_keys_cmd:n {#1}
      },
  }
```

(*End of definition for* `read-arg` , `silent` , *and* `unknown`.)

15.24.4 Internal functions for `\spread`

`\__spintent_spread_exec:nn`

The function `\__spintent_spread_exec:nn` receives the optional ⟨*keys*⟩ in '`#1`' and the {⟨*symbol*⟩} in '`#2`'. With `silent=true`, it passes the literal reading '`-`' to `\__spintent_spread_mathml:nn`. Otherwise, with `read-arg` given, that reading is used for this call, overriding any registered reading. With `read-arg` empty, it calls `\__spintent_luafun_spread_lookup:n`, using the registered reading when found and reporting `spread-unknown-sym` otherwise.

```
\cs_new_protected:Nn \__spintent_spread_exec:nn
  {
    \keys_set:nn { spintent / spread } {#1}
    \tl_if_blank:nTF {#2}
      { \msg_error:nn { spintent } { spread-empty-sym } }
      {
        \bool_if:NTF \l__spintent_spread_silent_bool
          { \__spintent_spread_mathml:nn {#2} { - } }
          {
            \tl_if_empty:NTF \l__spintent_spread_read_arg_tl
              {
                \__spintent_luafun_spread_lookup:n { \exp_not:n {#2} }
                \str_if_eq:VnTF \l__spintent_spread_luaset_status_str { found }
                  {
                    \__spintent_spread_mathml:nV {#2}
                      \l__spintent_spread_luaset_reading_str
                  }
                  {
                    \msg_error:nne { spintent } { spread-unknown-sym }
                      { \exp_not:n {#2} }
                  }
              }
              { \__spintent_spread_mathml:nV {#2} \l__spintent_spread_read_arg_tl }
          }
```

```
      }
  }
```

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

`\__spintent_spread_mathml:nn` `\__spintent_spread_mathml:nV`

The function `\__spintent_spread_mathml:nn` receives the {⟨*symbol*⟩} in '`#1`' and the reading in '`#2`'. It typesets the symbol, preceded by `\__spintent_inner_sep:`, inside a `\MathMLintent` whose literal is the reading preceded by '_'.

```
\cs_new_protected:Nn \__spintent_spread_mathml:nn
  {
    \__spintent_invisible_sep:
    \MathMLintent { _ #2 } { \__spintent_inner_sep: #1 }
  }
\cs_generate_variant:Nn \__spintent_spread_mathml:nn { nV }
```

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

`\spread`

The public command `\spread` enforces *math mode*, opens a local group, and delegates execution to `\__spintent_spread_exec:nn`.

```
\NewDocumentCommand \spread { O{} 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 16.*)

`\spnewread`

The public command `\spnewread` registers a new symbol/reading pair. It first normalizes the reading '`#2`' with `\__spintent_sanitize_affix:Nn`, storing the result in the scratch variable `\l__spintent_spread_read_tmp_tl`, and reports `spnewread-empty-args` when the symbol '`#1`' is blank or the normalized reading is empty. Otherwise it passes the symbol and the normalized reading to Lua via `\__spintent_luafun_spread_new:nn`, reporting `spnewread-duplicate` when the symbol is already registered.

```
\NewDocumentCommand \spnewread { m m }
  {
    \mode_if_math:TF
      {
        \msg_error:nnn { spintent } { need-text-mode } { \spnewread }
      }
      {
        \__spintent_sanitize_affix:Nn \l__spintent_spread_read_tmp_tl {#2}
        \bool_lazy_or:nnTF
          { \tl_if_blank_p:n {#1} } { \tl_if_empty_p:N \l__spintent_spread_read_tmp_tl }
          { \msg_error:nn { spintent } { spnewread-empty-args } }
          {
            \__spintent_luafun_spread_new:nn { \exp_not:n {#1} }
              { \exp_not:V \l__spintent_spread_read_tmp_tl }
            \str_if_eq:VnT \l__spintent_spread_luaset_status_str { duplicate }
              {
                \msg_error:nnn { spintent } { spnewread-duplicate } {#1}
              }
          }
      }
  }
```

(*End of definition for* `\spnewread`. *This function is documented on page 16.*)

15.25 The command `\spdsym`

The command `\spdsym` prints the division symbol with a configurable reading for `MathCAT`. The ⟨*keys*⟩ choose the printed symbol, the reading, and the optional prefix and suffix of the *intent*.

15.25.1 Definition of variables for \spdsym

\l__spintent_spdsym_symbol_tl
\l__spintent_spdsym_read_sym_tl
\l__spintent_spdsym_prefix_tl
\l__spintent_spdsym_suffix_tl

The variable \l__spintent_spdsym_symbol_tl stores the symbol that is printed, chosen by the key set-sym, and \l__spintent_spdsym_read_sym_tl stores the word used in the *intent*, chosen by the key read-sym. The variables \l__spintent_spdsym_prefix_tl and \l__spintent_spdsym_suffix_tl store the values of the keys prefix and suffix.

```
\tl_new:N \l__spintent_spdsym_symbol_tl
\tl_new:N \l__spintent_spdsym_read_sym_tl
\tl_new:N \l__spintent_spdsym_prefix_tl
\tl_new:N \l__spintent_spdsym_suffix_tl
```

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

15.25.2 Definition of keys for \spdsym

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.*)

15.25.3 Internal functions for \spdsym

__spintent_spdsym_mathml:

The internal function __spintent_spdsym_mathml: outputs the configured intent structure, attaching prefixes and suffixes if present.

```
\cs_new_protected:Nn \__spintent_spdsym_mathml:
```

```
{
  \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 }
}
```

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

`\spdsym` Defines the `\spdsym` operator. It takes a single optional argument for local setup and delegates the output to `\__spintent_spdsym_mathml:`.

```
\NewDocumentCommand \spdsym { O{} }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \keys_set:nn { spintent / spdsym } {#1}
          \__spintent_spdsym_mathml:
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spdsym } }
  }
```

(*End of definition for* `\spdsym`. *This function is documented on page* 17.)

15.26 The command \spmsym

The command `\spmsym` prints the multiplication symbol with a configurable reading for `MathCAT`. The ⟨*keys*⟩ choose the printed symbol, the reading, whether the symbol is printed, and the optional prefix and suffix of the *intent*.

15.26.1 Definition of variables for \spmsym

`\l__spintent_spmsym_symbol_tl` `\l__spintent_spmsym_read_sym_tl` `\l__spintent_spmsym_prefix_tl` `\l__spintent_spmsym_suffix_tl` The variable `\l__spintent_spmsym_symbol_tl` stores the symbol that is printed, chosen by the key `set-sym`, and `\l__spintent_spmsym_read_sym_tl` stores the word used in the *intent*, chosen by the key `read-sym`. The variables `\l__spintent_spmsym_prefix_tl` and `\l__spintent_spmsym_suffix_tl` store the values of the keys `prefix` and `suffix`.

```
\tl_new:N \l__spintent_spmsym_symbol_tl
\tl_new:N \l__spintent_spmsym_read_sym_tl
\tl_new:N \l__spintent_spmsym_prefix_tl
\tl_new:N \l__spintent_spmsym_suffix_tl
```

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

15.26.2 Definition of keys for \spmsym

`prefix` `suffix` `set-sym` `read-sym` `print-sym` `unknown` We define the ⟨*keys*⟩ `prefix`, `suffix`, `set-sym`, `read-sym` and `print-sym` 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   } }
```

©2026 by Pablo González L

```
        }
    },
  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} },
  print-sym .bool_set:N = \l__spintent_spmsym_print_sym_bool,
  print-sym .initial:n = true,
  print-sym .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.*)

15.26.3 Internal functions for `\spmsym`

`\__spintent_spmsym_mathml:`

The internal function `\__spintent_spmsym_mathml:` outputs the correct mathematical glyph based on the local ⟨*keys*⟩, attaching prefixes and suffixes if present. With `print-sym` false, it prints an invisible separator instead of the glyph.

```
\cs_new_protected:Nn \__spintent_spmsym_mathml:
  {
    \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_print_sym_bool
          { \l__spintent_spmsym_symbol_tl }
          {
            \mathopen { \c__spintent_invisible_sep_int }
          }
      }
  }
```

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

`\spmsym`

Defines the `\spmsym` operator. Similar to `\spdsym`, it takes a single optional argument for local setup and delegates the output to `\__spintent_spmsym_mathml:`.

```
\NewDocumentCommand \spmsym { O{} }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \keys_set:nn { spintent / spmsym } {#1}
          \__spintent_spmsym_mathml:
        \group_end:
```

©2026 by Pablo González L

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

(*End of definition for* `\spmsym`. *This function is documented on page 18.*)

15.27 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 checks that neither the {⟨*dividend*⟩} nor the {⟨*divisor*⟩} is *blank*.

15.27.1 Definition of error messages for `\spdiv`

spdiv-empty-arg

The "error" `spdiv-empty-arg` is raised when either the {⟨*dividend*⟩} or the {⟨*divisor*⟩} is missing or *blank*.

```
\msg_new:nnnn { spintent } { spdiv-empty-arg }
  {
    Empty~argument~for~'\token_to_str:N \spdiv'~\msg_line_context:.
  }
  {
    A~dividend~and~a~divisor~are~required.
  }
```

(*End of definition for* `spdiv-empty-arg`.)

15.27.2 Definition of keys for `\spdiv`

cifras-sep
read-sign
read-period-sep
notation-sym
set-sym
read-sym
wrap-parens
read-parens
unknown

We define the ⟨*keys*⟩ `cifras-sep`, `read-sign`, `read-period-sep`, `notation-sym`, `set-sym`, `read-sym`, `wrap-parens` and `read-parens`.

```
\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.*)

15.27.3 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_mathml:nn

The function `\__spintent_spdiv_mathml:nn` builds the `MathML` structure of the division. If `wrap-parens` is true, it adds the structural parentheses around `\__spintent_spdiv_render_op:nn`, read or silenced according to `read-parens`.

```
\cs_new_protected:Nn \__spintent_spdiv_mathml:nn
```

```
  {
    \bool_if:NTF \l__spintent_spdiv_wrap_parens_bool
      {
        \bool_if:NTF \l__spintent_spdiv_read_parens_bool
          {
            \lparen \__spintent_spdiv_render_op:nn {#1} {#2} \rparen
          }
          {
            \MathMLintent { :silent } { \lparen }
            \__spintent_spdiv_render_op:nn {#1} {#2}
            \MathMLintent { :silent } { \rparen }
          }
      }
      { \__spintent_spdiv_render_op:nn {#1} {#2} }
  }
```

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

`\__spintent_spdiv_exec:nnn` The function `\__spintent_spdiv_exec:nnn` sets ⟨*keys*⟩, validates {⟨*arguments*⟩}, and calls `\__spintent_spdiv_mathml:nn`.

```
\cs_new_protected:Nn \__spintent_spdiv_exec:nnn
  {
    \keys_set:nn { spintent / spdiv } {#1}
    \tl_if_blank:nTF {#2}
      { \msg_error:nn { spintent } { spdiv-empty-arg } }
      {
        \tl_if_blank:nTF {#3}
          { \msg_error:nn { spintent } { spdiv-empty-arg } }
          {
            \__spintent_spdiv_mathml:nn {#2} {#3}
          }
      }
  }
```

(*End of definition for* `\__spintent_spdiv_exec:nnn`.)

`\spdiv` Defines the user command `\spdiv`. It requires *math mode*.

```
\NewDocumentCommand \spdiv { O{} m m }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \__spintent_spdiv_exec: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 18.*)

15.28 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 checks that neither the {⟨*first factor*⟩} nor the {⟨*second factor*⟩} is *blank*.

15.28.1 Definition of error messages for `\spmul`

`spmul-empty-arg` The "error" `spmul-empty-arg` is raised when either the {⟨*first factor*⟩} or the {⟨*second factor*⟩} is missing or *blank*.

```
\msg_new:nnnn { spintent } { spmul-empty-arg }
  {
    Empty~argument~for~'\token_to_str:N \spmul'~\msg_line_context:.
  }
  {
    A~first~factor~and~a~second~factor~are~required.
  }
```

(*End of definition for* `spmul-empty-arg`.)

15.28.2 Definition of keys for \spmul

cifras-sep
read-sign
read-period-sep
notation-sym
set-sym
read-sym
wrap-parens
read-parens
unknown

We define the ⟨*keys*⟩ cifras-sep, read-sign, read-period-sep, notation-sym, set-sym, read-sym, wrap-parens and read-parens.

```
\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.*)

15.28.3 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_mathml:nn

The function __spintent_spmul_mathml:nn builds the MathML structure of the multiplication. If wrap-parens is true, it adds the structural parentheses around __spintent_spmul_render_op:nn, read or silenced according to read-parens.

```
\cs_new_protected:Nn \__spintent_spmul_mathml:nn
  {
    \bool_if:NTF \l__spintent_spmul_wrap_parens_bool
      {
        \bool_if:NTF \l__spintent_spmul_read_parens_bool
          {
            \lparen \__spintent_spmul_render_op:nn {#1} {#2} \rparen
          }
          {
            \MathMLintent { :silent } { \lparen }
            \__spintent_spmul_render_op:nn {#1} {#2}
            \MathMLintent { :silent } { \rparen }
          }
      }
      { \__spintent_spmul_render_op:nn {#1} {#2} }
  }
```

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

__spintent_spmul_exec:nnn

The function __spintent_spmul_exec:nnn sets ⟨*keys*⟩, validates {⟨*arguments*⟩}, and calls __spintent_spmul_mathml:nn.

```
\cs_new_protected:Nn \__spintent_spmul_exec:nnn
  {
    \keys_set:nn { spintent / spmul } {#1}
```

©2026 by Pablo González L

```
    \tl_if_blank:nTF {#2}
      { \msg_error:nn { spintent } { spmul-empty-arg } }
      {
        \tl_if_blank:nTF {#3}
          { \msg_error:nn { spintent } { spmul-empty-arg } }
          {
            \__spintent_spmul_mathml:nn {#2} {#3}
          }
      }
  }
```

(*End of definition for* `\__spintent_spmul_exec:nnn`.)

`\spmul` Defines the user command `\spmul`. It requires *math mode*.

```
\NewDocumentCommand \spmul { O{} m m }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \__spintent_spmul_exec: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 19.*)

15.29 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 send the validation of the arguments and the calculation to the Lua module `spintent.lua`, constructing a valid intent.

15.29.1 Definition of variables for `\spmcd` and `\spmcm`

`\l__spintent_mc_luaset_types_clist` `\l__spintent_mc_luaset_all_numeric_str` The variable `\l__spintent_mc_luaset_types_clist` stores the type of each argument, `number` or `other`, in the same order and separated by commas. The variable `\l__spintent_mc_luaset_all_numeric_str` stores `true` when all the arguments are natural numbers. Both are set by the Lua module `spintent.lua`.

```
\clist_new:N \l__spintent_mc_luaset_types_clist
\str_new:N   \l__spintent_mc_luaset_all_numeric_str
```

(*End of definition for* `\l__spintent_mc_luaset_types_clist` *and* `\l__spintent_mc_luaset_all_numeric_str`.)

`\l__spintent_mc_raw_args_seq` `\l__spintent_mc_types_seq` `\l__spintent_mc_args_seq` `\l__spintent_mc_idx_int` The variable `\l__spintent_mc_raw_args_seq` stores the arguments as they were given and `\l__spintent_mc_types_seq` stores their types. The variable `\l__spintent_mc_args_seq` stores the arguments ready to be typeset, with `\spnum` applied to those of type `number`. The variable `\l__spintent_mc_idx_int` stores the position of the argument being processed.

```
\seq_new:N \l__spintent_mc_raw_args_seq
\seq_new:N \l__spintent_mc_types_seq
\seq_new:N \l__spintent_mc_args_seq
\int_new:N \l__spintent_mc_idx_int
```

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

`\l__spintent_spmcd_luaset_mcd_value_tl` `\l__spintent_spmcm_luaset_mcm_value_tl` `\l__spintent_spmcd_read_terse_bool` `\l__spintent_spmcm_read_terse_bool` `\l__spintent_spmcd_prefix_tl` `\l__spintent_spmcm_prefix_tl` `\l__spintent_spmcd_print_result_bool` `\l__spintent_spmcm_print_result_bool` `\l__spintent_spmcd_print_upper_bool` `\l__spintent_spmcm_print_upper_bool` Each command has its own copy of these variables. The variables `\l__spintent_spmcd_luaset_mcd_value_tl` and `\l__spintent_spmcm_luaset_mcm_value_tl` store the result of the greatest common divisor and of the least common multiple, set by the Lua module `spintent.lua`. The variables `_read_terse_bool` and `_prefix_tl` store the values of the keys `read-sty` and `prefix`. The variables `_print_result_bool` and `_print_upper_bool` store the values of the keys `result` and `upper-case`, and are created by those keys.

```
\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_spmcd_luaset_mcd_value_tl` *and others.*)

15.29.2 Definition of keys for \spmcd and \spmcm

result
prefix
read-sty
upper-case
concept
silent
unknown

We define the shared ⟨*keys*⟩ result, prefix, read-sty, upper-case, concept and silent.

```
\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-sty   .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-sty   .initial:n = clear,
        read-sty   .value_required:n = true,
        read-sty / unknown .code:n =
          \msg_error:nneee { spintent } { unknown-choice }
            { read-sty } { 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     .bool_set:c = { l__spintent_sp #1 _silent_bool },
        silent     .initial:n = false,
        silent     .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.*)

15.29.3 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. Then it calls __spintent_mc_mathml_args:n.

```
\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 } }
          { number }
          {
            \seq_put_right:Nn \l__spintent_mc_args_seq { \spnum {##1} }
          }
```

```
            { \seq_put_right:Nn \l__spintent_mc_args_seq {##1} }
        }
      \__spintent_mc_mathml_args:n {#1}
    }
```

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

`\__spintent_mc_mathml_args:n` The function `\__spintent_mc_mathml_args:n` receives the short identifier in '`#1`' and puts the intents around the already-formatted `\l__spintent_mc_args_seq`.

```
\cs_new_protected:Nn \__spintent_mc_mathml_args:n
  {
    \bool_if:cTF { l__spintent_sp #1 _silent_bool }
      {
        \seq_set_map:NNn \l__spintent_mc_args_seq \l__spintent_mc_args_seq
          {
            \MathMLintent { _- } { \__spintent_inner_sep: ##1 }
          }
        \MathMLintent { :silent } { \c__spintent_invisible_sep_int } % prevent "por"
        \MathMLintent { :silent } { \lparen }
        % prevent "intervalo"
        \MathMLintent { :silent } { \c__spintent_invisible_sep_int }
        \seq_use:Nnnn \l__spintent_mc_args_seq
          { \MathMLintent { :silent } { , } }
          { \MathMLintent { :silent } { , } }
          { \MathMLintent { :silent } { , } }
      }
      {
        \MathMLintent { :silent } { \c__spintent_invisible_sep_int } % prevent doble "de"
        \bool_if:cTF { l__spintent_sp #1 _concept_bool }
          {
            \MathMLintent { _entre } { \lparen }
          }
          {
            \MathMLintent { :silent } { \lparen }
          }
        % prevent "intervalo"
        \MathMLintent { :silent } { \c__spintent_invisible_sep_int }
        \seq_use:Nnnn \l__spintent_mc_args_seq
          {
            \MathMLintent { :silent } { , }
            \MathMLintent { _y- } { \c__spintent_invisible_sep_int } % "y" medio
          }
          {
            \MathMLintent { :silent } { , }
          }
          {
            \MathMLintent { :silent } { , }
            \MathMLintent { _y- } { \c__spintent_invisible_sep_int } % "y" final
          }
      }
    \MathMLintent { :silent } { \rparen }
  }
```

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

`\__spintent_spmc_exec:nnnn` The internal function `\__spintent_spmc_exec:nnnn` is the shared function 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`'.

After setting the ⟨*keys*⟩, it calls `\__spintent_spmc_mathml:nn` to print the operator name.

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` and, when `result` is `true`, print the computed value with `\__spintent_spmc_result_mathml:n`. `\__spintent_mc_process_args:nn` prints the argument list itself either way. With `silent=true`, it wraps each argument in a literal '`_-`' intent and the printed result is silenced as well.

We should use `\mathrm` instead of `\operatorname`, which generates unwanted spacing when `\c__spintent_invisible_sep_int` is applied.

```
\cs_new_protected:Nn \__spintent_spmc_exec:nnnn
  {
```

```
    \keys_set:nn { spintent / sp #1 } {#3}
    \__spintent_spmc_mathml:nn {#1} {#2}
```

With '#4' given, it is classified with __spintent_luafun_mc_classify_args:n. If every argument is a number, the Lua calculation calculate_#1 is called, and the computed value is printed when result is true. If any argument is not a number, no calculation is attempted. In both cases __spintent_mc_process_args:n prints the argument list.

```
    \tl_if_novalue:nF {#4}
      {
        \seq_set_from_clist:Nn \l__spintent_mc_raw_args_seq {#4}
        \int_compare:nNnTF { \seq_count:N \l__spintent_mc_raw_args_seq } < { 2 }
          {
            \msg_error:nne { spintent } { mc-too-few-args } { \exp_not:c { sp #1 } }
          }
          {
            \__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}
              }
            \__spintent_mc_process_args:nn {#1} {#4}
            \__spintent_spmc_result_mathml:n {#1}
          }
      }
  }
```

(*End of definition for* __spintent_spmc_exec:nnnn.)

__spintent_spmc_mathml:nn

The function __spintent_spmc_mathml:nn receives the short identifier in '#1' and its full reading string in '#2', and builds the MathML structure of the operator name. With silent=true or concept=false, the operator name is silenced. With concept=true, the literal intent is #1 or #2 according to read-sty, prefixed with prefix when given.

```
\cs_new_protected:Nn \__spintent_spmc_mathml:nn
  {
    \bool_if:cTF { l__spintent_sp #1 _silent_bool }
      {
        \MathMLintent { :silent }
          {
            \mathrm
              {
                \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} {#2}
              }
              {
                \mathrm
                  {
                    \bool_if:cTF { l__spintent_sp #1 _print_upper_bool }
                      { \text_uppercase:n {#1} } {#1}
                  }
              }
          }
          {
            \MathMLintent { :silent }
              {
                \mathrm
```

```
                    {
                      \bool_if:cTF { l__spintent_sp #1 _print_upper_bool }
                        { \text_uppercase:n {#1} } {#1}
                    }
                }
            }
        }
    }
```

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

`\__spintent_spmc_result_mathml:n` The function `\__spintent_spmc_result_mathml:n` receives the short identifier in '`#1`' and builds the MathML structure of the computed value, when `result` is `true`. With `silent=true`, the result is silenced as well.

```
\cs_new_protected:Nn \__spintent_spmc_result_mathml:n
  {
    \str_if_eq:VnT \l__spintent_mc_luaset_all_numeric_str { true }
      {
        \bool_if:cT { l__spintent_sp #1 _print_result_bool }
          {
            \bool_if:cTF { l__spintent_sp #1 _silent_bool }
              {
                \MathMLintent { :silent } { = }
                \MathMLintent { _- }
                  {
                    \__spintent_inner_sep:
                    \tl_use:c { l__spintent_sp #1 _luaset_ #1 _value_tl }
                  }
              }
              {
                \MathMLintent { _es-igual-a } { = }
                \tl_use:c { l__spintent_sp #1 _luaset_ #1 _value_tl }
              }
          }
      }
  }
```

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

`\spmcd` `\spmcm` The commands `\spmcd` and `\spmcm` are the public commands 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_exec:nnnn`, passing along the appropriate reading strings and arguments.

```
\NewDocumentCommand \spmcd { O{} d() }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \__spintent_spmc_exec: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_exec: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 20.*)

15.30 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 send the validation of the arguments and the calculation to the Lua module `spintent.lua`, constructing a valid intent.

©2026 by Pablo González L

15.30.1 Definition of variables for \spnM and \spnD

\l__spintent_spnmd_luaset_is_number_str
\l__spintent_spnM_spnD_luaset_error_str

The variable \l__spintent_spnmd_luaset_is_number_str stores true when the {⟨*argument*⟩} is a positive integer. The variable \l__spintent_spnM_spnD_luaset_error_str stores true when the {⟨*argument*⟩} is zero. Both are set by the Lua module spintent.lua.

```
\str_new:N \l__spintent_spnM_spnD_luaset_error_str
\str_new:N \l__spintent_spnmd_luaset_is_number_str
```

(*End of definition for* \l__spintent_spnmd_luaset_is_number_str *and* \l__spintent_spnM_spnD_luaset_error_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

Each command has its own copy of these variables. The variables _luaset_result_tl store the multiples or the divisors, separated by commas, and the variables _luaset_is_truncated_str store true when the list is cut, that is, when not all the results are printed; both are set by the Lua module spintent.lua. The variables _prefix_tl and _print_result_num_tl store the values of the keys prefix and result-num, and the second ones are created by the key.

```
\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_spnM_luaset_result_tl *and others.*)

\l__spintent_md_out_seq
\l__spintent_spnM_spnD_intent_tl

The variable \l__spintent_md_out_seq stores the results to be printed, one by one. The variable \l__spintent_spnM_spnD_intent_tl holds the intent that __spintent_spnM_spnD_intent:nn builds.

```
\seq_new:N \l__spintent_md_out_seq
\tl_new:N  \l__spintent_spnM_spnD_intent_tl
```

(*End of definition for* \l__spintent_md_out_seq *and* \l__spintent_spnM_spnD_intent_tl.)

15.30.2 Definition of error messages for \spnM and \spnD

md-invalid-arg
mc-too-few-args

The "error" md-invalid-arg is raised when the argument for multiples or divisors is invalid.

```
\msg_new:nnnn { spintent } { md-invalid-arg }
  {
    Invalid~argument~for~'\tl_trim_spaces:n {#1}'~\msg_line_context:.
  }
  {
    The~argument~must~be~a~single~positive~integer.
  }
```

The "error" mc-too-few-args is raised when fewer than two arguments are provided.

```
\msg_new:nnnn { spintent } { mc-too-few-args }
  {
    Invalid~arguments~for~'\tl_trim_spaces:n {#1}'~\msg_line_context:.
  }
  {
    At~least~two~arguments~are~required.
  }
```

(*End of definition for* md-invalid-arg *and* mc-too-few-args.)

15.30.3 Definition of keys for \spnM and \spnD

result
silent
result-num
prefix
concept
unknown

We define the shared ⟨*keys*⟩ result, silent, 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     .bool_set:c = { l__spintent_sp #1 _silent_bool },
        silent     .initial:n = false,
        silent     .value_required:n = true,
        result-num .tl_set:c = { l__spintent_sp #1 _print_result_num_tl },
        result-num .initial:n = ,
```

©2026 by Pablo González L

```
        result-num .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,
        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.*)

15.30.4 Internal functions for `\spnM` and `\spnD`

`\__spintent_spnM_spnD_intent:nn`

The internal function `\__spintent_spnM_spnD_intent:nn` resolves the main reading intent. It takes the command identifier '`#1`', such as "nM" or "nD", and the base reading string '`#2`'. It prepends the localized prefix if defined, normalizes accentuation for *"múltiplos"*, appends '`-de`', and stores the result in `\l__spintent_spnM_spnD_intent_tl`.

```
\cs_new_protected:Nn \__spintent_spnM_spnD_intent:nn
  {
    \tl_set:Ne \l__spintent_spnM_spnD_intent_tl
      {
        _
        \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_spnM_spnD_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 put between the parentheses to prevent NVDA/MathCAT from applying default contextual reading rules. The argument is wrapped in a literal '`_-`' intent, not in `:silent`, so that intents of its own, such as those of `\spnum`, are silenced too.

```
\cs_new_protected:Nn \__spintent_spnM_spnD_silent:nn
  {
    \MathMLintent { :silent } { \operatorname {#1} }
    \MathMLintent { :silent } { \lparen }
    \MathMLintent { :silent } { \c__spintent_invisible_sep_int } % prevent doble "de"
    \MathMLintent { _- } { \__spintent_inner_sep: #2 }
    \MathMLintent { :silent } { \rparen }
  }
```

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

`\__spintent_spnM_spnD_result:nn`

The internal function `\__spintent_spnM_spnD_result:nn` calculates and formats the computed set. It takes the command identifier '`#1`' and the classified numeric argument '`#2`'. It invokes the Lua calculation, reporting `md-invalid-arg` if it fails. On success, it maps the resulting clist into a sequence and formats the printed output with custom intents for the separators and the conjunctions.

```
\cs_new_protected:Nn \__spintent_spnM_spnD_result:nn
  {
    \use:c { __spintent_luafun_calcular_ #1 :nn } {#2}
      {
        \tl_use:c { l__spintent_sp #1 _print_result_num_tl }
      }
    \str_if_eq:VnTF \l__spintent_spnM_spnD_luaset_error_str { true }
      {
        \msg_error:nne { spintent } { md-invalid-arg } { \exp_not:c { sp #1 } }
      }
```

©2026 by Pablo González L

```
      {
        \MathMLintent { :silent } { \c__spintent_invisible_sep_int } % prevent "por"
        \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 { :silent } { , }
            \str_if_eq:vnT { l__spintent_sp #1 _luaset_is_truncated_str } { false }
              {
                \MathMLintent { _y- } { \c__spintent_invisible_sep_int } % "y" (solo dos)
              }
          }
          { \MathMLintent { :silent } { , } }
          {
            \MathMLintent { :silent } { , }
            \str_if_eq:vnT { l__spintent_sp #1 _luaset_is_truncated_str } { false }
              {
                \MathMLintent { _y- } { \c__spintent_invisible_sep_int } % "y" final
              }
          }
        \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:nn`.)

`\__spintent_spnM_spnD_exec:nnnnn`

The internal function `\__spintent_spnM_spnD_exec:nnnnn` is the one called by the commands `\spnM` and `\spnD`. 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`'. It first sets the provided keys within the current scope. With '`#5`' given, `\__spintent_luafun_spnmd_classify:n` classifies it, and a zero is rejected with `md-invalid-arg`. Then it calls `\__spintent_spnM_spnD_mathml:nnnn`.

```
\cs_new_protected:Nn \__spintent_spnM_spnD_exec:nnnnn
  {
    \keys_set:nn { spintent / sp #1 } {#4}
    \tl_if_novalue:nF {#5}
      {
        \__spintent_luafun_spnmd_classify:n { \exp_not:n {#5} }
        \str_if_eq:VnT \l__spintent_spnM_spnD_luaset_error_str { true }
          { \msg_error:nne { spintent } { md-invalid-arg } { \exp_not:c { sp #1 } } }
      }
    \__spintent_spnM_spnD_mathml:nnnn {#1} {#2} {#3} {#5}
  }
```

(*End of definition for* `\__spintent_spnM_spnD_exec:nnnnn`.)

`\__spintent_spnM_spnD_mathml:nnnn`

The function `\__spintent_spnM_spnD_mathml:nnnn` takes the command identifier in '`#1`', the symbol in '`#2`', the base reading string in '`#3`', and the optional argument in '`#4`', and builds the `MathML` structure. The parentheses are always silenced.

With '`#4`' absent, the placeholder `n` stands in for the argument. With '`#4`' given, the classification stored in `\l__spintent_spnmd_luaset_is_number_str` decides whether it is wrapped with `\spnum`, and then `result` decides whether `\__spintent_spnM_spnD_result:nn` prints the computed set; otherwise the argument is typeset verbatim and no result is attempted. With `silent=true` the printed result is wrapped in a literal '`_-`' intent.

Either way, `silent` 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_mathml:nnnn
  {
    \tl_if_novalue:nTF {#4}
      {
        \bool_if:cTF { l__spintent_sp #1 _silent_bool }
          { \__spintent_spnM_spnD_silent:nn {#2} { n } }
          {
```

```
            \bool_if:cTF { l__spintent_sp #1 _concept_bool }
              {
                \__spintent_spnM_spnD_intent:nn {#1} {#3}
                \MathMLintent
                  { \l__spintent_spnM_spnD_intent_tl }
                  { \operatorname {#2} }
              }
              { \MathMLintent { :silent } { \operatorname {#2} } }
            \MathMLintent { :silent } { \lparen }
            % prevent doble "de"
            \MathMLintent { :silent } { \c__spintent_invisible_sep_int }
            n
            \MathMLintent { :silent } { \rparen }
          }
      }
      {
        \bool_if:cTF { l__spintent_sp #1 _silent_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 }
              {
                \__spintent_spnM_spnD_intent:nn {#1} {#3}
                \MathMLintent
                  { \l__spintent_spnM_spnD_intent_tl }
                  { \operatorname {#2} }
              }
              { \MathMLintent { :silent } { \operatorname {#2} } }
            \MathMLintent { :silent } { \lparen }
            % prevent doble "de"
            \MathMLintent { :silent } { \c__spintent_invisible_sep_int }
            \str_if_eq:VnTF \l__spintent_spnmd_luaset_is_number_str { true }
              { \spnum {#4} }
              {#4}
            \MathMLintent { :silent } { \rparen }
          }
        \str_if_eq:VnT \l__spintent_spnmd_luaset_is_number_str { true }
          {
            \bool_if:cT { l__spintent_sp #1 _print_result_bool }
              {
                \bool_if:cTF { l__spintent_sp #1 _silent_bool }
                  {
                    % prevent "por"
                    \MathMLintent { :silent } { \c__spintent_invisible_sep_int }
                    \MathMLintent { _- }
                      {
                        \__spintent_inner_sep:
                        \__spintent_spnM_spnD_result:nn {#1} {#4}
                      }
                  }
                  {
                    \__spintent_spnM_spnD_result:nn {#1} {#4}
                  }
              }
          }
      }
  }
```

(*End of definition for* `\__spintent_spnM_spnD_mathml:nnnn`.)

`\spnM` `\spnD` The commands `\spnM` and `\spnD` are the public commands for typesetting multiples and divisors, respectively. They enforce math mode, open a local group, and defer execution to the function `\__spintent_spnM_spnD_exec: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_exec: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_exec: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* 20.)

15.31 The command \sprazon

The user command `\sprazon` typesets a ratio a:b as a fraction, using `\dfrac` or `\frac` according to `use-dfrac`, read natively, without parsing in Lua.

15.31.1 Definition of variables for \sprazon

`\l__spintent_sprazon_cmd_str`

The string variable `\l__spintent_sprazon_cmd_str` holds the name of the command that the message of an unknown key reports: `sprazon`, or `spProp`, which reuses the keys of `\sprazon`.

```
\str_new:N \l__spintent_sprazon_cmd_str
\str_set:Nn \l__spintent_sprazon_cmd_str { sprazon }
```

(*End of definition for* `\l__spintent_sprazon_cmd_str`.)

15.31.2 Definition of keys for \sprazon

`silent` `use-dfrac` `inline` `unknown`

We define the ⟨*keys*⟩ `silent`, `use-dfrac` and `inline`.

```
\keys_define:nn { spintent / sprazon }
  {
    silent    .bool_set:N = \l__spintent_sprazon_silent_bool,
    silent    .initial:n = false,
    silent    .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,
    inline    .bool_set:N = \l__spintent_sprazon_inline_bool,
    inline    .initial:n = false,
    inline    .value_required:n = true,
    unknown   .code:n =
      {
        \str_set_eq:NN \l__spintent_command_name_str \l__spintent_sprazon_cmd_str
        \__spintent_unknown_keys_cmd:n {#1}
      },
  }
```

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

15.31.3 Internal functions for \sprazon

`\__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`'. It processes the keys and passes the numerator and the denominator to `\__spintent_sprazon_mathml:nn`.

```
\cs_new_protected:Nn \__spintent_sprazon_exec:nnn
  {
    \keys_set:nn { spintent / sprazon } {#1}
    \__spintent_sprazon_mathml:nn {#2} {#3}
  }
```

(*End of definition for* `\__spintent_sprazon_exec:nnn`.)

`\__spintent_sprazon_mathml:nn`

The function `\__spintent_sprazon_mathml:nn` receives the numerator in '`#1`' and the denominator in '`#2`'. With `silent`, the whole expression is silenced as one unit: `\frac` or `\dfrac`, according to `use-dfrac`, or `#1 : #2` with `inline`.

Otherwise, with `inline=false`, the default, the numerator and the denominator are tagged with a native `\MathMLarg`, as `num` and `denom`, inside `\frac` or `\dfrac`, and the outer `\MathMLintent` reads `_es-a:infix($num,$denom)`. With `inline=true`, the two references are tagged side by side and the `:` between them carries the `_es-a:infix($num,$denom)` intent.

```
\cs_new_protected:Nn \__spintent_sprazon_mathml:nn
  {
    \__spintent_invisible_sep:
    \bool_if:NTF \l__spintent_sprazon_silent_bool
      {
        \MathMLintent { _- }
          {
            \__spintent_inner_sep:
            \bool_if:NTF \l__spintent_sprazon_inline_bool
              { #1 : #2 }
              {
                \bool_if:NTF \l__spintent_sprazon_use_dfrac_bool
                  { \dfrac {#1} {#2} }
                  { \frac  {#1} {#2} }
              }
          }
      }
      {
        \bool_if:NTF \l__spintent_sprazon_inline_bool
          {
            \MathMLintent { _es-a:infix($num,$denom) }
              {
                \MathMLarg { num } {#1}
                :
                \MathMLarg { denom } {#2}
              }
          }
          {
            \MathMLintent { _es-a:infix($num,$denom) }
              {
                \__spintent_inner_sep:
                \bool_if:NTF \l__spintent_sprazon_use_dfrac_bool
                  {
                    \dfrac { \MathMLarg { num } {#1} } { \MathMLarg { denom } {#2} }
                  }
                  { \frac  { \MathMLarg { num } {#1} } { \MathMLarg { denom } {#2} } }
              }
          }
      }
  }
```

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

`\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 23.*)

15.32 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.

15.32.1 Internal functions for `\spProp`

`\spProp` The public command `\spProp` enforces *math mode*, and forwards the same ⟨*keys*⟩ in '`#1`' to both ratios: `a/b`, built from '`#2`' and '`#3`', and `c/d`, built from '`#4`' and '`#5`', joined by a native `\MathMLintent` reading `_como` over the `=` sign.

```
\NewDocumentCommand \spProp { O{} m m m m }
```

©2026 by Pablo González L

```
  {
    \mode_if_math:TF
      {
        \group_begin:
          \str_set:Nn \l__spintent_sprazon_cmd_str { spProp }
          \__spintent_sprazon_exec:nnn {#1} {#2} {#3}
          \bool_if:NTF \l__spintent_sprazon_silent_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 23.*)

15.33 The command \spang

The command `\spang` is the main public command for typesetting and tagging sexagesimal angle expressions. 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.

15.33.1 Definition of variables for \spang

`\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 variables store the values returned by the Lua module `spintent.lua`. The variables `\l__spintent_spang_luaset_grado_str`, `\l__spintent_spang_luaset_minuto_str` and `\l__spintent_spang_luaset_segundo_str` store the degrees, the minutes and the seconds, and are empty if missing. The variable `\l__spintent_spang_luaset_error_str` stores `true` or `false` if the {⟨*argument*⟩} is invalid.

```
\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.*)

15.33.2 Definition of error messages for \spang

spang-invalid-arg

The "error" spang-invalid-arg is raised when an argument fails the format checks, does not match the expected sexagesimal pattern, or contains invalid measurement symbols.

```
\msg_new:nnnn { spintent } { spang-invalid-arg }
  {
    Invalid~argument~'#2'~for~'\tl_trim_spaces:n {#1}'~\msg_line_context:.
  }
  {
    The~argument~must~have~the~form~'degrees',~'degrees:minutes'~or~
    'degrees:minutes:seconds'.
  }
```

(*End of definition for* `spang-invalid-arg`.)

15.33.3 Internal functions for \spang

`\__spintent_spang_exec:n`

The internal function `\__spintent_spang_exec:n` parses the angle input using `\__spintent_luafun_spang_pars` after resetting the error flag in `\l__spintent_spang_luaset_error_str`. If an invalid format is detected, it issues the spang-invalid-arg error. Otherwise, it calls `\__spintent_spang_mathml:`.

```
\cs_new_protected:Nn \__spintent_spang_exec: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 } { spang-invalid-arg } { \spang } {#1}
      }
      { \__spintent_spang_mathml: }
  }
```

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

©2026 by Pablo González L

__spintent_spang_mathml:

The internal function __spintent_spang_mathml: 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. Each value is expanded with \exp_args:Ne before it reaches \spunit, which protects its {⟨*argument*⟩} from expansion.

```
\cs_new_protected:Nn \__spintent_spang_mathml:
  {
    \str_if_empty:NF \l__spintent_spang_luaset_grado_str
      {
        \exp_args:Ne \spunit { \l__spintent_spang_luaset_grado_str ° }
      }
    \str_if_empty:NF \l__spintent_spang_luaset_minuto_str
      {
        \MathMLintent { _minutos:postfix($m) }
          {
            \MathMLarg { m }
              {
                \exp_args:Ne \spunit { \l__spintent_spang_luaset_minuto_str ' }
              }
          }
      }
    \str_if_empty:NF \l__spintent_spang_luaset_segundo_str
      {
        \MathMLintent { _segundos:postfix($s) }
          {
            \MathMLarg { s }
              {
                \exp_args:Ne \spunit { \l__spintent_spang_luaset_segundo_str " }
              }
          }
      }
  }
```

(*End of definition for* __spintent_spang_mathml:.)

\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_exec:n to process the {⟨*argument*⟩}.

```
\NewDocumentCommand \spang { m }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \__spintent_spang_exec:n {#1}
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spang } }
  }
```

(*End of definition for* \spang. *This function is documented on page* 29.)

15.34 The command \spnZ

The user-level command \spnZ formats integers, the set $\mathbb{Z}$. It includes specific options to 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.

15.34.1 Definition of error messages for \spnZ

spnZ-not-integer

The "error" spnZ-not-integer is raised when the {⟨*argument*⟩} contains decimal separators or attached units, which conflict with the constraints of a pure integer representation, or when it is empty.

```
\msg_new:nnnn { spintent } { spnZ-not-integer }
  {
    Invalid~argument~'#1'~for~'\token_to_str:N \spnZ'~\msg_line_context:.
  }
  {
    Only~integers~are~accepted;~decimal~separators~and~units~are~not~allowed.
  }
```

©2026 by Pablo González L

(*End of definition for* `spnZ-not-integer`.)

15.34.2 Definition of keys for \spnZ

`cifras-sep` `read-sign` `wrap-parens` `read-parens` `unknown`

We define the ⟨*keys*⟩ `cifras-sep`, `read-sign`, `wrap-parens` and `read-parens`.

```
\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.*)

15.34.3 Internal functions for \spnZ

`\__spintent_spnz_mathml:`

The function `\__spintent_spnz_mathml:` builds the `MathML` structure of the integer. If `wrap-parens` is true, it prints the number surrounded by parentheses, checking the state of `\l__spintent_spnz_read_parens_bool`, which the `read-parens` key controls: if it is `true`, the parentheses are read; otherwise it applies the `:silent` *intent property* to them. If `wrap-parens` is false, it prints the number alone with `\__spintent_spnum_print_num_start:`.

```
\cs_new_protected:Nn \__spintent_spnz_mathml:
  {
    \bool_if:NTF \l__spintent_spnz_wrap_parens_bool
      {
        \bool_if:NTF \l__spintent_spnz_read_parens_bool
          {
            \lparen \__spintent_spnum_print_num_start: \rparen
          }
          {
            \MathMLintent { :silent } { \lparen }
            \__spintent_spnum_print_num_start:
            \MathMLintent { :silent } { \rparen }
          }
      }
      { \__spintent_spnum_print_num_start: }
  }
```

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

`\__spintent_spnz_exec:nn`

The function `\__spintent_spnz_exec: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 calls `\__spintent_spnz_mathml:`.

```
\cs_new_protected:Nn \__spintent_spnz_exec: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 }
      { \__spintent_spnz_mathml: }
      { \msg_error:nnn { spintent } { spnZ-not-integer } {#2} }
  }
```

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

`\spnZ`

Defines the user command `\spnZ`, which is executed within a *group* and enforces *math mode*.

```
\NewDocumentCommand \spnZ { O{} m }
  {
```

```
    \mode_if_math:TF
      {
        \group_begin:
          \__spintent_spnz_exec:nn {#1} {#2}
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spnZ } }
  }
```

(*End of definition for* `\spnZ`. *This function is documented on page 22.*)

15.35 The command \spmQ

The command `\spmQ` typesets *"mixed numbers"* within Spanish mathematical contexts such as $3\frac{1}{2}$. It sends 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 intent for `MathCAT`.

15.35.1 Definition of variables for \spmQ

`\l__spintent_spmQ_join_word_str` `\l__spintent_spmQ_intent_read_sign_str`

The variable `\l__spintent_spmQ_join_word_str` stores the value set by the key `join-word`, which `MathCAT` verbalizes as a grammatical connector between the integer and fractional parts. The variable `\l__spintent_spmQ_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`.

```
\str_new:N \l__spintent_spmQ_join_word_str
\str_new:N \l__spintent_spmQ_intent_read_sign_str
```

(*End of definition for* `\l__spintent_spmQ_join_word_str` *and* `\l__spintent_spmQ_intent_read_sign_str`.)

`\l__spintent_spmQ_wrap_parens_l_tl` `\l__spintent_spmQ_wrap_parens_r_tl`

The variables `\l__spintent_spmQ_wrap_parens_l_tl` and `\l__spintent_spmQ_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`.

```
\tl_new:N \l__spintent_spmQ_wrap_parens_l_tl
\tl_new:N \l__spintent_spmQ_wrap_parens_r_tl
```

(*End of definition for* `\l__spintent_spmQ_wrap_parens_l_tl` *and* `\l__spintent_spmQ_wrap_parens_r_tl`.)

`\l__spintent_spmQ_error_bool`

The boolean variable `\l__spintent_spmQ_error_bool` is set to `true` by `\__spintent_spmQ_check_part:nn` when any of the three arguments is rejected, so that nothing is typeset.

```
\bool_new:N \l__spintent_spmQ_error_bool
```

(*End of definition for* `\l__spintent_spmQ_error_bool`.)

`\l__spintent_saved_luamml_flag_int`

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.

```
\int_new:N \l__spintent_saved_luamml_flag_int
```

(*End of definition for* `\l__spintent_saved_luamml_flag_int`.)

15.35.2 Definition of error messages for \spmQ

`spmQ-zero-denominator` `spmQ-not-integer` `spmQ-signed-part`

The "error" `spmQ-zero-denominator` is raised when a zero denominator is passed to `\spmQ`. The "error" `spmQ-not-integer` is raised when any of the three arguments is not actually an integer. The "error" `spmQ-signed-part` is raised when the numerator or the denominator carries a sign: only the integer part may be signed.

```
\msg_new:nnnn { spintent } { spmQ-zero-denominator }
  {
    Invalid~denominator~for~'\token_to_str:N \spmQ'~\msg_line_context:.
  }
  {
    The~denominator~of~a~fraction~cannot~be~zero.
  }
\msg_new:nnnn { spintent } { spmQ-not-integer }
  {
    Invalid~argument~'#1'~for~'\token_to_str:N \spmQ'~\msg_line_context:.
  }
```

```
  {
    Only~integers~are~accepted.
  }
\msg_new:nnnn { spintent } { spmQ-signed-part }
  {
    Invalid~argument~'#1'~for~'\token_to_str:N \spmQ'~\msg_line_context:.
  }
  {
    The~numerator~and~the~denominator~must~be~integers~without~sign;~only~the~integer~
    part~may~have~one.
  }
```

(*End of definition for* `spmQ-zero-denominator` , `spmQ-not-integer` , *and* `spmQ-signed-part`.)

15.35.3 Definition of keys for \spmQ

join-word
use-spnZ
wrap-parens
read-parens
use-dfrac
silent
unknown

We define the ⟨*keys*⟩ `join-word`, `use-spnZ`, `wrap-parens`, `read-parens`, `use-dfrac` and `silent`.

```
\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,
    use-dfrac   .bool_set:N = \l__spintent_spmQ_use_dfrac_bool,
    use-dfrac   .initial:n = false,
    use-dfrac   .value_required:n = true,
    wrap-parens .bool_set:N = \l__spintent_spmQ_wrap_parens_bool,
    wrap-parens .initial:n = false,
    wrap-parens .value_required:n = true,
    read-parens .bool_set:N = \l__spintent_spmQ_read_parens_bool,
    read-parens .initial:n = true,
    read-parens .value_required:n = true,
    silent      .bool_set:N = \l__spintent_spmQ_silent_bool,
    silent      .initial:n = false,
    silent      .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.*)

15.35.4 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_spmQ_wrap_parens_l_tl` and `\l__spintent_spmQ_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 `use-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_spmQ_wrap_parens_l_tl { \Bigl \lparen }
              \tl_set:Nn \l__spintent_spmQ_wrap_parens_r_tl { \Bigr \rparen }
            }
          { 1 }
            {
              \tl_set:Nn \l__spintent_spmQ_wrap_parens_l_tl { \Bigl \lparen }
              \tl_set:Nn \l__spintent_spmQ_wrap_parens_r_tl { \Bigr \rparen }
            }
          { 2 }
            {
              \tl_set:Nn \l__spintent_spmQ_wrap_parens_l_tl { \bigl \lparen }
              \tl_set:Nn \l__spintent_spmQ_wrap_parens_r_tl { \bigr \rparen }
            }
          { 3 }
            {
              \tl_set:Nn \l__spintent_spmQ_wrap_parens_l_tl { \bigl \lparen }
              \tl_set:Nn \l__spintent_spmQ_wrap_parens_r_tl { \bigr \rparen }
            }
        }
        {
          \tl_set:Nn \l__spintent_spmQ_wrap_parens_l_tl { \lparen }
          \tl_set:Nn \l__spintent_spmQ_wrap_parens_r_tl { \rparen }
        }
```

If the key `use-dfrac` is active, we must reset the variables `\l__spintent_spmQ_wrap_parens_l_tl` and `\l__spintent_spmQ_wrap_parens_r_tl` to `\Bigl`/`\Bigr` unconditionally.

```
    \bool_if:NT \l__spintent_spmQ_use_dfrac_bool
      {
        \tl_set:Nn \l__spintent_spmQ_wrap_parens_l_tl { \Bigl \lparen }
        \tl_set:Nn \l__spintent_spmQ_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_spmQ_wrap_parens_l_tl` or `\l__spintent_spmQ_wrap_parens_r_tl`.

If the variable `\l__spintent_spmQ_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_spmQ_wrap_parens_bool
      {
        \__spintent_spmQ_set_wrap_parens_size:
        \bool_if:NTF \l__spintent_spmQ_read_parens_bool
          { \l__spintent_spmQ_wrap_parens_l_tl }
          { \MathMLintent { :silent } { \l__spintent_spmQ_wrap_parens_l_tl } }
      }
  }
\cs_new_protected:Nn \__spintent_spmQ_stop_wrap_parens:
  {
    \bool_if:NT \l__spintent_spmQ_wrap_parens_bool
      {
        \__spintent_spmQ_set_wrap_parens_size:
        \bool_if:NTF \l__spintent_spmQ_read_parens_bool
          { \l__spintent_spmQ_wrap_parens_r_tl }
          { \MathMLintent { :silent } { \l__spintent_spmQ_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:` sets the grammatical connector between the integer part and the fraction by branching 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_scale_int:Vn, 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, which skips it 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, determined by the key use-dfrac, as its two arguments, and inserts the appropriately scaled sub_box node 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_use_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 set the shared variables from Lua, 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 to __spintent_spmQ_print_scale_int: to render the scaled integer. When a sign is present, it sets \l__spintent_spmQ_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_spmQ_intent_read_sign_str { _más:prefix($e) }
              }
              {
                \str_set:Nn
                  \l__spintent_spmQ_intent_read_sign_str { _menos:prefix($e) }
              }
```

```
            \MathMLintent { \l__spintent_spmQ_intent_read_sign_str }
              {
                \l__spintent_luaset_sign_tl \c__spintent_invisible_sep_int
                \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`', both already validated by `\__spintent_spmQ_check_part:nn`. The function branches on the two boolean variables `\l__spintent_spmQ_use_spnZ_bool` and `\l__spintent_spmQ_use_dfrac_bool`, controlled respectively by the keys `use-spnZ` and `use-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 as given. In both branches, the fraction command is selected with `\use:c`: if `use-dfrac` is `true`, `dfrac` is assembled; otherwise, `frac` is used. Inside the fraction the outer invisible separator is not typeset, only recorded as already placed, since the `\spnZ` of the numerator would put it there as the first element, and `MathCAT` would no longer read the fraction as a simple one.

