

spanish-mathml-intent

MATHML INTENT EN ESPAÑOL

V1.4 2026-11-02^{*}

©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	3.3 El comando <code>\spmoney</code>	11
1.1 Carga del paquete	2	3.3.1 Llaves para <code>\spmoney</code>	11
1.2 El paquete <code>babel</code>	3	3.3.2 Monedas y alias para <code>\spmoney</code>	12
1.3 Fuentes tipográficas	3	3.4 El comando <code>\spshort</code>	13
1.3.1 La llave <code>mathcal</code>	3	3.4.1 Llaves para <code>\spshort</code>	13
1.4 El paquete <code>hyperref</code>	3	3.4.2 Argumentos para <code>\spshort</code>	13
1.5 Interfaz de usuario	4	4 Utilidades generales	15
1.5.1 Lo que ofrece el paquete	4	5 Utilidades para símbolos	16
1.5.2 Lo que ofrece cada eje	4	6 Utilidades para números	18
1.5.3 Reglas para argumentos	5	7 Utilidades para geometría	24
1.5.4 Consulta rápida	5	8 Utilidades para álgebra	39
1.6 Un primer documento	5	9 Utilidades para estadística	43
1.7 Cómo comprobar el resultado	6	10 Modo <code>mathml-AF</code> y <code>spmml-i-tool</code>	49
1.7.1 Escuchar la lectura	6	10.1 Cuál usar y cuándo	49
1.7.2 Validar la estructura	6	10.2 De <code>mathml-SE</code> a <code>mathml-AF</code>	49
1.7.3 Instalar <code>rnv-wrapp</code> y <code>veraPDF</code>	7	10.3 La herramienta <code>spmml-i-tool</code>	49
1.8 Si algo no suena como esperaba	7	11 Ejemplos de PDF accesible y etiquetado	50
2 El comando <code>\setspintent</code>	7	12 Referencias	52
3 The big four	8	13 Change history	54
3.1 El comando <code>\spnum</code>	8	14 Índice de la Documentación	55
3.1.1 Llaves para <code>\spnum</code>	8	15 Implementation	59
3.2 El comando <code>\spunit</code>	9	16 Índice de la Implementación	250
3.2.1 Llaves para <code>\spunit</code>	9		
3.2.2 Argumentos para <code>\spunit</code>	9		
3.2.3 El comando <code>\spnewunit</code>	10		
3.2.4 El comando <code>\spunitalias</code>	11		

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 v1.4 revisada por última vez el 2026-11-02.

[†]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 $\text{\LaTeX}$, 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, $\text{\Lua\LaTeX}$ ofrece una plataforma especialmente adecuada para incorporar esta tecnología.

Debo agradecer a todo el equipo de $\text{\LaTeX}$ 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 $\text{\Lua\LaTeX}$, por lo que es necesario contar con una distribución moderna de $\text{\TeX}$ como $\text{\TeX}$ Live 2026. Ha sido probado con la clase estándar `article` proporcionada por $\text{\LaTeX}$ 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 $\text{\LaTeX}$ puede apoyarse en `Learn \LaTeX` [17], un sitio respaldado por el equipo de $\text{\LaTeX}$ con lecciones breves y ejemplos que se pueden probar en línea; como referencia general sirve el manual no oficial de $\text{\LaTeX} 2_{\epsilon}$ [18].

Se otorga permiso para copiar, distribuir y/o modificar este paquete bajo los términos de $\text{\LaTeX}$ 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 $\text{\LaTeX}$ [19] release 2026-11-01.

Este paquete solo se puede utilizar con $\text{\Lua\LaTeX}$ y *tagged* PDF activo. Está presente en $\text{\TeX}$ 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 $\text{\Lua\LaTeX}$ 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.

¹A *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 siga 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 Lua $\TeX$ debemos distinguir entre las fuentes tipográficas OpenType/TrueType versus las fuentes de 8bit estándar usadas por $\TeX$ 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 Lua $\TeX$ 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 `\spcÍrc`, `\spAcÍrc` y `\spPcÍrc`, 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`.