```
\cs_new_protected:Nn \__spintent_spmQ_print_spnZ_dfrac:nn
  {
    \bool_if:NTF \l__spintent_spmQ_use_spnZ_bool
      {
        \group_begin:
          \cs_set_protected:Nn \__spintent_invisible_sep:
            {
              \__spintent_luafun_invisible_sep_mark:
            }
          \use:c { \bool_if:NT \l__spintent_spmQ_use_dfrac_bool { d } frac }
            { \spnZ {#1} } { \spnZ {#2} }
        \group_end:
      }
      {
        \use:c { \bool_if:NT \l__spintent_spmQ_use_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 function of `\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 sets 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 `use-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_check_part:nn`

The function `\__spintent_spmQ_check_part:nn` validates one argument of `\spmQ`. It receives the role of the argument in '`#1`', which is `int`, `num` or `den`, and the argument itself in '`#2`', and calls `\__spintent_luafun_clean_split_and_set:n` on it. If `\l__spintent_luaset_has_integer_str` is `false`, the `spmQ-not-integer` "error" is raised. If the argument is an integer but carries a sign and its role is not `int`, the `spmQ-signed-part` "error" is raised. If the role is `den` and the value is zero, the `spmQ-zero-denominator` "error" is raised. Every rejected argument sets

`\l__spintent_spmQ_error_bool` to `true`; the arguments are rejected whatever the value of the key `silent`.

```
\cs_new_protected:Nn \__spintent_spmQ_check_part:nn
  {
    \__spintent_luafun_clean_split_and_set:n { \exp_not:n {#2} }
    \str_if_eq:VnTF \l__spintent_luaset_has_integer_str { true }
      {
        \bool_lazy_or:nnTF
          { \tl_if_empty_p:N \l__spintent_luaset_sign_tl }
          { \str_if_eq_p:nn {#1} { int } }
          {
            \str_if_eq:nnT {#1} { den }
              {
                \int_compare:nNnT { \l__spintent_luaset_part_int_tl } = { 0 }
                  {
                    \msg_error:nn { spintent } { spmQ-zero-denominator }
                    \bool_set_true:N \l__spintent_spmQ_error_bool
                  }
              }
          }
          {
            \msg_error:nnn { spintent } { spmQ-signed-part } {#2}
            \bool_set_true:N \l__spintent_spmQ_error_bool
          }
      }
      {
        \msg_error:nnn { spintent } { spmQ-not-integer } {#2}
        \bool_set_true:N \l__spintent_spmQ_error_bool
      }
  }
```

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

`\__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 validates the three arguments with `\__spintent_spmQ_check_part:nn`. If any of them is rejected, nothing is typeset.

Otherwise, it calls `\__spintent_spmQ_mathml:nnn`.

```
\cs_new_protected:Nn \__spintent_spmQ_exec:nnnn
  {
    \keys_set:nn { spintent / spmQ } {#1}
    \bool_set_false:N \l__spintent_spmQ_error_bool
    \__spintent_spmQ_check_part:nn { int } {#2}
    \__spintent_spmQ_check_part:nn { num } {#3}
    \__spintent_spmQ_check_part:nn { den } {#4}
    \bool_if:NF \l__spintent_spmQ_error_bool
      { \__spintent_spmQ_mathml:nnn {#2} {#3} {#4} }
  }
```

(*End of definition for* `\__spintent_spmQ_exec:nnnn`.)

`\__spintent_spmQ_mathml:nnn`

The internal function `\__spintent_spmQ_mathml:nnn` takes the integer '`#1`', the numerator '`#2`', and the denominator '`#3`', and calls `\__spintent_spmQ_print:nnn` to format the *"mixed number"*. With the key `silent` set to `true`, the whole *"mixed number"* is wrapped in a single `\MathMLintent{_-}`, preceded by the outer invisible separator and with the inner one first inside it, so MathCAT verbalizes nothing.

```
\cs_new_protected:Nn \__spintent_spmQ_mathml:nnn
  {
    \bool_if:NTF \l__spintent_spmQ_silent_bool
      {
        \__spintent_invisible_sep:
        \MathMLintent { _- }
          {
            \__spintent_inner_sep:
            \__spintent_spmQ_print:nnn {#1} {#2} {#3}
          }
      }
      { \__spintent_spmQ_print:nnn {#1} {#2} {#3} }
  }
```

(*End of definition for* `\__spintent_spmQ_mathml:nnn`.)

`\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* 22.)

15.36 The commands of the geo family

The commands described in this section provide a semantic interface for geometric objects in Spanish mathematical contexts. They delegate the parsing to `spintent.lua` and share the variables `\l__spintent_geo_luaset_error_str`, `\l__spintent_geo_luaset_intent_str`, `\l__spintent_geo_luaset_print_tl`, `\l__spintent_geo_luaset_letra_str`, `\l__spintent_geo_luaset_` and `\l__spintent_geo_luaset_tipo_str`, with which the intent is built. A single point is always annotated with `\__spintent_mathmlarg_point:nnn`. Every command activates `\l__spintent_scope_attr_int` inside its own group, so that the shared queue only claims content typeset by a geo command.

15.36.1 Definition of variables for the geo family

`\l__spintent_geo_luaset_error_str` `\l__spintent_geo_luaset_intent_str` `\l__spintent_geo_luaset_print_tl` These variables are shared by all the commands of the family and are set by the Lua module `spintent.lua`. The variable `\l__spintent_geo_luaset_error_str` stores `true` or `false` if the {⟨*argument*⟩} is invalid. The variable `\l__spintent_geo_luaset_intent_str` stores the body of the intent and `\l__spintent_geo_luaset_print_tl` stores what is printed.

```
\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
```

(*End of definition for* `\l__spintent_geo_luaset_error_str`, `\l__spintent_geo_luaset_intent_str`, *and* `\l__spintent_geo_luaset_print_tl`.)

`\l__spintent_geo_luaset_points_clist` `\l__spintent_geo_luaset_is_name_str` `\l__spintent_geo_luaset_letra_str` `\l__spintent_geo_luaset_sub_str` `\l__spintent_geo_luaset_tipo_str` The variable `\l__spintent_geo_luaset_points_clist` stores the points of the {⟨*argument*⟩} separated by commas and not joined, such as `A,B,C`. The variable `\l__spintent_geo_luaset_is_name_str` stores `true` when the {⟨*argument*⟩} is the name of a line, ray or segment and `false` when it is a list of points. When the {⟨*argument*⟩} is a single point or name, `\l__spintent_geo_luaset_letra_str` stores its letter, `\l__spintent_geo_luaset_sub_str` stores its subscript or prime and `\l__spintent_geo_luaset_tipo_str` stores `sub` or `prima`, and is empty if there is none.

```
\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_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_points_clist` *and others.*)

`\l__spintent_geo_luaset_poly_sym_str` `\l__spintent_geo_luaset_concept_str` The variable `\l__spintent_geo_luaset_poly_sym_str` stores `triangle`, `square` or nothing, according to the number of vertices of the polygon. The variable `\l__spintent_geo_luaset_concept_str` stores the name of the polygon or of the concept, such as `triángulo` or `área-del-triángulo`.

```
\str_new:N \l__spintent_geo_luaset_poly_sym_str
\str_new:N \l__spintent_geo_luaset_concept_str
```

(*End of definition for* `\l__spintent_geo_luaset_poly_sym_str` *and* `\l__spintent_geo_luaset_concept_str`.)

`\l__spintent_geo_luaset_angulo_case_str` `\l__spintent_geo_luaset_nombre_es_num_str` The variable `\l__spintent_geo_luaset_angulo_case_str` stores the kind of {⟨*argument*⟩} that Lua recognized for an angle: `tres-puntos`, `un-punto` or `nombre`. The variable `\l__spintent_geo_luaset_nombre_es_nu` stores `true` when that name is a number.

```
\str_new:N \l__spintent_geo_luaset_angulo_case_str
\str_new:N \l__spintent_geo_luaset_nombre_es_num_str
```

©2026 by Pablo González L

(*End of definition for* `\l__spintent_geo_luaset_angulo_case_str` *and* `\l__spintent_geo_luaset_nombre_es_num_str`.)

`\l__spintent_geo_luaset_letra_word_str`
`\l__spintent_geo_luaset_letra_conector_str`
`\l__spintent_geo_luaset_sub_spoken_str`

These variables are used by the commands for areas and perimeters of figures. When the {⟨*argument*⟩} is a letter of the table, `F`, `P`, `B`, `L` or `T`, the variable `\l__spintent_geo_luaset_letra_word_str` stores its word, such as `figura` or `lateral`, and `\l__spintent_geo_luaset_letra_conector_str` stores the connector that the word needs, such as `de-la`, which is empty if it needs none. The variable `\l__spintent_geo_luaset_sub_spoken_str` stores the subscript as it is spoken.

```
\str_new:N \l__spintent_geo_luaset_letra_word_str
\str_new:N \l__spintent_geo_luaset_letra_conector_str
\str_new:N \l__spintent_geo_luaset_sub_spoken_str
```

(*End of definition for* `\l__spintent_geo_luaset_letra_word_str`, `\l__spintent_geo_luaset_letra_conector_str`, *and* `\l__spintent_geo_luaset_sub_spoken_str`.)

15.37 The command \spcoord

The user command `\spcoord` typesets an ordered pair `(x,y)`. The comma or semicolon between the coordinates carries its own `:infix` intent, and each coordinate that starts with + or - is wrapped with `:prefix($vp)` or `:prefix($vn)`, so that the sign is read as *"más"* or *"menos"*.

15.37.1 Definition of variables for \spcoord

`\l__spintent_spcoord_luaset_error_str`
`\l__spintent_spcoord_luaset_sep_str`

These variables are set by the function `\__spintent_luafun_spcoord_parse_and_set:n`. The variable `\l__spintent_spcoord_luaset_error_str` stores `true` or `false` if the {⟨*argument*⟩} is invalid. The variable `\l__spintent_spcoord_luaset_sep_str` stores `,` or `;`, whichever the argument actually used.

```
\str_new:N \l__spintent_spcoord_luaset_error_str
\str_new:N \l__spintent_spcoord_luaset_sep_str
```

(*End of definition for* `\l__spintent_spcoord_luaset_error_str` *and* `\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_x_composite_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`
`\l__spintent_spcoord_luaset_y_composite_str`

The variables for the abscissa `x` and the ordinate `y` have the same meaning and are set by the same function: `_sign_str` is `+`, `-`, or empty; `_type_str` is `number` or `raw`; `_value_str` is the value with the sign already removed; `_has_frac_str` is `none`, `frac`, or `dfrac`; and `_composite_str` is `true` when the value is a sole subscripted, superscripted or radical node, `false` otherwise.

```
\clist_map_inline:nn { x , y }
  {
    \str_new:c { l__spintent_spcoord_luaset_ #1 _sign_str      }
    \str_new:c { l__spintent_spcoord_luaset_ #1 _type_str      }
    \str_new:c { l__spintent_spcoord_luaset_ #1 _value_str     }
    \str_new:c { l__spintent_spcoord_luaset_ #1 _has_frac_str  }
    \str_new:c { l__spintent_spcoord_luaset_ #1 _composite_str }
  }
```

(*End of definition for* `\l__spintent_spcoord_luaset_x_sign_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`

The variable `\l__spintent_spcoord_raw_arg_tl` holds the {⟨*argument*⟩} exactly as given. The variable `\l__spintent_spcoord_intent_str` holds the `:infix` word for the separator `_coma:infix` or `_punto-y-coma:infix`. The variables `\l__spintent_spcoord_wrap_parens_l_tl`/`\l__spintent_spcoord_wr` hold the opening/closing delimiter, sized by the function `\__spintent_spcoord_set_wrap_parens_size:`.

```
\tl_new:N  \l__spintent_spcoord_raw_arg_tl
\str_new:N \l__spintent_spcoord_intent_str
\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.*)

15.37.2 Definition of error messages for \spcoord

`spcoord-invalid-arg`

The "error" `spcoord-invalid-arg` is raised when the {⟨*argument*⟩} does not have the form *abscissa,ordinate* or *abscissa;ordinate*.

```
\msg_new:nnnn { spintent } { spcoord-invalid-arg }
  {
    Invalid~argument~'#1'~for~'\token_to_str:N \spcoord'~\msg_line_context:.
  }
  {
    The~argument~must~have~the~form~'abscissa,~ordinate'~or~'abscissa;~ordinate'.
  }
```

(*End of definition for* `spcoord-invalid-arg`.)

15.37.3 Definition of keys for \spcoord

silent unknown

We define the ⟨*key*⟩ `silent`.

```
\keys_define:nn { spintent / spcoord }
  {
    silent  .bool_set:N = \l__spintent_spcoord_silent_bool,
    silent  .initial:n = false,
    silent  .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` *and* `unknown`.)

15.37.4 Internal functions for \spcoord

__spintent_spcoord_set_wrap_parens_size:

With neither side carrying a fraction, the parentheses stay unsized. A fraction counts wherever it appears in the value, be it \frac, \dfrac or a \spmQ, which prints one, except inside a subscript or superscript, where it does not make the line taller. Otherwise, \mathstyle picks \Bigl/\Bigr for display, \bigl/\bigr for text. When either side is dfrac, the parentheses are forced to \Bigl/\Bigr regardless of \mathstyle, since \dfrac always renders at display height.

```
\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 the variable `_type_str` of the endpoint in '`#1`' equal to `number`, it calls `\spnum` on its variable `_value_str`. Otherwise, it typesets it as given, a `\frac`, a `\dfrac` or a letter expression. 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 the variable `_sign_str` of the endpoint in '`#1`' equal to `+` or `-`, it wraps the sign and `\__spintent_spcoord_value:n` with `_más:prefix($vp)` or `_menos:prefix($vn)`, referencing the value with With no sign, it calls `\__spintent_spcoord_value:n` directly. 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_inner_sep:n` Receives `x` or `y` in '`#1`'. With the variable `_composite_str` of that endpoint equal to `true`, places `\__spintent_inner_sep:` first, so that the group `\MathMLarg` wraps holds two children and keeps its `arg`: luamml drops it from a sole node that carries a subscript, a superscript or a radical.

```
\cs_new_protected:Nn \__spintent_spcoord_inner_sep:n
  {
    \str_if_eq:eeT
      { \str_use:c { l__spintent_spcoord_luaset_ #1 _composite_str } } { true }
      { \__spintent_inner_sep: }
  }
```

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

`\__spintent_spcoord_intent_infix:` Sets `\l__spintent_spcoord_intent_str` to `_coma:infix` or `_punto-y-coma:infix`, according to `\l__spintent_spcoord_luaset_sep_str`, and wraps both sides, each referenced with with it. The names `$x` and `$y` stand for the output of `\__spintent_spcoord_print:n`.

```
\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_inner_sep:n { x } \__spintent_spcoord_print:n { x } }
        \MathMLintent { :silent } { \l__spintent_spcoord_luaset_sep_str }
        \MathMLarg{y}
          { \__spintent_spcoord_inner_sep:n { y } \__spintent_spcoord_print:n { y } }
      }
  }
```

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

`\__spintent_spcoord_mathml:N` The argument is the boolean variable that holds the value of `silent` of the command that calls it. With `silent=true`, the parentheses and the argument are wrapped in a single '`_-`' intent, and `\l__spintent_spcoord_raw_arg_tl` is typeset as given. Otherwise, the parentheses are silenced individually and `\__spintent_spcoord_intent_infix:` provides the reading between them.

```
\cs_new_protected:Nn \__spintent_spcoord_mathml:N
  {
    \bool_if:NTF #1
      {
        \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:N`.)

`\__spintent_spcoord_exec:nn` Sets the ⟨*keys*⟩ in '`#1`' and `\l__spintent_spcoord_raw_arg_tl` to '`#2`'. Calls `\__spintent_luafun_spcoord_par` and reports `spcoord-invalid-arg` on error, whatever the value of `silent`. Then it sizes the parentheses and, if there was no error, calls `\__spintent_spcoord_mathml:N` preceded by an invisible separator placed *before* the opening parenthesis. The whole pair is wrapped in `\mathord`, which prevents `MathCAT` from reading `(1,2)` as an interval when an equals sign or `\in` follows it.

```
\cs_new_protected:Nn \__spintent_spcoord_exec:nn
  {
    \keys_set:nn { spintent / spcoord } {#1}
    \tl_set:Nn \l__spintent_spcoord_raw_arg_tl {#2}
    \__spintent_luafun_spcoord_parse_and_set:n { \exp_not:n {#2} }
    \str_if_eq:VnT \l__spintent_spcoord_luaset_error_str { true }
      { \msg_error:nne { 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 }
      {
        \mathord
          {
            % prevent "intervalo"
            \MathMLintent { :silent } { \c__spintent_invisible_sep_int }
            \__spintent_spcoord_mathml:N \l__spintent_spcoord_silent_bool
          }
      }
  }
```

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

`\spcoord` The public command `\spcoord` 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 24.*)

15.38 The command `\spinterval`

The user command `\spinterval` receives the {⟨*argument*⟩} written as a left delimiter `(`, `[` or `]`, two endpoints separated by a semicolon or by a single comma, and a right delimiter `)`, `]` or `[`. A delimiter `(` or `]` on the left, and `)` or `[` on the right, excludes its endpoint; the others include it. The type of interval, the endpoints and the separator are detected by the Lua module.

15.38.1 Definition of variables for `\spinterval`

`\l__spintent_spinterval_luaset_error_str`
`spintent_spinterval_luaset_error_code_str`

These variables are set by the function `\__spintent_luafun_spinterval_parse_and_set:n`. The variable `\l__spintent_spinterval_luaset_error_str` stores `true` or `false` if the {⟨*argument*⟩} is invalid. The variable `\l__spintent_spinterval_luaset_error_code_str` stores the kind of error: `delim`, `vacio`, `llaves`, `sep`, `ambiguo`, `inf-orden` or `inf-cerrado`, and is empty if there is none.

```
\str_new:N \l__spintent_spinterval_luaset_error_str
\str_new:N \l__spintent_spinterval_luaset_error_code_str
```

(*End of definition for* `\l__spintent_spinterval_luaset_error_str` *and* `\l__spintent_spinterval_luaset_error_code_str`.)

`_spintent_spinterval_luaset_left_sym_str`
`spintent_spinterval_luaset_right_sym_str`
`\l__spintent_spinterval_luaset_type_str`
`\l__spintent_spinterval_luaset_sep_str`
`\l__spintent_spinterval_luaset_tall_str`

The variables `\l__spintent_spinterval_luaset_left_sym_str` and `\l__spintent_spinterval_luaset_rig` store the delimiters of the interval as they were written. The variable `\l__spintent_spinterval_luaset_type_str` stores `open`, `closed`, `open-closed` or `closed-open`. The variable `\l__spintent_spinterval_luaset_sep_str` stores the separator, `,` or `;`. The variable `\l__spintent_spinterval_luaset_tall_str` stores `dfrac`, `frac` or `none`, according to the tallest fraction in the endpoints.

```
\str_new:N \l__spintent_spinterval_luaset_left_sym_str
\str_new:N \l__spintent_spinterval_luaset_right_sym_str
\str_new:N \l__spintent_spinterval_luaset_type_str
\str_new:N \l__spintent_spinterval_luaset_sep_str
\str_new:N \l__spintent_spinterval_luaset_tall_str
```

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

`\l__spintent_spinterval_luaset_a_sign_str`
`\l__spintent_spinterval_luaset_a_type_str`
`l__spintent_spinterval_luaset_a_value_str`
`\l__spintent_spinterval_luaset_a_inf_str`
`\l__spintent_spinterval_luaset_a_tall_str`
`pintent_spinterval_luaset_a_composite_str`
`_spintent_spinterval_luaset_a_integer_str`
`l__spintent_spinterval_luaset_a_short_str`
`__spintent_spinterval_luaset_a_braced_str`
`\l__spintent_spinterval_luaset_b_sign_str`
`\l__spintent_spinterval_luaset_b_type_str`
`l__spintent_spinterval_luaset_b_value_str`
`\l__spintent_spinterval_luaset_b_inf_str`
`\l__spintent_spinterval_luaset_b_tall_str`
`pintent_spinterval_luaset_b_composite_str`
`_spintent_spinterval_luaset_b_integer_str`
`l__spintent_spinterval_luaset_b_short_str`
`__spintent_spinterval_luaset_b_braced_str`

The same variables for each endpoint, `a` and `b`, set by the same function. The variable `_sign_str` is `+`, `-`, or empty; `_type_str` is `number`, `raw` or `infinity`; `_value_str` is the value with the sign already removed; `_inf_str` is `pos`, `neg` or `none`; `_tall_str` is `dfrac`, `frac` or `none`; and `_composite_str` is `true` when the value is a sole subscripted, superscripted or radical node. The variables `_integer_str` and `_short_str` are `true` when the value is an integer and when it is an integer of at most three digits, respectively; `_braced_str` holds the value with each comma protected by braces.

```
\clist_map_inline:nn { a , b }
  {
    \str_new:c { l__spintent_spinterval_luaset_ #1 _sign_str      }
    \str_new:c { l__spintent_spinterval_luaset_ #1 _type_str      }
    \str_new:c { l__spintent_spinterval_luaset_ #1 _value_str     }
    \str_new:c { l__spintent_spinterval_luaset_ #1 _inf_str       }
    \str_new:c { l__spintent_spinterval_luaset_ #1 _tall_str      }
    \str_new:c { l__spintent_spinterval_luaset_ #1 _composite_str }
    \str_new:c { l__spintent_spinterval_luaset_ #1 _integer_str   }
    \str_new:c { l__spintent_spinterval_luaset_ #1 _short_str     }
    \str_new:c { l__spintent_spinterval_luaset_ #1 _braced_str    }
  }
```

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

`\l__spintent_spinterval_prefix_tl`
`\l__spintent_spinterval_read_arg_tl`
`\l__spintent_spinterval_mode_str`
`\l__spintent_spinterval_size_str`

The variables `\l__spintent_spinterval_mode_str`, `\l__spintent_spinterval_prefix_tl`, `\l__spintent_spinterval_read_arg_tl` and `\l__spintent_spinterval_size_str` store the values set by the keys `read-cmd`, `prefix`, `read-arg` and `size`.

```
\tl_new:N  \l__spintent_spinterval_prefix_tl
```

```
\tl_new:N  \l__spintent_spinterval_read_arg_tl
\str_new:N \l__spintent_spinterval_mode_str
\str_new:N \l__spintent_spinterval_size_str
```

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

`\l__spintent_spinterval_reason_str`

The variable `\l__spintent_spinterval_reason_str` stores the text of the error reason, set from the error code returned by the Lua module.

```
\str_new:N \l__spintent_spinterval_reason_str
```

(*End of definition for* `\l__spintent_spinterval_reason_str`.)

`\l__spintent_spinterval_word_w_tl`
`\l__spintent_spinterval_word_g_tl`
`\l__spintent_spinterval_intent_arg_tl`
`\l__spintent_spinterval_intent_group_tl`

The variables `\l__spintent_spinterval_word_w_tl` and `\l__spintent_spinterval_word_g_tl` store the word that names the interval and the word that joins its endpoints. The variables `\l__spintent_spinterval_intent_arg_tl` and `\l__spintent_spinterval_intent_group_tl` store the intents of the whole interval and of the group of endpoints.

```
\tl_new:N \l__spintent_spinterval_word_w_tl
\tl_new:N \l__spintent_spinterval_word_g_tl
\tl_new:N \l__spintent_spinterval_intent_arg_tl
\tl_new:N \l__spintent_spinterval_intent_group_tl
```

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

`\l__spintent_spinterval_sym_l_tl`
`\l__spintent_spinterval_sym_r_tl`
`\l__spintent_spinterval_size_l_tl`
`\l__spintent_spinterval_size_r_tl`
`\l__spintent_spinterval_delim_l_tl`
`\l__spintent_spinterval_delim_r_tl`

The variables `\l__spintent_spinterval_sym_l_tl` and `\l__spintent_spinterval_sym_r_tl` store the delimiter symbols, the variables `\l__spintent_spinterval_size_l_tl` and `\l__spintent_spinterval_size` store their size commands, and the variables `\l__spintent_spinterval_delim_l_tl` and `\l__spintent_spinterval_delim_r_tl` store the delimiters ready to be typeset.

```
\tl_new:N \l__spintent_spinterval_sym_l_tl
\tl_new:N \l__spintent_spinterval_sym_r_tl
\tl_new:N \l__spintent_spinterval_size_l_tl
\tl_new:N \l__spintent_spinterval_size_r_tl
\tl_new:N \l__spintent_spinterval_delim_l_tl
\tl_new:N \l__spintent_spinterval_delim_r_tl
```

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

`\l__spintent_spinterval_tmp_tl`

The variable `\l__spintent_spinterval_tmp_tl` is a scratch variable.

```
\tl_new:N \l__spintent_spinterval_tmp_tl
```

(*End of definition for* `\l__spintent_spinterval_tmp_tl`.)

15.38.2 Definition of error messages for `\spinterval`

`spinterval-invalid-arg`

The "error" `spinterval-invalid-arg` is raised when the Lua module rejects the {⟨*argument*⟩}; it reports the argument and the reason set by `\__spintent_spinterval_set_reason:`.

```
\msg_new:nnnn { spintent } { spinterval-invalid-arg }
  {
    Invalid~argument~'#1'~for~'\token_to_str:N \spinterval'~(#2)~
    \msg_line_context:.
  }
  {
    Use~'(a;~b)'~or~'(a,~b)',~with~'(',~'['~or~']'~on~the~left~and~')',~']'~or~'['~
    on~the~right.
  }
```

(*End of definition for* `spinterval-invalid-arg`.)

15.38.3 Definition of keys for `\spinterval`

`read-cmd` `prefix` `read-arg` `size` `silent` `concept` `unknown`

We define the ⟨*keys*⟩ `read-cmd`, `prefix`, `read-arg`, `size`, `silent` and `concept`.

```
\keys_define:nn { spintent / spinterval }
  {
    read-cmd .choices:nn = { school , mathcat }
      {
        \int_case:nn { \l_keys_choice_int }
          {
            {1} { \str_set:Nn \l__spintent_spinterval_mode_str { school  } }
            {2} { \str_set:Nn \l__spintent_spinterval_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_spinterval_prefix_tl {#1} },
    prefix   .initial:n = ,
    prefix   .value_required:n = true,
    read-arg .code:n =
      { \__spintent_sanitize_affix:Nn \l__spintent_spinterval_read_arg_tl {#1} },
    read-arg .initial:n = ,
    read-arg .value_required:n = true,
    size     .choices:nn = { auto , none , big , Big , bigg , Bigg }
      {
        \int_case:nn { \l_keys_choice_int }
          {
            {1} { \str_set:Nn \l__spintent_spinterval_size_str { auto } }
            {2} { \str_set:Nn \l__spintent_spinterval_size_str { none } }
            {3} { \str_set:Nn \l__spintent_spinterval_size_str { big  } }
            {4} { \str_set:Nn \l__spintent_spinterval_size_str { Big  } }
            {5} { \str_set:Nn \l__spintent_spinterval_size_str { bigg } }
            {6} { \str_set:Nn \l__spintent_spinterval_size_str { Bigg } }
          }
      },
    size     .initial:n = auto,
    size     .value_required:n = true,
    size / unknown .code:n =
      \msg_error:nneee { spintent } { unknown-choice }
        { size } { auto,~none,~big,~Big,~bigg,~Bigg } { \exp_not:n {#1} },
    silent   .bool_set:N = \l__spintent_spinterval_silent_bool,
    silent   .initial:n = false,
    silent   .value_required:n = true,
    concept  .bool_set:N = \l__spintent_spinterval_concept_bool,
    concept  .initial:n = true,
    concept  .value_required:n = true,
    unknown  .code:n =
      {
        \str_set:Nn \l__spintent_command_name_str { spinterval }
        \__spintent_unknown_keys_cmd:n {#1}
      },
  }
```

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

15.38.4 Internal functions for `\spinterval`

`\__spintent_spinterval_set_reason:`

The internal function `\__spintent_spinterval_set_reason:` stores in `\l__spintent_spinterval_reason_str` the text that describes the error code returned by the Lua module, or `invalid argument` when the code is not known.

```
\cs_new_protected:Nn \__spintent_spinterval_set_reason:
  {
    \str_case:VnF \l__spintent_spinterval_luaset_error_code_str
      {
        { delim }
          {
            \str_set:Nn \l__spintent_spinterval_reason_str
              { it~must~start~with~'(',~'['~or~']'~and~end~with~')',~']'~or~'[' }
          }
        { vacio }
          { \str_set:Nn \l__spintent_spinterval_reason_str { an~endpoint~is~empty } }
        { sep }
          {
            \str_set:Nn \l__spintent_spinterval_reason_str
              { there~is~no~separator:~use~a~comma~or~a~semicolon }
          }
        { ambiguo }
          {
            \str_set:Nn \l__spintent_spinterval_reason_str
              {
```

```
                more~than~one~separator:~use~a~single~semicolon~and~the~commas~are~
                  decimal
              }
          }
        { llaves }
          { \str_set:Nn \l__spintent_spinterval_reason_str { unbalanced~braces } }
        { inf-orden }
          {
            \str_set:Nn \l__spintent_spinterval_reason_str
              { minus~infinity~goes~on~the~left~and~infinity~on~the~right }
          }
        { inf-cerrado }
          {
            \str_set:Nn \l__spintent_spinterval_reason_str
              {
                an~infinite~endpoint~cannot~be~included:~use~'('~or~']'~on~the~left,~
                ')'~or~'['~on~the~right
              }
          }
      }
      { \str_set:Nn \l__spintent_spinterval_reason_str { invalid~argument } }
  }
```

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

`\__spintent_spinterval_set_delims:`

The internal function `\__spintent_spinterval_set_delims:` sets the symbols of the left and right delimiters from the characters returned by the Lua module, and their size commands from the key `size`. With `big`, `Big`, `bigg` and `Bigg` the size is fixed, and with `none` there is no size command. With `auto` there is none unless an endpoint contains a fraction, in which case it is `\Bigl`/`\Bigr` in display style and `\bigl`/`\bigr` in text style, and always `\Bigl`/`\Bigr` when the fraction is a `\dfrac`. Without a size command each delimiter is wrapped in `\mathopen` or `\mathclose`; with one, the delimiter follows the size command.

```
\cs_new_protected:Nn \__spintent_spinterval_set_delims:
  {
    \str_case:Vn \l__spintent_spinterval_luaset_left_sym_str
      {
        { ( } { \tl_set:Nn \l__spintent_spinterval_sym_l_tl { \lparen } }
        { [ } { \tl_set:Nn \l__spintent_spinterval_sym_l_tl { \lbrack } }
        { ] } { \tl_set:Nn \l__spintent_spinterval_sym_l_tl { \rbrack } }
      }
    \str_case:Vn \l__spintent_spinterval_luaset_right_sym_str
      {
        { ) } { \tl_set:Nn \l__spintent_spinterval_sym_r_tl { \rparen } }
        { [ } { \tl_set:Nn \l__spintent_spinterval_sym_r_tl { \lbrack } }
        { ] } { \tl_set:Nn \l__spintent_spinterval_sym_r_tl { \rbrack } }
      }
    \tl_clear:N \l__spintent_spinterval_size_l_tl
    \tl_clear:N \l__spintent_spinterval_size_r_tl
    \str_case:Vn \l__spintent_spinterval_size_str
      {
        { big }
          {
            \tl_set:Nn \l__spintent_spinterval_size_l_tl { \bigl }
            \tl_set:Nn \l__spintent_spinterval_size_r_tl { \bigr }
          }
        { Big }
          {
            \tl_set:Nn \l__spintent_spinterval_size_l_tl { \Bigl }
            \tl_set:Nn \l__spintent_spinterval_size_r_tl { \Bigr }
          }
        { bigg }
          {
            \tl_set:Nn \l__spintent_spinterval_size_l_tl { \biggl }
            \tl_set:Nn \l__spintent_spinterval_size_r_tl { \biggr }
          }
        { Bigg }
          {
            \tl_set:Nn \l__spintent_spinterval_size_l_tl { \Biggl }
            \tl_set:Nn \l__spintent_spinterval_size_r_tl { \Biggr }
          }
        { auto }
          {
```

©2026 by Pablo González L

```
            \str_if_eq:VnF \l__spintent_spinterval_luaset_tall_str { none }
              {
                \int_case:nnF { \mathstyle }
                  {
                    { 0 }
                      {
                        \tl_set:Nn \l__spintent_spinterval_size_l_tl { \Bigl }
                        \tl_set:Nn \l__spintent_spinterval_size_r_tl { \Bigr }
                      }
                    { 1 }
                      {
                        \tl_set:Nn \l__spintent_spinterval_size_l_tl { \Bigl }
                        \tl_set:Nn \l__spintent_spinterval_size_r_tl { \Bigr }
                      }
                    { 2 }
                      {
                        \tl_set:Nn \l__spintent_spinterval_size_l_tl { \bigl }
                        \tl_set:Nn \l__spintent_spinterval_size_r_tl { \bigr }
                      }
                    { 3 }
                      {
                        \tl_set:Nn \l__spintent_spinterval_size_l_tl { \bigl }
                        \tl_set:Nn \l__spintent_spinterval_size_r_tl { \bigr }
                      }
                  }
                  { }
                \str_if_eq:VnT \l__spintent_spinterval_luaset_tall_str { dfrac }
                  {
                    \tl_set:Nn \l__spintent_spinterval_size_l_tl { \Bigl }
                    \tl_set:Nn \l__spintent_spinterval_size_r_tl { \Bigr }
                  }
              }
          }
        \tl_if_empty:NTF \l__spintent_spinterval_size_l_tl
          {
            \tl_set:Ne \l__spintent_spinterval_delim_l_tl
              { \exp_not:N \mathopen { \exp_not:V \l__spintent_spinterval_sym_l_tl } }
            \tl_set:Ne \l__spintent_spinterval_delim_r_tl
              { \exp_not:N \mathclose { \exp_not:V \l__spintent_spinterval_sym_r_tl } }
          }
          {
            \tl_set:Ne \l__spintent_spinterval_delim_l_tl
              {
                \exp_not:V \l__spintent_spinterval_size_l_tl \exp_not:V
                  \l__spintent_spinterval_sym_l_tl
              }
            \tl_set:Ne \l__spintent_spinterval_delim_r_tl
              {
                \exp_not:V \l__spintent_spinterval_size_r_tl \exp_not:V
                  \l__spintent_spinterval_sym_r_tl
              }
          }
      }
```

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

`\__spintent_spinterval_set_intents:`

The internal function `\__spintent_spinterval_set_intents:` sets the two words used in the intents from the type returned by the Lua module: `abierto-de` and `a` for an open interval, `cerrado-de` and `a` for a closed one, `abierto-en` and `y-cerrado-en` for an open-closed one, and `cerrado-en` and `y-abierto-en` for a closed-open one. The intent of the group of endpoints is `_<word>:infix($a,$b)` with the second word. The intent of the whole interval is '`_-`' when `silent` is `true`, `_` followed by the value of `read-arg` when it is not empty, and otherwise `_<word>:prefix($ab)` with the first word, preceded by the `prefix`, if any, and `intervalo-` when `concept` is `true`.

```
\cs_new_protected:Nn \__spintent_spinterval_set_intents:
  {
    \str_case:Vn \l__spintent_spinterval_luaset_type_str
      {
        { open }
          {
```

```
            \tl_set:Nn \l__spintent_spinterval_word_w_tl { abierto-de }
            \tl_set:Nn \l__spintent_spinterval_word_g_tl { a }
          }
        { closed }
          {
            \tl_set:Nn \l__spintent_spinterval_word_w_tl { cerrado-de }
            \tl_set:Nn \l__spintent_spinterval_word_g_tl { a }
          }
        { open-closed }
          {
            \tl_set:Nn \l__spintent_spinterval_word_w_tl { abierto-en }
            \tl_set:Nn \l__spintent_spinterval_word_g_tl { y-cerrado-en }
          }
        { closed-open }
          {
            \tl_set:Nn \l__spintent_spinterval_word_w_tl { cerrado-en }
            \tl_set:Nn \l__spintent_spinterval_word_g_tl { y-abierto-en }
          }
      }
    \tl_set:Ne \l__spintent_spinterval_intent_group_tl
      { _ \l__spintent_spinterval_word_g_tl :infix($a,$b) }
    \bool_if:NTF \l__spintent_spinterval_silent_bool
      { \tl_set:Nn \l__spintent_spinterval_intent_arg_tl { _- } }
      {
        \tl_if_empty:NTF \l__spintent_spinterval_read_arg_tl
          {
            \tl_set:Ne \l__spintent_spinterval_intent_arg_tl
              {
                _
                \bool_if:NT \l__spintent_spinterval_concept_bool
                  {
                    \tl_if_empty:NF \l__spintent_spinterval_prefix_tl
                      { \tl_use:N \l__spintent_spinterval_prefix_tl - }
                    intervalo-
                  }
                \l__spintent_spinterval_word_w_tl :prefix($ab)
              }
          }
          {
            \tl_set:Ne \l__spintent_spinterval_intent_arg_tl
              { _ \l__spintent_spinterval_read_arg_tl }
          }
      }
  }
```

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

`\__spintent_spinterval_inner_sep:n`

The internal function `\__spintent_spinterval_inner_sep:n` calls `\__spintent_inner_sep:` when the endpoint a or b passed in '`#1`' is composite.

```
\cs_new_protected:Nn \__spintent_spinterval_inner_sep:n
  {
    \str_if_eq:eeT
      { \str_use:c { l__spintent_spinterval_luaset_ #1 _composite_str } } { true }
      { \__spintent_inner_sep: }
  }
```

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

`\__spintent_spinterval_plain_number:n`

The internal function `\__spintent_spinterval_plain_number:n` typesets the value of the endpoint passed in '`#1`' as a number, with each comma protected by braces. Unless the value is an integer it is preceded by `\__spintent_inner_sep:`. An integer is preceded by a permanent silent ghost instead, which survives in mathml-AF and keeps MathCAT from merging `1,5` into a single decimal number.

```
\cs_new_protected:Nn \__spintent_spinterval_plain_number:n
  {
    \str_if_eq:eeTF { \str_use:c { l__spintent_spinterval_luaset_ #1 _integer_str } }
      { true }
      % keep "a,b" apart in mathml-AF
      { \MathMLintent { :silent } { \c__spintent_invisible_sep_int } }
      { \__spintent_inner_sep: }
    \str_use:c { l__spintent_spinterval_luaset_ #1 _braced_str }
  }
```

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

`\__spintent_spinterval_value:n`

The internal function `\__spintent_spinterval_value:n` typesets the value of the endpoint passed in '`#1`' according to its type: `\infty` for infinity; for a number, `\__spintent_inner_sep:` followed by `\spnum` when `read-cmd` is `school` and the value is not an integer of at most three digits, or `\__spintent_spinterval_plain_number:n` otherwise, since `\spnum` reads such an integer as MathCAT does; and the value as written for any other type.

```
\cs_new_protected:Nn \__spintent_spinterval_value:n
  {
    \str_case:enF { \str_use:c { l__spintent_spinterval_luaset_ #1 _type_str } }
      {
        { infinity } { \infty }
        { number }
          {
            \str_if_eq:VnTF \l__spintent_spinterval_mode_str { school }
              {
                \tl_set:Ne \l__spintent_spinterval_tmp_tl
                  { \str_use:c { l__spintent_spinterval_luaset_ #1 _value_str } }
                \str_if_eq:eeTF
                  { \str_use:c { l__spintent_spinterval_luaset_ #1 _short_str } }
                  { true }
                  { \__spintent_spinterval_plain_number:n {#1} }
                  {
                    \__spintent_inner_sep:
                    \exp_args:NV \spnum \l__spintent_spinterval_tmp_tl
                  }
              }
              { \__spintent_spinterval_plain_number:n {#1} }
          }
      }
      { \str_use:c { l__spintent_spinterval_luaset_ #1 _value_str } }
  }
```

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

`\__spintent_spinterval_print:n`

The internal function `\__spintent_spinterval_print:n` typesets the endpoint passed in '`#1`' with its sign. With `read-cmd=school` a sign + or - is wrapped, with its value, in a native `\MathMLintent` with the intent `_más:prefix($vp)` or `_menos:prefix($vn)`, the value tagged with `\MathMLarg` and preceded by `\__spintent_spinterval_inner_sep:n`; an endpoint without sign is typeset through `\__spintent_spinterval_value:n`. With `read-cmd=mathcat` the sign is typeset as is, followed by the value.

```
\cs_new_protected:Nn \__spintent_spinterval_print:n
  {
    \str_if_eq:VnTF \l__spintent_spinterval_mode_str { school }
      {
        \str_case:enF { \str_use:c { l__spintent_spinterval_luaset_ #1 _sign_str } }
          {
            { + }
              {
                \MathMLintent { _más:prefix($vp) }
                  {
                    +
                    \MathMLarg { vp }
                      {
                        \__spintent_spinterval_inner_sep:n {#1}
                          \__spintent_spinterval_value:n {#1}
                      }
                  }
              }
            { - }
              {
                \MathMLintent { _menos:prefix($vn) }
                  {
                    -
                    \MathMLarg { vn }
                      {
                        \__spintent_spinterval_inner_sep:n {#1}
                          \__spintent_spinterval_value:n {#1}
                      }
```

```
                    }
                }
            }
            { \__spintent_spinterval_value:n {#1} }
        }
        {
          \str_use:c { l__spintent_spinterval_luaset_ #1 _sign_str }
          \__spintent_spinterval_value:n {#1}
        }
    }
```

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

`\__spintent_spinterval_mathml:`

The internal function `\__spintent_spinterval_mathml:` builds the annotated interval with a native `\MathMLintent` that carries the intent of the whole interval. It contains the left delimiter silenced, a `\MathMLarg` named ab that wraps `\__spintent_inner_sep:` and a `\MathMLintent` with the intent of the group of endpoints, and the right delimiter silenced. The group holds the endpoints a and b, each tagged with `\MathMLarg` and typeset by `\__spintent_spinterval_print:n`, separated by the silenced separator.

```
\cs_new_protected:Nn \__spintent_spinterval_mathml:
  {
    \MathMLintent { \l__spintent_spinterval_intent_arg_tl }
      {
        \MathMLintent { :silent } { \l__spintent_spinterval_delim_l_tl }
        \MathMLarg { ab }
          {
            \__spintent_inner_sep:
            \MathMLintent { \l__spintent_spinterval_intent_group_tl }
              {
                \MathMLarg { a }
                  {
                    \__spintent_spinterval_inner_sep:n { a }
                      \__spintent_spinterval_print:n { a }
                  }
                \MathMLintent { :silent } { \l__spintent_spinterval_luaset_sep_str }
                \MathMLarg { b }
                  {
                    \__spintent_spinterval_inner_sep:n { b }
                      \__spintent_spinterval_print:n { b }
                  }
              }
          }
        \MathMLintent { :silent } { \l__spintent_spinterval_delim_r_tl }
      }
  }
```

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

`\__spintent_spinterval_exec:nn`

The internal function `\__spintent_spinterval_exec:nn` receives the optional ⟨*keys*⟩ passed in '`#1`' and the {⟨*argument*⟩} passed in '`#2`'. It calls `\__spintent_luafun_spinterval_parse_and_set:n`; if the Lua module reports an error, it reports `spinterval-invalid-arg` whatever the value of `silent`: an invalid {⟨*argument*⟩} is always rejected. When there is no error, it calls `\__spintent_spinterval_set_delims:` and `\__spintent_spinterval_set_intents:`, places `\__spintent_invisible_sep:` and delegates the rendering to `\__spintent_spinterval_mathml:`.

```
\cs_new_protected:Nn \__spintent_spinterval_exec:nn
  {
    \keys_set:nn { spintent / spinterval } {#1}
    \__spintent_luafun_spinterval_parse_and_set:n { \exp_not:n {#2} }
    \str_if_eq:VnTF \l__spintent_spinterval_luaset_error_str { true }
      {
        \__spintent_spinterval_set_reason:
        \msg_error:nnee { spintent } { spinterval-invalid-arg }
          { \exp_not:n {#2} } { \l__spintent_spinterval_reason_str }
      }
      {
        \__spintent_spinterval_set_delims:
        \__spintent_spinterval_set_intents:
        \__spintent_invisible_sep:
        \__spintent_spinterval_mathml:
      }
  }
```

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

`\spinterval` The public command `\spinterval` enforces *math mode*, opens a local group, and delegates to `\__spintent_spinterval_exec:nn`.

```
\NewDocumentCommand \spinterval { O{} m }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \attribute \l__spintent_scope_attr_int = 1
          \__spintent_spinterval_exec:nn {#1} {#2}
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spinterval } }
  }
```

(*End of definition for* `\spinterval`. *This function is documented on page 39.*)

15.39 The command \spfun

The user command `\spfun` typesets a function call such as `f(x)`, `f_{1}(x)`, `f''(x,y)`. The Lua function `\__spintent_luafun_spfun_parse_and_set:n` splits the {⟨*argument*⟩} into the function name, which is a letter with an optional subscript or primes left in the shared `\l__spintent_geo_luaset_*` variables, and the list of arguments, cut at the top-level commas, each one stored in `\l__spintent_spfun_luaset_arg_<i>_tl` together with a flag `\l__spintent_spfun_luaset_simple_<i>_str` telling whether it is a single node. The whole call is tagged with a native `\MathMLintent` whose `intent` is `$f($a1,$a2,...)`; the name is tagged `arg="f"` and each argument `arg="a<i>"`, while the parentheses and the commas are silenced. The name may carry the inverse exponent `^{-1}`, which sets the flag `\l__spintent_spfun_luaset_inv_str`; in that case the literal `intent` of the name ends in `-inversa`, the exponent is typeset silenced and the parentheses may be omitted, with no arguments and `$f` as the whole `intent`. An argument that is itself a call, `f(g(x))` or `f(g^{-1}(y))`, is analysed recursively by Lua, kind `call`, and typeset by `\__spintent_spfun_nested:n`, which runs the same code in a group, one level deeper; the `arg` names carry the depth.

15.39.1 Definition of variables for \spfun

`\l__spintent_spfun_luaset_error_str` `\l__spintent_spfun_luaset_nargs_str` `\l__spintent_spfun_luaset_inv_str` These variables are set by the Lua module `spintent.lua`. The variable `\l__spintent_spfun_luaset_error_str` stores `true` or `false` if the {⟨*argument*⟩} is invalid. The variable `\l__spintent_spfun_luaset_nargs_str` stores the number of arguments of the function and `\l__spintent_spfun_luaset_inv_str` stores `true` when the name carries the inverse exponent `^{-1}`.

```
\str_new:N \l__spintent_spfun_luaset_error_str
\str_new:N \l__spintent_spfun_luaset_nargs_str
\str_new:N \l__spintent_spfun_luaset_inv_str
```

(*End of definition for* `\l__spintent_spfun_luaset_error_str`, `\l__spintent_spfun_luaset_nargs_str`, *and* `\l__spintent_spfun_luaset_inv_str`.)

`\l__spintent_spfun_mode_str` `\l__spintent_spfun_prefix_tl` `\l__spintent_spfun_silent_bool` `\l__spintent_spfun_concept_bool` The variables `\l__spintent_spfun_mode_str`, `\l__spintent_spfun_prefix_tl`, `\l__spintent_spfun_siler` and `\l__spintent_spfun_concept_bool` store the values set by the keys `read-cmd`, `prefix`, `silent` and `concept`. The last two are created by the keys.

```
\str_new:N \l__spintent_spfun_mode_str
\tl_new:N  \l__spintent_spfun_prefix_tl
```

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

`\l__spintent_spfun_nargs_int` `\l__spintent_spfun_depth_int` `\l__spintent_spfun_tmp_str` The variable `\l__spintent_spfun_nargs_int` stores the number of arguments of the function, copied from `\l__spintent_spfun_luaset_nargs_str`. The variable `\l__spintent_spfun_depth_int` stores the nesting level of `\__spintent_spfun_nested:n`. The variable `\l__spintent_spfun_tmp_str` is a scratch variable that holds the value Lua left for the argument being typeset.

```
\int_new:N \l__spintent_spfun_nargs_int
\int_new:N \l__spintent_spfun_depth_int
\str_new:N \l__spintent_spfun_tmp_str
```

(*End of definition for* `\l__spintent_spfun_nargs_int`, `\l__spintent_spfun_depth_int`, *and* `\l__spintent_spfun_tmp_str`.)

`\l__spintent_spfun_refs_tl` `\l__spintent_spfun_call_tl` `\l__spintent_spfun_intent_arg_tl` `\l__spintent_spfun_kv_tl` The variable `\l__spintent_spfun_refs_tl` stores the list of references to the arguments and `\l__spintent_spfun_call_tl` stores the `$f($a1,...)` text of the `intent`. The variable `\l__spintent_spfun_intent_arg_tl` stores the `intent` of the whole call. The variable

`\l__spintent_spfun_kv_tl` stores the key-value list with the attributes `arg` and `intent` of the name. The variables `\l__spintent_spfun_fname_tl` and `\l__spintent_spfun_aname_tl` hold the names of the `arg` attributes built by `\__spintent_spfun_fname:` and `\__spintent_spfun_aname:n`.

```
\tl_new:N \l__spintent_spfun_refs_tl
\tl_new:N \l__spintent_spfun_call_tl
\tl_new:N \l__spintent_spfun_intent_arg_tl
\tl_new:N \l__spintent_spfun_kv_tl
\tl_new:N \l__spintent_spfun_fname_tl
\tl_new:N \l__spintent_spfun_aname_tl
```

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

15.39.2 Definition of error messages for `\spfun`

spfun-invalid-arg

The "error" `spfun-invalid-arg` is raised when the argument fails to parse: no valid name, no parentheses unless it is the inverse function, a malformed or misplaced inverse exponent, unbalanced delimiters, an empty argument or text after the closing parenthesis. It is raised whatever the value of `silent`.

```
\msg_new:nnnn { spintent } { spfun-invalid-arg }
  {
    Invalid~argument~'#1'~for~'\token_to_str:N \spfun'~\msg_line_context:.
  }
  {
    The~argument~must~have~the~form~'f(x)'~or~'f(x,y,...)',~where~'f'~is~a~letter~with~
    optional~primes~or~subscript.
  }
```

(*End of definition for* `spfun-invalid-arg`.)

15.39.3 Definition of keys for `\spfun`

read-cmd
prefix
concept
silent
unknown

We define the ⟨*keys*⟩ `read-cmd`, `prefix`, `concept` and `silent`.

```
\keys_define:nn { spintent / spfun }
  {
    read-cmd .choices:nn = { school , mathcat }
      {
        \int_case:nn { \l_keys_choice_int }
          {
            {1} { \str_set:Nn \l__spintent_spfun_mode_str { school  } }
            {2} { \str_set:Nn \l__spintent_spfun_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_spfun_prefix_tl {#1} },
    prefix   .initial:n = ,
    prefix   .value_required:n = true,
    silent   .bool_set:N = \l__spintent_spfun_silent_bool,
    silent   .initial:n = false,
    silent   .value_required:n = true,
    concept  .bool_set:N = \l__spintent_spfun_concept_bool,
    concept  .initial:n = false,
    concept  .value_required:n = true,
    unknown  .code:n =
      {
        \str_set:Nn \l__spintent_command_name_str { spfun }
        \__spintent_unknown_keys_cmd:n {#1}
      },
  }
```

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

15.39.4 Internal functions for `\spfun`

__spintent_spfun_fname:
__spintent_spfun_aname:n

Names of the `arg` attributes at the current nesting depth `\l__spintent_spfun_depth_int`. At depth 1 they are `f` and `a<i>`; at a deeper level, `f<d>` and `a<d>x<i>`. They are stored in `\l__spintent_spfun_fname_tl` and `\l__spintent_spfun_aname_tl`.

```
\cs_new_protected:Nn \__spintent_spfun_fname:
```

```
  {
    \tl_set:Ne \l__spintent_spfun_fname_tl
      {
        f
        \int_compare:nNnF { \l__spintent_spfun_depth_int } = { 1 }
          { \int_use:N \l__spintent_spfun_depth_int }
      }
  }
\cs_new_protected:Nn \__spintent_spfun_aname:n
  {
    \tl_set:Ne \l__spintent_spfun_aname_tl
      {
        a
        \int_compare:nNnF { \l__spintent_spfun_depth_int } = { 1 }
          { \int_use:N \l__spintent_spfun_depth_int x }
        #1
      }
  }
```

(*End of definition for* `\__spintent_spfun_fname:` *and* `\__spintent_spfun_aname:n`.)

`\__spintent_spfun_name_plain:` Typesets the name without any annotation: the letter, followed by the subscript or by the primes when there are any.

```
\cs_new_protected:Nn \__spintent_spfun_name_plain:
  {
    \l__spintent_geo_luaset_letra_str
    \str_case:VnF \l__spintent_geo_luaset_tipo_str
      {
        { sub }   { \c_math_subscript_token { \l__spintent_geo_luaset_sub_str } }
        { prima } { \l__spintent_geo_luaset_sub_str }
      }
      { }
    \str_if_eq:VnT \l__spintent_spfun_luaset_inv_str { true }
      { \c_math_superscript_token { -1 } }
  }
```

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

`\__spintent_spfun_name_school:` For `read-cmd=school` the name is tagged `arg="f"` with a literal `intent` such as `_f-sub-uno-` or `_F-doble-prima-` through the shared queue, so `MathCAT` neither announces a capital letter nor reads the Unicode double prime as seconds. A subscript is typeset silenced; a prime is not, since it is not spoken on the base. The inverse exponent `^{-1}`, when present, is typeset silenced too: the literal `intent` `_f-inversa-` already says everything.

```
\cs_new_protected:Nn \__spintent_spfun_name_school:
  {
    \__spintent_spfun_fname:
    \tl_set:Ne \l__spintent_spfun_kv_tl
      {
        arg = " \l__spintent_spfun_fname_tl ", intent =
          "_ \l__spintent_geo_luaset_intent_str -"
      }
    \__spintent_luafun_mrow_intent_arg:nn { } { \l__spintent_spfun_kv_tl }
    \str_case:VnF \l__spintent_geo_luaset_tipo_str
      {
        { sub }
          {
            \l__spintent_geo_luaset_letra_str
            \c_math_subscript_token
              { \MathMLintent { :silent } { \l__spintent_geo_luaset_sub_str } }
          }
        { prima }
          { \l__spintent_geo_luaset_letra_str \l__spintent_geo_luaset_sub_str }
      }
      { \l__spintent_geo_luaset_letra_str }
    \str_if_eq:VnT \l__spintent_spfun_luaset_inv_str { true }
      {
        \c_math_superscript_token
          { \MathMLintent { :silent } { -1 } }
      }
  }
```

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

`\__spintent_spfun_name_mathcat:` For `read-cmd=mathcat` the name is tagged `arg="f"` with a native `\MathMLarg`, without using the shared queue. With a subscript or primes, the inner separator keeps the `arg` on a wrapping `<mrow>`.

```
\cs_new_protected:Nn \__spintent_spfun_name_mathcat:
  {
    \__spintent_spfun_fname:
    \bool_lazy_and:nnTF
      { \str_if_eq_p:Vn \l__spintent_geo_luaset_tipo_str { } }
      { \str_if_eq_p:Vn \l__spintent_spfun_luaset_inv_str { false } }
      { \MathMLarg { \l__spintent_spfun_fname_tl } { \__spintent_spfun_name_plain: } }
      {
        \MathMLarg { \l__spintent_spfun_fname_tl }
          { \__spintent_inner_sep: \__spintent_spfun_name_plain: }
      }
  }
```

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

`\__spintent_spfun_arg:n` Tags argument number `#1` with `arg="a<i>"`, named by `\__spintent_spfun_aname:n`. A single node, which is a letter, a number or a command such as `\alpha`, carries its `arg` directly; any other argument starts with the inner separator, which forces an `<mrow>` to hold the `arg`. An argument that is itself a call, such as `g(x)` or `g^{-1}(y)`, is typeset by `\__spintent_spfun_nested:n`.

```
\cs_new_protected:Nn \__spintent_spfun_arg:n
  {
    \__spintent_spfun_aname:n {#1}
    \str_set_eq:Nc \l__spintent_spfun_tmp_str
      { l__spintent_spfun_luaset_simple_ #1 _str }
    \str_case:VnF \l__spintent_spfun_tmp_str
      {
        { true }
          {
            \MathMLarg { \l__spintent_spfun_aname_tl }
              { \tl_use:c { l__spintent_spfun_luaset_arg_ #1 _tl } }
          }
        { call }
          {
            \MathMLarg { \l__spintent_spfun_aname_tl }
              { \__spintent_inner_sep: \__spintent_spfun_nested:n {#1} }
          }
      }
      {
        \MathMLarg { \l__spintent_spfun_aname_tl }
          { \__spintent_inner_sep: \tl_use:c { l__spintent_spfun_luaset_arg_ #1 _tl } }
      }
  }
```

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

`\__spintent_spfun_args:` Walks the arguments, silencing the comma between them.

```
\cs_new_protected:Nn \__spintent_spfun_args:
  {
    \int_step_inline:nn { \l__spintent_spfun_nargs_int }
      {
        \int_compare:nNnF { ##1 } = { 1 }
          { \MathMLintent { :silent } { , } }
        \__spintent_spfun_arg:n {##1}
      }
  }
```

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

`\__spintent_spfun_args_plain:` Typesets the arguments with no annotation for `silent`.

```
\cs_new_protected:Nn \__spintent_spfun_args_plain:
  {
    \int_step_inline:nn { \l__spintent_spfun_nargs_int }
      {
        \int_compare:nNnF { ##1 } = { 1 } { , }
        \tl_use:c { l__spintent_spfun_luaset_arg_ ##1 _tl }
      }
  }
```

©2026 by Pablo González L

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

`\__spintent_spfun_set_intent:` Builds the concept `intent` in `\l__spintent_spfun_intent_arg_tl`: `$f($a1,$a2,...)` when `concept` is `false`, and `_prefix-función:prefix($f($a1,$a2,...))` otherwise. Without arguments, as for the inverse function written bare, the call is just `$f`.

```
\cs_new_protected:Nn \__spintent_spfun_set_intent:
  {
    \tl_clear:N \l__spintent_spfun_refs_tl
    \int_step_inline:nn { \l__spintent_spfun_nargs_int }
      {
        \__spintent_spfun_aname:n {##1}
        \tl_put_right:Ne \l__spintent_spfun_refs_tl
          { \int_compare:nNnF { ##1 } = { 1 } { , } $ \l__spintent_spfun_aname_tl }
      }
    \__spintent_spfun_fname:
    \int_compare:nNnTF { \l__spintent_spfun_nargs_int } = { 0 }
      { \tl_set:Ne \l__spintent_spfun_call_tl { $ \l__spintent_spfun_fname_tl } }
      {
        \tl_set:Ne \l__spintent_spfun_call_tl
          { $ \l__spintent_spfun_fname_tl ( \l__spintent_spfun_refs_tl ) }
      }
    \bool_if:NTF \l__spintent_spfun_concept_bool
      {
        \tl_set:Ne \l__spintent_spfun_intent_arg_tl
          {
            _
            \tl_if_empty:NF \l__spintent_spfun_prefix_tl
              { \tl_use:N \l__spintent_spfun_prefix_tl - }
            función:prefix( \l__spintent_spfun_call_tl )
          }
      }
      {
        \tl_set_eq:NN \l__spintent_spfun_intent_arg_tl \l__spintent_spfun_call_tl
      }
  }
```

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

`\__spintent_spfun_mathml:` Calls `\__spintent_invisible_sep:` first, then wraps name, parentheses and arguments in a native `\MathMLintent`. With `silent` set to `true` the intent is '_-' and nothing inside is annotated.

```
\cs_new_protected:Nn \__spintent_spfun_mathml:
  {
    \__spintent_invisible_sep:
    \__spintent_spfun_body:
  }
```

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

`\__spintent_spfun_body:` The call itself, without the outer separator: used by the top-level call and by every call nested in an argument.

```
\cs_new_protected:Nn \__spintent_spfun_body:
  {
    \bool_if:NTF \l__spintent_spfun_silent_bool
      {
        \MathMLintent { _- }
          {
            \__spintent_inner_sep:
            \__spintent_spfun_name_plain:
            \int_compare:nNnF { \l__spintent_spfun_nargs_int } = { 0 }
              {
                \lparen
                \__spintent_spfun_args_plain:
                \rparen
              }
          }
      }
      {
        \__spintent_spfun_set_intent:
        \MathMLintent { \l__spintent_spfun_intent_arg_tl }
          {
```

```
            \int_compare:nNnT { \l__spintent_spfun_nargs_int } = { 0 }
              { \__spintent_inner_sep: }
            \str_if_eq:VnTF \l__spintent_spfun_mode_str { school }
              { \__spintent_spfun_name_school: }
              { \__spintent_spfun_name_mathcat: }
            \int_compare:nNnF { \l__spintent_spfun_nargs_int } = { 0 }
              {
                \MathMLintent { :silent } { \lparen }
                \__spintent_spfun_args:
                \MathMLintent { :silent } { \rparen }
              }
          }
      }
  }
```

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

`\__spintent_spfun_nested:n` Typesets argument number `#1` as a call of its own. It opens a group, so the values that Lua leaves in the variables for the nested call, namely the name, the arguments and their number, are discarded when it ends. The nested call raises the depth, does not apply `concept` and goes through `\__spintent_spfun_body:` with no outer separator. It cannot fail to parse, since the first analysis already validated every level.

```
\cs_new_protected:Nn \__spintent_spfun_nested:n
  {
    \group_begin:
      \__spintent_luafun_spfun_parse_and_set:v { l__spintent_spfun_luaset_arg_ #1 _tl }
      \int_incr:N \l__spintent_spfun_depth_int
      \bool_set_false:N \l__spintent_spfun_concept_bool
      \int_set:NV \l__spintent_spfun_nargs_int \l__spintent_spfun_luaset_nargs_str
      \__spintent_spfun_body:
    \group_end:
  }
```

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

`\__spintent_spfun_exec:nn` Receives the optional ⟨*keys*⟩ in '`#1`' and the {⟨*argument*⟩} in '`#2`'. Parses first, reporting `spfun-invalid-arg` on error whatever `silent` is.

```
\cs_new_protected:Nn \__spintent_spfun_exec:nn
  {
    \keys_set:nn { spintent / spfun } {#1}
    \__spintent_luafun_spfun_parse_and_set:n { \exp_not:n {#2} }
    \str_if_eq:VnTF \l__spintent_spfun_luaset_error_str { true }
      {
        \msg_error:nne { spintent } { spfun-invalid-arg } { \exp_not:n {#2} }
      }
      {
        \int_set:NV \l__spintent_spfun_nargs_int \l__spintent_spfun_luaset_nargs_str
        \int_set:Nn \l__spintent_spfun_depth_int { 1 }
        \__spintent_spfun_mathml:
      }
  }
```

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

`\spfun` The public command `\spfun` enforces *math mode*, opens a local group, and delegates to `\__spintent_spfun_exec:nn`.

```
\NewDocumentCommand \spfun { O{} m }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \attribute \l__spintent_scope_attr_int = 1
          \__spintent_spfun_exec:nn {#1} {#2}
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spfun } }
  }
```

(*End of definition for* `\spfun`. *This function is documented on page* 40.)

15.40 The command \spfuncomp

The user command \spfuncomp typesets the linear composition of functions: f\circ g, f\circ g\circ h, (f\circ g) and the composition applied to arguments, (f\circ g)(x). The Lua function __spintent_luafun_spfuncomp_parse_and_set:n splits the {⟨*argument*⟩} at the top-level \circ, or the Unicode character U+2218; every function is a name as in \spfun and gets its literal intent, so MathCAT neither announces a capital letter nor reads the letter h, U+210E, as Planck's constant. The arguments of the application, if any, are left in the \spfun variables and typeset with its own functions, so an argument may be a call. The whole is tagged with a native \MathMLintent whose intent is _compuesta-con:infix($k1,$k2,...); the application adds $c($a1,...) around it, with the composition tagged arg="c". With two or more arguments they are grouped under their own intent, _coma:infix($a1,$a2,...), tagged arg="l", because MathCAT drops the commas of a call whose head is not a single letter. The \circ and the parentheses are silenced.

15.40.1 Definition of variables for \spfuncomp

\l__spintent_spfuncomp_luaset_error_str
\l__spintent_spfuncomp_luaset_n_str
\l__spintent_spfuncomp_luaset_paren_str
\l__spintent_spfuncomp_luaset_apply_str

These variables are set by the Lua module spintent.lua. The variable \l__spintent_spfuncomp_luaset_error_s stores true or false if the {⟨*argument*⟩} is invalid. The variable \l__spintent_spfuncomp_luaset_n_str stores the number of functions. The variables \l__spintent_spfuncomp_luaset_paren_str and \l__spintent_spfuncomp_luaset_apply_str store true when the composition is in parentheses and when it is applied to arguments.

```
\str_new:N \l__spintent_spfuncomp_luaset_error_str
\str_new:N \l__spintent_spfuncomp_luaset_n_str
\str_new:N \l__spintent_spfuncomp_luaset_paren_str
\str_new:N \l__spintent_spfuncomp_luaset_apply_str
```

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

\l__spintent_spfuncomp_n_int
\l__spintent_spfuncomp_refs_tl
\l__spintent_spfuncomp_intent_tl
\l__spintent_spfuncomp_kv_tl
\l__spintent_spfuncomp_tipo_str

The variable \l__spintent_spfuncomp_n_int stores the number of functions, copied from \l__spintent_spfuncomp_luaset_n_str. The variable \l__spintent_spfuncomp_refs_tl stores the references to the functions and \l__spintent_spfuncomp_intent_tl stores the intent being built. The variable \l__spintent_spfuncomp_kv_tl stores the key-value list with the attributes arg and intent of the function being typeset and \l__spintent_spfuncomp_tipo_str stores the kind of its name.

```
\int_new:N \l__spintent_spfuncomp_n_int
\tl_new:N  \l__spintent_spfuncomp_refs_tl
\tl_new:N  \l__spintent_spfuncomp_intent_tl
\tl_new:N  \l__spintent_spfuncomp_kv_tl
\str_new:N \l__spintent_spfuncomp_tipo_str
```

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

15.40.2 Definition of error messages for \spfuncomp

spfuncomp-invalid-arg

The "error" spfuncomp-invalid-arg is raised when the argument is not a composition of at least two function names, optionally in parentheses and optionally applied to arguments written in parentheses, whatever the value of silent.

```
\msg_new:nnnn { spintent } { spfuncomp-invalid-arg }
  {
    Invalid~argument~'#1'~for~'\token_to_str:N \spfuncomp'~\msg_line_context:.
  }
  {
    The~argument~must~be~two~or~more~functions~joined~by~'\token_to_str:N \circ'.
  }
```

(*End of definition for* spfuncomp-invalid-arg.)

15.40.3 Definition of keys for \spfuncomp

silent
unknown

We define the ⟨*key*⟩ silent.

```
\keys_define:nn { spintent / spfuncomp }
  {
    silent  .bool_set:N = \l__spintent_spfuncomp_silent_bool,
    silent  .initial:n = false,
    silent  .value_required:n = true,
    unknown .code:n =
      {
        \str_set:Nn \l__spintent_command_name_str { spfuncomp }
        \__spintent_unknown_keys_cmd:n {#1}
      },
  }
```

(*End of definition for* silent *and* unknown.)

©2026 by Pablo González L

15.40.4 Internal functions for \spfuncomp

__spintent_spfuncomp_fun:n

Function number #1 of the chain. Its first letter is tagged arg="k<i>" with the literal intent of the name through the shared queue. A subscript is typeset silenced; a prime is not; the inverse exponent ^{-1} is silenced as well.

```
\cs_new_protected:Nn \__spintent_spfuncomp_fun:n
  {
    \tl_set:Ne \l__spintent_spfuncomp_kv_tl
      {
        arg = " k #1 ",
        intent = "_ \use:c { l__spintent_spfuncomp_luaset_intent_ #1 _str } -"
      }
    \__spintent_luafun_mrow_intent_arg:nn { } { \l__spintent_spfuncomp_kv_tl }
    \str_set_eq:Nc \l__spintent_spfuncomp_tipo_str
      { l__spintent_spfuncomp_luaset_tipo_ #1 _str }
    \str_case:VnF \l__spintent_spfuncomp_tipo_str
      {
        { sub }
          {
            \use:c { l__spintent_spfuncomp_luaset_letra_ #1 _str }
            \c_math_subscript_token
              {
                \MathMLintent { :silent }
                  { \use:c { l__spintent_spfuncomp_luaset_sub_ #1 _str } }
              }
          }
        { prima }
          {
            \use:c { l__spintent_spfuncomp_luaset_letra_ #1 _str }
            \use:c { l__spintent_spfuncomp_luaset_sub_ #1 _str }
          }
      }
      { \use:c { l__spintent_spfuncomp_luaset_letra_ #1 _str } }
    \str_set_eq:Nc \l__spintent_spfuncomp_tipo_str
      { l__spintent_spfuncomp_luaset_inv_ #1 _str }
    \str_if_eq:VnT \l__spintent_spfuncomp_tipo_str { true }
      {
        \c_math_superscript_token
          { \MathMLintent { :silent } { -1 } }
      }
  }
```

(*End of definition for* __spintent_spfuncomp_fun:n.)

__spintent_spfuncomp_group:

The composition, with its own intent, in parentheses when the argument was written in parentheses.

```
\cs_new_protected:Nn \__spintent_spfuncomp_group:
  {
    \tl_clear:N \l__spintent_spfuncomp_refs_tl
    \int_step_inline:nn { \l__spintent_spfuncomp_n_int }
      {
        \tl_put_right:Ne \l__spintent_spfuncomp_refs_tl
          { \int_compare:nNnF { ##1 } = { 1 } { , } $k ##1 }
      }
    \tl_set:Ne \l__spintent_spfuncomp_intent_tl
      { _compuesta-con:infix( \l__spintent_spfuncomp_refs_tl ) }
    \MathMLintent { \l__spintent_spfuncomp_intent_tl }
      {
        \str_if_eq:VnT \l__spintent_spfuncomp_luaset_paren_str { true }
          { \MathMLintent { :silent } { \lparen } }
        \int_step_inline:nn { \l__spintent_spfuncomp_n_int }
          {
            \int_compare:nNnF { ##1 } = { 1 }
              { \MathMLintent { :silent } { \circ } }
            \__spintent_spfuncomp_fun:n {##1}
          }
        \str_if_eq:VnT \l__spintent_spfuncomp_luaset_paren_str { true }
          { \MathMLintent { :silent } { \rparen } }
      }
  }
```

(*End of definition for* __spintent_spfuncomp_group:.)

©2026 by Pablo González L

__spintent_spfuncomp_plain: Typesets the whole with no annotation for silent.

```
\cs_new_protected:Nn \__spintent_spfuncomp_plain:
  {
    \str_if_eq:VnT \l__spintent_spfuncomp_luaset_paren_str { true } { \lparen }
    \int_step_inline:nn { \l__spintent_spfuncomp_n_int }
      {
        \int_compare:nNnF { ##1 } = { 1 } { \circ }
        \tl_use:c { l__spintent_spfuncomp_luaset_print_ ##1 _tl }
      }
    \str_if_eq:VnT \l__spintent_spfuncomp_luaset_paren_str { true } { \rparen }
    \str_if_eq:VnT \l__spintent_spfuncomp_luaset_apply_str { true }
      {
        \lparen \__spintent_spfun_args_plain: \rparen
      }
  }
```

(*End of definition for* __spintent_spfuncomp_plain:.)

__spintent_spfuncomp_mathml: Calls __spintent_invisible_sep: first. With silent the intent is '_-'. Otherwise an application wraps the composition with arg="c", and the arguments in $c($a1,...); the arguments are processed as in \spfun, including calls nested in them.

```
\cs_new_protected:Nn \__spintent_spfuncomp_mathml:
  {
    \__spintent_invisible_sep:
    \str_if_eq:VnT \l__spintent_spfuncomp_luaset_apply_str { true }
      {
        \int_set:NV \l__spintent_spfun_nargs_int \l__spintent_spfun_luaset_nargs_str
      }
    \int_set:Nn \l__spintent_spfun_depth_int { 1 }
    \str_set:Nn \l__spintent_spfun_mode_str { school }
    \bool_set_false:N \l__spintent_spfun_silent_bool
    \bool_set_false:N \l__spintent_spfun_concept_bool
    \bool_if:NTF \l__spintent_spfuncomp_silent_bool
      {
        \MathMLintent { _- }
          { \__spintent_inner_sep: \__spintent_spfuncomp_plain: }
      }
      {
        \str_if_eq:VnTF \l__spintent_spfuncomp_luaset_apply_str { true }
          {
            \tl_clear:N \l__spintent_spfun_refs_tl
            \int_step_inline:nn { \l__spintent_spfun_nargs_int }
              {
                \__spintent_spfun_aname:n {##1}
                \tl_put_right:Ne \l__spintent_spfun_refs_tl
                  {
                    \int_compare:nNnF { ##1 } = { 1 } { , }
                    $ \l__spintent_spfun_aname_tl
                  }
              }
            \int_compare:nNnTF { \l__spintent_spfun_nargs_int } > { 1 }
              {
                \tl_set:Ne \l__spintent_spfun_call_tl { $c ( $l ) }
                \MathMLintent { \l__spintent_spfun_call_tl }
                  {
                    \MathMLarg { c }
                      { \__spintent_inner_sep: \__spintent_spfuncomp_group: }
                    \MathMLintent { :silent } { \lparen }
                    \MathMLarg { l }
                      {
                        \__spintent_inner_sep:
                        \MathMLintent
                          { _coma:infix ( \l__spintent_spfun_refs_tl ) }
                          { \__spintent_spfun_args: }
                      }
                    \MathMLintent { :silent } { \rparen }
                  }
              }
              {
                \tl_set:Ne \l__spintent_spfun_call_tl
```

```
                  { $c ( \l__spintent_spfun_refs_tl ) }
                \MathMLintent { \l__spintent_spfun_call_tl }
                  {
                    \MathMLarg { c }
                      { \__spintent_inner_sep: \__spintent_spfuncomp_group: }
                    \MathMLintent { :silent } { \lparen }
                    \__spintent_spfun_args:
                    \MathMLintent { :silent } { \rparen }
                  }
              }
          }
          { \__spintent_spfuncomp_group: }
      }
  }
```

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

`\__spintent_spfuncomp_exec:nn` Receives the optional ⟨*keys*⟩ in '`#1`' and the {⟨*argument*⟩} in '`#2`'. Parses first, reporting `spfuncomp-invalid-arg` on error whatever `silent` is.

```
\cs_new_protected:Nn \__spintent_spfuncomp_exec:nn
  {
    \keys_set:nn { spintent / spfuncomp } {#1}
    \__spintent_luafun_spfuncomp_parse_and_set:n { \exp_not:n {#2} }
    \str_if_eq:VnTF \l__spintent_spfuncomp_luaset_error_str { true }
      {
        \msg_error:nne { spintent } { spfuncomp-invalid-arg }
          { \exp_not:n {#2} }
      }
      {
        \int_set:NV \l__spintent_spfuncomp_n_int \l__spintent_spfuncomp_luaset_n_str
        \__spintent_spfuncomp_mathml:
      }
  }
```

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

`\spfuncomp` The public command `\spfuncomp` enforces *math mode*, opens a local group with the scope attribute, for the calls that an argument may contain, and delegates to `\__spintent_spfuncomp_exec:nn`.

```
\NewDocumentCommand \spfuncomp { O{} m }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \attribute \l__spintent_scope_attr_int = 1
          \__spintent_spfuncomp_exec:nn {#1} {#2}
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spfuncomp } }
  }
```

(*End of definition for* `\spfuncomp`. *This function is documented on page* 42.)

15.41 The commands `\spdom`, `\sprec` and `\spcod`

The user commands `\spdom`, `\sprec` and `\spcod` typeset the domain `\operatorname{Dom}(f)`, the range `\operatorname{Rec}(f)` and the codomain `\operatorname{Cod}(f)` of a function. They share one implementation, `\__spintent_spdom_exec:nnnnn`, which receives the command name, the operator, the spoken word, the ⟨*keys*⟩ and the {⟨*argument*⟩}. The Lua function `\__spintent_luafun_spdom_parse_and_set:n` accepts exactly a function name, as `\spfun` does: a letter with an optional subscript or primes, and the optional exponent `^{-1}`. It leaves it in the shared `\l__spintent_geo_luaset_*` variables. The whole is tagged with a native `\MathMLintent` whose `intent` is `_[prefix-]dominio-de:prefix($f)`, with `recorrido` or `codominio` in place of `dominio` for the other two commands; the operator, the parentheses are silenced and the name carries `arg="f"` wrapped in its own literal `intent`, such as `_f-`, `_F-doble-prima-` or `_f-inversa-`, so `MathCAT` neither announces a capital letter nor reads the inverse as a power. No `read-cmd` nor `concept` keys exist: the native reading of `Dom(f)` is not useful, and the concept is already the spoken word.

15.41.1 Definition of variables for `\spdom`

`\l__spintent_spdom_cmd_str` `\l__spintent_spdom_op_tl` `\l__spintent_spdom_word_tl` Each public command of this family sets these variables before calling the common code. The variable `\l__spintent_spdom_cmd_str` stores the name of the command, `\l__spintent_spdom_op_tl` the operator typeset by it and `\l__spintent_spdom_word_tl` the word used in the `intent`.

```
\str_new:N \l__spintent_spdom_cmd_str
```

```
\tl_new:N  \l__spintent_spdom_op_tl
\tl_new:N  \l__spintent_spdom_word_tl
```

(*End of definition for* `\l__spintent_spdom_cmd_str`, `\l__spintent_spdom_op_tl`, *and* `\l__spintent_spdom_word_tl`.)

`\l__spintent_spdom_prefix_tl` `\l__spintent_spdom_intent_arg_tl`

The variable `\l__spintent_spdom_prefix_tl` stores the value set by the key `prefix` and `\l__spintent_spdom_intent_arg_tl` stores the `intent` being built.

```
\tl_new:N \l__spintent_spdom_prefix_tl
\tl_new:N \l__spintent_spdom_intent_arg_tl
```

(*End of definition for* `\l__spintent_spdom_prefix_tl` *and* `\l__spintent_spdom_intent_arg_tl`.)

`\l__spintent_spdom_luaset_error_str`

This variable is set from Lua and stores `true` or `false` if the {⟨*argument*⟩} is not a function name.

```
\str_new:N \l__spintent_spdom_luaset_error_str
```

(*End of definition for* `\l__spintent_spdom_luaset_error_str`.)

15.41.2 Definition of error messages for \spdom

`spdom-invalid-arg`

The "error" `spdom-invalid-arg` is raised when the argument is not a function name. It receives the command name and the argument, and is raised whatever the value of `silent`.

```
\msg_new:nnnn { spintent } { spdom-invalid-arg }
  {
    Invalid~argument~'#2'~for~'\c_backslash_str #1'~\msg_line_context:.
  }
  {
    The~argument~must~be~a~function~name.
  }
```

(*End of definition for* `spdom-invalid-arg`.)

15.41.3 Definition of keys for \spdom

`prefix` `silent` `unknown`

We define the ⟨*keys*⟩ `prefix` and `silent`.

```
\keys_define:nn { spintent / spdom }
  {
    prefix  .code:n =
      { \__spintent_sanitize_affix:Nn \l__spintent_spdom_prefix_tl {#1} },
    prefix  .initial:n = ,
    prefix  .value_required:n = true,
    silent  .bool_set:N = \l__spintent_spdom_silent_bool,
    silent  .initial:n = false,
    silent  .value_required:n = true,
    unknown .code:n =
      {
        \str_set_eq:NN \l__spintent_command_name_str \l__spintent_spdom_cmd_str
        \__spintent_unknown_keys_cmd:n {#1}
      },
  }
```

(*End of definition for* `prefix`, `silent`, *and* `unknown`.)

15.41.4 Internal functions for \spdom

`\__spintent_spdom_name:`

Tags the name with `arg="f"` through a native `\MathMLarg`; inside it, a native `\MathMLintent` gives the name its literal `intent`. Both groups start with the inner separator, so each one gets its own `<mrow>` even when the name is a single node.

```
\cs_new_protected:Nn \__spintent_spdom_name:
  {
    \MathMLarg { f }
      {
        \__spintent_inner_sep:
        \MathMLintent { _ \l__spintent_geo_luaset_intent_str - }
          { \__spintent_inner_sep: \__spintent_spfun_name_plain: }
      }
  }
```

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

©2026 by Pablo González L

__spintent_spdom_mathml: Calls __spintent_invisible_sep: first, then wraps operator, parentheses and name in a native \MathMLintent. With silent set to true the intent is '_-' and nothing inside is annotated.

```
\cs_new_protected:Nn \__spintent_spdom_mathml:
  {
    \__spintent_invisible_sep:
    \bool_if:NTF \l__spintent_spdom_silent_bool
      {
        \MathMLintent { _- }
          {
            \__spintent_inner_sep:
            \operatorname { \l__spintent_spdom_op_tl }
            \lparen \__spintent_spfun_name_plain: \rparen
          }
      }
      {
        \tl_set:Ne \l__spintent_spdom_intent_arg_tl
          {
            _
            \tl_if_empty:NF \l__spintent_spdom_prefix_tl
              { \tl_use:N \l__spintent_spdom_prefix_tl - }
            \l__spintent_spdom_word_tl -de:prefix( $f )
          }
        \MathMLintent { \l__spintent_spdom_intent_arg_tl }
          {
            \MathMLintent { :silent } { \operatorname { \l__spintent_spdom_op_tl } }
            \MathMLintent { :silent } { \lparen }
            \__spintent_spdom_name:
            \MathMLintent { :silent } { \rparen }
          }
      }
  }
```

(*End of definition for* __spintent_spdom_mathml:.)

__spintent_spdom_exec:nnnnn Receives the command name in '#1', the operator in '#2', the spoken word in '#3', the optional ⟨*keys*⟩ in '#4' and the {⟨*argument*⟩} in '#5'. Parses first, reporting spdom-invalid-arg on error whatever silent is.

```
\cs_new_protected:Nn \__spintent_spdom_exec:nnnnn
  {
    \str_set:Nn \l__spintent_spdom_cmd_str {#1}
    \tl_set:Nn  \l__spintent_spdom_op_tl   {#2}
    \tl_set:Nn  \l__spintent_spdom_word_tl {#3}
    \keys_set:nn { spintent / spdom } {#4}
    \__spintent_luafun_spdom_parse_and_set:n { \exp_not:n {#5} }
    \str_if_eq:VnTF \l__spintent_spdom_luaset_error_str { true }
      {
        \msg_error:nnee { spintent } { spdom-invalid-arg }
          {#1} { \exp_not:n {#5} }
      }
      { \__spintent_spdom_mathml: }
  }
```

(*End of definition for* __spintent_spdom_exec:nnnnn.)

\spdom \sprec \spcod The public commands \spdom, \sprec and \spcod enforce *math mode*, open a local group, and delegate to __spintent_spdom_exec:nnnnn.

```
\NewDocumentCommand \spdom { O{} m }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \__spintent_spdom_exec:nnnnn { spdom } { Dom } { dominio } {#1} {#2}
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spdom } }
  }
\NewDocumentCommand \sprec { O{} m }
  {
    \mode_if_math:TF
      {
```

```
      \group_begin:
        \__spintent_spdom_exec:nnnnn { sprec } { Rec } { recorrido } {#1} {#2}
      \group_end:
    }
    { \msg_error:nnn { spintent } { need-math-mode } { \sprec } }
}
\NewDocumentCommand \spcod { O{} m }
  {
    \mode_if_math:TF
      {
      \group_begin:
        \__spintent_spdom_exec:nnnnn { spcod } { Cod } { codominio } {#1} {#2}
      \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spcod } }
  }
```

(*End of definition for* `\spdom`, `\sprec`, *and* `\spcod`. *These functions are documented on page 41.*)

15.42 The command \spabs

The user command `\spabs` typesets the absolute value `\lvert x\rvert` and reads it with the literal intent `valor-absoluto($x)`. Optionally, a second literal intent `fin-valor-absoluto` announces where it ends. It has the keys `silent`, `read-arg`, `size` and `close`.

15.42.1 Definition of variables for \spabs

`\l__spintent_spabs_read_arg_tl` `\l__spintent_spabs_size_str` `\l__spintent_spabs_close_mode_str`

The token list variable `\l__spintent_spabs_read_arg_tl` stores the key `read-arg`, and the string variables `\l__spintent_spabs_size_str` and `\l__spintent_spabs_close_mode_str` the keys `size` and `close`.

```
\tl_new:N  \l__spintent_spabs_read_arg_tl
\str_new:N \l__spintent_spabs_size_str
\str_new:N \l__spintent_spabs_close_mode_str
```

(*End of definition for* `\l__spintent_spabs_read_arg_tl`, `\l__spintent_spabs_size_str`, *and* `\l__spintent_spabs_close_mode_str`.)

`\l__spintent_spabs_delim_l_tl` `\l__spintent_spabs_delim_r_tl`

The opening and closing bars, with their size commands.

```
\tl_new:N \l__spintent_spabs_delim_l_tl
\tl_new:N \l__spintent_spabs_delim_r_tl
```

(*End of definition for* `\l__spintent_spabs_delim_l_tl` *and* `\l__spintent_spabs_delim_r_tl`.)

`\l__spintent_spabs_luaset_close_str` `\l__spintent_spabs_close_bool`

The string variable `\l__spintent_spabs_luaset_close_str` is set by the function `\__spintent_luafun_spabs_` and is `true` when the argument, without spaces and without a leading sign, has more than one token; the boolean variable `\l__spintent_spabs_close_bool` tells whether the closing is read.

```
\str_new:N  \l__spintent_spabs_luaset_close_str
\bool_new:N \l__spintent_spabs_close_bool
```

(*End of definition for* `\l__spintent_spabs_luaset_close_str` *and* `\l__spintent_spabs_close_bool`.)

15.42.2 Definition of error messages for \spabs

`spabs-empty-arg`

The "error" `spabs-empty-arg` is raised when the argument of `\spabs` is missing or *blank*.

```
\msg_new:nnnn { spintent } { spabs-empty-arg }
  {
    Empty~argument~for~'\token_to_str:N \spabs'~\msg_line_context:.
  }
  {
    The~expression~inside~the~absolute~value~is~required.
  }
```

(*End of definition for* `spabs-empty-arg`.)

15.42.3 Definition of keys for \spabs

`silent` `read-arg` `size` `close` `unknown`

We define the ⟨*keys*⟩ `silent`, `read-arg`, `size` and `close`.

```
\keys_define:nn { spintent / spabs }
  {
    silent   .bool_set:N = \l__spintent_spabs_silent_bool,
```

```
    silent   .initial:n = false,
    silent   .value_required:n = true,
    read-arg .code:n =
      { \__spintent_sanitize_affix:Nn \l__spintent_spabs_read_arg_tl {#1} },
    read-arg .initial:n = ,
    read-arg .value_required:n = true,
    size     .choices:nn = { none , big , Big , bigg , Bigg }
      {
        \int_case:nn { \l_keys_choice_int }
          {
            {1} { \str_set:Nn \l__spintent_spabs_size_str { none } }
            {2} { \str_set:Nn \l__spintent_spabs_size_str { big  } }
            {3} { \str_set:Nn \l__spintent_spabs_size_str { Big  } }
            {4} { \str_set:Nn \l__spintent_spabs_size_str { bigg } }
            {5} { \str_set:Nn \l__spintent_spabs_size_str { Bigg } }
          }
      },
    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} },
    close    .choices:nn = { auto , true , false }
      {
        \int_case:nn { \l_keys_choice_int }
          {
            {1} { \str_set:Nn \l__spintent_spabs_close_mode_str { auto  } }
            {2} { \str_set:Nn \l__spintent_spabs_close_mode_str { true  } }
            {3} { \str_set:Nn \l__spintent_spabs_close_mode_str { false } }
          }
      },
    close    .initial:n = auto,
    close    .value_required:n = true,
    close / unknown .code:n =
      \msg_error:nneee { spintent } { unknown-choice }
        { close } { auto,~true,~false } { \exp_not:n {#1} },
    unknown  .code:n =
      {
        \str_set:Nn \l__spintent_command_name_str { spabs }
        \__spintent_unknown_keys_cmd:n {#1}
      },
  }
```

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

15.42.4 Internal functions for `\spabs`

`\__spintent_spabs_set_delims:`

The function `\__spintent_spabs_set_delims:` sets the bars from the key `size`: the bare `\lvert` and `\rvert` for `none`, and `\bigl`, `\Bigl`, `\biggl` or `\Biggl`, and the closing versions, before the bars otherwise.

```
\cs_new_protected:Nn \__spintent_spabs_set_delims:
  {
    \str_case:Vn \l__spintent_spabs_size_str
      {
        { none }
          {
            \tl_set:Nn \l__spintent_spabs_delim_l_tl { \lvert }
            \tl_set:Nn \l__spintent_spabs_delim_r_tl { \rvert }
          }
        { big }
          {
            \tl_set:Nn \l__spintent_spabs_delim_l_tl { \bigl  \lvert }
            \tl_set:Nn \l__spintent_spabs_delim_r_tl { \bigr  \rvert }
          }
        { Big }
          {
            \tl_set:Nn \l__spintent_spabs_delim_l_tl { \Bigl  \lvert }
            \tl_set:Nn \l__spintent_spabs_delim_r_tl { \Bigr  \rvert }
          }
        { bigg }
          {
            \tl_set:Nn \l__spintent_spabs_delim_l_tl { \biggl \lvert }
            \tl_set:Nn \l__spintent_spabs_delim_r_tl { \biggr \rvert }
```

```
4256            }
4257          { Bigg }
4258            {
4259              \tl_set:Nn \l__spintent_spabs_delim_l_tl { \Biggl \lvert }
4260              \tl_set:Nn \l__spintent_spabs_delim_r_tl { \Biggr \rvert }
4261            }
4262        }
4263    }
```

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

`\__spintent_spabs_set_close:n` The function `\__spintent_spabs_set_close:n` sets `\l__spintent_spabs_close_bool` from `close`: always `true` with `true`, always `false` with `false`, and with `auto` `true` when Lua reports that the argument '`#1`', without spaces and without a leading + or -, has more than one token.

```
4264 \cs_new_protected:Nn \__spintent_spabs_set_close:n
4265   {
4266     \bool_set_false:N \l__spintent_spabs_close_bool
4267     \str_case:Vn \l__spintent_spabs_close_mode_str
4268       {
4269         { true } { \bool_set_true:N \l__spintent_spabs_close_bool }
4270         { auto }
4271           {
4272             \__spintent_luafun_spabs_close:n { \exp_not:n {#1} }
4273             \str_if_eq:VnT \l__spintent_spabs_luaset_close_str { true }
4274               { \bool_set_true:N \l__spintent_spabs_close_bool }
4275           }
4276       }
4277   }
```

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

`\__spintent_spabs_exec:nn` The function `\__spintent_spabs_exec:nn` sets the ⟨*keys*⟩ in '`#1`', sets the delimiters with `\__spintent_spabs_set_delims:` and calls `\__spintent_spabs_mathml:n` with the argument in '`#2`'.

```
4278 \cs_new_protected:Nn \__spintent_spabs_exec:nn
4279   {
4280     \keys_set:nn { spintent / spabs } {#1}
4281     \__spintent_spabs_set_delims:
4282     \__spintent_spabs_mathml:n {#2}
4283   }
```

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

`\__spintent_spabs_mathml:n` The function `\__spintent_spabs_mathml:n` prints the absolute value of '`#1`' inside one native `\MathMLintent`, preceded by the outer ghost `\__spintent_invisible_sep:` and followed by the inner ghost `\__spintent_inner_sep:`. With `silent=true` the intent is '`_-`'; with `read-arg` given it is `_`⟨*read-arg*⟩`:nofix`; otherwise it is `_valor-absoluto($x)`. In this last case, when `\__spintent_spabs_set_close:n` sets the closing, a native `\MathMLintent` with the literal intent `_fin-valor-absoluto` follows the bars as a sibling, holding `\__spintent_inner_sep:` and an invisible character, and the outer ghost is omitted because the formula already has two top-level nodes.

```
4284 \cs_new_protected:Nn \__spintent_spabs_mathml:n
4285   {
4286     \bool_if:NTF \l__spintent_spabs_silent_bool
4287       {
4288         \__spintent_invisible_sep:
4289         \MathMLintent { _- }
4290           {
4291             \l__spintent_spabs_delim_l_tl \__spintent_inner_sep: #1
4292               \l__spintent_spabs_delim_r_tl
4293           }
4294       }
4295       {
4296         \tl_if_empty:NTF \l__spintent_spabs_read_arg_tl
4297           {
4298             \__spintent_spabs_set_close:n {#1}
4299             \bool_if:NF \l__spintent_spabs_close_bool { \__spintent_invisible_sep: }
4300             \MathMLintent { _valor-absoluto($x) }
4301               {
4302                 \l__spintent_spabs_delim_l_tl
```

```
                \MathMLarg { x } { \__spintent_inner_sep: #1 }
                \l__spintent_spabs_delim_r_tl
              }
            \bool_if:NT \l__spintent_spabs_close_bool
              {
                \MathMLintent { _fin-valor-absoluto }
                  { \__spintent_inner_sep: \c__spintent_invisible_sep_int }
              }
          }
          {
            \__spintent_invisible_sep:
            \MathMLintent { _ \tl_use:N \l__spintent_spabs_read_arg_tl :nofix }
              {
                \l__spintent_spabs_delim_l_tl \__spintent_inner_sep: #1
                  \l__spintent_spabs_delim_r_tl
              }
          }
      }
  }
```

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

`\spabs` The public command `\spabs` enforces *math mode*, opens a local group and delegates execution to `\__spintent_spabs_exec:nn`.

```
\NewDocumentCommand \spabs { O{} m }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \attribute \l__spintent_scope_attr_int = 1
          \tl_if_blank:nTF {#2}
            { \msg_error:nn { spintent } { spabs-empty-arg } }
            { \__spintent_spabs_exec:nn {#1} {#2} }
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spabs } }
  }
```

(*End of definition for* `\spabs`. *This function is documented on page* 42.)

15.43 The function `\__spintent_mathmlarg_point:nnn`

`\l__spintent_mathmlarg_point_ref_tl` `\l__spintent_mathmlarg_point_kv_tl` These variables are scratch variables of `\__spintent_mathmlarg_point:nnn`. The variable `\l__spintent_mathmlarg_point_ref_tl` stores the reference name given in `#1` and `\l__spintent_mathmlarg_point_kv_tl` stores the key-value list with the attributes of the point.

```
\tl_new:N \l__spintent_mathmlarg_point_ref_tl
\tl_new:N \l__spintent_mathmlarg_point_kv_tl
```

(*End of definition for* `\l__spintent_mathmlarg_point_ref_tl` *and* `\l__spintent_mathmlarg_point_kv_tl`.)

`\l__spintent_mathmlarg_point_mode_str` `\l__spintent_mathmlarg_point_silent_bool` The variables `\l__spintent_mathmlarg_point_mode_str` and `\l__spintent_mathmlarg_point_silent_bool` store the state of the keys `read-cmd` and `silent` of the calling command, found by the name given in `#2`.

```
\str_new:N  \l__spintent_mathmlarg_point_mode_str
\bool_new:N \l__spintent_mathmlarg_point_silent_bool
```

(*End of definition for* `\l__spintent_mathmlarg_point_mode_str` *and* `\l__spintent_mathmlarg_point_silent_bool`.)

`\__spintent_mathmlarg_point:nnn` `\__spintent_mathmlarg_point:nnV` Annotates the point in '`#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 is tagged with `\__spintent_luafun_mrow_intent_arg:nn` and the subscript is appended separately, silenced. A subscript of more than one digit is typeset plain, with one extra silent claim per digit. For `read-cmd=mathcat`, the letter and the subscript are typeset together and tagged as a whole with 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_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_luaset_intent_str -"
                  }
                \__spintent_luafun_mrow_intent_arg:nn
                  { } { \l__spintent_mathmlarg_point_kv_tl }
                \int_compare:nNnTF { \str_count:N \l__spintent_geo_luaset_sub_str } >
                  { 1 }
                  {
                    \int_step_inline:nnnn { 1 } { 1 }
                      { \str_count:N \l__spintent_geo_luaset_sub_str }
                      {
                        \__spintent_luafun_mrow_intent_arg:nn
                          { } { intent = ":silent" }
                      }
                    {
                      \l__spintent_geo_luaset_letra_str
                      \c_math_subscript_token { \l__spintent_geo_luaset_sub_str }
                    }
                  }
                  {
                    {
                      \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, since a prime is not spoken on the base and there is 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_luaset_intent_str -"
                  }
                \__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, the same distinction applies to the combined `\l__spintent_geo_luaset_print_tl`: the shared queue for `school` or when `silent` is `true`, and a native `\MathMLarg` for `mathcat`. The two mechanisms are never mixed in the same formula.

```
        {
          \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
                {  } % must be empty
                { \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_luaset_intent_str -"
                    }
                  \__spintent_luafun_mrow_intent_arg:nn
                    {  } % must be empty
                    { \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`.)

15.44 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.

15.44.1 Definition of variables for `\spPoint`

`\l__spintent_spPoint_mode_str`
`\l__spintent_spPoint_prefix_tl`

The variables `\l__spintent_spPoint_mode_str` and `\l__spintent_spPoint_prefix_tl` store the values set by the keys `read-cmd` and `prefix`.

```
\str_new:N \l__spintent_spPoint_mode_str
\tl_new:N  \l__spintent_spPoint_prefix_tl
```

(*End of definition for* `\l__spintent_spPoint_mode_str` *and* `\l__spintent_spPoint_prefix_tl`.)

`\l__spintent_spPoint_intent_arg_tl`

The variable `\l__spintent_spPoint_intent_arg_tl` stores the `intent` being built.

```
\tl_new:N \l__spintent_spPoint_intent_arg_tl
```

(*End of definition for* `\l__spintent_spPoint_intent_arg_tl`.)

15.44.2 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~argument~'#1'~for~'\token_to_str:N \spPoint'~\msg_line_context:.
  }
  {
    The~argument~must~be~a~single~uppercase~letter~with~an~optional~subscript~(\_{...})~
    or~prime~(').
  }
```

(*End of definition for* spPoint-invalid-point.)

15.44.3 Definition of keys for \spPoint

read-cmd
prefix
silent
concept
unknown

We define the ⟨*keys*⟩ read-cmd, prefix, silent 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   .bool_set:N = \l__spintent_spPoint_silent_bool,
    silent   .initial:n = false,
    silent   .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.*)

15.44.4 Internal functions for \spPoint

__spintent_spPoint_exec:nnn

The internal function __spintent_spPoint_exec:nnn processes the {⟨*argument*⟩} passed in '#2'. It calls __spintent_luafun_geo_point_parse_and_set:n, reporting spPoint-invalid-point on error. If silent is true or concept is false, it sets \l__spintent_spPoint_intent_arg_tl to _($pt), or to '_-' when silent is true, whatever read-cmd is. Otherwise it sets it to _punto:prefix($pt), with prefix placed after the leading _ when given. Finally it calls __spintent_spPoint_mathml:n.

Before processing the {⟨*argument*⟩}, it sets the optional ⟨*keys*⟩ passed in '#1'.

```
\cs_new_protected:Nn \__spintent_spPoint_exec:nnn
  {
    \keys_set:nn { spintent / spPoint } {#1}
    \__spintent_luafun_geo_point_parse_and_set:n { \exp_not:n {#2} }
    \str_if_eq:VnT \l__spintent_geo_luaset_error_str { true }
      {
        \msg_error:nne { spintent } { spPoint-invalid-point } { \exp_not:n {#2} }
      }
    \bool_if:nTF
      {
```

```
      \bool_lazy_or_p:nn
        { \l__spintent_spPoint_silent_bool }
        { ! \l__spintent_spPoint_concept_bool }
    }
    {
      \tl_set:Nn \l__spintent_spPoint_intent_arg_tl { _($pt) }
      \bool_if:NT \l__spintent_spPoint_silent_bool
        {
          \tl_set:Nn \l__spintent_spPoint_intent_arg_tl { _- }
        }
    }
    {
      \tl_set:Ne \l__spintent_spPoint_intent_arg_tl
        {
          _
          \tl_if_empty:NF \l__spintent_spPoint_prefix_tl
            {
              \tl_use:N \l__spintent_spPoint_prefix_tl -
            }
          punto:prefix($pt)
        }
    }
  \__spintent_spPoint_mathml:n {#2}
  \tl_if_novalue:nF {#3}
    {
      \MathMLintent { :silent } { \c__spintent_invisible_sep_int } % prevent "por"
      \tl_set:Nn \l__spintent_spcoord_raw_arg_tl {#3}
      \__spintent_luafun_spcoord_parse_and_set:n { \exp_not:n {#3} }
      \str_if_eq:VnT \l__spintent_spcoord_luaset_error_str { true }
        { \msg_error:nne { 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:N \l__spintent_spPoint_silent_bool }
    }
}
```

(*End of definition for* `\__spintent_spPoint_exec:nnn`.)

`\__spintent_spPoint_mathml:n`

The function `\__spintent_spPoint_mathml:n` calls `\__spintent_invisible_sep:` and then wraps `\__spintent_mathmlarg_point:nnn`, which parses '`#1`', tags it `arg="pt"` and silences a subscript or prime, inside a native `\MathMLintent` carrying `\l__spintent_spPoint_intent_arg_tl` as the concept `intent`. Inside that group it calls `\__spintent_inner_sep:` first.

```
\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`.)

`\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 24.*)

15.45 The command \spvec

The user command \spvec supports two types of notation for the input {⟨*argument*⟩}: a pair of points, which gets \overrightarrow, or a one-letter name, which gets \vec instead. Both notations are detected by the Lua module. An optional coordinate, in parentheses, may follow the {⟨*argument*⟩}.

15.45.1 Definition of variables for \spvec

\l__spintent_spvec_mode_str
\l__spintent_spvec_prefix_tl

The variables \l__spintent_spvec_mode_str and \l__spintent_spvec_prefix_tl store the values set by the keys read-cmd and prefix.

```
\str_new:N \l__spintent_spvec_mode_str
\tl_new:N  \l__spintent_spvec_prefix_tl
```

(*End of definition for* \l__spintent_spvec_mode_str *and* \l__spintent_spvec_prefix_tl.)

\l__spintent_spvec_pt_seq
\l__spintent_spvec_pt_idx_int
\l__spintent_spvec_refs_tl

The variable \l__spintent_spvec_pt_seq stores the points of the argument, split at the commas, and \l__spintent_spvec_pt_idx_int counts the point being read. The variable \l__spintent_spvec_refs_tl stores the reference list $pts1,$pts2,... built from them.

```
\seq_new:N \l__spintent_spvec_pt_seq
\int_new:N \l__spintent_spvec_pt_idx_int
\tl_new:N  \l__spintent_spvec_refs_tl
```

(*End of definition for* \l__spintent_spvec_pt_seq, \l__spintent_spvec_pt_idx_int, *and* \l__spintent_spvec_refs_tl.)

\l__spintent_spvec_intent_arg_tl
\l__spintent_spvec_kv_tl

The variable \l__spintent_spvec_intent_arg_tl stores the intent being built and \l__spintent_spvec_kv_tl stores the key-value list with the attributes arg and intent of the points.

```
\tl_new:N \l__spintent_spvec_intent_arg_tl
\tl_new:N \l__spintent_spvec_kv_tl
```

(*End of definition for* \l__spintent_spvec_intent_arg_tl *and* \l__spintent_spvec_kv_tl.)

15.45.2 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:nnnn { spintent } { spvec-invalid-arg }
  {
    Invalid~argument~'#1'~for~'\token_to_str:N \spvec'~\msg_line_context:.
  }
  {
    The~argument~must~be~a~lowercase~letter~or~two~points~separated~by~a~comma.
  }
```

(*End of definition for* spvec-invalid-arg.)

15.45.3 Definition of keys for \spvec

silent
concept
autofit
read-cmd
prefix
unknown

We define the ⟨*keys*⟩ silent, concept, autofit, read-cmd and prefix.