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

1.6 Un primer documento

El siguiente documento está completo: se puede copiar, guardar con extensión `.tex` y compilar con `Lua®TeX`. 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  $\text{\spnum{3,14}}$  es una aproximación de  $\pi$ . El pasillo del
colegio mide  $\text{\spunit{23 m}}$  de largo y un cuaderno cuesta  $\text{\spmoney{500}}$ .

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

Material preparado por la  $\text{\spshort{prof.a}}$  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
$\text{\spnum{3,14}}$	3,14	«tres coma catorce»
$\text{\spunit{23 m}}$	23 m	«veintitrés metros»
$\text{\spmoney{500}}$	\$500	«quinientos pesos»
$\text{\spshort{prof.a}}$	Prof. ^a	«profesora»
$\text{\spPoly{ABC}}$	$\triangle ABC$	«triángulo A B C»
$\text{\spfun{f(x)}}$	$f(x)$	«f de x»
$\text{\spProb{A \cap B}}$	$P(A \cap B)$	«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 `spmml-i-tool -c archivo.pdf` ejecuta ambas comprobaciones de una vez (§10).

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 $\text{\LaTeX}$ 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`

<code>\spnum</code>	<code>\spnum{<número>}</code>	<code>\spnum{<número,número>}</code>	<code>\spnum{<número.número>}</code>	<code>\spnum{<número;número>}</code>
	<code>\spnum[<key = val>]{<número,número>}</code>		<code>\spnum[<key = val>]{<número,número;número e exponente>}</code>	
	<code>\spnum[<key = val>]{<número.número>}</code>		<code>\spnum[<key = val>]{<número.número;número e exponente>}</code>	

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 `3,14` 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 · 103` 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 `3,4` en la salida PDF y NVDA/MathCAT verbalizará «tres coma cuatro periódico» y `\spnum[read-period-sep=punto]{3;4}` imprimirá `3.4` 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 · 103` 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 × 103` 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».

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

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>}` válidos 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 s²’ 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/s²’ 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, °C	Grados Celsius	°C	«grados celsius»
°F	fahrenheit, degF, °F	Grados Fahrenheit	°F	«grados fahrenheit»
cal	caloria, calorías	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»
Ω	mho	Mho / Siemens	Ω	«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 $\langle nueva\ unidad \rangle$ no debe estar registrada como $\langle unidad \rangle$ o $\langle alias \rangle$ para `\spunit` descritos en §3.2.2. Si la $\langle nueva\ unidad \rangle$ ya está registrada se devuelve un “error” y se detiene la compilación del documento.

La declaración de la $\langle nueva\ unidad \rangle$ es «global» y se almacena en el módulo `spintent.lua` como una $\langle unidad \rangle$ válida para pasar como $\langle argumento \rangle$ a `\spunit`. Por ejemplo: `\spnewunit{px}{píxeles}` creará la $\langle nueva\ unidad \rangle$ ‘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*: $\langle nuevo\ alias \rangle$ que es el nuevo $\langle alias \rangle$ de $\langle argumento \rangle$ para `\spunit` y $\langle expresión\ de\ unidades \rangle$ que puede contener multiplicación o división de $\langle unidades \rangle$ ya registradas que se imprimirán en la salida PDF.

El $\langle nuevo\ alias \rangle$ no debe estar registrado como $\langle unidad \rangle$ o $\langle alias \rangle$ para `\spunit` descritos en §3.2.2 o estar definido como $\langle unidad \rangle$ por medio del comando `\spnewunit`. Si el $\langle nuevo\ alias \rangle$ ya está registrado se devuelve un “error” y se detiene la compilación del documento.

La declaración de un $\langle nuevo\ alias \rangle$ es «global» y se almacena en el módulo `spintent.lua` como un $\langle alias \rangle$ válido para pasar como $\langle argumento \rangle$ a `\spunit`. Por ejemplo: `\spunitalias{caudal}{m^3/s}` creará el $\langle alias \rangle$ ‘caudal’ y cuando se utilice `\spunit{50 caudal}` se imprimirá ‘50 m³/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* $\langle cifra \rangle$ 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 $\langle argumento \rangle$ 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 $\langle meta-key \rangle$ `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 $\{(\textit{argumento})\}$ 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 = {\textit{true} | \textit{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 = {\textit{true} | \textit{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 $\{(\textit{argumento})\}$ de `\spmoney`.