```
\keys_define:nn { spintent / spvec }
  {
    silent   .bool_set:N = \l__spintent_spvec_silent_bool,
    silent   .initial:n = false,
    silent   .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` *and others.*)

15.45.4 Internal functions for \spvec

`\__spintent_spvec_exec:nnn`

The internal function `\__spintent_spvec_exec:nnn` processes the {⟨*argument*⟩} passed in '`#2`'. 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`, since 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`.

Before processing the {⟨*argument*⟩}, it sets the optional ⟨*keys*⟩ passed in '`#1`'.

```
\cs_new_protected:Nn \__spintent_spvec_exec:nnn
  {
    \keys_set:nn { spintent / spvec } {#1}
    \__spintent_luafun_geo_parse_and_set:n { \exp_not:n {#2} }
    \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:nNnF { \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 {#2} } }
      { \__spintent_spvec_mathml:n {#2} }
    \tl_if_novalue:nF {#3}
      {
        \bool_if:NTF \l__spintent_spvec_silent_bool
      { \MathMLintent { :silent } { \equal } }
      { \equal } % vector notation =( , )
        % prevent "intervalo"
        \MathMLintent { :silent } { \c__spintent_invisible_sep_int }
        \tl_set:Nn \l__spintent_spcoord_raw_arg_tl {#3}
        \__spintent_luafun_spcoord_parse_and_set:n { \exp_not:n {#3} }
        \str_if_eq:VnT \l__spintent_spcoord_luaset_error_str { true }
          { \msg_error:nne { 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:N \l__spintent_spvec_silent_bool }
      }
  }
```

(*End of definition for* `\__spintent_spvec_exec:nnn`.)

`\__spintent_spvec_mathml:n`

The function `\__spintent_spvec_mathml:n` branches on `\l__spintent_geo_luaset_is_name_str`.

For a proper name that is a single letter with no prime, the concept `intent` is '`_-`' when `silent` is `true`, whatever `read-cmd` is, `_($pt)` when `concept` is `false`, and otherwise `_vector:prefix($pt)`, with `prefix` placed after the leading `_` when given. With `\l__spintent_geo_luaset_tipo_str` equal to `sub`, `read-cmd=school` tags `\vec{letra}` with `\__spintent_luafun_mrow_intent_arg:nn` and a literal `kv`, and appends the subscript silenced; `read-cmd=mathcat` tags `\vec{...}` with a native `\MathMLarg` and silences the subscript only under `silent`. The key `autofit` inserts a `\vphantom{b}` inside `\vec{...}` to normalize its height. Otherwise, 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_bool }
              { ! \l__spintent_spvec_concept_bool }
          }
          {
            \tl_set:Nn \l__spintent_spvec_intent_arg_tl { _($pt) }
            \bool_if:NT \l__spintent_spvec_silent_bool
              { \tl_set:Nn \l__spintent_spvec_intent_arg_tl { _- } }
          }
          {
            \tl_set:Ne \l__spintent_spvec_intent_arg_tl
              {
                _
                \tl_if_empty:NF \l__spintent_spvec_prefix_tl
                  { \tl_use:N \l__spintent_spvec_prefix_tl - }
                vector:prefix($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_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_bool
                          { \c_math_subscript_token { \l__spintent_geo_luaset_sub_str } }
```

```
                    }
                  \bool_if:NF \l__spintent_spvec_silent_bool
                    { \c_math_subscript_token { \l__spintent_geo_luaset_sub_str } }
                }
            }
        }
        {
          \bool_if:NTF \l__spintent_spvec_silent_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 '`_-`' when `silent` is `true`, `_(refs)` when `concept` is `false` or `_vector:prefix(refs)` otherwise, 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_idx_int }
        }
      \bool_if:nTF
        {
          \bool_lazy_or_p:nn
            { \l__spintent_spvec_silent_bool }
            { ! \l__spintent_spvec_concept_bool }
        }
        {
          \tl_set:Ne \l__spintent_spvec_intent_arg_tl
            { _( \l__spintent_spvec_refs_tl ) }
          \bool_if:NT \l__spintent_spvec_silent_bool
            { \tl_set:Nn \l__spintent_spvec_intent_arg_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:
```

©2026 by Pablo González L

```
        \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`.)

`\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 25.*)

15.46 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 a comma or a lowercase letter, it is formatted without any visual decorator. Both notations are detected by the Lua module.

15.46.1 Definition of variables for `\sprecta`

`\l__spintent_sprecta_mode_str` `\l__spintent_sprecta_prefix_tl` The variables `\l__spintent_sprecta_mode_str` and `\l__spintent_sprecta_prefix_tl` store the values set by the keys `read-cmd` and `prefix`.

```
\str_new:N \l__spintent_sprecta_mode_str
\tl_new:N  \l__spintent_sprecta_prefix_tl
```

(*End of definition for* `\l__spintent_sprecta_mode_str` *and* `\l__spintent_sprecta_prefix_tl`.)

`\l__spintent_sprecta_pt_seq` `\l__spintent_sprecta_pt_idx_int` `\l__spintent_sprecta_refs_tl` The variable `\l__spintent_sprecta_pt_seq` stores the points of the argument, split at the commas, and `\l__spintent_sprecta_pt_idx_int` counts the point being read. The variable `\l__spintent_sprecta_refs_tl` stores the reference list `$pts1,$pts2,...` built from them.

```
\seq_new:N \l__spintent_sprecta_pt_seq
\int_new:N \l__spintent_sprecta_pt_idx_int
\tl_new:N  \l__spintent_sprecta_refs_tl
```

(*End of definition for* `\l__spintent_sprecta_pt_seq`, `\l__spintent_sprecta_pt_idx_int`, *and* `\l__spintent_sprecta_refs_tl`.)

`\l__spintent_sprecta_intent_arg_tl` `\l__spintent_sprecta_kv_tl` The variable `\l__spintent_sprecta_intent_arg_tl` stores the `intent` being built and `\l__spintent_sprecta_kv_tl` stores the key-value list with the attributes `arg` and `intent` of the points.

```
\tl_new:N \l__spintent_sprecta_intent_arg_tl
\tl_new:N \l__spintent_sprecta_kv_tl
```

(*End of definition for* `\l__spintent_sprecta_intent_arg_tl` *and* `\l__spintent_sprecta_kv_tl`.)

15.46.2 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~argument~'#1'~for~'\token_to_str:N \sprecta'~\msg_line_context:.
  }
  {
    The~argument~must~be~a~letter~or~two~points~separated~by~a~comma.
  }
```

(*End of definition for* sprecta-invalid-points.)

15.46.3 Definition of keys for \sprecta

read-cmd
prefix
silent
concept
unknown

We define the ⟨*keys*⟩ read-cmd, prefix, silent and concept.

```
\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   .bool_set:N = \l__spintent_sprecta_silent_bool,
    silent   .initial:n = false,
    silent   .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 { sprecta }
        \__spintent_unknown_keys_cmd:n {#1}
      },
  }
```

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

15.46.4 Internal functions for \sprecta

__spintent_sprecta_exec:nn

The internal function __spintent_sprecta_exec:nn processes the {⟨*argument*⟩} passed in '#2'. Calls __spintent_luafun_geo_parse_and_set:n, reporting sprecta-invalid-points on error, then __spintent_sprecta_mathml:n.

Before processing the {⟨*argument*⟩}, it sets the optional ⟨*keys*⟩ passed in '#1'.

```
\cs_new_protected:Nn \__spintent_sprecta_exec:nn
  {
    \keys_set:nn { spintent / sprecta } {#1}
    \__spintent_luafun_geo_parse_and_set:n { \exp_not:n {#2} }
    \str_if_eq:VnT \l__spintent_geo_luaset_error_str { true }
      {
        \msg_error:nne { spintent } { sprecta-invalid-points } { \exp_not:n {#2} }
      }
    \__spintent_sprecta_mathml:n {#2}
  }
```

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

`\__spintent_sprecta_mathml:n`

The function `\__spintent_sprecta_mathml:n` branches on `\l__spintent_geo_luaset_is_name_str`. For a proper name, such as `m` or `l`, the concept `intent` is `_($pt)` when `silent` is `true` *or* `concept` is `false`, and `_recta:prefix($pt)` otherwise, with `prefix` placed after the leading `_` when given; with `silent=true` and `read-cmd` not `school` it is '`_-`'. With `read-cmd=school`, the `sub` and `prima` cases tag the letter with a literal `kv` that is '`_-`' when `silent` is `true`.

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_sprecta_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_sprecta_silent_bool }
              { ! \l__spintent_sprecta_concept_bool }
          }
          {
            \tl_set:Nn \l__spintent_sprecta_intent_arg_tl { _($pt) }
        \bool_if:NT \l__spintent_sprecta_silent_bool
          {
            \str_if_eq:VnF \l__spintent_sprecta_mode_str { school }
              { \tl_set:Nn \l__spintent_sprecta_intent_arg_tl { _- } }
          }
          }
          {
            \tl_set:Ne \l__spintent_sprecta_intent_arg_tl
              {
                _
                \tl_if_empty:NF \l__spintent_sprecta_prefix_tl
                  { \tl_use:N \l__spintent_sprecta_prefix_tl - }
                recta:prefix($pt)
              }
          }
        \str_case:VnF \l__spintent_geo_luaset_tipo_str
          {
            { sub }
              {
                \str_if_eq:VnTF \l__spintent_sprecta_mode_str { school }
                  {
                    \bool_if:NTF \l__spintent_sprecta_silent_bool
                      {
                        \tl_set:Nn \l__spintent_sprecta_kv_tl
                          { arg = "pt", intent = "_-" }
                      }
                      {
                        \tl_set:Ne \l__spintent_sprecta_kv_tl
                          {
                            arg = "pt", intent = "_ \l__spintent_geo_luaset_intent_str -"
                          }
                      }
                    \MathMLintent { \l__spintent_sprecta_intent_arg_tl }
                      {
                        \__spintent_inner_sep:
                        \__spintent_luafun_mrow_intent_arg:nn
                          { } { \l__spintent_sprecta_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 the base and the subscript together with a native `\MathMLarg`,

without using the shared queue.

```
                    {
                      \MathMLintent { \l__spintent_sprecta_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, since a prime is not spelled out on the base and there is nothing to repeat; `read-cmd=mathcat` tags base and mark together through a native `\MathMLarg`.

```
              { prima }
                {
                  \str_if_eq:VnTF \l__spintent_sprecta_mode_str { school }
                    {
                      \bool_if:NTF \l__spintent_sprecta_silent_bool
                        {
                          \tl_set:Nn \l__spintent_sprecta_kv_tl
                            { arg = "pt", intent = "_-" }
                        }
                        {
                          \tl_set:Ne \l__spintent_sprecta_kv_tl
                            {
                              arg = "pt", intent = "_ \l__spintent_geo_luaset_intent_str -"
                            }
                        }
                      \MathMLintent { \l__spintent_sprecta_intent_arg_tl }
                        {
                          \__spintent_inner_sep:
                          \__spintent_luafun_mrow_intent_arg:nn
                            { } { \l__spintent_sprecta_kv_tl }
                          {
                            \l__spintent_geo_luaset_letra_str
                              \l__spintent_geo_luaset_sub_str
                          }
                        }
                    }
                    {
                      \MathMLintent { \l__spintent_sprecta_intent_arg_tl }
                        {
                          {
                            \__spintent_inner_sep:
                            \MathMLarg { pt }
                              {
                                {
                                  \l__spintent_geo_luaset_letra_str
                                    \l__spintent_geo_luaset_sub_str
                                }
                              }
                          }
                        }
                    }
                }
            }
```

Otherwise, the same `kv` logic wraps the combined `\l__spintent_geo_luaset_print_tl` as a whole: with the shared queue for `school` and with its own `arg` alone for `mathcat`.

```
            {
```

```
4985            \bool_if:NTF \l__spintent_specta_silent_bool
4986              {
4987                \tl_set:Nn \l__spintent_specta_kv_tl { arg = "pt", intent = "_-" }
4988              }
4989              {
4990                \str_if_eq:VnTF \l__spintent_specta_mode_str { school }
4991                  {
4992                    \tl_set:Ne \l__spintent_specta_kv_tl
4993                      { arg = "pt", intent = "_ \l__spintent_geo_luaset_intent_str -" }
4994                  }
4995                  {
4996                    \tl_set:Nn \l__spintent_specta_kv_tl { arg = "pt" }
4997                  }
4998              }
4999            \__spintent_luafun_mrow_intent_arg:nn
5000              { \l__spintent_specta_intent_arg_tl }
5001              { \l__spintent_specta_kv_tl }
5002            { \l__spintent_geo_luaset_print_tl }
5003          }
5004      }
```

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 '`_-`' when `silent` is `true`, `_(refs)` when `concept` is `false` or `_recta:prefix(refs)` otherwise, applied with a native `\MathMLintent` wrapping `\overleftrightarrow`. The sequence is walked again, calling `\__spintent_mathmlarg_point:nnn` once per point.

```
5005      {
5006        \seq_set_from_clist:NN \l__spintent_specta_pt_seq
5007          \l__spintent_geo_luaset_points_clist
5008        \tl_clear:N \l__spintent_specta_refs_tl
5009        \int_zero:N \l__spintent_specta_pt_idx_int
5010        \seq_map_inline:Nn \l__spintent_specta_pt_seq
5011          {
5012            \int_incr:N \l__spintent_specta_pt_idx_int
5013            \tl_if_empty:NF \l__spintent_specta_refs_tl
5014              { \tl_put_right:Nn \l__spintent_specta_refs_tl { , } }
5015            \tl_put_right:Ne \l__spintent_specta_refs_tl
5016              { $ pt \int_use:N \l__spintent_specta_pt_idx_int }
5017          }
5018        \bool_if:nTF
5019          {
5020            \bool_lazy_or_p:nn
5021              { \l__spintent_specta_silent_bool }
5022              { ! \l__spintent_specta_concept_bool }
5023          }
5024          {
5025            \tl_set:Ne \l__spintent_specta_intent_arg_tl
5026              { _( \l__spintent_specta_refs_tl ) }
5027            \bool_if:NT \l__spintent_specta_silent_bool
5028              { \tl_set:Nn \l__spintent_specta_intent_arg_tl { _- } }
5029          }
5030          {
5031            \tl_set:Ne \l__spintent_specta_intent_arg_tl
5032              {
5033                _
5034                \tl_if_empty:NF \l__spintent_specta_prefix_tl
5035                  { \tl_use:N \l__spintent_specta_prefix_tl - }
5036                recta:prefix( \l__spintent_specta_refs_tl )
5037              }
5038          }
5039        \__spintent_invisible_sep:
5040        \MathMLintent { \l__spintent_specta_intent_arg_tl }
5041          {
5042            \__spintent_inner_sep:
5043            \overleftrightarrow
5044              {
5045                \int_zero:N \l__spintent_specta_pt_idx_int
5046                \seq_map_inline:Nn \l__spintent_specta_pt_seq
5047                  {
5048                    \int_incr:N \l__spintent_specta_pt_idx_int
5049                    \__spintent_mathmlarg_point:nnn
```

```
                        { pt \int_use:N \l__spintent_sprecta_pt_idx_int }
                        { sprecta }
                        { ##1 }
                    }
                }
            }
        }
    }
```

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

`\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* 25.)

15.47 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`.

15.47.1 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` stores the value set by the key `read-cmd`, the variable `\l__spintent_sprayo_prefix_tl` the value set by the key `prefix` and the variable `\l__spintent_sprayo_read_sty_str` 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`.)

`\l__spintent_sprayo_pt_seq` `\l__spintent_sprayo_pt_idx_int` `\l__spintent_sprayo_refs_tl` The variable `\l__spintent_sprayo_pt_seq` stores the points taken from `\l__spintent_geo_luaset_points_cli` by `\__spintent_sprayo_mathml:` and `\l__spintent_sprayo_pt_idx_int` counts the point being read. The variable `\l__spintent_sprayo_refs_tl` stores the reference list `$pts1,$pts2,...` built from them.

```
\seq_new:N \l__spintent_sprayo_pt_seq
\int_new:N \l__spintent_sprayo_pt_idx_int
\tl_new:N  \l__spintent_sprayo_refs_tl
```

(*End of definition for* `\l__spintent_sprayo_pt_seq`, `\l__spintent_sprayo_pt_idx_int`, *and* `\l__spintent_sprayo_refs_tl`.)

`\l__spintent_sprayo_intent_arg_tl` The variable `\l__spintent_sprayo_intent_arg_tl` stores the `intent` being built.

```
\tl_new:N \l__spintent_sprayo_intent_arg_tl
```

(*End of definition for* `\l__spintent_sprayo_intent_arg_tl`.)

15.47.2 Definition of error messages for `\sprayo`

`sprayo-invalid-points` The "error" `sprayo-invalid-points` is raised when the argument is not a comma-separated pair of uppercase letters.

```
\msg_new:nnnn { spintent } { sprayo-invalid-points }
  {
    Invalid~argument~'#1'~for~'\token_to_str:N \sprayo'~\msg_line_context:.
  }
  {
    The~argument~must~be~two~points~separated~by~a~comma.
  }
```

(*End of definition for* `sprayo-invalid-points`.)

©2026 by Pablo González L

15.47.3 Definition of keys for \sprayo

read-cmd
prefix
silent
concept
read-sty
unknown

We define the ⟨*keys*⟩ `read-cmd`, `prefix`, `silent`, `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   .bool_set:N = \l__spintent_sprayo_silent_bool,
    silent   .initial:n = false,
    silent   .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.*)

15.47.4 Internal functions for \sprayo

__spintent_sprayo_exec:nn

The internal function `\__spintent_sprayo_exec:nn` processes the {⟨*argument*⟩} passed in '`#2`'. 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:`. The command `\sprayo` always takes a list of points, so no `intent` is built here.

Before processing the {⟨*argument*⟩}, it sets the optional ⟨*keys*⟩ passed in '`#1`'.

```
\cs_new_protected:Nn \__spintent_sprayo_exec:nn
  {
    \keys_set:nn { spintent / sprayo } {#1}
    \__spintent_luafun_geo_parse_and_set:n { \exp_not:n {#2} }
    \bool_if:nT
      {
        \str_if_eq_p:Vn \l__spintent_geo_luaset_error_str { true } ||
        \str_if_eq_p:Vn \l__spintent_geo_luaset_is_name_str { true }
      }
      {
        \msg_error:nne { spintent } { sprayo-invalid-points } { \exp_not:n {#2} }
      }
    \__spintent_sprayo_mathml:
  }
```

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

`\__spintent_sprayo_mathml:`

The function `\__spintent_sprayo_mathml:` splits `\l__spintent_geo_luaset_points_clist`, which Lua sets with 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 '`_-`' when `silent` is `true`, `_(refs)` when `concept` is `false`, and otherwise `_\l__spintent_sprayo_read_sty_str:prefix(refs)`, where the word `rayo` or `semirrecta` comes from `read-sty`, with `prefix` placed after the leading _ when given. It is applied with a native `\MathMLintent` wrapping `\overrightarrow`, preceded by `\__spintent_inner_sep:`. The sequence is walked a second time, calling `\__spintent_mathmlarg_point:nnn` once per point with its `pts`*n* reference name and the values of `silent` and `read-cmd` looked up under `sprayo`. This call decides `arg` or the 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_int }
      }
    \bool_if:nTF
      {
        \bool_lazy_or_p:nn
          { \l__spintent_sprayo_silent_bool }
          { ! \l__spintent_sprayo_concept_bool }
      }
      {
        \tl_set:Ne \l__spintent_sprayo_intent_arg_tl
          { _( \l__spintent_sprayo_refs_tl ) }
        \bool_if:NT \l__spintent_sprayo_silent_bool
          { \tl_set:Nn \l__spintent_sprayo_intent_arg_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:`.)

`\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* 26.)

15.48 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 among lado, segmento and trazo, and the key read-arg overrides that term with a free-form string. The command \spmside also provides the key print-sym, which controls whether \operatorname{m} is typeset before the argument.

15.48.1 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_spmside_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 store the value set by the key read-sty.

The variables \l__spintent_spside_read_arg_tl and \l__spintent_spmside_read_arg_tl store 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.*)

\l__spintent_spside_intent_tl

The variable \l__spintent_spside_intent_tl stores the concept word, taken from read-arg or read-sty. It is shared by \spside and \spmside.

```
\tl_new:N \l__spintent_spside_intent_tl
```

(*End of definition for* \l__spintent_spside_intent_tl.)

\l__spintent_spside_pt_seq
\l__spintent_spside_pt_idx_int
\l__spintent_spside_refs_tl

The variable \l__spintent_spside_pt_seq stores the points taken from \l__spintent_geo_luaset_points_cli and \l__spintent_spside_pt_idx_int counts the point being read. The variable \l__spintent_spside_refs_t stores the reference list $pts1,$pts2,... built from them. They are shared by \spside and \spmside.

```
\seq_new:N \l__spintent_spside_pt_seq
\int_new:N \l__spintent_spside_pt_idx_int
\tl_new:N  \l__spintent_spside_refs_tl
```

(*End of definition for* \l__spintent_spside_pt_seq, \l__spintent_spside_pt_idx_int, *and* \l__spintent_spside_refs_tl.)

\l__spintent_spside_intent_arg_tl

The variable \l__spintent_spside_intent_arg_tl stores the intent being built. It is shared by \spside and \spmside.

```
\tl_new:N \l__spintent_spside_intent_arg_tl
```

(*End of definition for* \l__spintent_spside_intent_arg_tl.)

15.48.2 Definition of error messages for \spside and \spmside

spside-invalid-points
spmside-invalid-points

The "error"s spside-invalid-points and spmside-invalid-points are raised when the user includes an invalid *"point"* in the {⟨*argument*⟩} passed in '#1' to \spside or \spmside.

```
\clist_map_inline:nn { side , mside }
  {
    \msg_new:nnnn { spintent } { sp #1 -invalid-points }
      {
        Invalid~argument~'##2'~for~'\tl_trim_spaces:n {##1}'~\msg_line_context:.
      }
      {
        The~argument~must~be~two~points~separated~by~a~comma.
      }
  }
```

(*End of definition for* spside-invalid-points *and* spmside-invalid-points.)

15.48.3 Definition of keys for \spside and \spmside

read-cmd
prefix
silent
concept
read-sty
read-arg
print-sym
unknown

We define the shared ⟨*keys*⟩ read-cmd, prefix, silent, concept, read-sty, read-arg and print-sym.

```
\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    .bool_set:c = { l__spintent_sp #1 _silent_bool },
        silent    .initial:n = false,
        silent    .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-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 =
```

```
5278            {
5279              \str_set:Nn \l__spintent_command_name_str { sp #1 }
5280              \__spintent_unknown_keys_cmd:n {##1}
5281            },
5282        }
5283    }
```

The key print-sym is defined for both commands for simplicity, but has no effect on \spside.

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

15.48.4 Internal functions for \spside and \spmside

__spintent_spside_exec:nnn

The internal function __spintent_spside_exec:nnn receives the short name of the command, side or mside, in '#1' and the {⟨*argument*⟩} in '#3'. It calls __spintent_luafun_geo_parse_and_set:n to process '#3', 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 such as read-arg={medida de la hipotenusa}, with gender and article left to the user. No intent is built here: it needs the reference list, which __spintent_spside_mathml:n builds after splitting the points.

Before processing the {⟨*argument*⟩}, it sets the optional ⟨*keys*⟩ passed in '#2'.

```
5284 \cs_new_protected:Nn \__spintent_spside_exec:nnn
5285   {
5286     \keys_set:nn { spintent / sp #1 } {#2}
5287     \__spintent_luafun_geo_parse_and_set:n { \exp_not:n {#3} }
5288     \bool_if:nT
5289       {
5290         \str_if_eq_p:Vn \l__spintent_geo_luaset_error_str { true } ||
5291         \str_if_eq_p:Vn \l__spintent_geo_luaset_is_name_str { true }
5292       }
5293       {
5294         \msg_error:nnee { spintent } { sp #1 -invalid-points }
5295           { \c_backslash_str sp #1 } { \exp_not:n {#3} }
5296       }
5297     \tl_if_empty:cTF { l__spintent_sp #1 _read_arg_tl }
5298       {
5299         \tl_set:Ne \l__spintent_spside_intent_tl
5300           { \str_use:c { l__spintent_sp #1 _read_sty_str } }
5301         \str_if_eq:eeT {#1} { mside }
5302           {
5303             \tl_put_left:Nn \l__spintent_spside_intent_tl { medida-del- }
5304           }
5305       }
5306       {
5307         \tl_set:Ne \l__spintent_spside_intent_tl
5308           { \tl_use:c { l__spintent_sp #1 _read_arg_tl } }
5309         \bool_if:cT { l__spintent_sp #1 _concept_bool }
5310           {
5311             \str_if_eq:eeT {#1} { mside }
5312               {
5313                 \tl_put_left:Nn \l__spintent_spside_intent_tl { medida-del- }
5314               }
5315           }
5316       }
5317     \__spintent_spside_mathml:n {#1}
5318   }
```

(*End of definition for* __spintent_spside_exec:nnn.)

__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.

```
5319 \cs_new_protected:Nn \__spintent_spside_mathml:n
5320   {
5321     \seq_set_from_clist:NN \l__spintent_spside_pt_seq
5322       \l__spintent_geo_luaset_points_clist
5323     \tl_clear:N \l__spintent_spside_refs_tl
5324     \int_zero:N \l__spintent_spside_pt_idx_int
```

©2026 by Pablo González L

```
    \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_idx_int }
      }
```

The concept `intent` is '_-' when `silent` is `true`. Otherwise, if `read-arg` is empty, it is `_(refs)` when `concept` is `false` and `_\l__spintent_spside_intent_tl:prefix(refs)` when `true`; if `read-arg` is given, it is always `_\l__spintent_spside_intent_tl:prefix(refs)`. The `:prefix(...)` form is prefixed with `prefix` when given.

```
    \bool_if:cTF { l__spintent_sp #1 _silent_bool }
      {
        \tl_set:Nn \l__spintent_spside_intent_arg_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 )
              }
          }
      }
```

It is applied with a native `\MathMLintent`, preceded by `\__spintent_inner_sep:`. Inside, for `#1=side` the points are wrapped in `\overline`; for `#1=mside`, a silent `\operatorname{m}` precedes the `\overline`-wrapped points when `print-sym` is `true`, and the points are typeset bare, without `\overline`, when it is `false`. 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 the values of `silent` and `read-cmd` looked up under `spside` or `spmside`. This call decides `arg` or the 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_sym_bool }
              {
                \__spintent_luafun_mrow_intent_arg:nn
                  { } { intent = ":silent" }
                { \operatorname { m } }
                \overline { \__spintent_spside_points_loop:n {#1} }
              }
              {
                \__spintent_spside_points_loop:n {#1}
              }
```

©2026 by Pablo González L

```
5384            }
5385        }
5386    }
```

(*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`, and `mside` with and without `print-sym`; it takes the short command name in '`#1`' again since a separate macro does not inherit its caller's own `#1`.

```
5387 \cs_new_protected:Nn \__spintent_spside_points_loop:n
5388   {
5389     \int_zero:N \l__spintent_spside_pt_idx_int
5390     \seq_map_inline:Nn \l__spintent_spside_pt_seq
5391       {
5392         \int_incr:N \l__spintent_spside_pt_idx_int
5393         \__spintent_mathmlarg_point:nnn
5394           { pts \int_use:N \l__spintent_spside_pt_idx_int }
5395           { sp #1 }
5396           { ##1 }
5397       }
5398   }
```

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

`\spside` `\spmside` Both public commands `\spside` and `\spmside` enforces *math mode*, opens a local group, and delegates execution to `\__spintent_spside_exec:nnn`.

```
5399 \NewDocumentCommand \spside { O{} m }
5400   {
5401     \mode_if_math:TF
5402       {
5403         \group_begin:
5404           \attribute \l__spintent_scope_attr_int = 1
5405           \__spintent_spside_exec:nnn { side } {#1} {#2}
5406         \group_end:
5407       }
5408       { \msg_error:nnn { spintent } { need-math-mode } { \spside } }
5409   }
5410 \NewDocumentCommand \spmside { O{} m }
5411   {
5412     \mode_if_math:TF
5413       {
5414         \group_begin:
5415           \attribute \l__spintent_scope_attr_int = 1
5416           \__spintent_spside_exec:nnn { mside } {#1} {#2}
5417         \group_end:
5418       }
5419       { \msg_error:nnn { spintent } { need-math-mode } { \spmside } }
5420   }
```

(*End of definition for* `\spside` *and* `\spmside`. *These functions are documented on page 27.*)

15.49 The commands `\spangle` and `\spmangle`

The user commands `\spangle` and `\spmangle` provide a semantic interface for angles. The key `print-sym` 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; the key `recto` chooses the right-angle variant of whichever pair is active.

15.49.1 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_tl` store the values set by the key `prefix`.

The variables defined here are assigned using `.choices:nn` or `.code:n` within `\keys_define:nn`.

```
5421 \clist_map_inline:nn { angle , mangle }
5422   {
5423     \str_new:c { l__spintent_sp #1 _mode_str  }
5424     \tl_new:c  { l__spintent_sp #1 _prefix_tl }
5425   }
```

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

`\l__spintent_spangle_intent_tl` `\l__spintent_spangle_print_sym_now_bool`

The variable `\l__spintent_spangle_intent_tl` stores the concept word, `ángulo` or `medida-del-ángulo`, with the suffix `-recto` when `recto` is `true`. The variable `\l__spintent_spangle_print_sym_now_bool` tells whether the symbol is typeset at this moment. Both are shared by `\spangle` and `\spmangle`.

```
\tl_new:N   \l__spintent_spangle_intent_tl
\bool_new:N \l__spintent_spangle_print_sym_now_bool
```

(*End of definition for* `\l__spintent_spangle_intent_tl` *and* `\l__spintent_spangle_print_sym_now_bool`.)

`\l__spintent_spangle_pt_seq` `\l__spintent_spangle_pt_idx_int` `\l__spintent_spangle_refs_tl`

The variable `\l__spintent_spangle_pt_seq` stores the points taken from `\l__spintent_geo_luaset_points_cl` and `\l__spintent_spangle_pt_idx_int` counts the point being read. The variable `\l__spintent_spangle_refs_` stores the reference list built from them, in the `tres-puntos` and `un-punto` cases of `\__spintent_spangle_mathml:n`.

```
\seq_new:N \l__spintent_spangle_pt_seq
\int_new:N \l__spintent_spangle_pt_idx_int
\tl_new:N  \l__spintent_spangle_refs_tl
```

(*End of definition for* `\l__spintent_spangle_pt_seq`, `\l__spintent_spangle_pt_idx_int`, *and* `\l__spintent_spangle_refs_tl`.)

`\l__spintent_spangle_intent_arg_tl` `\l__spintent_spangle_kv_tl`

The variable `\l__spintent_spangle_intent_arg_tl` stores the `intent` being built. The variable `\l__spintent_spangle_kv_tl` stores the key-value list passed to `\__spintent_luafun_mrow_intent_arg:nn`.

```
\tl_new:N \l__spintent_spangle_intent_arg_tl
\tl_new:N \l__spintent_spangle_kv_tl
```

(*End of definition for* `\l__spintent_spangle_intent_arg_tl` *and* `\l__spintent_spangle_kv_tl`.)

15.49.2 Definition of error messages for `\spangle` and `\spmangle`

`spangle-invalid-arg` `spmangle-invalid-arg`

The "error"s `spangle-invalid-arg` and `spmangle-invalid-arg` are raised when the user includes an invalid {⟨*argument*⟩} passed in '`#1`' to `\spangle` or `\spmangle`.

```
\clist_map_inline:nn { angle , mangle }
  {
    \msg_new:nnnn { spintent } { sp #1 -invalid-arg }
      {
        Invalid~argument~'##2'~for~'\tl_trim_spaces:n {##1}'~\msg_line_context:.
      }
      {
        Use~a~point,~a~Greek~letter,~a~number~or~three~points~separated~by~
        commas.
      }
  }
```

(*End of definition for* `spangle-invalid-arg` *and* `spmangle-invalid-arg`.)

15.49.3 Definition of keys for `\spangle` and `\spmangle`

`read-cmd` `prefix` `recto` `print-sym` `silent` `concept` `unknown`

We define the shared ⟨*keys*⟩ `read-cmd`, `prefix`, `recto`, `print-sym`, `silent` 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-sym .bool_set:c = { l__spintent_sp #1 _print_sym_bool },
        print-sym .initial:n = false,
        print-sym .value_required:n = true,
        silent    .bool_set:c = { l__spintent_sp #1 _silent_bool },
        silent    .initial:n = false,
        silent    .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-sym is defined for both commands for simplicity, but has no effect on \spangle.

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

15.49.4 Internal functions for \spangle and \spmangle

__spintent_spangle_sym:n

The internal function __spintent_spangle_sym:n branches on #1, print-sym and recto to select one of four symbols: \angle/\rightanglesqr when print-sym 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_sym_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.)

__spintent_spangle_exec:nnn

The internal function __spintent_spangle_exec:nnn receives the short name of the command, angle or mangle, in '#1' and the {⟨*argument*⟩} in '#3'. It calls __spintent_luafun_geo_angulo_parse_and_set:n, reporting spangle-invalid-arg or spmangle-invalid-arg on error. For a proper name that is a single letter, not numeric, it reports the same error if the letter is lowercase, since an angle name takes an uppercase letter, a number or a Greek letter; the length check excludes a Greek letter command such as \alpha from this comparison. Then it 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.

Before processing the {⟨*argument*⟩}, it sets the optional ⟨*keys*⟩ passed in '#2'.

```
\cs_new_protected:Nn \__spintent_spangle_exec:nnn
  {
    \keys_set:nn { spintent / sp #1 } {#2}
    \__spintent_luafun_geo_angulo_parse_and_set:n { \exp_not:n {#3} }
    \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 {#3} }
      }
    \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:nNnT { \str_count:N \l__spintent_geo_luaset_letra_str } = { 1 }
            {
              \str_if_eq:eeF { \l__spintent_geo_luaset_letra_str }
                { \str_uppercase:n { \l__spintent_geo_luaset_letra_str } }
                {
                  \msg_error:nnee { spintent } { sp #1 -invalid-arg }
                    { \c_backslash_str sp #1 } { \exp_not:n {#3} }
                }
            }
        }
    \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_exec:nnn`.)

`\__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 with `\__spintent_luafun_mrow_intent_arg:nn` on its own, with `_\l__spintent_spangle_intent_tl` alone, without a `$`-reference, or with '`_-`' when `silent` is `true` *or* `concept` is `false`. With `silent=true`, the name that follows is silenced as a whole.

```
\cs_new_protected:Nn \__spintent_spangle_mathml:n
  {
    \__spintent_invisible_sep:
    \str_if_eq:VnTF \l__spintent_geo_luaset_angulo_case_str { nombre }
      {
        \bool_if:cTF { l__spintent_sp #1 _silent_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
                  }
              }
              {
                \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_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, since 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: it is spoken as part of the literal for `read-cmd=school` and natively for `read-cmd=mathcat`.

```
                    { 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` and `un-punto`, with the suffix `-en` added to the concept word in the latter, where a single point reads *"ángulo en A"*, `\l__spintent_geo_luaset_points_clist` is split into `\l__spintent_spangle_pt_seq` and walked to build `\l__spintent_spangle_refs_tl`, the reference list `$pt1,$pt2,...`. A single native `\MathMLintent` wraps the symbol and the points together,

with `_\l__spintent_spangle_intent_tl:prefix(refs)`, or '`_-`' when `silent` is `true` and `_(refs)` when `concept` is `false`. For `\spmangle` with `print-sym=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.

```
      {
        \str_if_eq:VnT \l__spintent_geo_luaset_angulo_case_str { un-punto }
          { \tl_put_right:Nn \l__spintent_spangle_intent_tl { -en } }
        \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_pt_idx_int }
          }
        \bool_if:cTF { l__spintent_sp #1 _silent_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_sym_now_bool
        \str_if_eq:eeT {#1} { mangle }
          {
            \bool_if:cT { l__spintent_sp #1 _print_sym_bool }
              { \bool_set_true:N \l__spintent_spangle_print_sym_now_bool }
          }
        \MathMLintent { \l__spintent_spangle_intent_arg_tl }
          {
            \__spintent_inner_sep:
            \bool_if:NT \l__spintent_spangle_print_sym_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`.)

`\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 28.*)

15.50 The commands `\sparc` and `\spmarc`

The user commands `\sparc` and `\spmarc` provide a semantic interface for arcs. The command `\spmarc` has no `print-sym` key: `\operatorname{m}` is always present, since $\mathrm{m}\,\overset{\frown}{AB}$ is the standard notation for the measure of an arc.

15.50.1 Definition of variables for `\sparc` and `\spmarc`

`\l__spintent_sparc_mode_str` `\l__spintent_spmarc_mode_str` `\l__spintent_sparc_prefix_tl` `\l__spintent_spmarc_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_spmarc_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.*)

`\l__spintent_sparc_intent_tl` The variable `\l__spintent_sparc_intent_tl` stores the concept word, `arco` or `medida-del-arco`. It is shared by `\sparc` and `\spmarc`.

```
\tl_new:N \l__spintent_sparc_intent_tl
```

(*End of definition for* `\l__spintent_sparc_intent_tl`.)

`\l__spintent_sparc_pt_seq` `\l__spintent_sparc_pt_idx_int` `\l__spintent_sparc_refs_tl` The variable `\l__spintent_sparc_pt_seq` stores the points taken from `\l__spintent_geo_luaset_points_clis` and `\l__spintent_sparc_pt_idx_int` counts the point being read. The variable `\l__spintent_sparc_refs_tl` stores the reference list `$pt1,$pt2,...` used in the `intent` of the concept.

```
\seq_new:N \l__spintent_sparc_pt_seq
\int_new:N \l__spintent_sparc_pt_idx_int
\tl_new:N  \l__spintent_sparc_refs_tl
```

(*End of definition for* `\l__spintent_sparc_pt_seq`, `\l__spintent_sparc_pt_idx_int`, *and* `\l__spintent_sparc_refs_tl`.)

`\l__spintent_sparc_intent_arg_tl` The variable `\l__spintent_sparc_intent_arg_tl` stores the `intent` being built.

```
\tl_new:N \l__spintent_sparc_intent_arg_tl
```

(*End of definition for* `\l__spintent_sparc_intent_arg_tl`.)

©2026 by Pablo González L

15.50.2 Definition of error messages for \sparc and \spmarc

sparc-invalid-points
spmarc-invalid-points

The "error"s sparc-invalid-points and spmarc-invalid-points are raised when the user includes an invalid {⟨*argument*⟩} passed in '#1' to \sparc or \spmarc.

```
\clist_map_inline:nn { arc , marc }
  {
    \msg_new:nnnn { spintent } { sp #1 -invalid-points }
      {
        Invalid~argument~'##2'~for~'\tl_trim_spaces:n {##1}'~\msg_line_context:.
      }
      {
        The~argument~must~be~two~or~three~points~separated~by~commas.
      }
  }
```

(*End of definition for* sparc-invalid-points *and* spmarc-invalid-points.)

15.50.3 Definition of keys for \sparc and \spmarc

read-cmd
prefix
silent
concept
unknown

We define the shared ⟨*keys*⟩ read-cmd, prefix, silent 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   .bool_set:c = { l__spintent_sp #1 _silent_bool },
        silent   .initial:n = false,
        silent   .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.*)

15.50.4 Internal functions for \sparc and \spmarc

__spintent_sparc_wide_arc:n

The function __spintent_sparc_wide_arc:n is 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:Nn \__spintent_sparc_wide_arc:n
  {
    \Umathaccent 7 \symoperators "023DC \scan_stop: {#1}
  }
```

(*End of definition for* __spintent_sparc_wide_arc:n.)

©2026 by Pablo González L

__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.)

__spintent_sparc_exec:nnn

The internal function __spintent_sparc_exec:nnn receives the short name of the command, arc or marc, in '#1' and the {⟨*argument*⟩} in '#3'. It calls __spintent_luafun_geo_parse_and_set:n to process '#3', 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_mathml:n builds after splitting the points.

Before processing the {⟨*argument*⟩}, it sets the optional ⟨*keys*⟩ passed in '#2'.

```
\cs_new_protected:Nn \__spintent_sparc_exec:nnn
  {
    \keys_set:nn { spintent / sp #1 } {#2}
    \__spintent_luafun_geo_parse_and_set:n { \exp_not:n {#3} }
    \bool_if:nT
      {
        \str_if_eq_p:Vn \l__spintent_geo_luaset_error_str { true } ||
        \str_if_eq_p:Vn \l__spintent_geo_luaset_is_name_str { true }
      }
      {
        \msg_error:nnee { spintent } { sp #1 -invalid-points }
          { \c_backslash_str sp #1 } { \exp_not:n {#3} }
      }
    \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_exec:nnn.)

__spintent_sparc_mathml:n

The function __spintent_sparc_mathml:n receives the short command name in '#1'. It splits \l__spintent_geo_luaset_points_clist 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_int }
      }
```

The concept intent is '_-' when silent is true, _(refs) when concept is false; otherwise _\l__spintent_sparc_intent_tl:prefix(refs), with prefix placed after the leading _ when given.

```
    \bool_if:cTF { l__spintent_sp #1 _silent_bool }
      {
        \tl_set:Nn \l__spintent_sparc_intent_arg_tl { _- }
      }
      {
        \bool_if:cTF { l__spintent_sp #1 _concept_bool }
```

```
5824            {
5825              \tl_set:Ne \l__spintent_sparc_intent_arg_tl
5826                {
5827                  _
5828                  \tl_if_empty:cF { l__spintent_sp #1 _prefix_tl }
5829                    { \tl_use:c { l__spintent_sp #1 _prefix_tl } - }
5830                  \l__spintent_sparc_intent_tl :prefix( \l__spintent_sparc_refs_tl )
5831                }
5832            }
5833            {
5834              \tl_set:Ne \l__spintent_sparc_intent_arg_tl
5835                { _( \l__spintent_sparc_refs_tl ) }
5836            }
5837        }
```

It is applied with a single native `\MathMLintent` wrapping `\__spintent_sparc_print_arc:n` directly, with the 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}` always precedes it for `\spmarc`, since there is no `print-sym` 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 `sparc` or `spmarc`.

```
5838      \__spintent_invisible_sep:
5839      \MathMLintent { \l__spintent_sparc_intent_arg_tl }
5840        {
5841          \__spintent_inner_sep:
5842          \str_if_eq:eeT {#1} { marc }
5843            {
5844              \__spintent_luafun_mrow_intent_arg:nn
5845                { } { intent = ":silent" }
5846              { \operatorname { m } }
5847            }
5848          \__spintent_sparc_print_arc:n
5849            {
5850              \int_zero:N \l__spintent_sparc_pt_idx_int
5851              \seq_map_inline:Nn \l__spintent_sparc_pt_seq
5852                {
5853                  \int_incr:N \l__spintent_sparc_pt_idx_int
5854                  \__spintent_mathmlarg_point:nnn
5855                    { pt \int_use:N \l__spintent_sparc_pt_idx_int }
5856                    { sp #1 }
5857                    { ##1 }
5858                }
5859            }
5860        }
5861    }
```

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

`\sparc` `\spmarc` Both public commands `\sparc` and `\spmarc` enforces *math mode*, opens a local group, and delegates execution to `\__spintent_sparc_exec:nnn`.

```
5862  \NewDocumentCommand \sparc { O{} m }
5863    {
5864      \mode_if_math:TF
5865        {
5866          \group_begin:
5867            \attribute \l__spintent_scope_attr_int = 1
5868            \__spintent_sparc_exec:nnn { arc } {#1} {#2}
5869          \group_end:
5870        }
5871        { \msg_error:nnn { spintent } { need-math-mode } { \sparc } }
5872    }
5873  \NewDocumentCommand \spmarc { O{} m }
5874    {
5875      \mode_if_math:TF
5876        {
5877          \group_begin:
5878            \attribute \l__spintent_scope_attr_int = 1
5879            \__spintent_sparc_exec:nnn { marc } {#1} {#2}
5880          \group_end:
5881        }
```

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

(*End of definition for* `\sparc` *and* `\spmarc`*. These functions are documented on page 30.*)

15.51 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`, which holds the full `intent` with the figure name.

15.51.1 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`*.*)

`\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.*)

15.51.2 Definition of error messages for \spPoly

`spPoly-invalid-points`

The "error" `spPoly-invalid-points` is raised when the argument is not a comma-separated list of at least three uppercase letters.

```
\msg_new:nnnn { spintent } { spPoly-invalid-points }
  {
    Invalid~argument~'#1'~for~'\token_to_str:N \spPoly'~\msg_line_context:.
  }
  {
    The~argument~must~be~a~list~of~at~least~three~points~separated~by~commas.
  }
```

(*End of definition for* `spPoly-invalid-points`*.*)

15.51.3 Definition of keys for \spPoly

`read-cmd` `prefix` `print-sym` `silent` `concept` `unknown`

We define the ⟨*keys*⟩ `read-cmd`, `prefix`, `print-sym`, `silent` 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    .bool_set:N = \l__spintent_spPoly_silent_bool,
    silent    .initial:n = false,
```

```
    silent    .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.*)

15.51.4 Internal functions for `\spPoly`

`\__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_int }
      }
  }
```

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

`\__spintent_spPoly_build_concept:` Builds `\l__spintent_spPoly_intent_arg_tl`, independently of `print-sym`: '`_-`' when `silent` is `true`, `_(refs)` when `concept` is `false`; otherwise `_\l__spintent_geo_luaset_concept_str:prefix(refs)`, with `prefix` placed after the leading `_` 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_bool
      {
        \tl_set:Nn \l__spintent_spPoly_intent_arg_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:` It 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`. This call decides `arg` or the literal `intent` per vertex.

```
\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_exec:nn` 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.

Before processing the {⟨*argument*⟩}, it sets the optional ⟨*keys*⟩ passed in '`#1`'.

```
\cs_new_protected:Nn \__spintent_spPoly_exec:nn
  {
    \keys_set:nn { spintent / spPoly } {#1}
    \__spintent_luafun_geo_poly_parse_and_set:n { \exp_not:n {#2} }
    \str_if_eq:VnT \l__spintent_geo_luaset_error_str { true }
      {
        \msg_error:nne { spintent } { spPoly-invalid-points }
          { \exp_not:n {#2} }
      }
    \__spintent_spPoly_mathml:
  }
```

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

`\__spintent_spPoly_mathml:` Builds the reference list with `\__spintent_spPoly_build_refs:` and the concept `intent` with `\__spintent_spPoly_build_concept:`, 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`.

The keys `print-sym` and `concept/silent` are independent. The key `print-sym` only decides whether the glyph from `\__spintent_spPoly_sym:`, silenced with `\__spintent_luafun_mrow_intent_arg:nn`, is attempted, and the glyph is skipped when `\l__spintent_geo_luaset_poly_sym_str` is empty, which happens for a pentagon and above because they have no symbol. It does not affect whether the concept word is spoken.

```
\cs_new_protected:Nn \__spintent_spPoly_mathml:
  {
    \__spintent_invisible_sep:
    \__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:`.)

`\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 31.*)

15.52 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()`. 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*⟩`}`.

15.52.1 Definition of variables for `\spAfig` and `\spPfig`

`\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.*)

`\l__spintent_geo_final_print_tl` `\l__spintent_spfig_symbol_tl`

These variables are used by `\__spintent_spfig_exec:nnn`. The variable `\l__spintent_geo_final_print_tl` stores the final visual placed under the subscript of `\mathcal` and `\l__spintent_spfig_symbol_tl` stores the symbol of the figure, `\triangle` or `\mdlgwhtsquare`, when it is typeset.

```
\tl_new:N \l__spintent_geo_final_print_tl
\tl_new:N \l__spintent_spfig_symbol_tl
```

(*End of definition for* `\l__spintent_geo_final_print_tl` *and* `\l__spintent_spfig_symbol_tl`.)

`\l__spintent_geo_print_sym_str`

The variable `\l__spintent_geo_print_sym_str` stores the value of the key `print-sym` of the command being executed.

```
\str_new:N \l__spintent_geo_print_sym_str
```

(*End of definition for* `\l__spintent_geo_print_sym_str`.)

`\l__spintent_spfig_op_tl` `\l__spintent_spfig_intent_tl` `\l__spintent_spfig_desc_tl`

The variable `\l__spintent_spfig_op_tl` stores *«área»* or *«perímetro»*. The variable `\l__spintent_spfig_intent_tl` stores the concept word for the vertices case, taken from `read-arg` or from the figure name detected. The variable `\l__spintent_spfig_desc_tl` stores the description of the figure that goes after the operator in the `intent`.

```
\tl_new:N \l__spintent_spfig_op_tl
\tl_new:N \l__spintent_spfig_intent_tl
\tl_new:N \l__spintent_spfig_desc_tl
```

(*End of definition for* `\l__spintent_spfig_op_tl`, `\l__spintent_spfig_intent_tl`, *and* `\l__spintent_spfig_desc_tl`.)

©2026 by Pablo González L

\l__spintent_spfig_pt_seq
\l__spintent_spfig_pt_idx_int
\l__spintent_spfig_refs_tl

The variable \l__spintent_spfig_pt_seq stores the points taken from \l__spintent_geo_luaset_points_clis and \l__spintent_spfig_pt_idx_int counts the point being read. The variable \l__spintent_spfig_refs_tl stores the reference list $pt1,$pt2,... built from them.

```
\seq_new:N \l__spintent_spfig_pt_seq
\int_new:N \l__spintent_spfig_pt_idx_int
\tl_new:N  \l__spintent_spfig_refs_tl
```

(*End of definition for* \l__spintent_spfig_pt_seq, \l__spintent_spfig_pt_idx_int, *and* \l__spintent_spfig_refs_tl.)

\l__spintent_spfig_intent_arg_tl
\l__spintent_geo_temp_tl

The variable \l__spintent_spfig_intent_arg_tl stores the intent being built and \l__spintent_geo_temp_t is a scratch copy of \l__spintent_spfig_desc_tl from which the connector is removed.

```
\tl_new:N \l__spintent_spfig_intent_arg_tl
\tl_new:N \l__spintent_geo_temp_tl
```

(*End of definition for* \l__spintent_spfig_intent_arg_tl *and* \l__spintent_geo_temp_tl.)

15.52.2 Definition of error messages for \spAfig and \spPfig

spAfig-invalid-points
spPfig-invalid-points
spAfig-invalid-arg
spPfig-invalid-arg
spAfig-sym-letter-conflict
spPfig-sym-letter-conflict
spAfig-sym-cardinality-conflict
spPfig-sym-cardinality-conflict
spAfig-read-arg-key
spPfig-read-arg-key

The ten "error"s are generated by the same map, using the same ##1/##2 doubling convention as \spside/\spmside. spAfig-invalid-points and spPfig-invalid-points are currently unused by __spintent_spfig_exec:nnn, which raises only spAfig-invalid-arg or spPfig-invalid-arg for any parse failure, regardless of whether a comma was present. spAfig-sym-letter-conflict, spPfig-sym-letter-conflict, spAfig-sym-cardinality-conflict and spPfig-sym-cardinality-confli 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~argument~'##2'~for~'\tl_trim_spaces:n {##1}'~\msg_line_context:.
      }
      {
        The~argument~must~be~a~list~of~points~separated~by~commas.
      }
    \msg_new:nnnn { spintent } { sp #1 -invalid-arg }
      {
        Invalid~argument~'##2'~for~'\tl_trim_spaces:n {##1}'~\msg_line_context:.
      }
      {
        See~the~manual~for~the~accepted~argument~forms.
      }
    \msg_new:nnnn { spintent } { sp #1 -sym-letter-conflict }
      {
        The~key~'print-sym'~cannot~be~combined~with~a~lettered~argument~in~'##1'~
        \msg_line_context:.
      }
      {
        See~the~documentation~of~'##1'~for~details~on~the~key~'print-sym'.
      }
    \msg_new:nnnn { spintent } { sp #1 -sym-cardinality-conflict }
      {
        The~key~'print-sym'~does~not~match~the~vertex~count~in~'##1'~\msg_line_context:.
      }
      {
        See~the~documentation~of~'##1'~for~details~on~the~key~'print-sym'.
      }
    \msg_new:nnnn { spintent } { sp #1 -read-arg-key }
      {
        The~key~'read-arg'~has~no~effect~with~a~single~argument~in~'##1'~
          \msg_line_context:.
      }
      {
        See~the~documentation~of~'##1'~for~details~on~the~key~'read-arg'.
      }
  }
```

(*End of definition for* spAfig-invalid-points *and others.*)

15.52.3 Definition of keys for \spAfig and \spPfig

read-cmd
prefix
print-sym
read-arg

We define the shared ⟨*keys*⟩ read-cmd, prefix, print-sym, read-arg, read-sub, silent and concept.

```
\clist_map_inline:nn { Afig , Pfig }
```

©2026 by Pablo González L

```
{
  \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    .bool_set:c = { l__spintent_sp #1 _silent_bool },
      silent    .initial:n = false,
      silent    .value_required:n = true,
      concept   .bool_set:c = { l__spintent_sp #1 _concept_bool },
      concept   .initial:n = true,
      concept   .value_required:n = true,
      concept*  .meta:n =
        {
          concept  = false,
          read-arg = { \str_if_eq:eeTF {#1} { Afig } { _área } { _perímetro } },
        },
      concept*  .value_forbidden: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.*)

15.52.4 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:`.)

`\__spintent_spfig_exec:nnn` Receives the short command name, `Afig` or `Pfig`, in '`#1`' and the {⟨*argument*⟩} in '`#3`'. Calls `\__spintent_luafun_geo_afig_parse_and_set:n` or `\__spintent_luafun_geo_pfig_parse_and_set:n`, reports `spAfig-invalid-arg` or `spPfig-invalid-arg` on failure, and warns with `spAfig-read-arg-key` or `spPfig-read-arg-key` when `read-arg` is given for a bare number/letter argument that has no vertices, where it has no effect and is ignored. Sets `\l__spintent_spfig_op_tl` to *«área»*/*«perímetro»*, then delegates to `\__spintent_spfig_mathml:n`.

Before processing the {⟨*argument*⟩}, it sets the optional ⟨*keys*⟩ passed in '`#2`'.

```
\cs_new_protected:Nn \__spintent_spfig_exec:nnn
  {
    \keys_set:nn { spintent / sp #1 } {#2}
    \str_if_eq:eeTF {#1} { Afig }
      { \__spintent_luafun_geo_afig_parse_and_set:n { \exp_not:n {#3} } }
      { \__spintent_luafun_geo_pfig_parse_and_set:n { \exp_not:n {#3} } }
    \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 {#3} }
      }
    \clist_if_empty:NT \l__spintent_geo_luaset_points_clist
      {
        \tl_if_empty:cF { l__spintent_sp #1 _read_arg_tl }
          {
            \msg_warning: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_exec:nnn`.)

`\__spintent_spfig_build_simple:n` Handles the argument-less, table-letter and bare number/letter cases, none of which has vertices. Builds `\l__spintent_geo_final_print_tl`, 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
```

```
6206                {
6207                  \str_if_eq:VnTF \l__spintent_geo_print_sym_str { triangle }
6208                    { triángulo } { cuadrado }
6209                }
6210            }
6211        }
```

With a bare number/letter, for which `\l__spintent_geo_luaset_letra_word_str` is empty, 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`.

```
6212        {
6213          \str_if_empty:NTF \l__spintent_geo_luaset_letra_word_str
6214            {
6215              \str_case:VnF \l__spintent_geo_print_sym_str
6216                {
6217                  { auto }
6218                    {
6219                      \tl_set:NV \l__spintent_geo_final_print_tl
6220                        \l__spintent_geo_luaset_print_tl
6221                    }
6222                  { none }
6223                    {
6224                      \tl_set:NV \l__spintent_geo_final_print_tl
6225                        \l__spintent_geo_luaset_print_tl
6226                    }
6227                }
6228                {
6229                  \tl_set:Nn \l__spintent_geo_final_print_tl
6230                    {
6231                      \__spintent_spfig_sym:
6232                      \c_math_subscript_token { \l__spintent_geo_luaset_print_tl }
6233                    }
6234                }
6235              \str_case:VnF \l__spintent_geo_print_sym_str
6236                {
6237                  { auto }
6238                    {
6239                      \tl_set:Ne \l__spintent_spfig_desc_tl
6240                        {
6241                          \bool_if:cT { l__spintent_sp #1 _read_sub_bool } { sub - }
6242                          \l__spintent_geo_luaset_sub_spoken_str
6243                        }
6244                    }
6245                  { none }
6246                    {
6247                      \tl_set:Ne \l__spintent_spfig_desc_tl
6248                        {
6249                          \bool_if:cT { l__spintent_sp #1 _read_sub_bool } { sub - }
6250                          \l__spintent_geo_luaset_sub_spoken_str
6251                        }
6252                    }
6253                }
6254                {
6255                  \tl_set:Ne \l__spintent_spfig_desc_tl
6256                    {
6257                      del-
6258                      \str_if_eq:VnTF \l__spintent_geo_print_sym_str { triangle }
6259                        { triángulo } { cuadrado }
6260                      -
6261                      \bool_if:cT { l__spintent_sp #1 _read_sub_bool } { sub - }
6262                      \l__spintent_geo_luaset_sub_spoken_str
6263                    }
6264                }
6265            }
```

A forced `print-sym` raises `spAfig-sym-letter-conflict` or `spPfig-sym-letter-conflict` when `\l__spintent_geo_luaset_letra_word_str` is non-empty, which happens with a table letter, that is `F`, `P`, `B`, `L` or `T`. The visual is the value as given. The description is its own word, connected as `\l__spintent_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-letter-conflict }
                    { \c_backslash_str sp #1 }
                }
              \tl_set:NV \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
                    }
                }
            }
        }
```

The variable `\l__spintent_spfig_intent_tl` is always the `word-` shape, without the leading `_`: `\l__spintent_spfig_op_tl` alone when `\l__spintent_spfig_desc_tl` is empty, both joined by a dash otherwise. `silent` overrides `\l__spintent_spfig_intent_arg_tl` with '`_-`'; with `silent=false` and `concept=false`, it becomes `_\l__spintent_spfig_desc_tl-`, where the descriptive part is alone, without *«área»*/*«perímetro»* nor, for a table letter, its connector `de-la`/`del`; with `concept=true` it is `_`, then `prefix` when given, and then `\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_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
              {
                _
                \tl_if_empty:cF { l__spintent_sp #1 _prefix_tl }
                  { \tl_use:c { l__spintent_sp #1 _prefix_tl } - }
                \l__spintent_spfig_intent_tl
              }
          }
          {
            \tl_set_eq:NN \l__spintent_geo_temp_tl \l__spintent_spfig_desc_tl
            \str_if_empty:NF \l__spintent_geo_luaset_letra_conector_str
              {
                \tl_replace_once:Nee \l__spintent_geo_temp_tl
                  { \l__spintent_geo_luaset_letra_conector_str - } { }
              }
            \tl_set:Ne \l__spintent_spfig_intent_arg_tl { _ \l__spintent_geo_temp_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_int }
      }
  }
```

(*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_refs:`.

The key `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 `\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`, which is 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 `spAfig-sym-cardinality-conflict` or `spPfig-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 }
          }
      }
```

The key `silent` sets `\l__spintent_spfig_intent_arg_tl` to '_-'. Otherwise, with `read-arg` empty, `concept` chooses between `_\l__spintent_spfig_intent_tl:prefix(refs)`, with `prefix` placed after the leading `_` when given, and `_(refs)` alone; with `read-arg` given, it is always `_\l__spintent_spfig_intent_tl:prefix(refs)`.

```
    \bool_if:cTF { l__spintent_sp #1 _silent_bool }
      {
        \tl_set:Nn \l__spintent_spfig_intent_arg_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 `spAfig` or `spPfig`. 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 with no `intent`.

With vertices, `\__spintent_spfig_build_refs:` builds the reference list and `\__spintent_spfig_build_concep` builds the concept `intent`, applied with a single native `\MathMLintent`. Inside, `\__spintent_mathcal:n{⟨A`

| $P\rangle\}$ is silenced with `\__spintent_luafun_mrow_intent_arg:nn`; the same is done with `\l__spintent_spfig_symbol_tl` when it is non-empty; the sequence is then walked by `\__spintent_spfig_points_loop:n`. All of this is nested inside a subscript, `\c_math_subscript_token`, which the recursive lookup of the shared `mathml_filter` callback reaches through the `sub` field of the noad.

The function `\__spintent_invisible_sep:` precedes both branches. It adds a second sibling so that the sole `<mrow>` does not collapse into `<math>`.

```
\cs_new_protected:Nn \__spintent_spfig_mathml:n
  {
    \__spintent_invisible_sep:
    \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 { P } }
            \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:n { P } }
              }
              \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`.)

`\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 31.*)

15.53 The commands \sprad and \spdiam

The user-level commands `\sprad` and `\spdiam` provide a semantic interface for the radius and diameter of a circle.

15.53.1 Definition of variables for \sprad and \spdiam

`\l__spintent_sp_rad_diam_letra_tl`
`\l__spintent_sp_rad_diam_word_tl`
`\l__spintent_sp_rad_diam_read_arg_full_tl`

The variables `\l__spintent_sp_rad_diam_letra_tl` and `\l__spintent_sp_rad_diam_word_tl` are scratch variables used by `\__spintent_sp_rad_diam_mathml:n`. `\l__spintent_sp_rad_diam_read_arg_full_tl` is used by `\__spintent_sp_rad_diam_read_arg_print:n` to hold the text to voice when `read-arg` is given.

```
\tl_new:N \l__spintent_sp_rad_diam_letra_tl
\tl_new:N \l__spintent_sp_rad_diam_word_tl
\tl_new:N \l__spintent_sp_rad_diam_read_arg_full_tl
```

(*End of definition for* `\l__spintent_sp_rad_diam_letra_tl`, `\l__spintent_sp_rad_diam_word_tl`, *and* `\l__spintent_sp_rad_diam_read_arg_full_tl`.)

`\l__spintent_sprad_mode_str`
`\l__spintent_sprad_sub_index_str`
`\l__spintent_sprad_read_arg_tl`
`\l__spintent_spdiam_mode_str`
`\l__spintent_spdiam_sub_index_str`
`\l__spintent_spdiam_read_arg_tl`

Each command has its own copy of the string variables `_mode_str` and `_sub_index_str` and of the token list variable `_read_arg_tl`, which hold the values of the keys `read-cmd`, `sub-index` and `read-arg`.

```
\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.*)

15.53.2 Definition of keys for \sprad and \spdiam

`read-cmd`
`upper-case`
`sub-index`
`read-sub`
`read-arg`
`concept`
`silent`
`print-sym`
`unknown`

We define the shared ⟨*keys*⟩ `read-cmd`, `upper-case`, `sub-index`, `read-sub`, `read-arg`, `concept`, `silent` 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      .bool_set:c = { l__spintent_sp #1 _silent_bool },
      silent      .initial:n = false,
      silent      .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 key `print-sym` is defined for both commands for simplicity, but has no effect on `\sprad`.

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

15.53.3 Internal functions for `\sprad` and `\spdiam`

`\__spintent_sp_rad_diam_mathml:n`

The function `\__spintent_sp_rad_diam_mathml:n` sets `\l__spintent_sp_rad_diam_letra_tl` to the bare letter, `r` or `R` for `\sprad` and `d` or `D` for `\spdiam`, according to `upper-case`; for `\spdiam` with `print-sym=true`, it silently typesets `\diameter` first. Receives the short command name in '`#1`': `rad` or `diam`.

```
\cs_new_protected:Nn \__spintent_sp_rad_diam_mathml: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=true`, the intent is '`_-`': the letter alone with the shared queue when there is no `sub-index`, or the letter and the subscript together in one native `\MathMLintent`, with the subscript silenced, when there is.

```
    \bool_if:cTF { l__spintent_sp #1 _silent_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"* or *"diámetro"*, or the text of `read-arg` instead of that word when given, followed by the `sub-index` if present, as `sub-`⟨*value*⟩ or ⟨*value*⟩ alone according to `read-sub`; the letter is not read. With no `sub-index`, the shared queue tags the letter alone. With one, for `read-cmd=school` the letter and the subscript are wrapped together in one native `\MathMLintent` with the subscript silenced, and for `read-cmd=mathcat` the shared queue tags the letter alone and the subscript is left bare.

```
    {
      \tl_if_empty:cTF { l__spintent_sp #1 _read_arg_tl }
        {
          \bool_if:cTF { l__spintent_sp #1 _concept_bool }
            {
              \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 } }
              \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 :nofix" }
                  { \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 }
                        }
                        {
                          \__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 :nofix" }
                      { \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, but only when `read-cmd` is `school`, built in the same way from the letter. With `sub-index`, the letter and the subscript are wrapped together and the subscript is silenced. With `read-cmd=mathcat`, the intent is left empty and `MathCAT` applies its own rules to the letter and its subscript.

```
            {
              \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 -" }
                      { \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 }
                        }
                        {
                          \__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 } }
                    }
                }
            }
        }
        { \__spintent_sp_rad_diam_read_arg_print:n {#1} }
    }
}
```

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

`\__spintent_sp_rad_diam_read_arg_print:n`

With `read-arg` given, `sub-index`/`read-sub` are ignored for the *spoken* content; the subscript, if any, still displays, silenced. With `concept=true`, the default, `\l__spintent_sp_rad_diam_read_arg_full_tl` becomes *"radio"*/*"diámetro"* followed by `read-arg`'s text; with `false`, `read-arg`'s text alone. Combined natively, regardless of `read-cmd`: with no `sub-index`, the shared queue tags the letter alone; with one, letter and subscript are wrapped together in one native `\MathMLintent`.

```
\cs_new_protected:Nn \__spintent_sp_rad_diam_read_arg_print:n
  {
    \bool_if:cTF { l__spintent_sp #1 _concept_bool }
      {
        \str_if_eq:eeTF {#1} { rad }
          {
            \tl_set:Ne \l__spintent_sp_rad_diam_read_arg_full_tl
              { radio- \tl_use:c { l__spintent_sp #1 _read_arg_tl } }
          }
          {
            \tl_set:Ne \l__spintent_sp_rad_diam_read_arg_full_tl
              { diámetro- \tl_use:c { l__spintent_sp #1 _read_arg_tl } }
          }
      }
      {
        \tl_set_eq:Nc \l__spintent_sp_rad_diam_read_arg_full_tl
          { l__spintent_sp #1 _read_arg_tl }
      }
    \str_if_empty:cTF { l__spintent_sp #1 _sub_index_str }
      {
        \__spintent_luafun_mrow_intent_arg:nn
          { } { intent = "_ \l__spintent_sp_rad_diam_read_arg_full_tl :nofix" }
        { \l__spintent_sp_rad_diam_letra_tl }
      }
      {
        \MathMLintent
          { _ \l__spintent_sp_rad_diam_read_arg_full_tl :nofix }
          {
            \__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 } }
```

```
            }
        }
    }
}
```

(*End of definition for* `\__spintent_sp_rad_diam_read_arg_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_mathml:n`.

```
\cs_new_protected:Nn \__spintent_sp_rad_diam_exec:nn
  {
    \keys_set:nn { spintent / sp #1 } {#2}
    \__spintent_sp_rad_diam_mathml: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:
          \attribute \l__spintent_scope_attr_int = 1
          \__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:
          \attribute \l__spintent_scope_attr_int = 1
          \__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* 34.)

15.54 The command `\spcirc`

The user command `\spcirc` provides a semantic interface for a circle or a circumference, using the delimited-optional argument syntax `d()`. The argument holds a required center point and an optional radius, separated by a comma. The shape of the radius decides whether it is a pure number, a measure, number + unit, a bare letter or a two-letter segment, and it is processed respectively with `\spnum`, `\spunit` and `\spside`/`\spmside`.

15.54.1 Definition of variables for `\spcirc`

`\__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`. The variable `\l__spintent_geo_luaset_circ_radio_type_str` holds one of `number`, `measure`, `label`, `segment`, or an empty string when no radius was given. `\l__spintent_geo_luaset_circ_radio_raw_str` holds the corresponding raw text, ready to be forwarded to the subcommand 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_read_arg_tl`

These variables store the values set by the keys. The variable `\l__spintent_spcirc_mode_str` stores `read-cmd` and `\l__spintent_spcirc_prefix_tl` stores `prefix`. The variable `\l__spintent_spcirc_print_sym_str` stores `print-sym`, either `usual` or `odot`. The variable `\l__spintent_spcirc_read_sty_str` stores `read-sty`, either `circun` or `circulo`. The variable `\l__spintent_spcirc_radio_sty_str` stores `radio-sty`, only meaningful when the radius is a segment. The variable `\l__spintent_spcirc_sub_index_str` stores `sub-index` and `\l__spintent_spcirc_read_arg_tl` stores `read-arg`.

```
\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
\tl_new:N  \l__spintent_spcirc_read_arg_tl
```

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

`\l__spintent_spcirc_articulo_str` `\l__spintent_spcirc_word_tl`

The variable `\l__spintent_spcirc_articulo_str` is set by `\__spintent_spcirc_word_and_article:` to `la` or `el`, matching the gender of the chosen word. The variable `\l__spintent_spcirc_word_tl` stores that word, `circunferencia` or `círculo`, or the `read-arg` override; it is used by `\__spintent_spcirc_exec:nn`.

```
\str_new:N \l__spintent_spcirc_articulo_str
\tl_new:N  \l__spintent_spcirc_word_tl
```

(*End of definition for* `\l__spintent_spcirc_articulo_str` *and* `\l__spintent_spcirc_word_tl`.)

`\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`

The variable `\l__spintent_spcirc_has_arg_bool` stores whether the optional argument `d()` was given. The variable `\l__spintent_spcirc_intent_tl` stores the word `circunferencia` or `círculo`, followed by the `sub-index` suffix if present. The variable `\l__spintent_spcirc_connector_tl` stores `de-centro-en`. The variables `\l__spintent_spcirc_intent_arg_tl` and `\l__spintent_spcirc_connector_arg_tl` store the `intent` passed to `\MathMLintent` for the word and for the connector, where the center is referenced as `$pt`. The variable `\l__spintent_spcirc_radio_arg_tl` stores the `intent` of the radius, spoken on the delimiter `,`.

```
\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.*)

15.54.2 Definition of error messages for `\spcirc`

`spcirc-invalid-arg`

The "error" `spcirc-invalid-arg` is 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~'#1'~for~'\token_to_str:N \spcirc'~\msg_line_context:.
  }
  {
    The~argument~must~have~the~form~'(center)'~or~'(center,~radius)'.
  }
```

(*End of definition for* `spcirc-invalid-arg`.)

15.54.3 Definition of keys for `\spcirc`

`read-cmd` `prefix` `print-sym` `read-sty` `diameter` `radio-sty` `sub-index` `read-sub` `read-arg` `silent` `concept` `spside` `spmside` `unknown`

We define the ⟨*keys*⟩ `read-cmd`, `prefix`, `print-sym`, `read-sty`, `diameter`, `radio-sty`, `sub-index`, `read-sub`, `read-arg`, `silent` and `concept`.

```
\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 }
                \keys_set:nn { spintent / spside  } { read-cmd = school }
                \keys_set:nn { spintent / spmside } { read-cmd = school }
              }
            {2}
              {
                \str_set:Nn \l__spintent_spcirc_mode_str { mathcat }
                \keys_set:nn { spintent / spside  } { read-cmd = mathcat }
                \keys_set:nn { spintent / spmside } { read-cmd = 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_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,
  print-sym / unknown .code:n =
    \msg_error:nneee { spintent } { unknown-choice }
      { print-sym } { usual,~odot } { \exp_not:n {#1} },
  read-sty  .choices:nn = { circun , circulo }
    {
      \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 { circulo } }
        }
    },
  read-sty  .initial:n = circun,
  read-sty  .value_required:n = true,
  read-sty / unknown .code:n =
    \msg_error:nneee { spintent } { unknown-choice }
      { read-sty } { circun,~circulo } { \exp_not:n {#1} },
  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,
  radio-sty / unknown .code:n =
    \msg_error:nneee { spintent } { unknown-choice }
      { radio-sty } { side,~mside } { \exp_not:n {#1} },
  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    .bool_set:N = \l__spintent_spcirc_silent_bool,
  silent    .initial:n = false,
  silent    .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.*)

15.54.4 Internal functions for `\spcirc`

`\__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:NV \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:` Branches 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 }
```

```
6907                { \exp_args:Ne \spside  { \l__spintent_geo_luaset_circ_radio_raw_str } }
6908                { \exp_args:Ne \spmside { \l__spintent_geo_luaset_circ_radio_raw_str } }
6909              }
6910          }
6911      }
```

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

`\__spintent_spcirc_build_concept:`

Builds the words `\l__spintent_spcirc_intent_tl`, with `\__spintent_spcirc_word_and_article:` and an optional `sub-index` suffix, and `\l__spintent_spcirc_connector_tl`, `de-centro-en`. Both are built even when `\l__spintent_spcirc_has_arg_bool` is `false`, although `\__spintent_spcirc_mathml:` does not use them in that case. The center is not concatenated into them: it is read by reference with `\__spintent_mathmlarg_point:nnn`.

```
6912  \cs_new_protected:Nn \__spintent_spcirc_build_concept:
6913    {
6914      \__spintent_spcirc_word_and_article:
6915      \tl_set:Ne \l__spintent_spcirc_intent_tl { \l__spintent_spcirc_word_tl }
6916      \str_if_empty:NF \l__spintent_spcirc_sub_index_str
6917        {
6918          \tl_put_right:Ne \l__spintent_spcirc_intent_tl
6919            {
6920              -
6921              \bool_if:NT \l__spintent_spcirc_read_sub_bool { sub - }
6922              \l__spintent_spcirc_sub_index_str
6923            }
6924        }
6925      \tl_set:Nn \l__spintent_spcirc_connector_tl { de-centro-en }
```

The key `silent`, when `true`, overrides `\l__spintent_spcirc_intent_arg_tl`, `\l__spintent_spcirc_connect` and `\l__spintent_spcirc_radio_arg_tl` with '`_-`', regardless of `read-arg`.

```
6926      \bool_if:NTF \l__spintent_spcirc_silent_bool
6927        {
6928          \tl_set:Nn \l__spintent_spcirc_intent_arg_tl     { _- }
6929          \tl_set:Nn \l__spintent_spcirc_connector_arg_tl  { _- }
6930          \tl_set:Nn \l__spintent_spcirc_radio_arg_tl      { _- }
6931        }
```

With `concept=true`, `\l__spintent_spcirc_intent_arg_tl` is `_\l__spintent_spcirc_intent_tl` alone, with `prefix` placed after the leading `_` when given; `\l__spintent_spcirc_connector_arg_tl` is `_\l__spintent_spcirc_connector_tl:prefix($pt)`; and `\l__spintent_spcirc_radio_arg_tl` is the literal `_y-radio-`, or `_y-diámetro-` when `diameter` is `true`, spoken on the delimiter `,` before the reading of `\__spintent_spcirc_radio_print:`.

```
6932        {
6933          \bool_if:NTF \l__spintent_spcirc_concept_bool
6934            {
6935              \tl_set:Ne \l__spintent_spcirc_intent_arg_tl
6936                {
6937                  _
6938                  \tl_if_empty:NF \l__spintent_spcirc_prefix_tl
6939                    { \tl_use:N \l__spintent_spcirc_prefix_tl - }
6940                  \l__spintent_spcirc_intent_tl
6941                }
6942              \tl_set:Ne \l__spintent_spcirc_connector_arg_tl
6943                { _ \l__spintent_spcirc_connector_tl :prefix( $ pt ) }
6944              \bool_if:NTF \l__spintent_spcirc_diameter_bool
6945                { \tl_set:Nn \l__spintent_spcirc_radio_arg_tl { _y-diámetro- } }
6946                { \tl_set:Nn \l__spintent_spcirc_radio_arg_tl { _y-radio- } }
6947            }
```

With `concept=false` and `read-arg` empty, the same '`_-`' and `_($pt)` silencing applies as with `silent`. With `read-arg` given, none of the three is silenced; the connector and the words `y-radio` and `y-diámetro` are fixed and `read-arg` does not change them.

```
6948            {
6949              \tl_if_empty:NTF \l__spintent_spcirc_read_arg_tl
6950                {
6951                  \tl_set:Nn \l__spintent_spcirc_intent_arg_tl    { _- }
6952                  \tl_set:Ne \l__spintent_spcirc_connector_arg_tl { _( $ pt ) }
6953                  \tl_set:Nn \l__spintent_spcirc_radio_arg_tl     { _- }
6954                }
```

```
6955                {
6956                  \tl_set:Ne \l__spintent_spcirc_intent_arg_tl
6957                    { _ \l__spintent_spcirc_intent_tl }
6958                  \tl_set:Ne \l__spintent_spcirc_connector_arg_tl
6959                    { _ \l__spintent_spcirc_connector_tl :prefix( $ pt ) }
6960                  \bool_if:NTF \l__spintent_spcirc_diameter_bool
6961                    { \tl_set:Nn \l__spintent_spcirc_radio_arg_tl { _y-diámetro- } }
6962                    { \tl_set:Nn \l__spintent_spcirc_radio_arg_tl { _y-radio- } }
6963                }
6964            }
6965        }
6966    }
```

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

`\__spintent_spcirc_exec:nn`

Receives the `d()` argument in '`#2`', which is an explicit empty group when omitted because `\spcirc` converts `-NoValue-` to `{ }` before calling `\__spintent_spcirc_exec:nn`. When the group is empty, the function sets `\l__spintent_spcirc_has_arg_bool` to `false` and does not call `\__spintent_luafun_geo_circ_parse_and_set:n`. Otherwise it calls that function, reports `spcirc-invalid-arg` on failure and delegates to `\__spintent_spcirc_mathml:`, which calls `\__spintent_spcirc_build_concept:`.

Before processing the {⟨*argument*⟩}, it sets the optional ⟨*keys*⟩ passed in '`#1`'.

```
6967 \cs_new_protected:Nn \__spintent_spcirc_exec:nn
6968   {
6969     \keys_set:nn { spintent / spcirc } {#1}
6970     \tl_if_empty:nTF {#2}
6971       {
6972         \bool_set_false:N \l__spintent_spcirc_has_arg_bool
6973       }
6974       {
6975         \bool_set_true:N \l__spintent_spcirc_has_arg_bool
6976         \__spintent_luafun_geo_circ_parse_and_set:n { \exp_not:n {#2} }
6977         \str_if_eq:VnT \l__spintent_geo_luaset_error_str { true }
6978           {
6979             \msg_error:nne { spintent } { spcirc-invalid-arg } { \exp_not:n {#2} }
6980           }
6981       }
6982     \__spintent_spcirc_mathml:
6983   }
```

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

`\__spintent_spcirc_mathml:`

Calls `\__spintent_spcirc_build_concept:` and wraps the symbol `\l__spintent_spcirc_intent_arg_tl` in its own native `\MathMLintent`. This is the whole result when `\l__spintent_spcirc_has_arg_bool` is `false`, that is, for `\spcirc` with no `(...)`.

Otherwise, the group `(center)`, held in `\l__spintent_spcirc_connector_arg_tl`, follows as a separate sibling, outside the `\MathMLintent` of the symbol. When a radius is present, the delimiter `,` after the center carries `\l__spintent_spcirc_radio_arg_tl` and `\__spintent_spcirc_radio_print:` follows as a sibling; with `silent=true`, it is wrapped in a literal '`_-`' `\MathMLintent` instead.

The symbol and, when `sub-index` is given, its subscript number are silenced separately with `\__spintent_luafun_mrow_intent_arg:nn`, one queue entry per leaf. The delimiter `(` is silenced in the same way but without the usual `{...}` around the character, so that `luamml` keeps its math class and produces `<mo>` instead of `<mi>`.

The final `)`, which follows `\__spintent_spcirc_radio_print:`, is typeset as `\rparen` inside its own native `\MathMLintent{:silent}`, outside the shared queue.

The center itself is read through `\__spintent_mathmlarg_point:nnn`, referenced as `$pt`.

```
6984 \cs_new_protected:Nn \__spintent_spcirc_mathml:
6985   {
6986     \__spintent_invisible_sep:
6987     \__spintent_spcirc_build_concept:
6988     \MathMLintent { \l__spintent_spcirc_intent_arg_tl }
6989       {
6990         \__spintent_inner_sep:
6991         \__spintent_luafun_mrow_intent_arg:nn
6992           { } { intent = ":silent" }
6993         { \__spintent_spcirc_sym: }
6994         \str_if_empty:NF \l__spintent_spcirc_sub_index_str
```

©2026 by Pablo González L

```
            {
              \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" } \lparen
                  \__spintent_mathmlarg_point:nnV { pt } { spcirc }
                    \l__spintent_geo_luaset_points_clist
                  \__spintent_luafun_mrow_intent_arg:nn { } { intent = ":silent" } \rparen
                }
            }
            {
              \MathMLintent { \l__spintent_spcirc_connector_arg_tl }
                {
                  \__spintent_inner_sep:
                  \__spintent_luafun_mrow_intent_arg:nn { } { intent = ":silent" } \lparen
                  \__spintent_mathmlarg_point:nnV { pt } { spcirc }
                    \l__spintent_geo_luaset_points_clist
                }
              \MathMLintent { \l__spintent_spcirc_radio_arg_tl } { , }
              \bool_if:NTF \l__spintent_spcirc_silent_bool
                {
                  \MathMLintent { _- }
                    {
                      \__spintent_inner_sep:
                      \__spintent_spcirc_radio_print:
                    }
                }
                { \__spintent_spcirc_radio_print: }
              \MathMLintent { :silent } { \rparen }
            }
        }
    }
```

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

`\spcirc` The public command `\spcirc` enforces math mode, opens a local group, and delegates execution to `\__spintent_spcirc_exec:nn`. An omitted radius forwards an explicit empty group instead of 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* 35.)

15.55 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.

15.55.1 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_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, 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 _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_spcircfig_luaset_comma_str

The variable \l__spintent_spcircfig_luaset_comma_str is set by __spintent_luafun_spcircfig_comma:n and stores true when the argument has a comma.

```
\str_new:N \l__spintent_spcircfig_luaset_comma_str
```

(*End of definition for* \l__spintent_spcircfig_luaset_comma_str.)

\l__spintent_spcircfig_short_bool

The variable \l__spintent_spcircfig_short_bool stores whether the argument was resolved as the short number or letter form, with __spintent_luafun_geo_afig_parse_and_set:n or __spintent_luafun_geo_pfig_parse_and_set:n, and not as a circle.

```
\bool_new:N \l__spintent_spcircfig_short_bool
```

(*End of definition for* \l__spintent_spcircfig_short_bool.)

\l__spintent_spcircfig_op_tl
\l__spintent_spcircfig_del_word_tl
\l__spintent_spcircfig_intent_tl
\l__spintent_spcircfig_intent_arg_tl

The variable \l__spintent_spcircfig_op_tl stores *«área»* or *«perímetro»*. The variable \l__spintent_spcircfig_del_word_tl, set by __spintent_spcircfig_del_word:n, stores the word that follows the operator. The variable \l__spintent_spcircfig_intent_tl stores the spoken phrase: that word, an optional sub-index suffix and, when a center is given, -con-centro-en; the center is read by reference with __spintent_mathmlarg_point:nnn. The variable \l__spintent_spcircfig_intent_arg_tl stores the intent being built.

```
\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_op_tl *and others.*)

15.55.2 Definition of error messages for \spAcirc and \spPcirc

spAcirc-invalid-arg
spPcirc-invalid-arg

The "error"s spAcirc-invalid-arg and spPcirc-invalid-arg are raised 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~'##2'~for~'\tl_trim_spaces:n {##1}'~\msg_line_context:.
      }
      {
        See~the~manual~for~the~accepted~argument~forms.
      }
```

(*End of definition for* spAcirc-invalid-arg *and* spPcirc-invalid-arg.)

15.55.3 Definition of keys for \spAcirc and \spPcirc

read-cmd
prefix
print-sym
read-sty
concept
sub-index
read-sub
read-arg
silent

We define the shared ⟨*keys*⟩ read-cmd, prefix, print-sym, read-sty, concept, sub-index, read-sub, read-arg and silent.

```
    \keys_define:nn { spintent / sp #1 }
      {
        read-cmd  .choices:nn = { school , mathcat }
```

©2026 by Pablo González L

```
7079            {
7080              \int_case:nn { \l_keys_choice_int }
7081                {
7082                  {1} { \str_set:cn { l__spintent_sp #1 _mode_str } { school  } }
7083                  {2} { \str_set:cn { l__spintent_sp #1 _mode_str } { mathcat } }
7084                }
7085            },
7086          read-cmd  .initial:n = school,
7087          read-cmd  .value_required:n = true,
7088          read-cmd / unknown .code:n =
7089            \msg_error:nneee { spintent } { unknown-choice }
7090              { read-cmd } { school,~mathcat } { \exp_not:n {##1} },
7091          prefix    .code:n =
7092            { \__spintent_sanitize_affix:cn { l__spintent_sp #1 _prefix_tl } {##1} },
7093          prefix    .initial:n = ,
7094          prefix    .value_required:n = true,
7095          print-sym .choices:nn = { usual , odot }
7096            {
7097              \int_case:nn { \l_keys_choice_int }
7098                {
7099                  {1} { \str_set:cn { l__spintent_sp #1 _print_sym_str } { usual } }
7100                  {2} { \str_set:cn { l__spintent_sp #1 _print_sym_str } { odot  } }
7101                }
7102            },
7103          print-sym .initial:n = odot,
7104          print-sym .value_required:n = true,
7105          print-sym / unknown .code:n =
7106            \msg_error:nneee { spintent } { unknown-choice }
7107              { print-sym } { usual,~odot } { \exp_not:n {##1} },
7108          read-sty  .choices:nn = { circun , circulo }
7109            {
7110              \int_case:nn { \l_keys_choice_int }
7111                {
7112                  {1} { \str_set:cn { l__spintent_sp #1 _read_sty_str } { circun } }
7113                  {2} { \str_set:cn { l__spintent_sp #1 _read_sty_str } { circulo } }
7114                }
7115            },
7116          read-sty  .initial:n = circun,
7117          read-sty  .value_required:n = true,
7118          read-sty / unknown .code:n =
7119            \msg_error:nneee { spintent } { unknown-choice }
7120              { read-sty } { circun,~circulo } { \exp_not:n {##1} },
7121          concept   .bool_set:c = { l__spintent_sp #1 _concept_bool },
7122          concept   .initial:n = true,
7123          concept   .value_required:n = true,
7124          sub-index .code:n =
7125            { \str_set:cn { l__spintent_sp #1 _sub_index_str } {##1} },
7126          sub-index .initial:n = ,
7127          sub-index .value_required:n = true,
7128          read-sub  .bool_set:c = { l__spintent_sp #1 _read_sub_bool },
7129          read-sub  .initial:n = false,
7130          read-sub  .value_required:n = true,
7131          read-arg  .code:n =
7132            { \__spintent_sanitize_affix:cn { l__spintent_sp #1 _read_arg_tl } {##1} },
7133          read-arg  .initial:n = ,
7134          read-arg  .value_required:n = true,
7135          silent    .bool_set:c = { l__spintent_sp #1 _silent_bool },
7136          silent    .initial:n = false,
7137          silent    .value_required:n = true,
7138          unknown   .code:n =
7139            {
7140              \str_set:Nn \l__spintent_command_name_str { sp #1 }
7141              \__spintent_unknown_keys_cmd:n {##1}
7142            },
7143        }
7144    }
```

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

15.55.4 Internal functions for \spAcirc and \spPcirc

__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 the word of __spintent_spcircfig_word_and_article:n, prefixed with del- or de-la-. 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 or letter case: builds \l__spintent_spcircfig_intent_arg_tl as _\l__spintent_spcircfig_op_tl-⟨*number word*⟩, with prefix placed after the leading _ when given and without :prefix, or as '_-' when silent 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_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
```

©2026 by Pablo González L

```
                {
                  _
                  \tl_if_empty:cF { l__spintent_sp #1 _prefix_tl }
                    { \tl_use:c { l__spintent_sp #1 _prefix_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
                }
            }
            {
              \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_intent_tl`. 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`. It is '`_-`' when `silent` is `true`, whatever `#2` is. With `concept=false` and `read-arg` empty, it is '`_-`', or `_(`$pt`)` when `#2` is non-empty. Otherwise it is `_\l__spintent_spcircfig_intent_tl`, with `:prefix(`$pt`)` appended only when `#2` is non-empty and with `prefix` placed after the leading `_` when given. With `concept=true`, `\l__spintent_spcircfig_intent_tl` is `\l__spintent_spcircfig_op_tl-\l__spintent_spcircfig_del_w` set by `\__spintent_spcircfig_del_word:n`; with `concept=false` and `read-arg` given, it is `read-arg` as given, without `\l__spintent_spcircfig_op_tl`, and the user writes the article if needed. In both cases 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_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_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
              \tl_if_empty:nF {#2} { :prefix ( $ 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
                  \tl_if_empty:nF {#2} { :prefix ( $ pt ) }
                }
            }
        }
    }
}
```

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

`\__spintent_spcircfig_exec:nnn`

Receives the short command name in '`#1`' and the argument in '`#3`'. Sets `\l__spintent_spcircfig_op_tl` to *«área»* or *«perímetro»*. With `#3` empty, it sets `\l__spintent_spcircfig_short_bool` to `false` and parses nothing else. With a comma in `#3`, it reports `spAcirc-invalid-arg` or `spPcirc-invalid-arg`. Without a comma, it calls `\__spintent_luafun_geo_circ_parse_and_set:n`; on error, it tries `\__spintent_luafun_geo_afig_parse_and_set:n` or `\__spintent_luafun_geo_pfig_parse_and_set:n` for the short number or letter form, and a table-letter match is an error. It reports `spAcirc-invalid-arg` or `spPcirc-invalid-arg` on failure, and then delegates to `\__spintent_spcircfig_build_short:n` or `\__spintent_spcircfig_build_concept:nn` and to `\__spintent_spcircfig_mathml:nn`.

The commands `\spAcirc` and `\spPcirc` take no radius or diameter: the argument names the figure, `A_0`, and the values needed to compute it are written elsewhere in the document.

Before processing the {⟨*argument*⟩}, it sets the optional ⟨*keys*⟩ passed in '`#2`'.

```
\cs_new_protected:Nn \__spintent_spcircfig_exec:nnn
  {
    \keys_set:nn { spintent / sp #1 } {#2}
    \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 {#3}
      {
        \bool_set_false:N \l__spintent_spcircfig_short_bool
      }
      {
        \bool_set_false:N \l__spintent_spcircfig_short_bool
        \__spintent_luafun_spcircfig_comma:n { \exp_not:n {#3} }
        \str_if_eq:VnTF \l__spintent_spcircfig_luaset_comma_str { true }
          {
            \msg_error:nnee { spintent } { sp #1 -invalid-arg }
              { \c_backslash_str sp #1 } { \exp_not:n {#3} }
          }
          {
            \__spintent_luafun_geo_circ_parse_and_set:n { \exp_not:n {#3} }
            \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 {#3} } }
                  { \__spintent_luafun_geo_pfig_parse_and_set:n { \exp_not:n {#3} } }
                \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 {#3} }
                }
            }
        }
    }
  \bool_if:NTF \l__spintent_spcircfig_short_bool
    { \__spintent_spcircfig_build_short:n {#1} }
    { \__spintent_spcircfig_build_concept:nn {#1} {#3} }
  \__spintent_spcircfig_mathml:nn {#1} {#3}
}
```

(*End of definition for* `\__spintent_spcircfig_exec:nnn`.)

`\__spintent_spcircfig_mathml:nn`

Renders `\l__spintent_spcircfig_intent_arg_tl`, already built by `\__spintent_spcircfig_exec:nnn`. There are three cases: the short case prints the symbol with the short value as its subscript; an empty `#2` prints the symbol alone; a center prints the symbol, plus the optional `sub-index` subscript, and the center between parentheses. In every case a single native `\MathMLintent` wraps `\__spintent_mathcal:n{⟨A | P⟩}` and the whole subscript. The center is read with `\__spintent_mathmlarg_point:nnn`, referenced as `$pt`. Receives the short command name in '`#1`' and the argument in '`#2`'.

```
\cs_new_protected:Nn \__spintent_spcircfig_mathml:nn
  {
    \__spintent_invisible_sep:
    \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 { P } }
        }
        \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_points_clist
                    \MathMLintent { :silent } { \rparen }
                  }
              }
          }
      }
  }
```

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

`\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 36.*)

15.56 The command `\spVol`

The user-level command `\spVol` provides a semantic interface for the volume of a solid, always typesetting `\mathcal{V}`.

15.56.1 Definition of variables for `\spVol`

`\l__spintent_spVol_mode_str` `\l__spintent_spVol_sub_index_str` `\l__spintent_spVol_read_arg_tl` The variable `\l__spintent_spVol_mode_str` stores the value of the key `read-cmd`, `school` or `mathcat`. The variable `\l__spintent_spVol_sub_index_str` stores the raw text of `sub-index` and `\l__spintent_spVol_read_arg_tl` the sanitized text of `read-arg`.

```
\str_new:N \l__spintent_spVol_mode_str
\str_new:N \l__spintent_spVol_sub_index_str
\tl_new:N  \l__spintent_spVol_read_arg_tl
```

(*End of definition for* `\l__spintent_spVol_mode_str`, `\l__spintent_spVol_sub_index_str`, *and* `\l__spintent_spVol_read_arg_tl`.)

`\l__spintent_spVol_word_tl` `\l__spintent_spVol_suffix_tl` `\l__spintent_spVol_native_bool` The variable `\l__spintent_spVol_word_tl` stores the word to voice, *«volumen»*, and `\l__spintent_spVol_suffix_tl` the text attached after it. The variable `\l__spintent_spVol_native_bool` records whether that text is combined natively with the subscript, silenced, or left for `MathCAT` to read on its own.

```
\tl_new:N   \l__spintent_spVol_word_tl
\tl_new:N   \l__spintent_spVol_suffix_tl
\bool_new:N \l__spintent_spVol_native_bool
```

(*End of definition for* `\l__spintent_spVol_word_tl`, `\l__spintent_spVol_suffix_tl`, *and* `\l__spintent_spVol_native_bool`.)

`\l__spintent_spVol_read_arg_full_tl` The variable `\l__spintent_spVol_read_arg_full_tl` stores the text to voice when `read-arg` is given.

```
\tl_new:N \l__spintent_spVol_read_arg_full_tl
```

(*End of definition for* `\l__spintent_spVol_read_arg_full_tl`.)

15.56.2 Definition of keys for \spVol

read-cmd
sub-index
read-sub
read-arg
concept
silent
unknown

We define the ⟨*keys*⟩ read-cmd, sub-index, read-sub, read-arg, concept and silent.

```
\keys_define:nn { spintent / spVol }
  {
    read-cmd  .choices:nn = { school , mathcat }
      {
        \int_case:nn { \l_keys_choice_int }
          {
            {1} { \str_set:Nn \l__spintent_spVol_mode_str { school  } }
            {2} { \str_set:Nn \l__spintent_spVol_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} },
    sub-index .code:n =
      { \str_set:Nn \l__spintent_spVol_sub_index_str {#1} },
    sub-index .initial:n = ,
    sub-index .value_required:n = true,
    read-sub  .bool_set:N = \l__spintent_spVol_read_sub_bool,
    read-sub  .initial:n = false,
    read-sub  .value_required:n = true,
    read-arg  .code:n =
      { \__spintent_sanitize_affix:Nn \l__spintent_spVol_read_arg_tl {#1} },
    read-arg  .initial:n = ,
    read-arg  .value_required:n = true,
    concept   .bool_set:N = \l__spintent_spVol_concept_bool,
    concept   .initial:n = true,
    concept   .value_required:n = true,
    silent    .bool_set:N = \l__spintent_spVol_silent_bool,
    silent    .initial:n = false,
    silent    .value_required:n = true,
    unknown   .code:n =
      {
        \str_set:Nn \l__spintent_command_name_str { spVol }
        \__spintent_unknown_keys_cmd:n {#1}
      },
  }
```

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

15.56.3 Internal functions for \spVol

__spintent_spVol_mathml:

The function __spintent_spVol_mathml: calls __spintent_spVol_silent: when silent is true, or __spintent_spVol_verbal: otherwise.

```
\cs_new_protected:Nn \__spintent_spVol_mathml:
  {
    \__spintent_invisible_sep:
    \bool_if:NTF \l__spintent_spVol_silent_bool
      { \__spintent_spVol_silent: }
      { \__spintent_spVol_verbal: }
  }
```

(*End of definition for* __spintent_spVol_mathml:.)

__spintent_spVol_silent:

The function __spintent_spVol_silent: typesets \mathcal{V} with the intent '_-': the symbol alone with the shared queue when there is no sub-index, or the symbol and the subscript together in one native \MathMLintent, with the subscript silenced, when there is.

```
\cs_new_protected:Nn \__spintent_spVol_silent:
  {
    \str_if_empty:NTF \l__spintent_spVol_sub_index_str
      {
        \__spintent_luafun_mrow_intent_arg:nn
          { } { intent = "_-" }
        { \mathcal{V} }
      }
      {
```

```
      \MathMLintent { _- }
        {
          \__spintent_inner_sep:
          \mathcal{V}
          \c_math_subscript_token
            {
              \MathMLintent { :silent }
                { \l__spintent_spVol_sub_index_str }
            }
        }
    }
```

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

`\__spintent_spVol_verbal:`

The function `\__spintent_spVol_verbal:` delegates to `\__spintent_spVol_read_arg_print:` when `read-arg` is given and, otherwise, with `concept=false`, to `\__spintent_spVol_symbol_print:`. With `concept=true` it sets `\l__spintent_spVol_word_tl` to *«volumen»* and `\l__spintent_spVol_suffix_tl` to sub-⟨*value*⟩ or ⟨*value*⟩, according to `read-sub`; the suffix is combined natively only with `read-cmd=school`, and with `read-cmd=mathcat` it is left to MathCAT. It delegates the typesetting to `\__spintent_spVol_attach_subscript:`.

```
\cs_new_protected:Nn \__spintent_spVol_verbal:
  {
    \tl_if_empty:NTF \l__spintent_spVol_read_arg_tl
      {
        \bool_if:NTF \l__spintent_spVol_concept_bool
          {
            \tl_set:Nn \l__spintent_spVol_word_tl { volumen }
            \str_if_eq:VnTF \l__spintent_spVol_mode_str { school }
              { \bool_set_true:N  \l__spintent_spVol_native_bool }
              { \bool_set_false:N \l__spintent_spVol_native_bool }
            \tl_set:Ne \l__spintent_spVol_suffix_tl
              {
                \bool_if:NT \l__spintent_spVol_read_sub_bool { sub- }
                \l__spintent_spVol_sub_index_str
              }
            \__spintent_spVol_attach_subscript:
          }
          { \__spintent_spVol_symbol_print: }
      }
      { \__spintent_spVol_read_arg_print: }
  }
```

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

`\__spintent_spVol_symbol_print:`

With `concept=false`, the symbol itself is read, but only when `read-cmd` is `school`: with no `sub-index`, the shared queue tags the symbol with the intent `_V-`; with one, the symbol and the subscript are wrapped together in one native `\MathMLintent`, with the subscript silenced. With `read-cmd=mathcat`, the intent is left empty and MathCAT applies its own rules to the symbol and its subscript.

```
\cs_new_protected:Nn \__spintent_spVol_symbol_print:
  {
    \str_if_empty:NTF \l__spintent_spVol_sub_index_str
      {
        \str_if_eq:VnTF \l__spintent_spVol_mode_str { school }
          {
            \__spintent_luafun_mrow_intent_arg:nn
              { } { intent = "_V-" }
              { \mathcal{V} }
          }
          { \mathcal{V} }
      }
      {
        \str_if_eq:VnTF \l__spintent_spVol_mode_str { school }
          {
            \MathMLintent
              {
                _V-
                \bool_if:NT \l__spintent_spVol_read_sub_bool { sub- }
                \l__spintent_spVol_sub_index_str
              }
```

```
                {
                  \__spintent_inner_sep:
                  \mathcal{V}
                  \c_math_subscript_token
                    {
                      \MathMLintent { :silent }
                        { \l__spintent_spVol_sub_index_str }
                    }
                }
            }
            {
              \mathcal{V}
              \c_math_subscript_token
                { \l__spintent_spVol_sub_index_str }
            }
        }
    }
```

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

`\__spintent_spVol_attach_subscript:`

The function `\__spintent_spVol_attach_subscript:` typesets `\mathcal{V}`. An empty `sub-index` tags the symbol alone through the shared queue, and `\l__spintent_spVol_suffix_tl` is discarded since there is no subscript to attach it to. A given one combines symbol, subscript, and `\l__spintent_spVol_suffix_tl` in one native `\MathMLintent`, subscript silenced, when `\l__spintent_spVol_native_bool` is `true`, or tags the symbol alone through the shared queue with the subscript left bare when `false`.

```
\cs_new_protected:Nn \__spintent_spVol_attach_subscript:
  {
    \str_if_empty:NTF \l__spintent_spVol_sub_index_str
      {
        \__spintent_luafun_mrow_intent_arg:nn
          { } { intent = "_ \l__spintent_spVol_word_tl :nofix" }
        { \mathcal{V} }
      }
      {
        \bool_if:NTF \l__spintent_spVol_native_bool
          {
            \MathMLintent
              { _ \l__spintent_spVol_word_tl - \l__spintent_spVol_suffix_tl :nofix }
              {
                \__spintent_inner_sep:
                \mathcal{V}
                \c_math_subscript_token
                  {
                    \MathMLintent { :silent }
                      { \l__spintent_spVol_sub_index_str }
                  }
              }
          }
          {
            \__spintent_luafun_mrow_intent_arg:nn
              { } { intent = "_ \l__spintent_spVol_word_tl :nofix" }
            { \mathcal{V} }
            \c_math_subscript_token
              { \l__spintent_spVol_sub_index_str }
          }
      }
  }
```

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

`\__spintent_spVol_read_arg_print:`

With `read-arg` given, `sub-index`/`read-sub` are ignored for the *spoken* content; the subscript, if any, still displays, silenced. With `concept=true`, `\l__spintent_spVol_read_arg_full_tl` becomes *«volumen»* followed by `read-arg`'s text; with `false`, `read-arg`'s text alone. Combined natively regardless of `read-cmd`: with no `sub-index`, the shared queue tags the symbol alone; with one, symbol and subscript are wrapped together in one native `\MathMLintent`.

```
\cs_new_protected:Nn \__spintent_spVol_read_arg_print:
  {
    \bool_if:NTF \l__spintent_spVol_concept_bool
      {
```

```
      \tl_set:Ne \l__spintent_spVol_read_arg_full_tl
        { volumen- \tl_use:N \l__spintent_spVol_read_arg_tl }
    }
    {
      \tl_set_eq:NN \l__spintent_spVol_read_arg_full_tl \l__spintent_spVol_read_arg_tl
    }
  \str_if_empty:NTF \l__spintent_spVol_sub_index_str
    {
      \__spintent_luafun_mrow_intent_arg:nn
        { } { intent = "_ \l__spintent_spVol_read_arg_full_tl :nofix" }
      { \mathcal{V} }
    }
    {
      \MathMLintent
        { _ \l__spintent_spVol_read_arg_full_tl :nofix }
        {
          \__spintent_inner_sep:
          \mathcal{V}
          \c_math_subscript_token
            {
              \MathMLintent { :silent }
                { \l__spintent_spVol_sub_index_str }
            }
        }
    }
}
```

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

`\spVol` The public command `\spVol` enforces *math mode*, opens a local group, and delegates execution to `\__spintent_spVol_mathml:`.

```
\NewDocumentCommand \spVol { O{} }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \attribute \l__spintent_scope_attr_int = 1
          \keys_set:nn { spintent / spVol } {#1}
          \__spintent_spVol_mathml:
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spVol } }
  }
```

(*End of definition for* `\spVol`. *This function is documented on page* 39.)

15.57 The command `\spProm`

The user command `\spProm` typesets its {⟨*argument*⟩} with a bar on top, `\bar`, the usual notation for a mean, and reads it *«promedio de»* followed by the argument. The argument is not parsed: it is typeset as given and read by `MathCAT` with its own rules.