Tabla 4: Monedas y alias soportados

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

3.4 El comando \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. ^{as}	«doctoras»
prof.a	—	Prof. ^a	«profesora»
d.a	d. ^a	D. ^a	«doña»
m.a	—	M. ^a	«maría»
n.o	n. ^o	N. ^o	«número»
n.os	—	N. ^{os}	«números»
c.ia	—	C. ^{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 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 $\{\langle \textit{siglo} \rangle\}$. 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{\langle horas : minutos \rangle}``\sptime{\langle horas : minutos : segundos \rangle}`

El comando `\sptime` trabaja en *modo matemático* y recibe el *argumento obligatorio* $\{\langle \textit{horas} : \textit{minutos} \rangle\}$ o $\{\langle \textit{horas} : \textit{minutos} : \textit{segundos} \rangle\}$.

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{\langle DD/MM/YYYY \rangle}``\spdate{\langle YYYY/MM/DD \rangle}``\spdate{\langle YYYY-MM-DD \rangle}``\spdate{\langle DD-MM-YYYY \rangle}`

El comando `\spdate` trabaja en *modo matemático* y recibe el *argumento obligatorio* $\{\langle \textit{DD/MM/YYYY} \rangle\}$, $\{\langle \textit{DD-MM-YYYY} \rangle\}$ o $\{\langle \textit{YYYY-MM-DD} \rangle\}$.

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 $\{\langle \textit{YYYY/MM/DD} \rangle\}$, internamente se convertirá en $\{\langle \textit{DD/MM/YYYY} \rangle\}$ 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{\langle \textit{símbolo} \rangle}{\langle \textit{verbalización NVDA} \rangle}`

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

El argumento $\{\langle \textit{símbolo} \rangle\}$ 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 ‘ $\cong$ ’ en la salida PDF y NVDA/MathCAT verbalizará «es congruente con».

- El $\{\langle \textit{símbolo} \rangle\}$ 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 $\{\langle \textit{verbalización NVDA} \rangle\}$ 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{\langle \textit{símbolo} \rangle}``\spread[key = val]{\langle \textit{símbolo} \rangle}``\spread[read-arg={\langle \textit{verbalización NVDA} \rangle}]{\langle \textit{comandos LATEX \rangle}`

El comando `\spread` trabaja en *modo matemático* y recibe el *argumento obligatorio* de la forma $\{\langle \textit{símbolo} \rangle\}$ previamente registrado por el comando `\spnewread` o $\{\langle \textit{comandos L^AT_EX \rangle}$ para ser usado junto a la llave `read-arg`.

Si el argumento es $\{\langle \textit{símbolo} \rangle\}$, $\$ \backslash \textit{spread} \{ \backslash \textit{cong} \} \$$ imprimirá ‘ $\cong$ ’ en la salida PDF y el argumento $\{\langle \textit{verbalización NVDA} \rangle\}$ establecido al registrar $\{\langle \textit{símbolo} \rangle\}$ con $\backslash \textit{spnewread}$ se verbalizará en NVDA/MathCAT.

Si el argumento es $\{\langle \textit{comandos L\TeX} \rangle\}$ y no se establece la llave `read-arg` se devuelve un “error” y se detiene la compilación del documento.

Llaves para `\spread`

`read-arg` = $\{\langle \textit{verbalización NVDA} \rangle\}$ por defecto: *no usado*

Establece la *verbalización* del argumento $\{\langle \textit{comandos L\TeX} \rangle\}$. El valor de esta llave *siempre* debe pasar-se entre llaves ‘{ }’ y sobrescribe la verbalización del $\{\langle \textit{símbolo} \rangle\}$ si está registrado previamente. Por ejemplo: $\$ \backslash \textit{spread} [\textit{read-arg} = \{ \textit{es-paralela-a} \}] \{ \backslash \textit{parallel} \} \$$ imprimirá ‘ $\parallel$ ’ en la salida PDF y NV-DA/MathCAT verbalizará «*es paralela a*».

`silent` = $\{\langle \textit{true} \mid \textit{false} \rangle\}$ por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT del argumento $\{\langle \textit{símbolo} \rangle\}$ o $\{\langle \textit{comandos L\TeX} \rangle\}$ 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 $\{\langle \textit{símbolo} \rangle\}$ esté registrado o usar la llave `read-arg` con $\{\langle \textit{comandos L\TeX} \rangle\}$. Dependiendo de lo que contenga el $\{\langle \textit{argumento} \rangle\}$, puede que `silent` no sea suficiente, en ese caso utilice `read-arg={_}`.

El comando `\spusep`

`\spusep` $\backslash \textit{spusep} \{ \backslash \{ [[([]] \} \} \}$ $\backslash \textit{spusep} [\textit{key} = \textit{val}] \{ \backslash \{ [[([]] \} \} \}$

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

Si no se proporciona el *argumento opcional* $[\textit{key} = \textit{val}]$, $\$ \backslash \textit{spusep} \{ \}$ 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 Lua $\TeX$ 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, $[\textit{key} = \textit{val}]$, o delimitado por paréntesis, $(\textit{número}, \textit{número}, \dots)$, 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, $\TeX$ lo interpretará como el comienzo del argumento opcional. Por ejemplo, $\$ \backslash \textit{spmedia} \backslash \textit{spusep} \{ [] \} \$$ imprime el símbolo de `\spmedia` seguido de ‘[’, mientras que `\spmedia[` haría que $\TeX$ lea lo que sigue como llaves $[\textit{key} = \textit{val}]$.

Llaves para `\spusep`

`size` = $\{\langle \textit{none} \mid \textit{big} \mid \textit{Big} \mid \textit{bigg} \mid \textit{Bigg} \rangle\}$ por defecto: *none*

Establece el *tamaño del paréntesis* pasado como $\{\langle \textit{argumento} \rangle\}$ 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 $\{\langle \textit{argumento} \rangle\}$ en el tamaño por defecto del modo matemático en curso; si se establece en `big`, se usa `\bigl/\bigr` antes del $\{\langle \textit{argumento} \rangle\}$; si se establece en `Big`, se usa `\Bigl/\Bigr` antes del $\{\langle \textit{argumento} \rangle\}$; si se establece en `bigg`, se usa `\biggl/\biggr` antes del $\{\langle \textit{argumento} \rangle\}$; y si se establece en `Bigg`, se usa `\Biggl/\Biggr` antes del $\{\langle \textit{argumento} \rangle\}$.

`silent` = $\{\langle \textit{true} \mid \textit{false} \rangle\}$ por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT del $\{\langle \textit{argumento} \rangle\}$ 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` $\backslash \textit{spdsym} \quad \backslash \textit{spdsym} [\textit{key} = \textit{val}]$

El comando `\spdsym` trabaja en *modo matemático* y recibe el *argumento opcional* $[\textit{key} = \textit{val}]$ 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* $[\textit{key} = \textit{val}]$, $\$ \backslash \textit{spdsym} \$$ imprimirá ‘:’ en la salida PDF y NV-DA/MathCAT verbalizará «*dividido*».

Llaves para `\spdsym`

`prefix` = $\{\langle \textit{prefijo} \rangle\}$ por defecto: *no usado*

Establece el *prefijo* que será *verbalizado* por NVDA/MathCAT «*antes*» de verbalizar el $\{\langle \textit{valor} \rangle\}$ 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 = { < 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».

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 · 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 ‘D(*n*)’ en la salida PDF y NVDA/MathCAT verbalizará «divisores de ene». Si se usa el argumento (⟨*número*⟩), `\spnD(6)` imprimirá ‘D(6)’ en la salida PDF y NVDA/MathCAT verbalizará «divisores de seis». Si se usa el argumento (⟨*letra*⟩), `\spnD(m)` imprimirá ‘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 ‘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`

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 `\spgcd`

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

El comando `\spgcd` 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*, `\spgcd` imprimirá ‘gcd’ en la salida PDF y NVDA/MathCAT verbalizará «máximo común divisor».

Si se usa el argumento (*número, número, ...*), `\spgcd(2, 3, 6)` imprimirá ‘gcd(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, ...*), `\spgcd(m, n)` imprimirá ‘gcd(*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.