15.57.1 Definition of variables for `\spProm`

`\l__spintent_spProm_read_arg_tl` `\l__spintent_spProm_intent_tl` The variable `\l__spintent_spProm_read_arg_tl` holds the normalized text of the `read-arg` key, and `\l__spintent_spProm_intent_tl` holds the name of the intent used by `\__spintent_spProm_exec:nn`.

```
\tl_new:N \l__spintent_spProm_read_arg_tl
\tl_new:N \l__spintent_spProm_intent_tl
```

(*End of definition for* `\l__spintent_spProm_read_arg_tl` *and* `\l__spintent_spProm_intent_tl`.)

15.57.2 Definition of keys for `\spProm`

`silent` `read-arg` `unknown` We define the ⟨*keys*⟩ `silent` and `read-arg`.

```
\keys_define:nn { spintent / spProm }
  {
    silent   .bool_set:N = \l__spintent_spProm_silent_bool,
    silent   .initial:n = false,
    silent   .value_required:n = true,
    read-arg .code:n =
      { \__spintent_sanitize_affix:Nn \l__spintent_spProm_read_arg_tl {#1} },
```

```
    read-arg .initial:n = ,
    read-arg .value_required:n = true,
    unknown  .code:n =
      {
        \str_set:Nn \l__spintent_command_name_str { spProm }
        \__spintent_unknown_keys_cmd:n {#1}
      },
  }
```

(*End of definition for* `silent` , `read-arg` , *and* `unknown`.)

15.57.3 Internal functions for `\spProm`

`\__spintent_spProm_exec:nn`

The function `\__spintent_spProm_exec:nn` receives the optional ⟨*keys*⟩ in '`#1`' and the {⟨*argument*⟩} in '`#2`'. It sets the ⟨*keys*⟩ and calls `\__spintent_spProm_mathml:n`.

```
\cs_new_protected:Nn \__spintent_spProm_exec:nn
  {
    \keys_set:nn { spintent / spProm } {#1}
    \__spintent_spProm_mathml:n {#2}
  }
```

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

`\__spintent_spProm_mathml:n`

The function `\__spintent_spProm_mathml:n` receives the {⟨*argument*⟩} in '`#1`'. Under `silent` the whole expression is silenced as one unit with the intent '`_-`'.

Otherwise the {⟨*argument*⟩} is tagged through a native `\MathMLarg`, `a`, inside `\bar`, and the outer `\MathMLintent` reads `_promedio-de:prefix($a)`. When `read-arg` is not empty, its text replaces `promedio-de`. The inner ghost `\__spintent_inner_sep:` keeps the content of the intent from collapsing into a single child; a second one inside the `\MathMLarg` does the same for the argument, whose `arg` attribute would otherwise be lost when the argument is a single compound element, such as a letter with a subscript.

```
\cs_new_protected:Nn \__spintent_spProm_mathml:n
  {
    \__spintent_invisible_sep:
    \bool_if:NTF \l__spintent_spProm_silent_bool
      {
        \MathMLintent { _- }
          {
            \__spintent_inner_sep:
            \bar {#1}
          }
      }
      {
        \tl_if_empty:NTF \l__spintent_spProm_read_arg_tl
          { \tl_set:Nn \l__spintent_spProm_intent_tl { promedio-de } }
          {
            \tl_set_eq:NN \l__spintent_spProm_intent_tl \l__spintent_spProm_read_arg_tl
          }
        \MathMLintent { _ \l__spintent_spProm_intent_tl :prefix($a) }
          {
            \__spintent_inner_sep:
            \bar { \MathMLarg { a } { \__spintent_inner_sep: #1 } }
          }
      }
  }
```

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

`\spProm`

The public command `\spProm` enforces *math mode*, opens a local group, and delegates execution to `\__spintent_spProm_exec:nn`.

```
\NewDocumentCommand \spProm { O{} m }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \attribute \l__spintent_scope_attr_int = 1
          \__spintent_spProm_exec:nn {#1} {#2}
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spProm } }
  }
```

(*End of definition for* `\spProm`. *This function is documented on page* 43.)

©2026 by Pablo González L

15.58 The commands of the stat family

The user commands \spmedia, \spmediana, \spmoda, \spdesv, \spvarz, \sprango, \spriq, \spmin and \spmax typeset the usual symbols of descriptive statistics: MA for the arithmetic mean, MD for the median, MO for the mode, s for the standard deviation, s^{2} for the variance, R for the range, RIQ for the interquartile range, x_{\min} for the minimum, and x_{\max} for the maximum. Each one is read with the name of the concept; \spdesv and \spvarz also read their letter.

15.58.1 Definition of variables for the stat family

\l__spintent_spmedia_sub_index_str
\l__spintent_spmedia_read_arg_tl
\l__spintent_spmediana_sub_index_str
\l__spintent_spmediana_read_arg_tl
\l__spintent_spmoda_sub_index_str
\l__spintent_spmoda_read_arg_tl
\l__spintent_spdesv_sub_index_str
\l__spintent_spdesv_read_arg_tl
\l__spintent_spvarz_sub_index_str
\l__spintent_spvarz_read_arg_tl
\l__spintent_sprango_sub_index_str
\l__spintent_sprango_read_arg_tl
\l__spintent_spriq_sub_index_str
\l__spintent_spriq_read_arg_tl
\l__spintent_spmin_sub_index_str
\l__spintent_spmin_read_arg_tl
\l__spintent_spmax_sub_index_str
\l__spintent_spmax_read_arg_tl

Each command of the family has its own copy of the variables _sub_index_str and _read_arg_tl, which hold the values of the keys sub-index and read-arg, so each command can be configured separately with \setspintent. Since the symbol of \spmin and \spmax already carries a subscript, they have no sub-index nor read-sub keys; their variable _sub_index_str exists only because __spintent_stat_intent:nnnn reads it, and always keeps its initial value.

```
\clist_map_inline:nn
  {
    spmedia , spmediana , spmoda , spdesv , spvarz , sprango , spriq ,
    spmin , spmax
  }
  {
    \str_new:c  { l__spintent_ #1 _sub_index_str }
    \tl_new:c   { l__spintent_ #1 _read_arg_tl   }
  }
```

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

\l__spintent_stat_intent_tl

The variable \l__spintent_stat_intent_tl holds the name of the intent that __spintent_stat_intent:nnnn builds, until __spintent_stat_mathml:nnnn uses it.

```
\tl_new:N \l__spintent_stat_intent_tl
```

(*End of definition for* \l__spintent_stat_intent_tl.)

15.58.2 Definition of keys for the stat family

silent
sub-index
read-sub
read-arg
unknown

We define the ⟨*keys*⟩ silent, sub-index, read-sub and read-arg for the seven commands \spmedia, \spmediana, \spmoda, \spdesv, \spvarz, \sprango and \spriq.

```
\clist_map_inline:nn
  {
    spmedia , spmediana , spmoda , spdesv , spvarz , sprango , spriq
  }
  {
    \keys_define:nn { spintent / #1 }
      {
        silent    .bool_set:c = { l__spintent_ #1 _silent_bool },
        silent    .initial:n = false,
        silent    .value_required:n = true,
        sub-index .code:n =
          { \str_set:cn { l__spintent_ #1 _sub_index_str } {##1} },
        sub-index .initial:n = ,
        sub-index .value_required:n = true,
        read-sub  .bool_set:c = { l__spintent_ #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_ #1 _read_arg_tl } {##1} },
        read-arg  .initial:n = ,
        read-arg  .value_required:n = true,
        unknown   .code:n =
          {
            \str_set:Nn \l__spintent_command_name_str {#1}
            \__spintent_unknown_keys_cmd:n {##1}
          },
      }
  }
```

(*End of definition for* silent *and others.*)

sigma

The key sigma exists only for \spdesv and \spvarz, and it starts in false.

```
\clist_map_inline:nn { spdesv , spvarz }
```

©2026 by Pablo González L

```
  {
    \keys_define:nn { spintent / #1 }
      {
        sigma     .bool_set:c = { l__spintent_ #1 _sigma_bool },
        sigma     .initial:n = false,
        sigma     .value_required:n = true,
        read-sub  .initial:n = true,
      }
  }
```

For \spdesv and \spvarz, which read a letter, read-sub starts in true, since s sub x sounds more natural than s de x and it avoids hearing the ⟨x⟩ as part of the exponent in varianza s de x al cuadrado; for the rest of the commands it starts in false.

(*End of definition for* sigma.)

silent
read-arg
unknown

Since the symbol of \spmin and \spmax already carries a subscript, we define for them only the ⟨*keys*⟩ silent and read-arg.

```
\clist_map_inline:nn { spmin , spmax }
  {
    \keys_define:nn { spintent / #1 }
      {
        silent   .bool_set:c = { l__spintent_ #1 _silent_bool },
        silent   .initial:n = false,
        silent   .value_required:n = true,
        read-arg .code:n =
          { \__spintent_sanitize_affix:cn { l__spintent_ #1 _read_arg_tl } {##1} },
        read-arg .initial:n = ,
        read-arg .value_required:n = true,
        unknown  .code:n =
          {
            \str_set:Nn \l__spintent_command_name_str {#1}
            \__spintent_unknown_keys_cmd:n {##1}
          },
      }
  }
```

(*End of definition for* silent , read-arg , *and* unknown.)

15.58.3 Internal functions for the stat family

__spintent_stat_typeset:nnn

The function __spintent_stat_typeset:nnn typesets the symbol. It receives the short name of the command in '#1', the symbol in '#2' and the exponent in '#3', empty when there is none. The symbol is followed by the sub-index, if present, with its own \MathMLintent set to :silent, and then by the exponent.

```
\cs_new_protected:Nn \__spintent_stat_typeset:nnn
  {
    #2
    \str_if_empty:cF { l__spintent_ #1 _sub_index_str }
      {
        \c_math_subscript_token
          {
            \MathMLintent { :silent }
              { \str_use:c { l__spintent_ #1 _sub_index_str } }
          }
      }
    \tl_if_blank:nF {#3}
      { \c_math_superscript_token {#3} }
  }
```

(*End of definition for* __spintent_stat_typeset:nnn.)

__spintent_stat_intent:nnnn
__spintent_stat_intent:nnn
__spintent_stat_intent:nn

The function __spintent_stat_intent:nnnn builds the name of the intent of the symbols of the family. It receives the short name of the command in '#1', the name of the concept in '#2', the name of the letter in '#3', empty when the symbol is not a letter, and the spoken form of the exponent in '#4', empty when there is none, the last three already written as they go in the intent, with hyphens. It stores the name of the intent in \l__spintent_stat_intent_tl.

With silent=true, the intent is '_-'. With read-arg given, the intent is _⟨*read-arg*⟩:nofix and sub-index and read-sub are ignored for the spoken content; the subscript, if any, still displays, silenced. Otherwise the spoken content is the concept, followed by the letter and, when there is a sub-index, by de-⟨*value*⟩,

©2026 by Pablo González L

or by sub-⟨*value*⟩ when read-sub is true, which is its default for \spdesv and \spvarz, and last by the exponent. With no sub-index the intent ends in :nofix.

The functions __spintent_stat_intent:nnn and __spintent_stat_intent:nn are the short forms for the symbols with a letter but no exponent, and for those with neither a letter nor an exponent. They give the empty values to the full form, so no command has to.

```
\cs_new_protected:Nn \__spintent_stat_intent:nnnn
  {
    \tl_set:Ne \l__spintent_stat_intent_tl
      {
        \bool_if:cTF { l__spintent_ #1 _silent_bool }
          { _- }
          {
            \tl_if_empty:cTF { l__spintent_ #1 _read_arg_tl }
              {
                _ #2
                \tl_if_blank:nF {#3} { - #3 }
                \str_if_empty:cTF { l__spintent_ #1 _sub_index_str }
                  {
                    \tl_if_blank:nF {#4} { - #4 }
                    :nofix
                  }
                  {
                    -
                    \bool_if:cTF { l__spintent_ #1 _read_sub_bool } { sub } { de }
                    - \str_use:c { l__spintent_ #1 _sub_index_str }
                    \tl_if_blank:nF {#4} { - #4 }
                  }
              }
              { _ \tl_use:c { l__spintent_ #1 _read_arg_tl } :nofix }
          }
      }
  }
\cs_new_protected:Nn \__spintent_stat_intent:nnn
  {
    \__spintent_stat_intent:nnnn {#1} {#2} {#3} { }
  }
\cs_new_protected:Nn \__spintent_stat_intent:nn
  {
    \__spintent_stat_intent:nnnn {#1} {#2} { } { }
  }
```

(*End of definition for* __spintent_stat_intent:nnnn, __spintent_stat_intent:nnn, *and* __spintent_stat_intent:nn.)

__spintent_stat_mathml:nnnn

The function __spintent_stat_mathml:nnnn receives the short name of the command in '#1', the symbol in '#2', the exponent in '#3', empty when there is none, and the name of the intent in '#4', as stored in \l__spintent_stat_intent_tl by __spintent_stat_intent:nnnn. The symbol, with its subscript and exponent, is always typeset by __spintent_stat_typeset:nnn inside one native \MathMLintent, preceded by the inner ghost __spintent_inner_sep:.

```
\cs_new_protected:Nn \__spintent_stat_mathml:nnnn
  {
    \__spintent_invisible_sep:
    \MathMLintent {#4}
      {
        \__spintent_inner_sep:
        \__spintent_stat_typeset:nnn {#1} {#2} {#3}
      }
  }
```

(*End of definition for* __spintent_stat_mathml:nnnn.)

__spintent_stat_exec:nnn
__spintent_stat_simple:nnn
__spintent_stat_letter:nnnn

The function __spintent_stat_exec:nnn holds what all the commands of the family share. It receives the short name of the command in '#1', the optional ⟨*keys*⟩ in '#2' and the code that prints the symbol in '#3'. It enforces *math mode*, opens a local group, sets the ⟨*keys*⟩ and only then runs '#3', so that the code can depend on the ⟨*keys*⟩ already set, as sigma does.

The function __spintent_stat_simple:nnn is the short form of __spintent_stat_mathml:nnnn for the symbols with neither a letter nor an exponent. It receives the short name in '#1', the symbol in '#2' and the name of the concept in '#3'.

©2026 by Pablo González L

The function __spintent_stat_letter:nnnn is the short form for the symbols with a letter but no exponent. It receives the same three arguments and the name of the letter in '#4'.

```
\cs_new_protected:Nn \__spintent_stat_exec:nnn
  {
    \mode_if_math:TF
      {
        \group_begin:
          \attribute \l__spintent_scope_attr_int = 1
          \keys_set:nn { spintent / #1 } {#2}
          #3
        \group_end:
      }
      { \msg_error:nne { spintent } { need-math-mode } { \exp_not:c {#1} } }
  }
\cs_new_protected:Nn \__spintent_stat_simple:nnn
  {
    \__spintent_stat_intent:nn {#1} {#3}
    \__spintent_stat_mathml:nnnn {#1} {#2} { } { \l__spintent_stat_intent_tl }
  }
\cs_new_protected:Nn \__spintent_stat_letter:nnnn
  {
    \__spintent_stat_intent:nnn {#1} {#3} {#4}
    \__spintent_stat_mathml:nnnn {#1} {#2} { } { \l__spintent_stat_intent_tl }
  }
```

(*End of definition for* __spintent_stat_exec:nnn, __spintent_stat_simple:nnn, *and* __spintent_stat_letter:nnnn.)

\spmedia
\spmediana
\spmoda

The public commands \spmedia, \spmediana and \spmoda delegate execution to __spintent_stat_exec:nnn, which prints the symbols MA, MD and MO through __spintent_stat_simple:nnn with the concepts media-aritmética, mediana and moda.

```
\NewDocumentCommand \spmedia { O{} }
  {
    \__spintent_stat_exec:nnn { spmedia } {#1}
      { \__spintent_stat_simple:nnn { spmedia } { \mathrm{MA} } { media-aritmética } }
  }
\NewDocumentCommand \spmediana { O{} }
  {
    \__spintent_stat_exec:nnn { spmediana } {#1}
      { \__spintent_stat_simple:nnn { spmediana } { \mathrm{MD} } { mediana } }
  }
\NewDocumentCommand \spmoda { O{} }
  {
    \__spintent_stat_exec:nnn { spmoda } {#1}
      { \__spintent_stat_simple:nnn { spmoda } { \mathrm{MO} } { moda } }
  }
```

(*End of definition for* \spmedia, \spmediana, *and* \spmoda. *These functions are documented on page* 43.)

\spdesv
\spvarz

The public commands \spdesv and \spvarz work the same way, but \spdesv calls __spintent_stat_letter:nnnn and \spvarz calls __spintent_stat_mathml:nnnn directly. The symbol is s, or \sigma when sigma is true, and its name is part of the spoken content; \spvarz adds the exponent 2, spoken al-cuadrado. The concepts are desviación-estándar and varianza.

```
\NewDocumentCommand \spdesv { O{} }
  {
    \__spintent_stat_exec:nnn { spdesv } {#1}
      {
        \bool_if:NTF \l__spintent_spdesv_sigma_bool
          {
            \__spintent_stat_letter:nnnn { spdesv } { \sigma }
              { desviación-estándar } { sigma }
          }
          {
            \__spintent_stat_letter:nnnn { spdesv } { s }
              { desviación-estándar } { s }
          }
      }
  }
\NewDocumentCommand \spvarz { O{} }
  {
```

```
    \__spintent_stat_exec:nnn { spvarz } {#1}
      {
        \bool_if:NTF \l__spintent_spvarz_sigma_bool
          {
            \__spintent_stat_intent:nnnn { spvarz } { varianza } { sigma }
              { al-cuadrado }
            \__spintent_stat_mathml:nnnn { spvarz } { \sigma } { 2 }
              { \l__spintent_stat_intent_tl }
          }
          {
            \__spintent_stat_intent:nnnn { spvarz } { varianza } { s }
              { al-cuadrado }
            \__spintent_stat_mathml:nnnn { spvarz } { s } { 2 }
              { \l__spintent_stat_intent_tl }
          }
      }
  }
```

(*End of definition for* `\spdesv` *and* `\spvarz`*. These functions are documented on page 43.*)

`\sprango` `\spriq` `\spmin` `\spmax`

The public commands `\sprango`, `\spriq`, `\spmin` and `\spmax` work like the first three, with the symbols `R`, `RIQ`, `x_{\min}` and `x_{\max}` and the concepts `rango`, `rango-intercuartil`, `mínimo` and `máximo`.

```
\NewDocumentCommand \sprango { O{} }
  {
    \__spintent_stat_exec:nnn { sprango } {#1}
      { \__spintent_stat_simple:nnn { sprango } { R } { rango } }
  }
\NewDocumentCommand \spriq { O{} }
  {
    \__spintent_stat_exec:nnn { spriq } {#1}
      { \__spintent_stat_simple:nnn { spriq } { \mathrm{RIQ} } { rango-intercuartil } }
  }
\NewDocumentCommand \spmin { O{} }
  {
    \__spintent_stat_exec:nnn { spmin } {#1}
      {
        \__spintent_stat_simple:nnn { spmin }
          { x \c_math_subscript_token { \min } } { mínimo }
      }
  }
\NewDocumentCommand \spmax { O{} }
  {
    \__spintent_stat_exec:nnn { spmax } {#1}
      {
        \__spintent_stat_simple:nnn { spmax }
          { x \c_math_subscript_token { \max } } { máximo }
      }
  }
```

(*End of definition for* `\sprango` *and others. These functions are documented on page 44.*)

15.59 The commands of the `stat_pos` family

The user commands `\spcuartil`, `\spdecil` and `\spPerc` typeset the measures of position: `Q_{k}` for quartiles, `D_{k}` for deciles, and `P_{k}` for percentiles. The index `k` is the {⟨*argument*⟩}: an integer from `1` to `3`, `1` to `9` and `1` to `99` respectively. Each one is read with the name of the measure and the index, for example, *«cuartil 1»*.

15.59.1 Definition of variables for the `stat_pos` family

`\l__spintent_spcuartil_read_arg_tl` `\l__spintent_spdecil_read_arg_tl` `\l__spintent_spPerc_read_arg_tl`

Each command of the family has its own copy of the variable `_read_arg_tl`, so each command can be configured separately with `\setspintent`.

```
\clist_map_inline:nn { spcuartil , spdecil , spPerc }
  {
    \tl_new:c { l__spintent_ #1 _read_arg_tl }
  }
```

(*End of definition for* `\l__spintent_spcuartil_read_arg_tl`*,* `\l__spintent_spdecil_read_arg_tl`*, and* `\l__spintent_spPerc_read_arg_tl`*.*)

\l__spintent_stat_pos_error_bool
\l__spintent_stat_pos_value_int

The variable \l__spintent_stat_pos_error_bool is set to true by __spintent_stat_pos_check:nnn when the index is rejected, so that nothing is typeset. The variable \l__spintent_stat_pos_value_int holds the accepted index without leading zeros.

```
\bool_new:N \l__spintent_stat_pos_error_bool
\int_new:N  \l__spintent_stat_pos_value_int
```

(*End of definition for* \l__spintent_stat_pos_error_bool *and* \l__spintent_stat_pos_value_int.)

15.59.2 Definition of error messages for the stat_pos family

stat-pos-invalid-index

The "error" stat-pos-invalid-index is raised when the index passed to \spcuartil, \spdecil or \spPerc is not an integer without sign between 1 and the maximum of the measure.

```
\msg_new:nnnn { spintent } { stat-pos-invalid-index }
  {
    Invalid~index~'#2'~for~'\tl_trim_spaces:n {#1}'~\msg_line_context:.
  }
  {
    Only~an~integer~without~sign~between~'1'~and~'#3'~is~accepted.
  }
```

(*End of definition for* stat-pos-invalid-index.)

15.59.3 Definition of keys for the stat_pos family

silent
read-arg
unknown

We define the ⟨*keys*⟩ silent and read-arg for the three commands.

```
\clist_map_inline:nn { spcuartil , spdecil , spPerc }
  {
    \keys_define:nn { spintent / #1 }
      {
        silent   .bool_set:c = { l__spintent_ #1 _silent_bool },
        silent   .initial:n = false,
        silent   .value_required:n = true,
        read-arg .code:n =
          { \__spintent_sanitize_affix:cn { l__spintent_ #1 _read_arg_tl } {##1} },
        read-arg .initial:n = ,
        read-arg .value_required:n = true,
        unknown  .code:n =
          {
            \str_set:Nn \l__spintent_command_name_str {#1}
            \__spintent_unknown_keys_cmd:n {##1}
          },
      }
  }
```

(*End of definition for* silent , read-arg , *and* unknown.)

15.59.4 Internal functions for the stat_pos family

__spintent_stat_pos_check:nnn

The function __spintent_stat_pos_check:nnn receives the short name of the command in '#1', the maximum of the index in '#2' and the index in '#3'. It calls __spintent_luafun_clean_split_and_set:n on the index and accepts it only if it is an integer without sign between 1 and '#2'. Otherwise it raises stat-pos-invalid-index, whatever the value of silent, and sets \l__spintent_stat_pos_error_bool to true.

```
\cs_new_protected:Nn \__spintent_stat_pos_check:nnn
  {
    \__spintent_luafun_clean_split_and_set:n { \exp_not:n {#3} }
    \bool_set_true:N \l__spintent_stat_pos_error_bool
    \bool_lazy_and:nnT
      { \str_if_eq_p:Vn \l__spintent_luaset_has_integer_str { true } }
      { \tl_if_empty_p:N \l__spintent_luaset_sign_tl }
      {
        \bool_lazy_and:nnT
          { \int_compare_p:nNn { \l__spintent_luaset_only_part_int_str } > { 0 } }
          { \int_compare_p:nNn { \l__spintent_luaset_only_part_int_str } < { #2 + 1 } }
          { \bool_set_false:N \l__spintent_stat_pos_error_bool }
      }
    \bool_if:NT \l__spintent_stat_pos_error_bool
      {
        \msg_error:nneee { spintent } { stat-pos-invalid-index }
          { \token_to_str:c {#1} } { \exp_not:n {#3} } {#2}
      }
  }
```

(*End of definition for* `\__spintent_stat_pos_check:nnn`.)

`\__spintent_stat_pos_typeset:nn`

The function `\__spintent_stat_pos_typeset:nn` typesets the symbol in '`#1`' with the index in '`#2`' as a subscript, with its own `\MathMLintent` set to `:silent`.

```
\cs_new_protected:Nn \__spintent_stat_pos_typeset:nn
  {
    #1
    \c_math_subscript_token { \MathMLintent { :silent } {#2} }
  }
```

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

`\__spintent_stat_pos_mathml:nnnn`
`\__spintent_stat_pos_mathml:nnnV`

The function `\__spintent_stat_pos_mathml:nnnn` receives the short name of the command in '`#1`', the symbol in '`#2`', the name of the measure in '`#3`' and the index in '`#4`'. The symbol with its index is always typeset by `\__spintent_stat_pos_typeset:nn` inside one native `\MathMLintent`, preceded by the inner ghost `\__spintent_inner_sep:`.

With `silent=true`, the intent is '`_-`'. With `read-arg` given, the intent is `_`⟨*read-arg*⟩`:nofix`. Otherwise the spoken content is the name of the measure followed by the index, as in `_cuartil-1:nofix`.

```
\cs_new_protected:Nn \__spintent_stat_pos_mathml:nnnn
  {
    \__spintent_invisible_sep:
    \bool_if:cTF { l__spintent_ #1 _silent_bool }
      {
        \MathMLintent { _- }
          {
            \__spintent_inner_sep:
            \__spintent_stat_pos_typeset:nn {#2} {#4}
          }
      }
      {
        \tl_if_empty:cTF { l__spintent_ #1 _read_arg_tl }
          {
            \MathMLintent { _ #3 - #4 :nofix }
              {
                \__spintent_inner_sep:
                \__spintent_stat_pos_typeset:nn {#2} {#4}
              }
          }
          {
            \MathMLintent { _ \tl_use:c { l__spintent_ #1 _read_arg_tl } :nofix }
              {
                \__spintent_inner_sep:
                \__spintent_stat_pos_typeset:nn {#2} {#4}
              }
          }
      }
  }
\cs_generate_variant:Nn \__spintent_stat_pos_mathml:nnnn { nnnV }
```

(*End of definition for* `\__spintent_stat_pos_mathml:nnnn`.)

`\__spintent_stat_pos_exec:nnnnn`

The function `\__spintent_stat_pos_exec:nnnnn` receives the short name of the command in '`#1`', the symbol in '`#2`', the name of the measure in '`#3`', the maximum of the index in '`#4`' and the index itself in '`#5`'. It validates the index with `\__spintent_stat_pos_check:nnn` and, if it is accepted, stores its value in `\l__spintent_stat_pos_value_int`, so `025` is printed and read as `25`, and prints it through `\__spintent_stat_pos_mathml:nnnn`.

```
\cs_new_protected:Nn \__spintent_stat_pos_exec:nnnnn
  {
    \__spintent_stat_pos_check:nnn {#1} {#4} {#5}
    \bool_if:NF \l__spintent_stat_pos_error_bool
      {
        \int_set:NV \l__spintent_stat_pos_value_int \l__spintent_luaset_only_part_int_str
        \__spintent_stat_pos_mathml:nnnV {#1} {#2} {#3} \l__spintent_stat_pos_value_int
      }
  }
```

(*End of definition for* `\__spintent_stat_pos_exec:nnnnn`.)

\spcuartil
\spdecil
\spPerc

The public commands \spcuartil, \spdecil and \spPerc enforce *math mode*, open a local group, set the ⟨*keys*⟩ and delegate execution to __spintent_stat_pos_exec:nnnnn with the symbols Q, D and P, the measures cuartil, decil and percentil and the maxima 3, 9 and 99.

```
\NewDocumentCommand \spcuartil { O{} m }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \attribute \l__spintent_scope_attr_int = 1
          \keys_set:nn { spintent / spcuartil } {#1}
          \__spintent_stat_pos_exec:nnnnn { spcuartil } { Q } { cuartil } { 3 } {#2}
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spcuartil } }
  }
\NewDocumentCommand \spdecil { O{} m }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \attribute \l__spintent_scope_attr_int = 1
          \keys_set:nn { spintent / spdecil } {#1}
          \__spintent_stat_pos_exec:nnnnn { spdecil } { D } { decil } { 9 } {#2}
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spdecil } }
  }
\NewDocumentCommand \spPerc { O{} m }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \attribute \l__spintent_scope_attr_int = 1
          \keys_set:nn { spintent / spPerc } {#1}
          \__spintent_stat_pos_exec:nnnnn { spPerc } { P } { percentil } { 99 } {#2}
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spPerc } }
  }
```

(*End of definition for* \spcuartil, \spdecil, *and* \spPerc. *These functions are documented on page* 44.)

15.60 The command \spFrec

The user command \spFrec typesets the usual symbols of the frequencies of a table of data: f for the absolute frequency, F for the accumulated absolute, f_{r} for the relative, F_{r} for the accumulated relative, f_{r}\% for the percentage, and F_{r}\% for the accumulated percentage. The symbol is the {⟨*argument*⟩}, written exactly as it is printed, and it is read with the name of the frequency, for example, *«frecuencia absoluta acumulada»*.

15.60.1 Definition of variables for \spFrec

\l__spintent_spFrec_sub_index_str
\l__spintent_spFrec_read_arg_tl

The string variable \l__spintent_spFrec_sub_index_str and the token list variable \l__spintent_spFrec_read_arg_tl hold the values of the keys sub-index and read-arg.

```
\str_new:N \l__spintent_spFrec_sub_index_str
\tl_new:N  \l__spintent_spFrec_read_arg_tl
```

(*End of definition for* \l__spintent_spFrec_sub_index_str *and* \l__spintent_spFrec_read_arg_tl.)

\l__spintent_spFrec_symbol_str
\l__spintent_spFrec_concept_tl

The string variable \l__spintent_spFrec_symbol_str holds the {⟨*argument*⟩} without spaces, and the token list variable \l__spintent_spFrec_concept_tl the name of the frequency, empty while the symbol is not in the table.

```
\str_new:N \l__spintent_spFrec_symbol_str
\tl_new:N  \l__spintent_spFrec_concept_tl
```

(*End of definition for* \l__spintent_spFrec_symbol_str *and* \l__spintent_spFrec_concept_tl.)

©2026 by Pablo González L

15.60.2 Definition of error messages for \spFrec

spFrec-invalid-symbol
spFrec-invalid-sub-index

The "error" spFrec-invalid-symbol is raised when the symbol is not in the table and the key read-arg was not given. The "error" spFrec-invalid-sub-index is raised when the key sub-index is given with a symbol other than f or F.

```
\msg_new:nnnn { spintent } { spFrec-invalid-symbol }
  {
    Invalid~symbol~'#1'~for~'\token_to_str:N \spFrec'~\msg_line_context:.
  }
  {
    Use~'f',~'F',~'f_{r}',~'F_{r}',~'f_{r}\iow_char:N \%'~or~'F_{r}\iow_char:N \%',~or~
    give~the~key~'read-arg'.
  }
\msg_new:nnnn { spintent } { spFrec-invalid-sub-index }
  {
    Invalid~key~'sub-index'~for~the~symbol~'#1'~\msg_line_context:.
  }
  {
    Only~the~symbols~'f'~and~'F'~accept~the~key~'sub-index'.
  }
```

(*End of definition for* spFrec-invalid-symbol *and* spFrec-invalid-sub-index.)

15.60.3 Definition of keys for \spFrec

silent
sub-index
read-sub
read-arg
unknown

We define the ⟨*keys*⟩ silent, sub-index, read-sub and read-arg, the third one starting in true.

```
\keys_define:nn { spintent / spFrec }
  {
    silent    .bool_set:N = \l__spintent_spFrec_silent_bool,
    silent    .initial:n = false,
    silent    .value_required:n = true,
    sub-index .code:n =
      { \str_set:Nn \l__spintent_spFrec_sub_index_str {#1} },
    sub-index .initial:n = ,
    sub-index .value_required:n = true,
    read-sub  .bool_set:N = \l__spintent_spFrec_read_sub_bool,
    read-sub  .initial:n = true,
    read-sub  .value_required:n = true,
    read-arg  .code:n =
      { \__spintent_sanitize_affix:Nn \l__spintent_spFrec_read_arg_tl {#1} },
    read-arg  .initial:n = ,
    read-arg  .value_required:n = true,
    unknown   .code:n =
      {
        \str_set:Nn \l__spintent_command_name_str { spFrec }
        \__spintent_unknown_keys_cmd:n {#1}
      },
  }
```

(*End of definition for* silent *and others.*)

15.60.4 Internal functions for \spFrec

__spintent_spFrec_exec:n

The function __spintent_spFrec_exec:n receives the symbol in '#1'. It removes the spaces and looks the result up in the table, setting \l__spintent_spFrec_concept_tl to frecuencia-absoluta, frecuencia-absoluta-acumulada, frecuencia-relativa, frecuencia-relativa-acumulada, frecuencia-porcentual or frecuencia-porcentual-acumulada.

If the symbol is not in the table, it is rejected with spFrec-invalid-symbol, unless the key read-arg was given: then the symbol is printed as it was written and read as read-arg says. A sub-index is accepted only with the symbols f and F; with any other symbol, even with read-arg, it is rejected with spFrec-invalid-sub-index. The {⟨*argument*⟩} is rejected whatever the value of the key silent. When it is accepted, the function delegates the printing to __spintent_stat_simple:nnn.

```
\cs_new_protected:Nn \__spintent_spFrec_exec:n
  {
    \str_set:Ne \l__spintent_spFrec_symbol_str { \tl_to_str:n {#1} }
    \str_remove_all:Nn \l__spintent_spFrec_symbol_str { ~ }
    \tl_clear:N \l__spintent_spFrec_concept_tl
    \str_case:Vn \l__spintent_spFrec_symbol_str
      {
        { f }
```

©2026 by Pablo González L

```
          { \tl_set:Nn \l__spintent_spFrec_concept_tl { frecuencia-absoluta } }
        { F }
          { \tl_set:Nn \l__spintent_spFrec_concept_tl { frecuencia-absoluta-acumulada } }
        { f_{r} }
          { \tl_set:Nn \l__spintent_spFrec_concept_tl { frecuencia-relativa } }
        { F_{r} }
          { \tl_set:Nn \l__spintent_spFrec_concept_tl { frecuencia-relativa-acumulada } }
        { f_{r}\% }
          { \tl_set:Nn \l__spintent_spFrec_concept_tl { frecuencia-porcentual } }
        { F_{r}\% }
          {
            \tl_set:Nn \l__spintent_spFrec_concept_tl { frecuencia-porcentual-acumulada }
          }
      }
    \bool_lazy_and:nnTF
      { \tl_if_empty_p:N \l__spintent_spFrec_concept_tl }
      { \tl_if_empty_p:N \l__spintent_spFrec_read_arg_tl }
      {
        \msg_error:nnn { spintent } { spFrec-invalid-symbol } {#1}
      }
      {
        \bool_lazy_and:nnTF
          { \bool_not_p:n { \str_if_empty_p:N \l__spintent_spFrec_sub_index_str } }
          {
            \bool_not_p:n
              {
                \bool_lazy_or_p:nn
                  { \str_if_eq_p:Vn \l__spintent_spFrec_symbol_str { f } }
                  { \str_if_eq_p:Vn \l__spintent_spFrec_symbol_str { F } }
              }
          }
          {
            \msg_error:nnn { spintent } { spFrec-invalid-sub-index } {#1}
          }
          {
            \__spintent_stat_simple:nnn { spFrec } {#1}
              { \l__spintent_spFrec_concept_tl }
          }
      }
  }
```

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

`\spFrec` The public command `\spFrec` enforces *math mode*, opens a local group, sets the ⟨*keys*⟩ and delegates execution to `\__spintent_spFrec_exec:n`.

```
\NewDocumentCommand \spFrec { O{} m }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \attribute \l__spintent_scope_attr_int = 1
          \keys_set:nn { spintent / spFrec } {#1}
          \__spintent_spFrec_exec:n {#2}
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spFrec } }
  }
```

(*End of definition for* `\spFrec`. *This function is documented on page* 45.)

15.61 The command `\spcase`

The user command `\spcase` typesets the symbols of the cases of a probability: `C_{f}` for the favorable cases, `C_{p}` for the possible ones, and `C_{t}` for the total. The letter is the {⟨*argument*⟩}, and the symbol is read with the name of the cases, for example, *«casos favorables»*.

15.61.1 Definition of variables for `\spcase`

`\l__spintent_spcase_sub_index_str` `\l__spintent_spcase_read_arg_tl` The token list variable `\l__spintent_spcase_read_arg_tl` holds the value of the key `read-arg`. This command has no `sub-index` key, so the string variable `\l__spintent_spcase_sub_index_str` exists only because `\__spintent_stat_intent:nnnn` reads it, and always keeps its initial value.

```
\str_new:N \l__spintent_spcase_sub_index_str
```

```
\tl_new:N  \l__spintent_spcase_read_arg_tl
```

(*End of definition for* `\l__spintent_spcase_sub_index_str` *and* `\l__spintent_spcase_read_arg_tl`.)

`\l__spintent_spcase_letter_str` `\l__spintent_spcase_concept_tl`

The string variable `\l__spintent_spcase_letter_str` holds the {⟨*argument*⟩} without spaces, and the token list variable `\l__spintent_spcase_concept_tl` the name of the cases, empty while the letter is not in the table.

```
\str_new:N \l__spintent_spcase_letter_str
\tl_new:N  \l__spintent_spcase_concept_tl
```

(*End of definition for* `\l__spintent_spcase_letter_str` *and* `\l__spintent_spcase_concept_tl`.)

15.61.2 Definition of error messages for `\spcase`

`spcase-invalid-letter`

The "error" `spcase-invalid-letter` is raised when the {⟨*argument*⟩} is not one of the letters `f`, `p` or `t`.

```
\msg_new:nnnn { spintent } { spcase-invalid-letter }
  {
    Invalid~letter~'#1'~for~'\token_to_str:N \spcase'~\msg_line_context:.
  }
  {
    The~letter~must~be~'f',~'p'~or~'t'.
  }
```

(*End of definition for* `spcase-invalid-letter`.)

15.61.3 Definition of keys for `\spcase`

`silent` `read-arg` `unknown`

We define the ⟨*keys*⟩ `silent` and `read-arg`.

```
\keys_define:nn { spintent / spcase }
  {
    silent   .bool_set:N = \l__spintent_spcase_silent_bool,
    silent   .initial:n = false,
    silent   .value_required:n = true,
    read-arg .code:n =
      { \__spintent_sanitize_affix:Nn \l__spintent_spcase_read_arg_tl {#1} },
    read-arg .initial:n = ,
    read-arg .value_required:n = true,
    unknown  .code:n =
      {
        \str_set:Nn \l__spintent_command_name_str { spcase }
        \__spintent_unknown_keys_cmd:n {#1}
      },
  }
```

(*End of definition for* `silent` , `read-arg` , *and* `unknown`.)

15.61.4 Internal functions for `\spcase`

`\__spintent_spcase_exec:n`

The function `\__spintent_spcase_exec:n` receives the letter in '`#1`'. It removes the spaces and looks the result up in the table, setting `\l__spintent_spcase_concept_tl` to `casos-favorables`, `casos-posibles` or `casos-totales`. If the letter is not in the table, it is rejected with `spcase-invalid-letter` whatever the value of the keys `silent` and `read-arg`. Otherwise the function delegates the printing of the symbol, with the letter as its subscript, to `\__spintent_stat_simple:nnn`.

```
\cs_new_protected:Nn \__spintent_spcase_exec:n
  {
    \str_set:Ne \l__spintent_spcase_letter_str { \tl_to_str:n {#1} }
    \str_remove_all:Nn \l__spintent_spcase_letter_str { ~ }
    \tl_clear:N \l__spintent_spcase_concept_tl
    \str_case:Vn \l__spintent_spcase_letter_str
      {
        { f } { \tl_set:Nn \l__spintent_spcase_concept_tl { casos-favorables } }
        { p } { \tl_set:Nn \l__spintent_spcase_concept_tl { casos-posibles } }
        { t } { \tl_set:Nn \l__spintent_spcase_concept_tl { casos-totales } }
      }
    \tl_if_empty:NTF \l__spintent_spcase_concept_tl
      { \msg_error:nnn { spintent } { spcase-invalid-letter } {#1} }
      {
        \__spintent_stat_simple:nnn { spcase }
          { C \c_math_subscript_token { \str_use:N \l__spintent_spcase_letter_str } }
          { \l__spintent_spcase_concept_tl }
      }
  }
```

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

`\spcase` The public command `\spcase` enforces *math mode*, opens a local group, sets the ⟨*keys*⟩ and delegates execution to `\__spintent_spcase_exec:n`.

```
\NewDocumentCommand \spcase { O{} m }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \attribute \l__spintent_scope_attr_int = 1
          \keys_set:nn { spintent / spcase } {#1}
          \__spintent_spcase_exec:n {#2}
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spcase } }
  }
```

(*End of definition for* `\spcase`. *This function is documented on page* 45.)

15.62 The command \spOmega

The user command `\spOmega` typesets the symbol `\Omega` of the sample space, or `\#\Omega`, which is its cardinality, with the key `card`. It is read *«espacio muestral»* or *«cardinalidad del espacio muestral»*.

15.62.1 Definition of variables for \spOmega

`\l__spintent_spOmega_sub_index_str` `\l__spintent_spOmega_read_arg_tl` `\l__spintent_spOmega_read_sub_bool` The string variable `\l__spintent_spOmega_sub_index_str` and the token list variable `\l__spintent_spOmega_re` hold the values of the keys `sub-index` and `read-arg`. This command has no `read-sub` key, so the boolean variable `\l__spintent_spOmega_read_sub_bool` starts as `true` and the subscript is always spoken with *«sub»*.

```
\str_new:N       \l__spintent_spOmega_sub_index_str
\tl_new:N        \l__spintent_spOmega_read_arg_tl
\bool_new:N      \l__spintent_spOmega_read_sub_bool
\bool_set_true:N \l__spintent_spOmega_read_sub_bool
```

(*End of definition for* `\l__spintent_spOmega_sub_index_str`, `\l__spintent_spOmega_read_arg_tl`, *and* `\l__spintent_spOmega_read_sub_bool`.)

15.62.2 Definition of keys for \spOmega

`silent` `sub-index` `read-arg` `card` `unknown` We define the ⟨*keys*⟩ `silent`, `sub-index`, `read-arg` and `card`, the last one starting in `false`.

```
\keys_define:nn { spintent / spOmega }
  {
    silent    .bool_set:N = \l__spintent_spOmega_silent_bool,
    silent    .initial:n = false,
    silent    .value_required:n = true,
    sub-index .code:n =
      { \str_set:Nn \l__spintent_spOmega_sub_index_str {#1} },
    sub-index .initial:n = ,
    sub-index .value_required:n = true,
    read-arg  .code:n =
      { \__spintent_sanitize_affix:Nn \l__spintent_spOmega_read_arg_tl {#1} },
    read-arg  .initial:n = ,
    read-arg  .value_required:n = true,
    card      .bool_set:N = \l__spintent_spOmega_card_bool,
    card      .initial:n = false,
    card      .value_required:n = true,
    unknown   .code:n =
      {
        \str_set:Nn \l__spintent_command_name_str { spOmega }
        \__spintent_unknown_keys_cmd:n {#1}
      },
  }
```

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

15.62.3 Internal functions for \spOmega

`\spOmega` The public command `\spOmega` enforces *math mode*, opens a local group, sets the ⟨*keys*⟩ and delegates execution to `\__spintent_stat_simple:nnn`, with the symbol `\Omega` and the concept `espacio-muestral`, or the symbol `\#\Omega` and the concept `cardinalidad-del-espacio-muestral` when `card` is `true`.

```
\NewDocumentCommand \spOmega { O{} }
```

```
  {
    \mode_if_math:TF
      {
        \group_begin:
          \attribute \l__spintent_scope_attr_int = 1
          \keys_set:nn { spintent / spOmega } {#1}
          \bool_if:NTF \l__spintent_spOmega_card_bool
            {
              \__spintent_stat_simple:nnn { spOmega } { \# \Omega }
                { cardinalidad-del-espacio-muestral }
            }
            {
              \__spintent_stat_simple:nnn { spOmega } { \Omega }
                { espacio-muestral }
            }
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spOmega } }
  }
```

(*End of definition for* `\spOmega`. *This function is documented on page 46.*)

15.63 The command \spPval

The user command `\spPval` typesets the value of a probability as a fraction a/b, using `\dfrac` or `\frac` according to `use-dfrac`, read natively as *«a de b»*, without parsing in Lua.

15.63.1 Definition of keys for \spPval

silent
use-dfrac
unknown

We define the ⟨*keys*⟩ `silent` and `use-dfrac`.

```
\keys_define:nn { spintent / spPval }
  {
    silent    .bool_set:N = \l__spintent_spPval_silent_bool,
    silent    .initial:n = false,
    silent    .value_required:n = true,
    use-dfrac .bool_set:N = \l__spintent_spPval_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 { spPval }
        \__spintent_unknown_keys_cmd:n {#1}
      },
  }
```

(*End of definition for* `silent`, `use-dfrac`, *and* `unknown`.)

15.63.2 Internal functions for \spPval

__spintent_spPval_exec:nnn

The function `\__spintent_spPval_exec:nnn` receives the optional ⟨*keys*⟩ in '`#1`', the numerator in '`#2`' and the denominator in '`#3`'. It sets the ⟨*keys*⟩ and calls `\__spintent_spPval_mathml:nn`.

```
\cs_new_protected:Nn \__spintent_spPval_exec:nnn
  {
    \keys_set:nn { spintent / spPval } {#1}
    \__spintent_spPval_mathml:nn {#2} {#3}
  }
```

(*End of definition for* `\__spintent_spPval_exec:nnn`.)

__spintent_spPval_mathml:nn

The function `\__spintent_spPval_mathml:nn` receives the numerator in '`#1`' and the denominator in '`#2`'. Under `silent`, the fraction, `\frac` or `\dfrac` per `use-dfrac`, is silenced as one unit. Otherwise numerator and denominator are each tagged through a native `\MathMLarg`, as `num` and `denom`, inside the fraction, and the outer `\MathMLintent` reads `_de:infix($num,$denom)`.

```
\cs_new_protected:Nn \__spintent_spPval_mathml:nn
  {
    \__spintent_invisible_sep:
    \bool_if:NTF \l__spintent_spPval_silent_bool
      {
        \MathMLintent { _- }
          {
            \__spintent_inner_sep:
```

```
            \bool_if:NTF \l__spintent_spPval_use_dfrac_bool
              { \dfrac {#1} {#2} }
              { \frac  {#1} {#2} }
          }
      }
      {
        \MathMLintent { _de:infix($num,$denom) }
          {
            \__spintent_inner_sep:
            \bool_if:NTF \l__spintent_spPval_use_dfrac_bool
              {
                \dfrac { \MathMLarg { num } {#1} } { \MathMLarg { denom } {#2} }
              }
              { \frac  { \MathMLarg { num } {#1} } { \MathMLarg { denom } {#2} } }
          }
      }
  }
```

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

`\spPval` The public command `\spPval` enforces *math mode*, opens a local group and delegates execution to `\__spintent_spPval_exec:nnn`.

```
\NewDocumentCommand \spPval { O{} m m }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \__spintent_spPval_exec:nnn {#1} {#2} {#3}
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spPval } }
  }
```

(*End of definition for* `\spPval`. *This function is documented on page 46.*)

15.64 The command \spProb

The user command `\spProb` typesets the probability of an event as `P({`⟨*event*⟩`})`. The `{`⟨*argument*⟩`}` is written as it is printed: a single event, or two events joined by exactly one of `\cap`, `\cup`, `\mid` or the character '`|`', which are read *«probabilidad de A»*, *«probabilidad de A y B»*, *«probabilidad de A o B»* and *«probabilidad de A dado B»*.

15.64.1 Definition of variables for \spProb

`\l__spintent_spProb_luaset_error_str` `\l__spintent_spProb_luaset_op_str` `\l__spintent_spProb_luaset_a_str` `\l__spintent_spProb_luaset_b_str` Variables set by the function `\__spintent_luafun_spProb_parse_and_set:n`. The string variable `\l__spintent_spProb_luaset_error_str` is `true` when the `{`⟨*argument*⟩`}` is invalid. The string variable `\l__spintent_spProb_luaset_op_str` is `none`, `cap`, `cup` or `mid`, and `\l__spintent_spProb_luaset_a_str` and `\l__spintent_spProb_luaset_b_str` hold the event before and after the operator; the second one is empty when there is no operator.

```
\str_new:N \l__spintent_spProb_luaset_error_str
\str_new:N \l__spintent_spProb_luaset_op_str
\str_new:N \l__spintent_spProb_luaset_a_str
\str_new:N \l__spintent_spProb_luaset_b_str
```

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

`\l__spintent_spProb_read_arg_tl` The token list variable `\l__spintent_spProb_read_arg_tl` stores the key `read-arg`.

```
\tl_new:N \l__spintent_spProb_read_arg_tl
```

(*End of definition for* `\l__spintent_spProb_read_arg_tl`.)

`\l__spintent_spProb_nested_bool` The boolean variable `\l__spintent_spProb_nested_bool` makes the redefinition of `\cap`, `\cup` and `\mid` happen only once when the groups are nested, for example, a `\spcomp` inside `\spProb`, so that the saved meanings are the original ones.

```
\bool_new:N \l__spintent_spProb_nested_bool
```

(*End of definition for* `\l__spintent_spProb_nested_bool`.)

15.64.2 Definition of error messages for \spProb

spProb-invalid-arg

The "error" spProb-invalid-arg is raised when the {⟨*argument*⟩} is empty, has unbalanced braces or parentheses, has an empty event, or has more than one operator.

```
\msg_new:nnnn { spintent } { spProb-invalid-arg }
  {
    Invalid~argument~'#1'~for~'\token_to_str:N \spProb'~\msg_line_context:.
  }
  {
    Use~one~event,~or~two~joined~by~one~of~'\token_to_str:N \cap',~
    '\token_to_str:N \cup',~'\token_to_str:N \mid'~or~'|'~(outside~braces~
    and~parentheses).
  }
```

(*End of definition for* spProb-invalid-arg.)

15.64.3 Definition of keys for \spProb

silent
read-arg
unknown

We define the ⟨*keys*⟩ silent and read-arg.

```
\keys_define:nn { spintent / spProb }
  {
    silent   .bool_set:N = \l__spintent_spProb_silent_bool,
    silent   .initial:n = false,
    silent   .value_required:n = true,
    read-arg .code:n =
      { \__spintent_sanitize_affix:Nn \l__spintent_spProb_read_arg_tl {#1} },
    read-arg .initial:n = ,
    read-arg .value_required:n = true,
    unknown  .code:n =
      {
        \str_set:Nn \l__spintent_command_name_str { spProb }
        \__spintent_unknown_keys_cmd:n {#1}
      },
  }
```

(*End of definition for* silent, read-arg, *and* unknown.)

15.64.4 Internal functions for \spProb

__spintent_spProb_cap:
__spintent_spProb_cup:
__spintent_spProb_mid:
__spintent_spProb_nested_ops:
__spintent_spProb_event:N

The functions __spintent_spProb_cap:, __spintent_spProb_cup: and __spintent_spProb_mid: wrap the saved meaning of \cap, \cup and \mid, each in its own \MathMLintent, which is _y-, _o and _dado respectively, while keeping its math class and therefore its spacing. The function __spintent_spProb_nested_ops: saves the three meanings and redefines, locally, \cap, \cup and \mid as these functions. The function __spintent_spProb_event:N typesets the event stored in its argument inside a group with these definitions, so the operators nested in braces or parentheses are also read; the operators at the top level never go through it, they are printed by __spintent_spProb_mathml:.

```
\cs_new_protected:Nn \__spintent_spProb_cap:
  {
    \mathbin { \MathMLintent { _y- } { \__spintent_spProb_cap_orig: } }
  }
\cs_new_protected:Nn \__spintent_spProb_cup:
  {
    \mathbin { \MathMLintent { _o } { \__spintent_spProb_cup_orig: } }
  }
\cs_new_protected:Nn \__spintent_spProb_mid:
  {
    \mathrel { \MathMLintent { _dado } { \__spintent_spProb_mid_orig: } }
  }
\cs_new_protected:Nn \__spintent_spProb_nested_ops:
  {
    \bool_if:NF \l__spintent_spProb_nested_bool
      {
        \bool_set_true:N \l__spintent_spProb_nested_bool
        \cs_set_eq:NN \__spintent_spProb_cap_orig: \cap
        \cs_set_eq:NN \__spintent_spProb_cup_orig: \cup
        \cs_set_eq:NN \__spintent_spProb_mid_orig: \mid
        \cs_set_eq:NN \cap \__spintent_spProb_cap:
        \cs_set_eq:NN \cup \__spintent_spProb_cup:
        \cs_set_eq:NN \mid \__spintent_spProb_mid:
      }
  }
```

©2026 by Pablo González L

```
\cs_new_protected:Nn \__spintent_spProb_event:N
  {
    \group_begin:
      \__spintent_spProb_nested_ops:
      \str_use:N #1
    \group_end:
  }
```

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

`\__spintent_spProb_content:` The function `\__spintent_spProb_content:` typesets the event, or the two events and the operator, without any annotation.

```
\cs_new_protected:Nn \__spintent_spProb_content:
  {
    \str_use:N \l__spintent_spProb_luaset_a_str
    \str_case:Vn \l__spintent_spProb_luaset_op_str
      {
        { cap } { \cap \str_use:N \l__spintent_spProb_luaset_b_str }
        { cup } { \cup \str_use:N \l__spintent_spProb_luaset_b_str }
        { mid } { \mid \str_use:N \l__spintent_spProb_luaset_b_str }
      }
  }
```

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

`\__spintent_spProb_mathml:` The function `\__spintent_spProb_mathml:` prints `P(`⟨*content*⟩`)` inside one native `\MathMLintent`, preceded by the outer ghost `\__spintent_invisible_sep:` and by the inner ghost `\__spintent_inner_sep:`. With `silent=true`, the intent is '`_-`'. With `read-arg` given, the intent is `_`⟨*read-arg*⟩`:nofix`. Otherwise the intent is `_probabilidad-de:prefix($a)` with the single event tagged as `a`; with an operator, the intent is `_probabilidad-de:prefix($p)`, where `p` is the group of the two events, which carries its own intent `_y-:infix($a,$b)`, `_o:infix($a,$b)` or `_dado:infix($a,$b)` according to the operator, with the two events tagged as `a` and `b`.

```
\cs_new_protected:Nn \__spintent_spProb_mathml:
  {
    \__spintent_invisible_sep:
    \bool_if:NTF \l__spintent_spProb_silent_bool
      {
        \MathMLintent { _- }
          {
            \__spintent_inner_sep:
            P \lparen \__spintent_spProb_content: \rparen
          }
      }
      {
        \tl_if_empty:NTF \l__spintent_spProb_read_arg_tl
          {
            \str_if_eq:VnTF \l__spintent_spProb_luaset_op_str { none }
              {
                \MathMLintent { _probabilidad-de:prefix($a) }
                  {
                    \__spintent_inner_sep:
                    P
                    \lparen
                      \MathMLarg { a }
                        {
                          \__spintent_inner_sep: \__spintent_spProb_event:N
                            \l__spintent_spProb_luaset_a_str
                        }
                    \rparen
                  }
              }
              {
                \MathMLintent { _probabilidad-de:prefix($p) }
                  {
                    \__spintent_inner_sep:
                    P
                    \lparen
                      \MathMLarg { p }
                        {
```

```
                        \MathMLintent
                          {
                            _
                            \str_case:Vn \l__spintent_spProb_luaset_op_str
                              { { cap } { y- } { cup } { o } { mid } { dado } }
                            :infix($a,$b)
                          }
                          {
                            \__spintent_inner_sep:
                            \MathMLarg { a }
                              {
                                \__spintent_inner_sep: \__spintent_spProb_event:N
                                  \l__spintent_spProb_luaset_a_str
                              }
                            \str_case:Vn \l__spintent_spProb_luaset_op_str
                              { { cap } { \cap } { cup } { \cup } { mid } { \mid } }
                            \MathMLarg { b }
                              {
                                \__spintent_inner_sep: \__spintent_spProb_event:N
                                  \l__spintent_spProb_luaset_b_str
                              }
                          }
                      }
                    \rparen
                }
            }
        }
        {
          \MathMLintent { _ \tl_use:N \l__spintent_spProb_read_arg_tl :nofix }
            {
              \__spintent_inner_sep:
              P \lparen \__spintent_spProb_content: \rparen
            }
        }
    }
}
```

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

`\__spintent_spProb_exec:nn` The function `\__spintent_spProb_exec:nn` sets the ⟨*keys*⟩ in '`#1`' and calls `\__spintent_luafun_spProb_parse_` with the {⟨*argument*⟩} in '`#2`'. If the Lua module reports an error, it is rejected with `spProb-invalid-arg` whatever the value of the key `silent`, and nothing is printed; otherwise it delegates the printing to `\__spintent_spProb_mathml:`.

```
\cs_new_protected:Nn \__spintent_spProb_exec:nn
  {
    \keys_set:nn { spintent / spProb } {#1}
    \__spintent_luafun_spProb_parse_and_set:n { \exp_not:n {#2} }
    \str_if_eq:VnTF \l__spintent_spProb_luaset_error_str { true }
      { \msg_error:nnn { spintent } { spProb-invalid-arg } {#2} }
      { \__spintent_spProb_mathml: }
  }
```

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

`\spProb` The public command `\spProb` enforces *math mode*, opens a local group and delegates execution to `\__spintent_spProb_exec:nn`.

```
\NewDocumentCommand \spProb { O{} m }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \attribute \l__spintent_scope_attr_int = 1
          \__spintent_spProb_exec:nn {#1} {#2}
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spProb } }
  }
```

(*End of definition for* `\spProb`. *This function is documented on page* 47.)

15.65 The command \spcomp

The user command \spcomp typesets the complement of an event, as ⟨*event*⟩^{c} or, with the key bar, as \overline{⟨*event*⟩}, and reads it *«complemento de»* the event.

15.65.1 Definition of variables for \spcomp

\l__spintent_spcomp_read_arg_tl
\l__spintent_spcomp_wrap_parens_str

The variable \l__spintent_spcomp_read_arg_tl stores the key read-arg and \l__spintent_spcomp_wrap_par the key wrap-parens.

```
\tl_new:N  \l__spintent_spcomp_read_arg_tl
\str_new:N \l__spintent_spcomp_wrap_parens_str
```

(*End of definition for* \l__spintent_spcomp_read_arg_tl *and* \l__spintent_spcomp_wrap_parens_str.)

\l__spintent_spcomp_luaset_compound_str
\l__spintent_spcomp_compound_bool
\l__spintent_spcomp_do_paren_bool

The function __spintent_luafun_spcomp_compound:n decides whether the event contains \cap, \cup, \setminus or \mid, whatever the keys are, and leaves the answer in \l__spintent_spcomp_luaset_compound_str. The boolean variable \l__spintent_spcomp_compound_bool is true in that case. The boolean variable \l__spintent_spcomp_do_paren_bool tells whether the event is printed in parentheses once the key and the event have been examined.

```
\str_new:N  \l__spintent_spcomp_luaset_compound_str
\bool_new:N \l__spintent_spcomp_compound_bool
\bool_new:N \l__spintent_spcomp_do_paren_bool
```

(*End of definition for* \l__spintent_spcomp_luaset_compound_str, \l__spintent_spcomp_compound_bool, *and* \l__spintent_spcomp_do_paren_bool.)

15.65.2 Definition of error messages for \spcomp

spcomp-empty-arg

The "error" spcomp-empty-arg is raised when the argument of \spcomp is missing or *blank*.

```
\msg_new:nnnn { spintent } { spcomp-empty-arg }
  {
    Empty~argument~for~'\token_to_str:N \spcomp'~\msg_line_context:.
  }
  {
    The~event~is~required.
  }
```

(*End of definition for* spcomp-empty-arg.)

15.65.3 Definition of keys for \spcomp

silent
bar
wrap-parens
read-arg
unknown

We define the ⟨*keys*⟩ silent, bar, wrap-parens and read-arg.

```
\keys_define:nn { spintent / spcomp }
  {
    silent      .bool_set:N = \l__spintent_spcomp_silent_bool,
    silent      .initial:n = false,
    silent      .value_required:n = true,
    bar         .bool_set:N = \l__spintent_spcomp_bar_bool,
    bar         .initial:n = false,
    bar         .value_required:n = true,
    wrap-parens .choices:nn = { auto , true , false }
      {
        \int_case:nn { \l_keys_choice_int }
          {
            {1} { \str_set:Nn \l__spintent_spcomp_wrap_parens_str { auto } }
            {2} { \str_set:Nn \l__spintent_spcomp_wrap_parens_str { true } }
            {3} { \str_set:Nn \l__spintent_spcomp_wrap_parens_str { false } }
          }
      },
    wrap-parens .initial:n = auto,
    wrap-parens .value_required:n = true,
    wrap-parens / unknown .code:n =
      \msg_error:nneee { spintent } { unknown-choice }
        { wrap-parens } { auto,~true,~false } { \exp_not:n {#1} },
    read-arg    .code:n =
      { \__spintent_sanitize_affix:Nn \l__spintent_spcomp_read_arg_tl {#1} },
    read-arg    .initial:n = ,
    read-arg    .value_required:n = true,
    unknown     .code:n =
      {
        \str_set:Nn \l__spintent_command_name_str { spcomp }
```

```
8463          \__spintent_unknown_keys_cmd:n {#1}
8464        },
8465    }
```

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

15.65.4 Internal functions for \spcomp

`\__spintent_spcomp_set_paren:n`
`\__spintent_spcomp_event:n`

The function `\__spintent_spcomp_set_paren:n` sets `\l__spintent_spcomp_compound_bool` and `\l__spintent_spcomp_do_paren_bool`. The first one is true when Lua reports that the event contains `\cap`, `\cup`, `\setminus` or `\mid`. The second one follows the key `wrap-parens`: with `auto` it is true when the event is compound, and it is always false when the key `bar` is set, since the overline already covers the event. The function `\__spintent_spcomp_event:n` typesets the event in a group where the operators nested in it are read as in `\__spintent_spProb_event:N`, and in parentheses when `\l__spintent_spcomp_do_paren_bool` is true. The parentheses belong to the event, so that they are inside the `\MathMLarg` and are read.

```
8466 \cs_new_protected:Nn \__spintent_spcomp_set_paren:n
8467   {
8468     \bool_set_false:N \l__spintent_spcomp_do_paren_bool
8469     \bool_set_false:N \l__spintent_spcomp_compound_bool
8470     \__spintent_luafun_spcomp_compound:n { \exp_not:n {#1} }
8471     \str_if_eq:VnT \l__spintent_spcomp_luaset_compound_str { true }
8472       { \bool_set_true:N \l__spintent_spcomp_compound_bool }
8473     \bool_if:NF \l__spintent_spcomp_bar_bool
8474       {
8475         \str_case:Vn \l__spintent_spcomp_wrap_parens_str
8476           {
8477             { true } { \bool_set_true:N \l__spintent_spcomp_do_paren_bool }
8478             { auto }
8479               {
8480                 \bool_if:NT \l__spintent_spcomp_compound_bool
8481                   { \bool_set_true:N \l__spintent_spcomp_do_paren_bool }
8482               }
8483           }
8484       }
8485   }
8486 \cs_new_protected:Nn \__spintent_spcomp_event:n
8487   {
8488     \group_begin:
8489       \__spintent_spProb_nested_ops:
8490       \bool_if:NTF \l__spintent_spcomp_do_paren_bool
8491         { \lparen #1 \rparen }
8492         {#1}
8493     \group_end:
8494   }
```

(*End of definition for* `\__spintent_spcomp_set_paren:n` *and* `\__spintent_spcomp_event:n`.)

`\__spintent_spcomp_typeset:n`

The function `\__spintent_spcomp_typeset:n` typesets its argument as the complement of the event, with the exponent c or with an overline according to the key `bar`.

```
8495 \cs_new_protected:Nn \__spintent_spcomp_typeset:n
8496   {
8497     \bool_if:NTF \l__spintent_spcomp_bar_bool
8498       { \overline {#1} }
8499       { {#1} ^ { c } }
8500   }
```

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

`\__spintent_spcomp_exec:nn`

The function `\__spintent_spcomp_exec:nn` sets the ⟨*keys*⟩ in '`#1`', calls `\__spintent_spcomp_set_paren:n` with the event in '`#2`' and then `\__spintent_spcomp_mathml:n`.

```
8501 \cs_new_protected:Nn \__spintent_spcomp_exec:nn
8502   {
8503     \keys_set:nn { spintent / spcomp } {#1}
8504     \__spintent_spcomp_set_paren:n {#2}
8505     \__spintent_spcomp_mathml:n {#2}
8506   }
```

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

__spintent_spcomp_mathml:n

The function __spintent_spcomp_mathml:n prints the complement of the event '#1' inside one native \MathMLintent, preceded by the outer ghost __spintent_invisible_sep: and by the inner ghost __spintent_inner_sep:. With silent =true the intent is '_-'; with read-arg given it is _⟨*read-arg*⟩:nofix; otherwise it is _complemento-de:prefix($a) with the event tagged as a; when the key bar is set and the event is compound, the intent is _complemento-de-la-expresión:prefix($a), because the overline is not read and the reading would not tell what the complement covers.

```
\cs_new_protected:Nn \__spintent_spcomp_mathml:n
  {
    \__spintent_invisible_sep:
    \bool_if:NTF \l__spintent_spcomp_silent_bool
      {
        \MathMLintent { _- }
          {
            \__spintent_inner_sep:
            \__spintent_spcomp_typeset:n { \__spintent_spcomp_event:n {#1} }
          }
      }
      {
        \tl_if_empty:NTF \l__spintent_spcomp_read_arg_tl
          {
            \MathMLintent
              {
                \bool_lazy_and:nnTF
                  { \bool_if_p:N \l__spintent_spcomp_bar_bool }
                  { \bool_if_p:N \l__spintent_spcomp_compound_bool }
                  { _complemento-de-la-expresión:prefix($a) }
                  { _complemento-de:prefix($a) }
              }
              {
                \__spintent_inner_sep:
                \__spintent_spcomp_typeset:n
                  {
                    \MathMLarg { a }
                      { \__spintent_inner_sep: \__spintent_spcomp_event:n {#1} }
                  }
              }
          }
          {
            \MathMLintent { _ \tl_use:N \l__spintent_spcomp_read_arg_tl :nofix }
              {
                \__spintent_inner_sep:
                \__spintent_spcomp_typeset:n { \__spintent_spcomp_event:n {#1} }
              }
          }
      }
  }
```

(*End of definition for* __spintent_spcomp_mathml:n.)

\spcomp

The public command \spcomp enforces *math mode*, opens a local group and delegates execution to __spintent_spcomp_exec:nn.

```
\NewDocumentCommand \spcomp { O{} m }
  {
    \mode_if_math:TF
      {
        \group_begin:
          \attribute \l__spintent_scope_attr_int = 1
          \tl_if_blank:nTF {#2}
            { \msg_error:nn { spintent } { spcomp-empty-arg } }
            { \__spintent_spcomp_exec:nn {#1} {#2} }
        \group_end:
      }
      { \msg_error:nnn { spintent } { need-math-mode } { \spcomp } }
  }
```

(*End of definition for* \spcomp. *This function is documented on page* 47.)

15.66 The command \spcomb

The user command \spcomb typesets the counting symbols of combinatorics: combinations, variations, arrangements and permutations, with or without repetition, and circular permutations. The keys kind,

`repeat` and `circular` choose the symbol and the reading; the key `print-sty` chooses between the binomial coefficient and the letter form.

15.66.1 Definition of variables for `\spcomb`

`\l__spintent_spcomb_luaset_single_str`
`\l__spintent_spcomb_luaset_raw_n_str`
`\l__spintent_spcomb_luaset_raw_k_str`

These variables are set by the function `\__spintent_luafun_spcomb_split:n`, which splits the argument of the command at its first comma of top level, ignoring the commas inside braces, parentheses or brackets. The variable `\l__spintent_spcomb_luaset_raw_n_str` stores what precedes the comma and `\l__spintent_spcomb_luaset_raw_k_str` what follows it. The variable `\l__spintent_spcomb_luaset_single_str` is `true` when the argument has no such comma.

```
\str_new:N \l__spintent_spcomb_luaset_single_str
\str_new:N \l__spintent_spcomb_luaset_raw_n_str
\str_new:N \l__spintent_spcomb_luaset_raw_k_str
```

(*End of definition for* `\l__spintent_spcomb_luaset_single_str`, `\l__spintent_spcomb_luaset_raw_n_str`, *and* `\l__spintent_spcomb_luaset_raw_k_str`.)

`\l__spintent_spcomb_luaset_error_str`
`\l__spintent_spcomb_luaset_n_str`
`\l__spintent_spcomb_luaset_k_str`
`\l__spintent_spcomb_luaset_list_tl`

These variables are set by the function `\__spintent_luafun_spcomb_prepare:nnnn`. The variable `\l__spintent_spcomb_luaset_error_str` is `true` when the arguments are invalid. The variables `\l__spintent_spcomb_luaset_n_str` and `\l__spintent_spcomb_luaset_k_str` store the arguments without the surrounding spaces. In the `list` mode, `\l__spintent_spcomb_luaset_list_tl` stores the multiplicities, each one in its own braces.

```
\str_new:N \l__spintent_spcomb_luaset_error_str
\str_new:N \l__spintent_spcomb_luaset_n_str
\str_new:N \l__spintent_spcomb_luaset_k_str
\tl_new:N  \l__spintent_spcomb_luaset_list_tl
```

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

`\l__spintent_spcomb_read_arg_tl`
`\l__spintent_spcomb_kind_str`
`\l__spintent_spcomb_print_sty_str`

The token list variable `\l__spintent_spcomb_read_arg_tl` stores the key `read-arg`, and the string variables store the keys `kind` and `print-sty`.

```
\tl_new:N  \l__spintent_spcomb_read_arg_tl
\str_new:N \l__spintent_spcomb_kind_str
\str_new:N \l__spintent_spcomb_print_sty_str
```

(*End of definition for* `\l__spintent_spcomb_read_arg_tl`, `\l__spintent_spcomb_kind_str`, *and* `\l__spintent_spcomb_print_sty_str`.)

`\l__spintent_spcomb_letter_tl`
`\l__spintent_spcomb_concept_tl`
`\l__spintent_spcomb_mode_str`
`\l__spintent_spcomb_problem_str`
`\l__spintent_spcomb_use_binom_bool`
`\l__spintent_spcomb_list_seq`
`\l__spintent_spcomb_last_tl`

Variables computed from the keys. The token list variable `\l__spintent_spcomb_letter_tl` holds the letters of the symbol: `C`, `V`, `A` or `P`, followed by `R` and/or `C`; and `\l__spintent_spcomb_concept_tl` the name of the concept that is read, before the final `-de`. The string variable `\l__spintent_spcomb_mode_str` is `one`, `pair` or `list` according to the arguments the symbol takes, and `\l__spintent_spcomb_problem_str` describes an incompatible combination of keys, it is empty when there is none. The boolean variable `\l__spintent_spcomb_use_binom_bool` is true when the symbol is printed with `\binom`. The sequence `\l__spintent_spcomb_list_seq` holds the multiplicities to print, and the token list `\l__spintent_spcomb_last_tl` the last one.

```
\tl_new:N   \l__spintent_spcomb_letter_tl
\tl_new:N   \l__spintent_spcomb_concept_tl
\str_new:N  \l__spintent_spcomb_mode_str
\str_new:N  \l__spintent_spcomb_problem_str
\bool_new:N \l__spintent_spcomb_use_binom_bool
\seq_new:N  \l__spintent_spcomb_list_seq
\tl_new:N   \l__spintent_spcomb_last_tl
```

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

15.66.2 Definition of error messages for `\spcomb`

`spcomb-invalid-keys`
`spcomb-missing-arg`
`spcomb-extra-arg`
`spcomb-invalid-arg`

The "error" `spcomb-invalid-keys` is raised for a combination of keys that does not exist: `circular` without `kind=permutation`, `circular` with `repeat`, or `print-sty=binom` for anything but combinations without repetition. The "error" `spcomb-missing-arg` is raised when `k` is missing and the symbol needs it. The "error" `spcomb-extra-arg` is raised when `k` is given and the symbol does not take it. The "error" `spcomb-invalid-arg` is raised when the arguments are empty, when the list of multiplicities has fewer than two elements or an empty one, or when integers are not possible, because `k` is greater than `n` or the multiplicities add more than `n`.

```
\msg_new:nnnn { spintent } { spcomb-invalid-keys }
  {
    Incompatible~keys~for~'\token_to_str:N \spcomb':~'#1'~\msg_line_context:.
```

```
  }
  {
    'circular'~needs~'kind=permutation'~and~'repeat=false';~'print-sty=binom'~needs~
    'kind=combination'~and~'repeat=false'.
  }
\msg_new:nnnn { spintent } { spcomb-missing-arg }
  {
    Missing~'k'~(after~the~comma)~for~'\token_to_str:N \spcomb'~\msg_line_context:.
  }
  {
    This~kind~of~symbol~needs~'k'~or~the~list~of~repetitions~after~the~
    comma:~'\token_to_str:N \spcomb {n,k}'.
  }
\msg_new:nnnn { spintent } { spcomb-extra-arg }
  {
    Unexpected~comma~and~'k'~for~'\token_to_str:N \spcomb'~\msg_line_context:.
  }
  {
    Circular~permutations~take~only~'n':~'\token_to_str:N \spcomb {n}'.
  }
\msg_new:nnnn { spintent } { spcomb-invalid-arg }
  {
    Invalid~arguments~'#1'~and~'#2'~for~'\token_to_str:N \spcomb'~\msg_line_context:.
  }
  {
    Use~'n',~'n,k'~or~'n,a,b,...'~(at~least~two~repetitions,~none~empty);~if~
    integers,~'k~<=~n'~and~'a+b+...~<=~n'.
  }
```

(*End of definition for* `spcomb-invalid-keys` *and others.*)

15.66.3 Definition of keys for `\spcomb`

silent
kind
repeat
circular
print-sty
read-arg
unknown

We define the ⟨*keys*⟩ `silent`, `kind`, `repeat`, `circular`, `print-sty` and `read-arg`.

```
\keys_define:nn { spintent / spcomb }
  {
    silent    .bool_set:N = \l__spintent_spcomb_silent_bool,
    silent    .initial:n = false,
    silent    .value_required:n = true,
    repeat    .bool_set:N = \l__spintent_spcomb_repeat_bool,
    repeat    .initial:n = false,
    repeat    .value_required:n = true,
    circular  .bool_set:N = \l__spintent_spcomb_circular_bool,
    circular  .initial:n = false,
    circular  .value_required:n = true,
    kind      .choices:nn =
      { combination , variation , arrangement , permutation }
      {
        \int_case:nn { \l_keys_choice_int }
          {
            {1} { \str_set:Nn \l__spintent_spcomb_kind_str { combination } }
            {2} { \str_set:Nn \l__spintent_spcomb_kind_str { variation } }
            {3} { \str_set:Nn \l__spintent_spcomb_kind_str { arrangement } }
            {4} { \str_set:Nn \l__spintent_spcomb_kind_str { permutation } }
          }
      },
    kind      .initial:n = combination,
    kind      .value_required:n = true,
    kind / unknown .code:n =
      \msg_error:nneee { spintent } { unknown-choice }
        { kind } { combination,~variation,~arrangement,~permutation }
        { \exp_not:n {#1} },
    print-sty .choices:nn = { auto , binom , letter }
      {
        \int_case:nn { \l_keys_choice_int }
          {
            {1} { \str_set:Nn \l__spintent_spcomb_print_sty_str { auto } }
            {2} { \str_set:Nn \l__spintent_spcomb_print_sty_str { binom } }
            {3} { \str_set:Nn \l__spintent_spcomb_print_sty_str { letter } }
          }
      },
    print-sty .initial:n = auto,
```

```
    print-sty .value_required:n = true,
    print-sty / unknown .code:n =
      \msg_error:nneee { spintent } { unknown-choice }
        { print-sty } { auto,~binom,~letter } { \exp_not:n {#1} },
    read-arg  .code:n =
      { \__spintent_sanitize_affix:Nn \l__spintent_spcomb_read_arg_tl {#1} },
    read-arg  .initial:n = ,
    read-arg  .value_required:n = true,
    unknown   .code:n =
      {
        \str_set:Nn \l__spintent_command_name_str { spcomb }
        \__spintent_unknown_keys_cmd:n {#1}
      },
  }
```

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

15.66.4 Internal functions for \spcomb

`\__spintent_spcomb_setup:`

The function `\__spintent_spcomb_setup:` computes the letters and the concept from the keys, decides whether `\binom` is used, and leaves in `\l__spintent_spcomb_problem_str` the description of an incompatible combination of keys, if any.

```
\cs_new_protected:Nn \__spintent_spcomb_setup:
  {
    \str_clear:N \l__spintent_spcomb_problem_str
    \str_case:Vn \l__spintent_spcomb_kind_str
      {
        { combination }
          {
            \tl_set:Nn \l__spintent_spcomb_letter_tl  { C }
            \tl_set:Nn \l__spintent_spcomb_concept_tl { combinación }
          }
        { variation }
          {
            \tl_set:Nn \l__spintent_spcomb_letter_tl  { V }
            \tl_set:Nn \l__spintent_spcomb_concept_tl { variación }
          }
        { arrangement }
          {
            \tl_set:Nn \l__spintent_spcomb_letter_tl  { A }
            \tl_set:Nn \l__spintent_spcomb_concept_tl { arreglo }
          }
        { permutation }
          {
            \tl_set:Nn \l__spintent_spcomb_letter_tl  { P }
            \tl_set:Nn \l__spintent_spcomb_concept_tl { permutación }
          }
      }
    \bool_if:NT \l__spintent_spcomb_circular_bool
      {
        \bool_lazy_and:nnTF
          { \str_if_eq_p:Vn \l__spintent_spcomb_kind_str { permutation } }
          { ! \bool_if_p:N \l__spintent_spcomb_repeat_bool }
          {
            \tl_put_right:Nn \l__spintent_spcomb_letter_tl  { C }
            \tl_put_right:Nn \l__spintent_spcomb_concept_tl { -circular }
          }
          {
            \str_set:Nn \l__spintent_spcomb_problem_str
              { 'circular'~requires~'kind=permutation'~and~'repeat=false' }
          }
      }
    \bool_if:NT \l__spintent_spcomb_repeat_bool
      {
        \tl_put_right:Nn \l__spintent_spcomb_letter_tl  { R }
        \tl_put_right:Nn \l__spintent_spcomb_concept_tl { -con-repetición }
      }
    \bool_set_false:N \l__spintent_spcomb_use_binom_bool
    \bool_lazy_and:nnTF
      { \str_if_eq_p:Vn \l__spintent_spcomb_kind_str { combination } }
      { ! \bool_if_p:N \l__spintent_spcomb_repeat_bool }
```

```
      {
        \str_if_eq:VnF \l__spintent_spcomb_print_sty_str { letter }
          { \bool_set_true:N \l__spintent_spcomb_use_binom_bool }
      }
      {
        \str_if_eq:VnT \l__spintent_spcomb_print_sty_str { binom }
          {
            \str_if_empty:NT \l__spintent_spcomb_problem_str
              {
                \str_set:Nn \l__spintent_spcomb_problem_str
                  { 'print-sty=binom'~requires~'kind=combination'~and~'repeat=false' }
              }
          }
      }
  }
```

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

`\__spintent_spcomb_arg:nn` `\__spintent_spcomb_arg_ann:nn` `\__spintent_spcomb_arg_plain:nn` `\__spintent_spcomb_sep:` `\__spintent_spcomb_sep_ann:` `\__spintent_spcomb_sep_plain:` `\__spintent_spcomb_last:n` `\__spintent_spcomb_last_ann:n` `\__spintent_spcomb_last_plain:n`

The functions `\__spintent_spcomb_arg:nn`, `\__spintent_spcomb_sep:` and `\__spintent_spcomb_last:n` are the places where the symbol is annotated. They are set locally to the annotated versions, which use an `\MathMLarg` with the inner ghost, a silent comma, and a last multiplicity that carries the intent `_y-:prefix($l)` so that it reads *"y 4"*, or to the plain ones, the content alone and a plain comma, so that the same function `\__spintent_spcomb_symbol:` prints the symbol in both cases.

```
\cs_new_protected:Nn \__spintent_spcomb_arg_ann:nn
  {
    \MathMLarg {#1} { \__spintent_inner_sep: #2 }
  }
\cs_new_protected:Nn \__spintent_spcomb_arg_plain:nn
  {
    #2
  }
\cs_new_eq:NN \__spintent_spcomb_arg:nn \__spintent_spcomb_arg_plain:nn
\cs_new_protected:Nn \__spintent_spcomb_sep_ann:
  {
    \MathMLintent { :silent } { , }
  }
\cs_new_protected:Nn \__spintent_spcomb_sep_plain:
  {
    ,
  }
\cs_new_eq:NN \__spintent_spcomb_sep: \__spintent_spcomb_sep_plain:
\cs_new_protected:Nn \__spintent_spcomb_last_ann:n
  {
    \MathMLintent { _y-:prefix($l) }
      { \__spintent_inner_sep: \MathMLarg { l } { \__spintent_inner_sep: #1 } }
  }
\cs_new_protected:Nn \__spintent_spcomb_last_plain:n
  {
    #1
  }
\cs_new_eq:NN \__spintent_spcomb_last:n \__spintent_spcomb_last_plain:n
```

(*End of definition for* `\__spintent_spcomb_arg:nn` *and others.*)

`\__spintent_spcomb_list:`

The function `\__spintent_spcomb_list:` typesets the multiplicities separated by commas; the last one reads with a *"y"* before it, *"3 3 y 4"*, through the intent `_y-:prefix($l)`.

```
\cs_new_protected:Nn \__spintent_spcomb_list:
  {
    \seq_clear:N \l__spintent_spcomb_list_seq
    \tl_map_inline:Nn \l__spintent_spcomb_luaset_list_tl
      { \seq_put_right:Nn \l__spintent_spcomb_list_seq {##1} }
    \seq_pop_right:NN \l__spintent_spcomb_list_seq \l__spintent_spcomb_last_tl
    \seq_use:Nn \l__spintent_spcomb_list_seq { \__spintent_spcomb_sep: }
    \__spintent_spcomb_sep:
    \exp_args:NV \__spintent_spcomb_last:n \l__spintent_spcomb_last_tl
  }
```

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

__spintent_spcomb_symbol: The function __spintent_spcomb_symbol: typesets the symbol: \binom with n and k, or the letters with k as the subscript and n as the superscript. With one argument n is the subscript, and with a list of multiplicities they form the superscript. The letters are grouped so the scripts belong to all of them.

```
\cs_new_protected:Nn \__spintent_spcomb_symbol:
  {
    \bool_if:NTF \l__spintent_spcomb_use_binom_bool
      {
        \int_compare:nNnTF { \mathstyle } < { 2 } { \Bigl } { \bigl } \lparen
        \genfrac { } { } { 0pt } { }
          {
            \__spintent_spcomb_arg:nn { n }
              { \str_use:N \l__spintent_spcomb_luaset_n_str }
          }
          {
            \__spintent_spcomb_arg:nn { k }
              { \str_use:N \l__spintent_spcomb_luaset_k_str }
          }
        \int_compare:nNnTF { \mathstyle } < { 2 } { \Bigr } { \bigr } \rparen
      }
      {
        { \tl_use:N \l__spintent_spcomb_letter_tl }
        \str_case:Vn \l__spintent_spcomb_mode_str
          {
            { pair }
              {
                \c_math_subscript_token
                  {
                    \__spintent_spcomb_arg:nn { k }
                      { \str_use:N \l__spintent_spcomb_luaset_k_str }
                  }
                ^
                  {
                    \__spintent_spcomb_arg:nn { n }
                      { \str_use:N \l__spintent_spcomb_luaset_n_str }
                  }
              }
            { one }
              {
                \c_math_subscript_token
                  {
                    \__spintent_spcomb_arg:nn { n }
                      { \str_use:N \l__spintent_spcomb_luaset_n_str }
                  }
              }
            { list }
              {
                \c_math_subscript_token
                  {
                    \__spintent_spcomb_arg:nn { n }
                      { \str_use:N \l__spintent_spcomb_luaset_n_str }
                  }
                ^
                  {
                    \__spintent_spcomb_arg:nn { m } { \__spintent_spcomb_list: }
                  }
              }
          }
      }
  }
```

(*End of definition for* __spintent_spcomb_symbol:.)

__spintent_spcomb_mathml: The function __spintent_spcomb_mathml: prints the symbol inside one native \MathMLintent, preceded by the outer ghost __spintent_invisible_sep: and by the inner ghost __spintent_inner_sep:.

With silent=true the intent is '_-'. With read-arg given it is _⟨*read-arg*⟩:nofix. Otherwise the intent of the whole symbol is the concept followed by -de: with one argument it is _⟨*concept*⟩-de:prefix($c), where c wraps the symbol with the intent _elementos:postfix($n); with pair it is _⟨*concept*⟩-de:prefix($c), where c is the symbol, which carries its own intent _elementos-tomados-de-a:infix($n,$k), except for the binomial coefficient, whose single intent is _sobre:infix($n,$k); with list it is _⟨*concept*⟩-de:prefix($c), where c is the symbol, which carries

the intent _elementos-con-repeticiones:infix($n,$m), and m is the group of the multiplicities, typeset by __spintent_spcomb_list:.

```
\cs_new_protected:Nn \__spintent_spcomb_mathml:
  {
    \__spintent_invisible_sep:
    \bool_if:NTF \l__spintent_spcomb_silent_bool
      {
        \cs_set_eq:NN \__spintent_spcomb_arg:nn \__spintent_spcomb_arg_plain:nn
        \cs_set_eq:NN \__spintent_spcomb_sep: \__spintent_spcomb_sep_plain:
        \cs_set_eq:NN \__spintent_spcomb_last:n \__spintent_spcomb_last_plain:n
        \MathMLintent { _- }
          { \__spintent_inner_sep: \__spintent_spcomb_symbol: }
      }
      {
        \tl_if_empty:NTF \l__spintent_spcomb_read_arg_tl
          {
            \cs_set_eq:NN \__spintent_spcomb_arg:nn \__spintent_spcomb_arg_ann:nn
            \cs_set_eq:NN \__spintent_spcomb_sep: \__spintent_spcomb_sep_ann:
            \cs_set_eq:NN \__spintent_spcomb_last:n \__spintent_spcomb_last_ann:n
            \str_case:Vn \l__spintent_spcomb_mode_str
              {
                { one }
                  {
                    \MathMLintent
                      { _ \tl_use:N \l__spintent_spcomb_concept_tl -de:prefix($c) }
                      {
                        \__spintent_inner_sep:
                        \MathMLarg { c }
                          {
                            \MathMLintent { _elementos:postfix($n) }
                              { \__spintent_inner_sep: \__spintent_spcomb_symbol: }
                          }
                      }
                  }
                { pair }
                  {
                    \bool_if:NTF \l__spintent_spcomb_use_binom_bool
                      {
                        \MathMLintent { _sobre:infix($n,$k) }
                          { \__spintent_inner_sep: \__spintent_spcomb_symbol: }
                      }
                      {
                        \MathMLintent
                          { _ \tl_use:N \l__spintent_spcomb_concept_tl -de:prefix($c) }
                          {
                            \__spintent_inner_sep:
                            \MathMLarg { c }
                              {
                                \MathMLintent { _elementos-tomados-de-a:infix($n,$k) }
                                  { \__spintent_inner_sep: \__spintent_spcomb_symbol: }
                              }
                          }
                      }
                  }
                { list }
                  {
                    \MathMLintent
                      { _ \tl_use:N \l__spintent_spcomb_concept_tl -de:prefix($c) }
                      {
                        \__spintent_inner_sep:
                        \MathMLarg { c }
                          {
                            \MathMLintent { _elementos-con-repeticiones:infix($n,$m) }
                              { \__spintent_inner_sep: \__spintent_spcomb_symbol: }
                          }
                      }
                  }
              }
          }
          {
            \cs_set_eq:NN \__spintent_spcomb_arg:nn \__spintent_spcomb_arg_plain:nn
```

```
8887              \cs_set_eq:NN \__spintent_spcomb_sep: \__spintent_spcomb_sep_plain:
8888              \cs_set_eq:NN \__spintent_spcomb_last:n \__spintent_spcomb_last_plain:n
8889              \MathMLintent { _ \tl_use:N \l__spintent_spcomb_read_arg_tl :nofix }
8890                { \__spintent_inner_sep: \__spintent_spcomb_symbol: }
8891            }
8892        }
8893    }
```

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

`\__spintent_spcomb_exec:nn` The function `\__spintent_spcomb_exec:nn` sets the ⟨*keys*⟩ in '`#1`', rejects an incompatible combination of keys, asks Lua to split the argument '`#2`' at its first comma, decides the mode from the keys and from the variable `\l__spintent_spcomb_luaset_single_str`, and calls `\__spintent_luafun_spcomb_prepare:nnnn` with the pieces. Whatever the value of `silent`, an invalid case is rejected with its error message and nothing is printed; otherwise the printing is delegated to `\__spintent_spcomb_mathml:`.

```
8894 \cs_new_protected:Nn \__spintent_spcomb_exec:nn
8895   {
8896     \keys_set:nn { spintent / spcomb } {#1}
8897     \__spintent_spcomb_setup:
8898     \str_if_empty:NTF \l__spintent_spcomb_problem_str
8899       {
8900         \__spintent_luafun_spcomb_split:n { \exp_not:n {#2} }
8901         \bool_if:NTF \l__spintent_spcomb_circular_bool
8902           {
8903             \str_if_eq:VnTF \l__spintent_spcomb_luaset_single_str { true }
8904               { \str_set:Nn \l__spintent_spcomb_mode_str { one } }
8905               { \str_set:Nn \l__spintent_spcomb_mode_str { extra } }
8906           }
8907           {
8908             \bool_lazy_and:nnTF
8909               { \str_if_eq_p:Vn \l__spintent_spcomb_kind_str { permutation } }
8910               { \bool_if_p:N \l__spintent_spcomb_repeat_bool }
8911               {
8912                 \str_if_eq:VnTF \l__spintent_spcomb_luaset_single_str { true }
8913                   { \str_set:Nn \l__spintent_spcomb_mode_str { missing } }
8914                   { \str_set:Nn \l__spintent_spcomb_mode_str { list } }
8915               }
8916               {
8917                 \str_if_eq:VnTF \l__spintent_spcomb_luaset_single_str { true }
8918                   {
8919                     \str_if_eq:VnTF \l__spintent_spcomb_kind_str { permutation }
8920                       { \str_set:Nn \l__spintent_spcomb_mode_str { one } }
8921                       { \str_set:Nn \l__spintent_spcomb_mode_str { missing } }
8922                   }
8923                   { \str_set:Nn \l__spintent_spcomb_mode_str { pair } }
8924               }
8925           }
8926         \str_case:VnF \l__spintent_spcomb_mode_str
8927           {
8928             { extra }
8929               { \msg_error:nn { spintent } { spcomb-extra-arg } }
8930             { missing }
8931               { \msg_error:nn { spintent } { spcomb-missing-arg } }
8932           }
8933           {
8934             \__spintent_luafun_spcomb_prepare:nnnn
8935               { \l__spintent_spcomb_mode_str }
8936               { \l__spintent_spcomb_luaset_raw_n_str }
8937               { \l__spintent_spcomb_luaset_raw_k_str }
8938               {
8939                 \bool_if:NTF \l__spintent_spcomb_repeat_bool
8940                   { false } { true }
8941               }
8942             \str_if_eq:VnTF \l__spintent_spcomb_luaset_error_str { true }
8943               {
8944                 \msg_error:nnee { spintent } { spcomb-invalid-arg }
8945                   { \l__spintent_spcomb_luaset_raw_n_str }
8946                   { \l__spintent_spcomb_luaset_raw_k_str }
8947               }
8948               { \__spintent_spcomb_mathml: }
8949           }
```

©2026 by Pablo González L

```
      }
      {
        \msg_error:nne { spintent } { spcomb-invalid-keys }
          { \l__spintent_spcomb_problem_str }
      }
  }
```

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

`\__spintent_spcomb_begin:nn` The function `\__spintent_spcomb_begin:nn` opens a local group, activates the scope attribute and delegates execution to `\__spintent_spcomb_exec:nn`.

```
\cs_new_protected:Nn \__spintent_spcomb_begin:nn
  {
    \group_begin:
      \attribute \l__spintent_scope_attr_int = 1
      \__spintent_spcomb_exec:nn {#1} {#2}
    \group_end:
  }
```

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

`\spcomb` The public command `\spcomb` enforces *math mode* and delegates execution to `\__spintent_spcomb_begin:nn`, which opens a local group. The mandatory argument '`#2`' is `n` or `n,k`, and Lua splits it.

```
\NewDocumentCommand \spcomb { O{} m }
  {
    \mode_if_math:TF
      { \__spintent_spcomb_begin:nn {#1} {#2} }
      { \msg_error:nnn { spintent } { need-math-mode } { \spcomb } }
  }
```

(*End of definition for* `\spcomb`. *This function is documented on page 48.*)

15.67 The command \spfact

The user command `\spfact` typesets the factorial `n!` and reads it *«n factorial»*. It has the keys `silent` and `read-arg`.

15.67.1 Definition of variables for \spfact

`\l__spintent_spfact_read_arg_tl` The token list variable `\l__spintent_spfact_read_arg_tl` stores the key `read-arg`.

```
\tl_new:N \l__spintent_spfact_read_arg_tl
```

(*End of definition for* `\l__spintent_spfact_read_arg_tl`.)

15.67.2 Definition of error messages for \spfact

`spfact-empty-arg` The "error" `spfact-empty-arg` is raised when the argument of `\spfact` is missing or *blank*.

```
\msg_new:nnnn { spintent } { spfact-empty-arg }
  {
    Empty~argument~for~'\token_to_str:N \spfact'~\msg_line_context:.
  }
  {
    The~number~n~is~required.
  }
```

(*End of definition for* `spfact-empty-arg`.)

15.67.3 Definition of keys for \spfact

`silent` `read-arg` `unknown` We define the ⟨*keys*⟩ `silent` and `read-arg`.

```
\keys_define:nn { spintent / spfact }
  {
    silent   .bool_set:N = \l__spintent_spfact_silent_bool,
    silent   .initial:n = false,
    silent   .value_required:n = true,
    read-arg .code:n =
      { \__spintent_sanitize_affix:Nn \l__spintent_spfact_read_arg_tl {#1} },
    read-arg .initial:n = ,
    read-arg .value_required:n = true,
    unknown  .code:n =
      {
```

```
8988          \str_set:Nn \l__spintent_command_name_str { spfact }
8989          \__spintent_unknown_keys_cmd:n {#1}
8990        },
8991    }
```

(*End of definition for* `silent` , `read-arg` , *and* `unknown`.)

15.67.4 Internal functions for `\spfact`

`\__spintent_spfact_exec:nn` The function `\__spintent_spfact_exec:nn` sets the ⟨*keys*⟩ in '`#1`' and calls `\__spintent_spfact_mathml:n` with the argument in '`#2`'.

```
8992 \cs_new_protected:Nn \__spintent_spfact_exec:nn
8993   {
8994     \keys_set:nn { spintent / spfact } {#1}
8995     \__spintent_spfact_mathml:n {#2}
8996   }
```

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

`\__spintent_spfact_mathml:n` The function `\__spintent_spfact_mathml:n` prints the factorial of '`#1`' inside one native `\MathMLintent`, preceded by the outer ghost `\__spintent_invisible_sep:` and followed by the inner ghost `\__spintent_inner_sep:`. With `silent=true` the intent is '`_-`'; with `read-arg` given it is `_`⟨*read-arg*⟩`:nofix`; otherwise it is `_factorial:postfix($n)`.

```
8997 \cs_new_protected:Nn \__spintent_spfact_mathml:n
8998   {
8999     \__spintent_invisible_sep:
9000     \bool_if:NTF \l__spintent_spfact_silent_bool
9001       { \MathMLintent { _- } { \__spintent_inner_sep: #1 \mathexclam } }
9002       {
9003         \tl_if_empty:NTF \l__spintent_spfact_read_arg_tl
9004           {
9005             \MathMLintent { _factorial:postfix($n) }
9006               {
9007                 \__spintent_inner_sep:
9008                 \MathMLarg { n } { \__spintent_inner_sep: #1 }
9009                 \mathexclam
9010               }
9011           }
9012           {
9013             \MathMLintent { _ \tl_use:N \l__spintent_spfact_read_arg_tl :nofix }
9014               { \__spintent_inner_sep: #1 \mathexclam }
9015           }
9016       }
9017   }
```

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

`\spfact` The public command `\spfact` enforces *math mode*, opens a local group and delegates execution to `\__spintent_spfact_exec:nn`.

```
9018 \NewDocumentCommand \spfact { O{} m }
9019   {
9020     \mode_if_math:TF
9021       {
9022         \group_begin:
9023           \attribute \l__spintent_scope_attr_int = 1
9024           \tl_if_blank:nTF {#2}
9025             { \msg_error:nn { spintent } { spfact-empty-arg } }
9026             { \__spintent_spfact_exec:nn {#1} {#2} }
9027         \group_end:
9028       }
9029       { \msg_error:nnn { spintent } { need-math-mode } { \spfact } }
9030   }
```

(*End of definition for* `\spfact`. *This function is documented on page* 48.)