Llaves para `\spgcd`

`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*: `\spgcd[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 `\spmc`

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

El comando `\spmc` 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*, `\spmc` imprimirá ‘mcm’ en la salida PDF y NVDA/MathCAT verbalizará «*mínimo común múltiplo*».

Si se usa el argumento (*número, número, ...*), `\spmc(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, ...*), `\spmc(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 `\spmc`

`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*

Establece si se *imprime* el resultado de calcular el «*mínimo común múltiplo*» del argumento (*⟨número, número, ...⟩*). Si se establece en *true*: `$\spmc{result=true}(2, 3, 6)$` imprimirá ‘mcm(2, 3, 6) = 6’ en la salida PDF y NVDA/MathCAT verbalizará «*mínimo común múltiplo entre dos, tres y seis es igual a seis*»; si se establece en *false* no se imprimirá 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 ‘2⁵/₇’ 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 `$\frac{5}{7}$` 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 ‘1⁵/₇’ 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 ‘2⁵/₇’ 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 ‘2⁵/₇’ 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

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

`use-dfrac = { \langle true | false \rangle }` por defecto: *false*

Establece si la fracción del *número mixto* se imprime usando `\dfrac` en lugar de `\frac`. Si se establece en *true*, $\$ \backslash \text{spmQ}[\text{use-dfrac}=\text{true}]{2}{5}{7} \$$ 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 = { \langle true | false \rangle }` 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 = { \langle true | false \rangle }` 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 = { \langle true | false \rangle }` 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` $\backslash \text{sprazon}[\langle \text{antecedente} \rangle]{\langle \text{consecuente} \rangle}$ $\backslash \text{sprazon}[\langle \text{key} = \text{val} \rangle]{\langle \text{antecedente} \rangle}{\langle \text{consecuente} \rangle}$

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

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

- El $\{\langle \text{antecedente} \rangle\}$ y el $\{\langle \text{consecuente} \rangle\}$ se tipografían directamente, sin procesamiento semántico adicional.

Llaves para `\sprazon`

`silent = { \langle true | false \rangle }` 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 = { \langle true | false \rangle }` por defecto: *false*

Establece si la fracción de la *razón* se imprime usando `\dfrac` en lugar de `\frac`. Si se establece en *true*, $\$ \backslash \text{sprazon}[\text{use-dfrac}=\text{true}]\{3\}{4} \$$ 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 = { \langle true | false \rangle }` por defecto: *false*

Establece si la *razón* se imprime en línea con dos puntos ‘:’ en lugar de una fracción. Si se establece en *true*, $\$ \backslash \text{sprazon}[\text{inline}=\text{true}]\{3\}{4} \$$ 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` $\backslash \text{spProp}\{\langle a \rangle\}\{\langle b \rangle\}\{\langle c \rangle\}\{\langle d \rangle\}$ $\backslash \text{spProp}[\langle \text{key} = \text{val} \rangle]\{\langle a \rangle\}\{\langle b \rangle\}\{\langle c \rangle\}\{\langle d \rangle\}$

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

Si no se proporciona el *argumento opcional* $[\langle \text{key} = \text{val} \rangle]$, $\$ \backslash \text{spProp}\{3\}\{4\}\{6\}\{8\} \$$ 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 = { \langle true | false \rangle }` 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.

`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`

<code>\spcoord{\abscisa, ordenada}</code>	<code>\spcoord[key = val]{\abscisa, ordenada}</code>
<code>\spcoord{\abscisa; ordenada}</code>	<code>\spcoord[key = val]{\abscisa; ordenada}</code>

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 ‘(x₁, y₁)’ 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`

<code>\spPoint{\punto}</code>	<code>\spPoint[key = val]{\punto}</code>
<code>\spPoint{\punto}(\abscisa, ordenada)</code>	<code>\spPoint[key = val]{\punto}(\abscisa, ordenada)</code>
<code>\spPoint{\punto}(\abscisa; ordenada)</code>	<code>\spPoint[key = val]{\punto}(\abscisa; ordenada)</code>

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 ‘A₁’ 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 ‘A₁(-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`

<code>\spvec {⟨letra⟩}</code>	<code>\spvec {⟨punto, punto⟩}</code>
<code>\spvec {⟨letra⟩} (⟨abscisa, ordenada⟩)</code>	<code>\spvec {⟨punto, punto⟩} (⟨abscisa, ordenada⟩)</code>
<code>\spvec [key = val] {⟨letra⟩} (⟨abscisa, ordenada⟩)</code>	<code>\spvec [key = val] {⟨punto, punto⟩} (⟨abscisa, ordenada⟩)</code>

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 `\spscoord`.