15.68 Finish package

Finish package implementation.

```
9031 \file_input_stop:
9032 ⟨/package⟩
```

16 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

autofit 4574

B

bar 8434

C

card 8162
cifras-sep 308, 585, 871, 1959, 2034, 2692
circular 8608
Document class:
  l3doc *58*
close 4188
cmd-key-unknown 215
cmd-key-value-unknown 215
Commands provided by spanish-mathml-intent:
  \setspintent *64*
  \spAcirc *213*
  \spAfig *195*
  \spFrec *228*
  \spOmega *230*
  \spPcirc *213*
  \spPerc *226*
  \spPfig *195*
  \spPoint *157*
  \spPoly *187*
  \spProb *235*
  \spProm *218*
  \spProp *115*
  \spPval *232*
  \spVol *217*
  \spabs *153*
  \spangle *180*
  \spang *117*
  \sparc *183*
  \spcase *230*
  \spcirc *206*
  \spcod *149*
  \spcomb *246*
  \spcomp *238*
  \spcoord *129*
  \spcuartil *226*
  \spdate *93*
  \spdecil *226*
  \spdesv *222*
  \spdiam *200*
  \spdiv *103*
  \spdom *149*
  \spdsym *100*
  \spfact *247*
  \spfuncomp *147*
  \spfun *143*
  \spinterval *138*
  \spmQ *125*
  \spmangle *180*
  \spmarc *183*
  \spmax *223*
  \spmcd *109*
  \spmcm *109*
  \spmediana *222*
  \spmedia *222*
  \spmin *223*
  \spmoda *222*
  \spmoney *84*
  \spmside *174*
  \spmsym *101*
  \spmul *105*
  \spnD *113*
  \spnM *113*
  \spnZ *118*
  \spnewread *98*
  \spnewunit *77*
  \spnum *71*
  \sprad *200*
  \sprango *223*
  \sprayo *169*
  \sprazon *115*
  \spread *98*
  \sprecta *167*
  \sprec *149*
  \spriq *223*
  \spshort *87*
  \spside *174*
  \spsiglo *90*
  \sptime *92*
  \spunitalias *78*
  \spunit *76*
  \spusep *96*
  \spvarz *222*
  \spvec *162*
concept 2114, 2352, 3238, 3630, 4464, 4574, 4823, 5083, 5227, 5444, 5741, 5897, 6090, 6504, 6766, 7076, 7393
concept* 6090
currency 871

D

diameter 6766

I

inline 2537
int commands:
  \int_set:Nn 70

J

join-word 2770

K

Keys for \spabs provided by spanish-mathml-intent:
  close *150, 152*
  read-arg *150*
  silent *150*
  size *150, 151*
Keys for \spAcirc provided by spanish-mathml-intent:
  concept *207*
  prefix *207*
  print-sym *207, 209*
  read-arg *207*
  read-cmd *207*
  read-sty *207*
  read-sub *207*

©2026 by Pablo González L

silent . . . 207
sub-index . . . 207
Keys for \spAfig provided by spanish-mathml-intent:
concept . . . 188
prefix . . . 187, 188
print-sym . . . 187, 188
read-arg . . . 187, 188, 190
read-cmd . . . 187, 188
read-sub . . . 188
silent . . . 188
Keys for \spangle provided by spanish-mathml-intent:
concept . . . 175
prefix . . . 174, 175
print-sym . . . 175
read-cmd . . . 174, 175
recto . . . 174, 175
silent . . . 175
Keys for \sparc provided by spanish-mathml-intent:
concept . . . 181
prefix . . . 180, 181
read-cmd . . . 180, 181
silent . . . 181
Keys for \spcase provided by spanish-mathml-intent:
read-arg . . . 228, 229
silent . . . 229
Keys for \spcirc provided by spanish-mathml-intent:
concept . . . 201
diameter . . . 201
prefix . . . 200, 201
print-sym . . . 200, 201, 203
radio-sty . . . 200, 201
read-arg . . . 200, 201
read-cmd . . . 200, 201
read-sty . . . 200, 201
read-sub . . . 201
silent . . . 201
sub-index . . . 200, 201
Keys for \spcomb provided by spanish-mathml-intent:
circular . . . 239, 240
kind . . . 238–240
print-sty . . . 239, 240
read-arg . . . 239, 240
repeat . . . 239, 240
silent . . . 240
Keys for \spcomp provided by spanish-mathml-intent:
bar . . . 236–238
read-arg . . . 236
silent . . . 236
wrap-parens . . . 236, 237
Keys for \spcoord provided by spanish-mathml-intent:
silent . . . 127
Keys for \spcuartil provided by spanish-mathml-intent:
read-arg . . . 224
silent . . . 224
Keys for \spdecil provided by spanish-mathml-intent:
read-arg . . . 224
silent . . . 224
Keys for \spdesv provided by spanish-mathml-intent:
read-arg . . . 219
read-sub . . . 219
sigma . . . 219
silent . . . 219
sub-index . . . 219
Keys for \spdiam provided by spanish-mathml-intent:
concept . . . 196
read-arg . . . 196
read-cmd . . . 196
read-sub . . . 196
silent . . . 196
sub-index . . . 196
upper-case . . . 196
Keys for \spdiv provided by spanish-mathml-intent:
cifras-sep . . . 102
notation-sym . . . 102
read-parens . . . 102
read-period-sep . . . 102
read-sign . . . 102
read-sym . . . 102
set-sym . . . 102
wrap-parens . . . 102
Keys for \spdom provided by spanish-mathml-intent:
prefix . . . 148
silent . . . 148
Keys for \spdsym provided by spanish-mathml-intent:
prefix . . . 99
read-sym . . . 99
set-sym . . . 99
suffix . . . 99
Keys for \spfact provided by spanish-mathml-intent:
read-arg . . . 246
silent . . . 246
Keys for \spFrec provided by spanish-mathml-intent:
read-arg . . . 226, 227
read-sub . . . 227
silent . . . 227
sub-index . . . 226, 227
Keys for \spfuncomp provided by spanish-mathml-intent:
silent . . . 144
Keys for \spfun provided by spanish-mathml-intent:
concept . . . 138, 139
prefix . . . 138, 139
read-cmd . . . 138, 139
silent . . . 138, 139
Keys for \spinterval provided by spanish-mathml-intent:
concept . . . 131, 134
prefix . . . 130, 131, 134
read-arg . . . 130, 131, 134
read-cmd . . . 130, 131, 136
silent . . . 131, 134, 137
size . . . 130, 131, 133
Keys for \spmangle provided by spanish-mathml-intent:
concept . . . 175
prefix . . . 174, 175
print-sym . . . 174, 175
read-cmd . . . 174, 175
recto . . . 174, 175
silent . . . 175
Keys for \spmarc provided by spanish-mathml-intent:
concept . . . 181
prefix . . . 180, 181
read-cmd . . . 180, 181
silent . . . 181
Keys for \spmax provided by spanish-mathml-intent:
read-arg . . . 220
silent . . . 220
Keys for \spmcd provided by spanish-mathml-intent:
concept . . . 106
prefix . . . 106
read-sty . . . 106
result . . . 106

©2026 by Pablo González L

silent . . . 106
upper-case . . . 106
Keys for \spmcm provided by spanish-mathml-intent:
concept . . . 106
prefix . . . 106
read-sty . . . 106
result . . . 106
silent . . . 106
upper-case . . . 106
Keys for \spmediana provided by spanish-mathml-intent:
read-arg . . . 219
read-sub . . . 219
silent . . . 219
sub-index . . . 219
Keys for \spmedia provided by spanish-mathml-intent:
read-arg . . . 219
read-sub . . . 219
silent . . . 219
sub-index . . . 219
Keys for \spmin provided by spanish-mathml-intent:
read-arg . . . 220
silent . . . 220
Keys for \spmoda provided by spanish-mathml-intent:
read-arg . . . 219
read-sub . . . 219
silent . . . 219
sub-index . . . 219
Keys for \spmoney provided by spanish-mathml-intent:
cifras-sep . . . 79
currency . . . 79, 80
money-math . . . 79
sub-units . . . 79
Keys for \spmQ provided by spanish-mathml-intent:
join-word . . . 119–121
read-parens . . . 120, 121
silent . . . 120, 124
use-dfrac . . . 120–123
use-spnZ . . . 120, 123
wrap-parens . . . 119–121
Keys for \spmside provided by spanish-mathml-intent:
concept . . . 171
prefix . . . 170, 171
print-sym . . . 170, 171, 196
read-arg . . . 170, 171
read-cmd . . . 170, 171, 173
read-sty . . . 170, 171
silent . . . 171, 173
Keys for \spmsym provided by spanish-mathml-intent:
prefix . . . 100
print-sym . . . 100, 101
read-sym . . . 100
set-sym . . . 100
suffix . . . 100
Keys for \spmul provided by spanish-mathml-intent:
cifras-sep . . . 104
notation-sym . . . 104
read-parens . . . 104
read-period-sep . . . 104
read-sign . . . 104
read-sym . . . 104
set-sym . . . 104
wrap-parens . . . 104
Keys for \spnD provided by spanish-mathml-intent:
concept . . . 110
prefix . . . 110
result-num . . . 110
result . . . 110
silent . . . 110
Keys for \spnM provided by spanish-mathml-intent:
concept . . . 110
prefix . . . 110
result-num . . . 110
result . . . 110
silent . . . 110
Keys for \spnum provided by spanish-mathml-intent:
cifras-sep . . . 65, 67, 68
notation-sym . . . 67
read-period-sep . . . 67, 69
read-sign . . . 67
Keys for \spnZ provided by spanish-mathml-intent:
cifras-sep . . . 118
read-parens . . . 118
read-sign . . . 118
wrap-parens . . . 118
Keys for \spOmega provided by spanish-mathml-intent:
card . . . 230
read-arg . . . 230
silent . . . 230
sub-index . . . 230
Keys for \spPcirc provided by spanish-mathml-intent:
concept . . . 207
prefix . . . 207
print-sym . . . 207, 209
read-arg . . . 207
read-cmd . . . 207
read-sty . . . 207
read-sub . . . 207
silent . . . 207
sub-index . . . 207
Keys for \spPerc provided by spanish-mathml-intent:
read-arg . . . 224
silent . . . 224
Keys for \spPfig provided by spanish-mathml-intent:
concept . . . 188
prefix . . . 187, 188
print-sym . . . 187, 188
read-arg . . . 187, 188, 190
read-cmd . . . 187, 188
read-sub . . . 188
silent . . . 188
Keys for \spPoint provided by spanish-mathml-intent:
concept . . . 156
prefix . . . 155, 156
read-cmd . . . 155, 156
silent . . . 156
Keys for \spPoly provided by spanish-mathml-intent:
concept . . . 184
prefix . . . 184
print-sym . . . 184
read-cmd . . . 184
silent . . . 184
Keys for \spProb provided by spanish-mathml-intent:
read-arg . . . 232, 233
silent . . . 233
Keys for \spProm provided by spanish-mathml-intent:
read-arg . . . 217
silent . . . 217
Keys for \spPval provided by spanish-mathml-intent:
silent . . . 231
use-dfrac . . . 231

©2026 by Pablo González L

Keys for \sprad provided by spanish-mathml-intent:
concept . . . 196
read-arg . . . 196
read-cmd . . . 196
read-sub . . . 196
silent . . . 196
sub-index . . . 196
upper-case . . . 196
Keys for \sprango provided by spanish-mathml-intent:
read-arg . . . 219
read-sub . . . 219
silent . . . 219
sub-index . . . 219
Keys for \sprayo provided by spanish-mathml-intent:
concept . . . 168
prefix . . . 167, 168
read-cmd . . . 167–169
read-sty . . . 167, 168
silent . . . 168, 169
Keys for \sprazon provided by spanish-mathml-intent:
inline . . . 114
silent . . . 114
use-dfrac . . . 114
Keys for \spread provided by spanish-mathml-intent:
read-arg . . . 97
silent . . . 97
Keys for \sprecta provided by spanish-mathml-intent:
concept . . . 163
prefix . . . 162, 163
read-cmd . . . 162, 163
silent . . . 163
Keys for \spriq provided by spanish-mathml-intent:
read-arg . . . 219
read-sub . . . 219
silent . . . 219
sub-index . . . 219
Keys for \spshort provided by spanish-mathml-intent:
read-arg . . . 85
small-caps . . . 85
Keys for \spside provided by spanish-mathml-intent:
concept . . . 171
prefix . . . 170, 171
read-arg . . . 170, 171
read-cmd . . . 170, 171, 173
read-sty . . . 170, 171
silent . . . 171, 173
Keys for \spsiglo provided by spanish-mathml-intent:
print-era . . . 88, 90
Keys for \spunit provided by spanish-mathml-intent:
cifras-sep . . . 73
notation-sym . . . 73
print-cdot . . . 73
read-cdot . . . 73
read-period-sep . . . 73
read-slash . . . 73
Keys for \spusep provided by spanish-mathml-intent:
silent . . . 94
size . . . 94
Keys for \spvarz provided by spanish-mathml-intent:
read-arg . . . 219
read-sub . . . 219
sigma . . . 219
silent . . . 219
sub-index . . . 219
Keys for \spvec provided by spanish-mathml-intent:
autofit . . . 158
concept . . . 158
prefix . . . 158
read-cmd . . . 158
silent . . . 158
Keys for \spVol provided by spanish-mathml-intent:
concept . . . 214
read-arg . . . 214
read-cmd . . . 214
read-sub . . . 214
silent . . . 214
sub-index . . . 214
Keys for \usepackage provided by spanish-mathml-intent:
AFghost . . . 58
mathcal . . . 58
kind . . . 8608

M

mathcal-fallback . . . 86
mc-too-few-args . . . 2338
md-invalid-arg . . . 2338
Lua module provided by spanish-mathml-intent:
spintent.lua 58–61, 65, 66, 72, 77, 78, 80, 85, 88–91, 93, 105, 109, 110, 116, 119, 125, 138, 144, 155
money-math . . . 871

N

notation-sym . . . 308, 585, 1959, 2034

P

Packages :
expl3 . . . 58, 65, 66
lua-unicode-math . . . 59
luamml . . . 61, 119, 122, 183, 186, 205
tagpdf . . . 85, 89
unicode-math . . . 59, 95
prefix 1798, 1873, 2114, 2352, 3238, 3630, 4077, 4464, 4574, 4823, 5083, 5227, 5444, 5741, 5897, 6090, 6766, 7076
print-cdot . . . 585
print-era . . . 1303
print-sty . . . 8608
print-sym . 1873, 5227, 5444, 5897, 6090, 6504, 6766, 7076

R

radio-sty . . . 6766
read-arg . . 1161, 1714, 3238, 4188, 5227, 6090, 6504, 6766, 7076, 7393, 7594, 7659, 7697, 7883, 8019, 8112, 8162, 8273, 8434, 8608, 8977
read-cdot . . . 585
read-cmd . . 3238, 3630, 4464, 4574, 4823, 5083, 5227, 5444, 5741, 5897, 6090, 6504, 6766, 7076, 7393
read-parens . . . 1959, 2034, 2692, 2770
read-period-sep . . . 308, 585, 1959, 2034
read-sign . . . 308, 1959, 2034, 2692
read-slash . . . 585
read-sty . . . 2114, 5083, 5227, 6766, 7076
read-sub . . . 6090, 6504, 6766, 7076, 7393, 7659, 8019
read-sym . . . 1798, 1873, 1959, 2034
recto . . . 5444
repeat . . . 8608
result . . . 2114, 2352
result-num . . . 2352
right-angle-sqr-fallback . . . 86

©2026 by Pablo González L

S

seq commands:
\seq_set_from_clist:Nn ... 70
set-sym ... 1798, 1873, 1959, 2034
\setspintent ... 6, 63, 202
sigma ... 7687
silent 1572, 1714, 2114, 2352, 2537, 2770, 3032, 3238, 3630, 3891, 4077, 4188, 4464, 4574, 4823, 5083, 5227, 5444, 5741, 5897, 6090, 6504, 6766, 7076, 7393, 7594, 7659, 7697, 7883, 8019, 8112, 8162, 8204, 8273, 8434, 8608, 8977
size ... 1572, 3238, 4188
small-caps ... 1161
\spabs ... 41, 150, 4322
spabs-empty-arg ... 4181
\spAcirc ... 35, 206, 7360
spAcirc-invalid-arg ... 7067
\spAfig ... 30, 187, 6469
spAfig-invalid-arg ... 6050
spAfig-invalid-points ... 6050
spAfig-read-arg-key ... 6050
spAfig-sym-cardinality-conflict ... 6050
spAfig-sym-letter-conflict ... 6050
\spang ... 29, 116, 2675
spang-invalid-arg ... 2630
\spangle ... 27, 174, 5699
spangle-invalid-arg ... 5433
\sparc ... 29, 180, 5862
sparc-invalid-points ... 5731
\spcase ... 44, 228, 8146
spcase-invalid-letter ... 8105
\spcirc ... 34, 200, 7039
spcirc-invalid-arg ... 6759
\spcod ... 40, 147, 4144
\spcomb ... 47, 238, 8963
spcomb-extra-arg ... 8577
spcomb-invalid-arg ... 8577
spcomb-invalid-keys ... 8577
spcomb-missing-arg ... 8577
\spcomp ... 46, 236, 8547
spcomp-empty-arg ... 8427
\spcoord ... 23, 126, 3183
spcoord-invalid-arg ... 3025
\spcuartil ... 43, 7964
\spdate ... 15, 92, 1544
spdate-invalid-arg ... 1519
\spdecil ... 43, 7964
\spdesv ... 42, 7810
\spdiam ... 33, 196, 6720
\spdiv ... 18, 102, 2017
spdiv-empty-arg ... 1952
\spdom ... 40, 147, 4144
spdom-invalid-arg ... 4070
\spdsym ... 17, 98, 1858
\spfact ... 47, 246, 9018
spfact-empty-arg ... 8970
\spFrec ... 44, 226, 8089
spFrec-invalid-sub-index ... 8004
spFrec-invalid-symbol ... 8004
\spfun ... 40, 138, 3864
spfun-invalid-arg ... 3622
\spfuncomp ... 41, 144, 4053
spfuncomp-invalid-arg ... 3884
spintent internal commands:
__spintent_collapse_hyphens:N ... 134
\l__spintent_command_name_str ... 215
\l__spintent_geo_final_print_tl ... 6039
\l__spintent_geo_luaset_angulo_case_-str ... 3006
\l__spintent_geo_luaset_circ_radio_raw_-str ... 6742
\l__spintent_geo_luaset_circ_radio_type_-str ... 6742
\l__spintent_geo_luaset_concept_str ... 3004
\l__spintent_geo_luaset_error_str ... 2996
\l__spintent_geo_luaset_intent_str ... 2996
\l__spintent_geo_luaset_is_name_str ... 2999
\l__spintent_geo_luaset_letra_conector_-str ... 3008
\l__spintent_geo_luaset_letra_str ... 2999
\l__spintent_geo_luaset_letra_word_str 3008
\l__spintent_geo_luaset_nombre_es_num_-str ... 3006
\l__spintent_geo_luaset_points_clist ... 2999
\l__spintent_geo_luaset_poly_sym_str ... 3004
\l__spintent_geo_luaset_print_tl ... 2996
\l__spintent_geo_luaset_sub_spoken_str 3008
\l__spintent_geo_luaset_sub_str ... 2999
\l__spintent_geo_luaset_tipo_str ... 2999
\l__spintent_geo_print_sym_str ... 6041
\l__spintent_geo_temp_tl ... 6048
__spintent_inner_sep: ... 126
__spintent_invisible_sep: ... 122
\c__spintent_invisible_sep_int ... 130
__spintent_luafun_clean_split_and_-set:n ... 252
__spintent_luafun_mrow_intent_arg:nn ... 122
__spintent_luafun_set_debug_ghost:n ... 77
__spintent_luafun_spfun_parse_and_set:n 82
__spintent_luafun_spmoney_lookup_data:n 82
__spintent_luafun_spmQ_scale_int:nn ... 82
__spintent_luafun_spunit_parse_-product:nn ... 82
\l__spintent_luaset_arg_above_tl ... 270
\l__spintent_luaset_arg_below_tl ... 270
\l__spintent_luaset_cal_fam_ok_str ... 73
\l__spintent_luaset_dec_is_all_zeros_-str ... 263
\l__spintent_luaset_dec_sep_tl ... 252
\l__spintent_luaset_decimal_period_str 263
\l__spintent_luaset_denom_has_number_-str ... 270
\l__spintent_luaset_exponent_tl ... 263
\l__spintent_luaset_format_part_dec_str 258
\l__spintent_luaset_format_part_int_str 258
\l__spintent_luaset_has_exponent_str ... 263
\l__spintent_luaset_has_integer_str ... 252
\l__spintent_luaset_has_units_str ... 270
\l__spintent_luaset_millons_str ... 263
\l__spintent_luaset_not_decimal_not_-period_str ... 263
\l__spintent_luaset_not_decimal_period_-str ... 263
\l__spintent_luaset_only_part_dec_str ... 258

©2026 by Pablo González L

\l__spintent_luaset_only_part_int_str . 258
\l__spintent_luaset_only_part_period_-
str 258
\l__spintent_luaset_part_dec_tl 252
\l__spintent_luaset_part_int_tl 252
\l__spintent_luaset_part_period_tl . . . 252
\l__spintent_luaset_scope_attr_str 73
\l__spintent_luaset_sign_tl 252
\l__spintent_luaset_status_str 270
\l__spintent_luaset_sym_fam_ok_str 73
__spintent_mathcal:n 100
__spintent_mathmlarg_point:nnn 4339
\l__spintent_mathmlarg_point_kv_tl . . . 4335
\l__spintent_mathmlarg_point_mode_str . 4337
\l__spintent_mathmlarg_point_ref_tl . . . 4335
\l__spintent_mathmlarg_point_silent_-
bool 4337
\l__spintent_mc_args_seq 2104
\l__spintent_mc_idx_int 2104
\l__spintent_mc_luaset_all_numeric_str 2102
\l__spintent_mc_luaset_types_clist . . . 2102
__spintent_mc_mathml_args:n 2175
__spintent_mc_process_args:nn 2155
\l__spintent_mc_raw_args_seq 2104
\l__spintent_mc_types_seq 2104
\l__spintent_md_out_seq 2336
__spintent_print_spunit_hierarchy: . . . 675
__spintent_right_angle_sqr: 113
__spintent_sanitize_affix:Nn 142
\l__spintent_saved_luamml_flag_int . . . 2747
\l__spintent_scope_attr_int 80
\l__spintent_setspintent_input_arg_cmd_-
tl 165
__spintent_setspintent_parse_args:nn . 185
__spintent_setspintent_set_keys:nn . . . 174
__spintent_sp_rad_diam_exec:nn 6715
\l__spintent_sp_rad_diam_letra_tl 6495
__spintent_sp_rad_diam_mathml:n 6551
\l__spintent_sp_rad_diam_read_arg_full_-
tl 6495
__spintent_sp_rad_diam_read_arg_-
print:n 6677
\l__spintent_sp_rad_diam_word_tl 6495
\l__spintent_spabs_close_bool 4179
\l__spintent_spabs_close_mode_str 4174
\l__spintent_spabs_delim_l_tl 4177
\l__spintent_spabs_delim_r_tl 4177
__spintent_spabs_exec:nn 4278
\l__spintent_spabs_luaset_close_str . . . 4179
__spintent_spabs_mathml:n 4284
\l__spintent_spabs_read_arg_tl 4174
__spintent_spabs_set_close:n 4264
__spintent_spabs_set_delims: 4233
\l__spintent_spabs_size_str 4174
\l__spintent_spAcirc_mode_str 7052
\l__spintent_spAcirc_prefix_tl 7052
\l__spintent_spAcirc_print_sym_str . . . 7052
\l__spintent_spAcirc_read_arg_tl 7052
\l__spintent_spAcirc_read_sty_str 7052
\l__spintent_spAcirc_sub_index_str . . . 7052
\l__spintent_spAfig_mode_str 6032
\l__spintent_spAfig_prefix_tl 6032
\l__spintent_spAfig_print_sym_str 6032
\l__spintent_spAfig_read_arg_tl 6032
__spintent_spang_exec:n 2638
\l__spintent_spang_luaset_error_str . . . 2626
\l__spintent_spang_luaset_grado_str . . . 2626
\l__spintent_spang_luaset_minuto_str . . 2626
\l__spintent_spang_luaset_segundo_str . 2626
__spintent_spang_mathml: 2648
__spintent_spangle_exec:nnn 5503
\l__spintent_spangle_intent_arg_tl . . . 5431
\l__spintent_spangle_intent_tl 5426
\l__spintent_spangle_kv_tl 5431
__spintent_spangle_mathml:n 5540
\l__spintent_spangle_mode_str 5421
\l__spintent_spangle_prefix_tl 5421
\l__spintent_spangle_print_sym_now_-
bool 5426
\l__spintent_spangle_pt_idx_int 5428
\l__spintent_spangle_pt_seq 5428
\l__spintent_spangle_refs_tl 5428
__spintent_spangle_sym:n 5484
__spintent_sparc_exec:nnn 5785
\l__spintent_sparc_intent_arg_tl 5730
\l__spintent_sparc_intent_tl 5726
__spintent_sparc_mathml:n 5803
\l__spintent_sparc_mode_str 5721
\l__spintent_sparc_prefix_tl 5721
__spintent_sparc_print_arc:n 5779
\l__spintent_sparc_pt_idx_int 5727
\l__spintent_sparc_pt_seq 5727
\l__spintent_sparc_refs_tl 5727
__spintent_sparc_wide_arc:n 5775
\l__spintent_spcase_concept_tl 8103
__spintent_spcase_exec:n 8127
\l__spintent_spcase_letter_str 8103
\l__spintent_spcase_read_arg_tl 8101
\l__spintent_spcase_sub_index_str 8101
\l__spintent_spcirc_articulo_str 6751
__spintent_spcirc_build_concept: 6912
\l__spintent_spcirc_connector_arg_tl . . 6753
\l__spintent_spcirc_connector_tl 6753
__spintent_spcirc_exec:nn 6967
\l__spintent_spcirc_has_arg_bool 6753
\l__spintent_spcirc_intent_arg_tl 6753
\l__spintent_spcirc_intent_tl 6753
__spintent_spcirc_mathml: 6984
\l__spintent_spcirc_mode_str 6744
\l__spintent_spcirc_prefix_tl 6744
\l__spintent_spcirc_print_sym_str 6744
\l__spintent_spcirc_radio_arg_tl 6753
__spintent_spcirc_radio_print: 6895
\l__spintent_spcirc_radio_sty_str 6744
\l__spintent_spcirc_read_arg_tl 6744
\l__spintent_spcirc_read_sty_str 6744
\l__spintent_spcirc_sub_index_str 6744
__spintent_spcirc_sym: 6864
__spintent_spcirc_word_and_article: . . 6874
\l__spintent_spcirc_word_tl 6751
__spintent_spcircfig_build_concept:nn 7223
__spintent_spcircfig_build_short:n . . . 7182
__spintent_spcircfig_del_word:n 7169

©2026 by Pablo González L

\l__spintent_spcircfig_del_word_tl ... 7063
__spintent_spcircfig_exec:nnn 7270
__spintent_spcircfig_finish_intent:nn 7207
\l__spintent_spcircfig_intent_arg_tl .. 7063
\l__spintent_spcircfig_intent_tl 7063
\l__spintent_spcircfig_luaset_comma_-
str 7061
__spintent_spcircfig_mathml:nn 7317
\l__spintent_spcircfig_op_tl 7063
\l__spintent_spcircfig_short_bool 7062
__spintent_spcircfig_sym:n 7145
__spintent_spcircfig_word_and_-
article:n 7151
__spintent_spcomb_arg:nn 8724
__spintent_spcomb_arg_ann:nn 8724
__spintent_spcomb_arg_plain:nn 8724
__spintent_spcomb_begin:nn 8956
\l__spintent_spcomb_concept_tl 8570
__spintent_spcomb_exec:nn 8894
\l__spintent_spcomb_kind_str 8567
__spintent_spcomb_last:n 8724
__spintent_spcomb_last_ann:n 8724
__spintent_spcomb_last_plain:n 8724
\l__spintent_spcomb_last_tl 8570
\l__spintent_spcomb_letter_tl 8570
__spintent_spcomb_list: 8752
\l__spintent_spcomb_list_seq 8570
\l__spintent_spcomb_luaset_error_str .. 8563
\l__spintent_spcomb_luaset_k_str 8563
\l__spintent_spcomb_luaset_list_tl ... 8563
\l__spintent_spcomb_luaset_n_str 8563
\l__spintent_spcomb_luaset_raw_k_str .. 8560
\l__spintent_spcomb_luaset_raw_n_str .. 8560
\l__spintent_spcomb_luaset_single_str . 8560
__spintent_spcomb_mathml: 8818
\l__spintent_spcomb_mode_str 8570
\l__spintent_spcomb_print_sty_str 8567
\l__spintent_spcomb_problem_str 8570
\l__spintent_spcomb_read_arg_tl 8567
__spintent_spcomb_sep: 8724
__spintent_spcomb_sep_ann: 8724
__spintent_spcomb_sep_plain: 8724
__spintent_spcomb_setup: 8660
__spintent_spcomb_symbol: 8762
\l__spintent_spcomb_use_binom_bool ... 8570
\l__spintent_spcomp_compound_bool 8424
\l__spintent_spcomp_do_paren_bool 8424
__spintent_spcomp_event:n 8466
__spintent_spcomp_exec:nn 8501
\l__spintent_spcomp_luaset_compound_-
str 8424
__spintent_spcomp_mathml:n 8507
\l__spintent_spcomp_read_arg_tl 8422
__spintent_spcomp_set_paren:n 8466
__spintent_spcomp_typeset:n 8495
\l__spintent_spcomp_wrap_parens_str ... 8422
__spintent_spcoord_exec:nn 3165
__spintent_spcoord_inner_sep:n 3128
__spintent_spcoord_intent_infix: 3134
\l__spintent_spcoord_intent_str 3021
\l__spintent_spcoord_luaset_error_str . 3011
\l__spintent_spcoord_luaset_sep_str ... 3011
\l__spintent_spcoord_luaset_x_composite_-
str 3013
\l__spintent_spcoord_luaset_x_has_frac_-
str 3013
\l__spintent_spcoord_luaset_x_sign_str 3013
\l__spintent_spcoord_luaset_x_type_str 3013
\l__spintent_spcoord_luaset_x_value_-
str 3013
\l__spintent_spcoord_luaset_y_composite_-
str 3013
\l__spintent_spcoord_luaset_y_has_frac_-
str 3013
\l__spintent_spcoord_luaset_y_sign_str 3013
\l__spintent_spcoord_luaset_y_type_str 3013
\l__spintent_spcoord_luaset_y_value_-
str 3013
__spintent_spcoord_mathml:N 3148
__spintent_spcoord_print:n 3105
\l__spintent_spcoord_raw_arg_tl 3021
__spintent_spcoord_set_wrap_parens_-
size: 3043
__spintent_spcoord_value:n 3096
\l__spintent_spcoord_wrap_parens_l_tl . 3021
\l__spintent_spcoord_wrap_parens_r_tl . 3021
\l__spintent_spcuartil_read_arg_tl ... 7870
__spintent_spdate_exec:n 1527
\l__spintent_spdate_luaset_error_str .. 1517
\l__spintent_spdate_luaset_output_str . 1517
__spintent_spdate_mathml: 1537
\l__spintent_spdecil_read_arg_tl 7870
\l__spintent_spdesv_read_arg_tl 7649
\l__spintent_spdesv_sub_index_str 7649
\l__spintent_spdiam_mode_str 6498
\l__spintent_spdiam_read_arg_tl 6498
\l__spintent_spdiam_sub_index_str 6498
__spintent_spdiv_exec:nnn 2004
__spintent_spdiv_mathml:nn 1988
__spintent_spdiv_render_op:nn 1979
\l__spintent_spdom_cmd_str 4064
__spintent_spdom_exec:nnnnn 4130
\l__spintent_spdom_intent_arg_tl 4067
\l__spintent_spdom_luaset_error_str ... 4069
__spintent_spdom_mathml: 4101
__spintent_spdom_name: 4092
\l__spintent_spdom_op_tl 4064
\l__spintent_spdom_prefix_tl 4067
\l__spintent_spdom_word_tl 4064
__spintent_spdsym_mathml: 1845
\l__spintent_spdsym_prefix_tl 1794
\l__spintent_spdsym_read_sym_tl 1794
\l__spintent_spdsym_suffix_tl 1794
\l__spintent_spdsym_symbol_tl 1794
__spintent_spfact_exec:nn 8992
__spintent_spfact_mathml:n 8997
\l__spintent_spfact_read_arg_tl 8969
__spintent_spfig_build_concept_-
vertices:n 6336
__spintent_spfig_build_refs: 6321
__spintent_spfig_build_simple:n 6186
\l__spintent_spfig_desc_tl 6042
__spintent_spfig_exec:nnn 6160

©2026 by Pablo González L

\l__spintent_spfig_intent_arg_tl 6048
\l__spintent_spfig_intent_tl 6042
__spintent_spfig_mathml:n 6427
\l__spintent_spfig_op_tl 6042
__spintent_spfig_points_loop:n 6415
\l__spintent_spfig_pt_idx_int 6045
\l__spintent_spfig_pt_seq 6045
\l__spintent_spfig_refs_tl 6045
__spintent_spfig_sym: 6152
\l__spintent_spfig_symbol_tl 6039
\l__spintent_spFrec_concept_tl 8002
__spintent_spFrec_exec:n 8041
\l__spintent_spFrec_read_arg_tl 8000
\l__spintent_spFrec_sub_index_str 8000
\l__spintent_spFrec_symbol_str 8002
__spintent_spfun_aname:n 3661
\l__spintent_spfun_aname_tl 3616
__spintent_spfun_arg:n 3731
__spintent_spfun_args: 3754
__spintent_spfun_args_plain: 3763
__spintent_spfun_body: 3806
\l__spintent_spfun_call_tl 3616
\l__spintent_spfun_concept_bool 3611
\l__spintent_spfun_depth_int 3613
__spintent_spfun_exec:nn 3850
__spintent_spfun_fname: 3661
\l__spintent_spfun_fname_tl 3616
\l__spintent_spfun_intent_arg_tl 3616
\l__spintent_spfun_kv_tl 3616
\l__spintent_spfun_luaset_error_str ... 3608
\l__spintent_spfun_luaset_inv_str 3608
\l__spintent_spfun_luaset_nargs_str ... 3608
__spintent_spfun_mathml: 3801
\l__spintent_spfun_mode_str 3611
__spintent_spfun_name_mathcat: 3719
__spintent_spfun_name_plain: 3680
__spintent_spfun_name_school: 3692
\l__spintent_spfun_nargs_int 3613
__spintent_spfun_nested:n 3840
\l__spintent_spfun_prefix_tl 3611
\l__spintent_spfun_refs_tl 3616
__spintent_spfun_set_intent: 3771
\l__spintent_spfun_silent_bool 3611
\l__spintent_spfun_tmp_str 3613
__spintent_spfuncomp_exec:nn 4039
__spintent_spfuncomp_fun:n 3902
__spintent_spfuncomp_group: 3938
\l__spintent_spfuncomp_intent_tl 3879
\l__spintent_spfuncomp_kv_tl 3879
\l__spintent_spfuncomp_luaset_apply_-
str 3875
\l__spintent_spfuncomp_luaset_error_-
str 3875
\l__spintent_spfuncomp_luaset_n_str ... 3875
\l__spintent_spfuncomp_luaset_paren_-
str 3875
__spintent_spfuncomp_mathml: 3976
\l__spintent_spfuncomp_n_int 3879
__spintent_spfuncomp_plain: 3962
\l__spintent_spfuncomp_refs_tl 3879
\l__spintent_spfuncomp_tipo_str 3879
\l__spintent_spinterval_delim_l_tl ... 3222
\l__spintent_spinterval_delim_r_tl ... 3222
__spintent_spinterval_exec:nn 3580
__spintent_spinterval_inner_sep:n ... 3476
\l__spintent_spinterval_intent_arg_tl . 3218
\l__spintent_spinterval_intent_group_-
tl 3218
\l__spintent_spinterval_luaset_a_braced_-
str 3201
\l__spintent_spinterval_luaset_a_-
composite_str 3201
\l__spintent_spinterval_luaset_a_inf_-
str 3201
\l__spintent_spinterval_luaset_a_integer_-
str 3201
\l__spintent_spinterval_luaset_a_short_-
str 3201
\l__spintent_spinterval_luaset_a_sign_-
str 3201
\l__spintent_spinterval_luaset_a_tall_-
str 3201
\l__spintent_spinterval_luaset_a_type_-
str 3201
\l__spintent_spinterval_luaset_a_value_-
str 3201
\l__spintent_spinterval_luaset_b_braced_-
str 3201
\l__spintent_spinterval_luaset_b_-
composite_str 3201
\l__spintent_spinterval_luaset_b_inf_-
str 3201
\l__spintent_spinterval_luaset_b_integer_-
str 3201
\l__spintent_spinterval_luaset_b_short_-
str 3201
\l__spintent_spinterval_luaset_b_sign_-
str 3201
\l__spintent_spinterval_luaset_b_tall_-
str 3201
\l__spintent_spinterval_luaset_b_type_-
str 3201
\l__spintent_spinterval_luaset_b_value_-
str 3201
\l__spintent_spinterval_luaset_error_-
code_str 3194
\l__spintent_spinterval_luaset_error_-
str 3194
\l__spintent_spinterval_luaset_left_sym_-
str 3196
\l__spintent_spinterval_luaset_right_sym_-
str 3196
\l__spintent_spinterval_luaset_sep_str 3196
\l__spintent_spinterval_luaset_tall_-
str 3196
\l__spintent_spinterval_luaset_type_-
str 3196
__spintent_spinterval_mathml: 3554
\l__spintent_spinterval_mode_str 3213
__spintent_spinterval_plain_number:n . 3482
\l__spintent_spinterval_prefix_tl 3213
__spintent_spinterval_print:n 3516
\l__spintent_spinterval_read_arg_tl ... 3213
\l__spintent_spinterval_reason_str ... 3217
__spintent_spinterval_set_delims: ... 3332

©2026 by Pablo González L

__spintent_spinterval_set_intents: ... 3426
__spintent_spinterval_set_reason: ... 3290
\l__spintent_spinterval_size_l_tl 3222
\l__spintent_spinterval_size_r_tl 3222
\l__spintent_spinterval_size_str 3213
\l__spintent_spinterval_sym_l_tl 3222
\l__spintent_spinterval_sym_r_tl 3222
\l__spintent_spinterval_tmp_tl 3228
__spintent_spinterval_value:n 3491
\l__spintent_spinterval_word_g_tl 3218
\l__spintent_spinterval_word_w_tl 3218
\l__spintent_spmangle_mode_str 5421
\l__spintent_spmangle_prefix_tl 5421
\l__spintent_spmarc_mode_str 5721
\l__spintent_spmarc_prefix_tl 5721
\l__spintent_spmax_read_arg_tl 7649
\l__spintent_spmax_sub_index_str 7649
__spintent_spmc_exec:nnnn 2218
__spintent_spmc_mathml:nn 2240
__spintent_spmc_result_mathml:n 2286
\l__spintent_spmcd_luaset_mcd_value_tl 2108
\l__spintent_spmcd_prefix_tl 2108
\l__spintent_spmcd_print_result_bool .. 2108
\l__spintent_spmcd_print_upper_bool ... 2108
\l__spintent_spmcd_read_terse_bool ... 2108
\l__spintent_spmcm_luaset_mcm_value_tl 2108
\l__spintent_spmcm_prefix_tl 2108
\l__spintent_spmcm_print_result_bool .. 2108
\l__spintent_spmcm_print_upper_bool ... 2108
\l__spintent_spmcm_read_terse_bool ... 2108
\l__spintent_spmedia_read_arg_tl 7649
\l__spintent_spmedia_sub_index_str ... 7649
\l__spintent_spmediana_read_arg_tl ... 7649
\l__spintent_spmediana_sub_index_str .. 7649
\l__spintent_spmin_read_arg_tl 7649
\l__spintent_spmin_sub_index_str 7649
\l__spintent_spmoda_read_arg_tl 7649
\l__spintent_spmoda_sub_index_str 7649
__spintent_spmoney_build_tree: 1080
__spintent_spmoney_check_sub_units: .. 1031
__spintent_spmoney_intent_int: 990
\l__spintent_spmoney_intent_money_str . 852
\l__spintent_spmoney_intent_sign_str .. 852
\l__spintent_spmoney_intent_sub_unit_-
str 852
\l__spintent_spmoney_luaset_conde_str . 842
\l__spintent_spmoney_luaset_currency_arg_-
str 847
\l__spintent_spmoney_luaset_extracted_-
curr_str 850
\l__spintent_spmoney_luaset_extracted_-
num_tl 850
\l__spintent_spmoney_luaset_plur_str .. 842
\l__spintent_spmoney_luaset_position_-
str 840
\l__spintent_spmoney_luaset_print_iso_-
str 847
\l__spintent_spmoney_luaset_sing_str .. 842
\l__spintent_spmoney_luaset_status_str 847
\l__spintent_spmoney_luaset_subplur_str 842
\l__spintent_spmoney_luaset_subsing_str 842
\l__spintent_spmoney_luaset_symbol_str 840
__spintent_spmoney_mathml: 961
__spintent_spmoney_normalize_key:n ... 889
__spintent_spmoney_parse_and_route:n . 1121
__spintent_spmoney_print_dec_sub_-
unit: 1080
__spintent_spmoney_print_sep_con: ... 1080
__spintent_spmoney_print_symbol: 923
__spintent_spmoney_render_layout: ... 940
__spintent_spmoney_render_number_node: 952
__spintent_spmoney_render_only_integer_-
node: 952
\l__spintent_spmoney_res_main_voice_str 852
__spintent_spmoney_set_and_print_sub_-
units: 1031
__spintent_spmoney_set_currency:n ... 889
\l__spintent_spmoney_sub_units_bool ... 852
\l__spintent_spmoney_subnum_int 852
__spintent_spmoney_validate_money: ... 904
__spintent_spmQ_check_part:nn 2935
\l__spintent_spmQ_error_bool 2746
__spintent_spmQ_exec:nnnn 2963
\l__spintent_spmQ_intent_read_sign_str 2742
\l__spintent_spmQ_join_word_str 2742
__spintent_spmQ_mathml:nnn 2973
__spintent_spmQ_print:nnn 2927
__spintent_spmQ_print_scale_int: 2872
__spintent_spmQ_print_scale_int:n ... 2881
__spintent_spmQ_print_spnZ_dfrac:nn .. 2909
__spintent_spmQ_set_join_word: 2863
__spintent_spmQ_set_wrap_parens_size: 2808
__spintent_spmQ_start_wrap_parens: ... 2843
__spintent_spmQ_stop_wrap_parens: ... 2843
\l__spintent_spmQ_wrap_parens_l_tl ... 2744
\l__spintent_spmQ_wrap_parens_r_tl ... 2744
\l__spintent_spmside_mode_str 5205
\l__spintent_spmside_prefix_tl 5205
\l__spintent_spmside_read_arg_tl 5205
\l__spintent_spmside_read_sty_str 5205
__spintent_spmsym_mathml: 1922
\l__spintent_spmsym_prefix_tl 1869
\l__spintent_spmsym_read_sym_tl 1869
\l__spintent_spmsym_suffix_tl 1869
\l__spintent_spmsym_symbol_tl 1869
__spintent_spmul_exec:nnn 2079
__spintent_spmul_mathml:nn 2063
__spintent_spmul_render_op:nn 2054
\l__spintent_spnD_luaset_is_truncated_-
str 2330
\l__spintent_spnD_luaset_result_tl ... 2330
\l__spintent_spnD_prefix_tl 2330
\l__spintent_spnD_print_result_num_tl . 2330
\l__spintent_spnM_luaset_is_truncated_-
str 2330
\l__spintent_spnM_luaset_result_tl ... 2330
\l__spintent_spnM_prefix_tl 2330
\l__spintent_spnM_print_result_num_tl . 2330
__spintent_spnM_spnD_exec:nnnnn 2437
__spintent_spnM_spnD_intent:nn 2379
\l__spintent_spnM_spnD_intent_tl 2336
\l__spintent_spnM_spnD_luaset_error_-
str 2328
__spintent_spnM_spnD_mathml:nnnn 2448

©2026 by Pablo González L

__spintent_spnM_spnD_result:nn 2399
__spintent_spnM_spnD_silent:nn 2391
\l__spintent_spnmd_luaset_is_number_-
str 2328
\l__spintent_spnum_intent_read_dec_sep_-
str 289
\l__spintent_spnum_intent_read_sign_str 289
__spintent_spnum_mathml: 471
\l__spintent_spnum_notation_sym_tl ... 289
__spintent_spnum_print_integer: 471
__spintent_spnum_print_num_start: ... 471
__spintent_spnum_print_part_dec: 450
__spintent_spnum_print_part_period: .. 450
__spintent_spnum_print_period_bar: ... 378
\l__spintent_spnum_read_period_sep_str 289
\l__spintent_spnum_read_terse_bool ... 289
__spintent_spnum_render_dec: 361
__spintent_spnum_render_decimal_sep: . 415
__spintent_spnum_render_int: 361
__spintent_spnum_render_only_period: . 378
__spintent_spnum_render_part:n 361
__spintent_spnum_render_period_sep: .. 378
__spintent_spnz_exec:nn 2724
__spintent_spnz_mathml: 2708
\l__spintent_spOmega_read_arg_tl 8158
\l__spintent_spOmega_read_sub_bool ... 8158
\l__spintent_spOmega_sub_index_str ... 8158
\l__spintent_spPcirc_mode_str 7052
\l__spintent_spPcirc_prefix_tl 7052
\l__spintent_spPcirc_print_sym_str ... 7052
\l__spintent_spPcirc_read_arg_tl 7052
\l__spintent_spPcirc_read_sty_str 7052
\l__spintent_spPcirc_sub_index_str ... 7052
\l__spintent_spPerc_read_arg_tl 7870
\l__spintent_spPfig_mode_str 6032
\l__spintent_spPfig_prefix_tl 6032
\l__spintent_spPfig_print_sym_str 6032
\l__spintent_spPfig_read_arg_tl 6032
__spintent_spPoint_exec:nnn 4495
\l__spintent_spPoint_intent_arg_tl ... 4455
__spintent_spPoint_mathml:n 4540
\l__spintent_spPoint_mode_str 4453
\l__spintent_spPoint_prefix_tl 4453
__spintent_spPoly_build_concept: 5954
__spintent_spPoly_build_refs: 5939
__spintent_spPoly_exec:nn 5990
\l__spintent_spPoly_intent_arg_tl 5886
__spintent_spPoly_mathml: 6001
\l__spintent_spPoly_mode_str 5884
__spintent_spPoly_points_loop: 5978
\l__spintent_spPoly_prefix_tl 5884
\l__spintent_spPoly_pt_idx_int 5886
\l__spintent_spPoly_pt_seq 5886
\l__spintent_spPoly_refs_tl 5886
__spintent_spPoly_sym: 5931
__spintent_spProb_cap: 8288
__spintent_spProb_content: 8320
__spintent_spProb_cup: 8288
__spintent_spProb_event:N 8288
__spintent_spProb_exec:nn 8403
\l__spintent_spProb_luaset_a_str 8258
\l__spintent_spProb_luaset_b_str 8258
\l__spintent_spProb_luaset_error_str .. 8258
\l__spintent_spProb_luaset_op_str 8258
__spintent_spProb_mathml: 8330
\l__spintent_spProb_mid: 8288
\l__spintent_spProb_nested_bool 8263
__spintent_spProb_nested_ops: 8288
\l__spintent_spProb_read_arg_tl 8262
__spintent_spProm_exec:nn 7609
\l__spintent_spProm_intent_tl 7592
__spintent_spProm_mathml:n 7614
\l__spintent_spProm_read_arg_tl 7592
__spintent_spPval_exec:nnn 8218
__spintent_spPval_mathml:nn 8223
\l__spintent_sprad_mode_str 6498
\l__spintent_sprad_read_arg_tl 6498
\l__spintent_sprad_sub_index_str 6498
\l__spintent_sprango_read_arg_tl 7649
\l__spintent_sprango_sub_index_str ... 7649
__spintent_sprayo_exec:nn 5127
\l__spintent_sprayo_intent_arg_tl 5075
__spintent_sprayo_mathml: 5141
\l__spintent_sprayo_mode_str 5069
\l__spintent_sprayo_prefix_tl 5069
\l__spintent_sprayo_pt_idx_int 5072
\l__spintent_sprayo_pt_seq 5072
\l__spintent_sprayo_read_sty_str 5069
\l__spintent_sprayo_refs_tl 5072
__spintent_sprazon_cmd_str 2535
__spintent_sprazon_exec:nnn 2554
__spintent_sprazon_mathml:nn 2559
__spintent_spread_exec:nn 1731
\l__spintent_spread_luaset_reading_str 1681
\l__spintent_spread_luaset_status_str . 1681
__spintent_spread_mathml:nn 1757
\l__spintent_spread_read_arg_tl 1681
\l__spintent_spread_read_tmp_tl 1681
__spintent_sprecta_exec:nn 4854
\l__spintent_sprecta_intent_arg_tl ... 4814
\l__spintent_sprecta_kv_tl 4814
__spintent_sprecta_mathml:n 4864
\l__spintent_sprecta_mode_str 4809
\l__spintent_sprecta_prefix_tl 4809
\l__spintent_sprecta_pt_idx_int 4811
\l__spintent_sprecta_pt_seq 4811
\l__spintent_sprecta_refs_tl 4811
\l__spintent_spriq_read_arg_tl 7649
\l__spintent_spriq_sub_index_str 7649
__spintent_spshort_exec:n 1200
__spintent_spshort_execute_layout: ... 1235
\l__spintent_spshort_luaset_base_str .. 1148
\l__spintent_spshort_luaset_expand_str 1148
\l__spintent_spshort_luaset_layout_str 1148
\l__spintent_spshort_luaset_output_str 1148
\l__spintent_spshort_luaset_status_str 1148
\l__spintent_spshort_luaset_suffix_str 1148
__spintent_spshort_print:nn 1188
__spintent_spshort_render_bypass:n ... 1211
__spintent_spshort_render_error:n ... 1211
__spintent_spshort_render_fallback: .. 1211
__spintent_spshort_render_success: ... 1211
__spintent_spshort_tag_span:nn 1175

©2026 by Pablo González L

__spintent_spside_exec:nnn 5284
\l__spintent_spside_intent_arg_tl 5216
\l__spintent_spside_intent_tl 5212
__spintent_spside_mathml:n 5319
\l__spintent_spside_mode_str 5205
__spintent_spside_points_loop:n 5387
\l__spintent_spside_prefix_tl 5205
\l__spintent_spside_pt_idx_int 5213
\l__spintent_spside_pt_seq 5213
\l__spintent_spside_read_arg_tl 5205
\l__spintent_spside_read_sty_str 5205
\l__spintent_spside_refs_tl 5213
__spintent_spsiglo_exec:n 1375
\l__spintent_spsiglo_luaset_arabic_str 1292
\l__spintent_spsiglo_luaset_error_str . 1292
\l__spintent_spsiglo_luaset_roman_min_-
str 1292
__spintent_spsiglo_mathml: 1375
\l__spintent_spsiglo_print_era_str . . . 1292
__spintent_spsiglo_render_simple:nn . . 1326
__spintent_spsiglo_render_with_-
era:nnn 1326
__spintent_spsiglo_tag_span:nn 1362
__spintent_spsiglo_text: 1375
__spintent_sptime_build_seconds:nn . . . 1449
\l__spintent_sptime_luaset_base_str . . . 1436
\l__spintent_sptime_luaset_error_str . . 1436
\l__spintent_sptime_luaset_has_seconds_-
str 1436
\l__spintent_sptime_luaset_is_fraction_-
str 1436
\l__spintent_sptime_luaset_seconds_str 1436
__spintent_sptime_mathml: 1467
__spintent_spunit_exec:n 687
\l__spintent_spunit_exponent_tl 560
__spintent_spunit_format_canonical: . . 619
__spintent_spunit_load_term:nn 627
\l__spintent_spunit_luaset_above_found_-
str 552
\l__spintent_spunit_luaset_above_n_str 552
\l__spintent_spunit_luaset_above_sexa_-
str 552
\l__spintent_spunit_luaset_below_found_-
str 552
\l__spintent_spunit_luaset_below_n_str 552
\l__spintent_spunit_luaset_canonical_-
str 557
\l__spintent_spunit_luaset_read_str . . . 557
\l__spintent_spunit_luaset_status_str . 557
__spintent_spunit_mathml: 730
__spintent_spunit_print_product:n . . . 670
__spintent_spunit_print_term:nn 636
\l__spintent_spunit_read_slash_str . . . 560
\l__spintent_spusep_arg_tl 1554
__spintent_spusep_exec:n 1631
\l__spintent_spusep_left_tl 1554
__spintent_spusep_mathml:nn 1620
\l__spintent_spusep_right_tl 1554
\l__spintent_spvarz_read_arg_tl 7649
\l__spintent_spvarz_sub_index_str 7649
__spintent_spvec_exec:nnn 4608
\l__spintent_spvec_intent_arg_tl 4565
\l__spintent_spvec_kv_tl 4565
__spintent_spvec_mathml:n 4641
\l__spintent_spvec_mode_str 4560
\l__spintent_spvec_prefix_tl 4560
\l__spintent_spvec_pt_idx_int 4562
\l__spintent_spvec_pt_seq 4562
\l__spintent_spvec_refs_tl 4562
__spintent_spVol_attach_subscript: . . . 7518
__spintent_spVol_mathml: 7431
\l__spintent_spVol_mode_str 7386
\l__spintent_spVol_native_bool 7389
\l__spintent_spVol_read_arg_full_tl . . . 7392
__spintent_spVol_read_arg_print: 7550
\l__spintent_spVol_read_arg_tl 7386
__spintent_spVol_silent: 7438
\l__spintent_spVol_sub_index_str 7386
\l__spintent_spVol_suffix_tl 7389
__spintent_spVol_symbol_print: 7480
__spintent_spVol_verbal: 7459
\l__spintent_spVol_word_tl 7389
__spintent_stat_exec:nnn 7773
__spintent_stat_intent:nn 7729
__spintent_stat_intent:nnn 7729
__spintent_stat_intent:nnnn 7729
\l__spintent_stat_intent_tl 7658
__spintent_stat_letter:nnnn 7773
__spintent_stat_mathml:nnnn 7764
__spintent_stat_pos_check:nnn 7901
\l__spintent_stat_pos_error_bool 7874
__spintent_stat_pos_exec:nnnnn 7955
__spintent_stat_pos_mathml:nnnn 7925
__spintent_stat_pos_typeset:nn 7920
\l__spintent_stat_pos_value_int 7874
__spintent_stat_simple:nnn 7773
__spintent_stat_typeset:nnn 7715
__spintent_unknown_keys_cmd:n 241
__spintent_unknown_keys_cmd:nn 241
spintent.lua 76
\spinterval *39*, *130*, 3597
spinterval-invalid-arg 3229
\spmangle *28*, *174*, 5699
spmangle-invalid-arg 5433
\spmarc *29*, *180*, 5862
spmarc-invalid-points 5731
\spmax *43*, 7844
\spmcd *19*, *105*, 2308
\spmcm *20*, *105*, 2308
\spmedia *42*, 7795
\spmediana *42*, 7795
\spmin *43*, 7844
\spmoda *42*, 7795
\spmoney *10*, *78*, 1137
spmoney-invalid-currency 857
spmoney-no-subunits 857
\spmQ *21*, *119*, 2986
spmQ-not-integer 2748
spmQ-signed-part 2748
spmQ-zero-denominator 2748
\spmside *26*, *170*, 5399
spmside 6766

©2026 by Pablo González L

spmside-invalid-points 5217
\spmsym 17, 100, 1941
\spmul 18, 103, 2092
spmul-empty-arg 2027
\spnD 18, 109, 2515
\spnewread 15, 1773
spnewread-duplicate 1685
spnewread-empty-args 1685
\spnewunit 10, 77, 808
spnewunit-duplicate 786
\spnM 19, 109, 2515
\spnum 7, 66, 534
spnum-has-units 294
spnum-no-digits 294
\spnZ 21, 117, 2732
spnZ-not-integer 2685
\spOmega 45, 230, 8184
\spPcirc 37, 206, 7360
spPcirc-invalid-arg 7067
\spPerc 43, 7964
\spPfig 32, 187, 6469
spPfig-invalid-arg 6050
spPfig-invalid-points 6050
spPfig-read-arg-key 6050
spPfig-sym-cardinality-conflict 6050
spPfig-sym-letter-conflict 6050
\spPoint 23, 155, 4549
spPoint-invalid-point 4456
\spPoly 30, 184, 6021
spPoly-invalid-points 5890
\spProb 46, 232, 8411
spProb-invalid-arg 8264
\spProm 42, 217, 7638
\spProp 22, 115, 2609
\spPval 45, 231, 8248
\sprad 33, 196, 6720
\sprango 43, 7844
\sprayo 25, 167, 5194
sprayo-invalid-points 5076
\sprazon 22, 114, 2599
\spread 16, 96, 1763
spread-empty-sym 1685
spread-unknown-sym 1685
\sprec 40, 147, 4144
\sprecta 25, 162, 5058
sprecta-invalid-points 4816
\spriq 43, 7844
\spshort 12, 85, 1279
spshort-unknown-abbreviation 1154
\spside 26, 170, 5399
spside 6766
spside-invalid-points 5217
\spsiglo 15, 88, 1428
spsiglo-invalid-arg 1296
\sptime 15, 91, 1499
sptime-invalid-arg 1441
\spunit 8, 72, 774
spunit-inline-numeric-denominator 562
spunit-invalid-unit 562
spunit-no-units 562
\spunitalias 10, 77, 822
spunitalias-duplicate 786
spunitalias-invalid-expr 786
\spusep 16, 94, 1670
spusep-empty-arg 1557
spusep-invalid-arg 1557
\spvarz 42, 7810
\spvec 24, 158, 4798
spvec-invalid-arg 4567
\spVol 38, 213, 7580
stat-pos-invalid-index 7876
str commands:
\str_lowercase:n 70
sub-index 6504, 6766, 7076, 7393, 7659, 8019, 8162
sub-units 871
suffix 1798, 1873

T

Tool provided by spanish-mathml-intent:
spmmli-tool 60, 62

U

unknown . . 308, 585, 871, 1161, 1303, 1572, 1714, 1798, 1873, 1959, 2034, 2114, 2352, 2537, 2692, 2770, 3032, 3238, 3630, 3891, 4077, 4188, 4464, 4574, 4823, 5083, 5227, 5444, 5741, 5897, 6090, 6504, 6766, 7076, 7393, 7594, 7659, 7697, 7883, 8019, 8112, 8162, 8204, 8273, 8434, 8608, 8977
unknown-choice 215
unknown-command-setup 165
upper-case 2114, 6504
use-dfrac 2537, 2770, 8204
use-spnZ 2770

W

wrap-parens 1959, 2034, 2692, 2770, 8434

©2026 by Pablo González L