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`

<code>\sprecta {⟨punto, punto⟩}</code>	<code>\sprecta [key = val] {⟨punto, punto⟩}</code>
<code>\sprecta {⟨letra⟩}</code>	<code>\sprecta [key = val] {⟨letra⟩}</code>

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 `\overleftarrow` en la impresión; si el *argumento* es de la forma {⟨letra⟩}, mayúscula o minúscula, se imprimirá solo la {⟨letra⟩} sin aplicar `\overleftarrow`.

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

Llaves para $\backslash \text{sprecta}$

`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»: $\$ \backslash \text{sprecta}\{AB\} \$$ 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 $\backslash \text{sprayo}$

$\backslash \text{sprayo}$ $\backslash \text{sprayo}\{\langle \text{punto, punto} \rangle\}$ $\backslash \text{sprayo}[\langle \text{key} = \text{val} \rangle]\{\langle \text{punto, punto} \rangle\}$

El comando $\backslash \text{sprayo}$ 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 $\backslash \text{overrightarrow}$ en la impresión.

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

Llaves para $\backslash \text{sprayo}$

`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»: $\$ \backslash \text{sprayo}\{AB\} \$$ 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 $\$ \backslash \text{sprayo}[\text{read-sty=rayo}]\{AB\} \$$, NVDA/MathCAT verbalizará «rayo A B»; si se utiliza $\$ \backslash \text{sprayo}[\text{read-sty=semirrecta}]\{AB\} \$$, 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».

El comando `\spside`

$$\backslash\mathrm{spside}\{\langle\text{punto}, \text{punto}\rangle\} \quad \backslash\mathrm{spside}[\langle\text{key} = \text{val}\rangle]\{\langle\text{punto}, \text{punto}\rangle\}$$

El comando `\spside` trabaja en *modo matemático* y recibe el *argumento obligatorio* $\{\langle\text{punto}, \text{punto}\rangle\}$ con prima(s) ‘ $\prime$ ’ o subíndice(s) ‘ $_{\dots}$ ’ opcionales. Representa un *lado*, *segmento* o *trazo* determinado por dos puntos, aplicando `\overline` en la impresión.

Si no se proporciona el *argumento opcional* $[\langle\text{key} = \text{val}\rangle]$, $\backslash\mathrm{spside}\{AB\}$ imprimirá ‘ $\overline{AB}$ ’ en la salida PDF y NVDA/MathCAT verbalizará «*lado A B*».

Llaves para `\spside`

`read-cmd` = $\{\langle\text{school} \mid \text{mathcat}\rangle\}$ por defecto: *school*

Establece la *forma* en la que NVDA/MathCAT verbalizará el argumento $\{\langle\text{punto}, \text{punto}\rangle\}$. Si se establece en *school*, NVDA/MathCAT verbaliza las letras del argumento sin el prefijo «mayúscula»: $\backslash\mathrm{spside}\{AB\}$ 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` = $\{\langle\text{lado} \mid \text{segmento} \mid \text{trazo}\rangle\}$ por defecto: *lado*

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

`prefix` = $\{\langle\text{prefijo}\rangle\}$ 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` = $\{\langle\text{true} \mid \text{false}\rangle\}$ 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 $\{\langle\text{argumento}\rangle\}$; si se establece en *false*, NVDA/MathCAT solo verbalizará el $\{\langle\text{argumento}\rangle\}$ acorde al valor establecido por la llave `read-cmd`.

`read-arg` = $\{\langle\text{verbalización NVDA}\rangle\}$ 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: $\backslash\mathrm{spside}[\text{read-arg}=\{\text{hipotenusa}\}]\{AB\}$ imprimirá ‘ $\overline{AB}$ ’ en la salida PDF y NVDA/MathCAT verbalizará «*hipotenusa A B*».

`silent` = $\{\langle\text{true} \mid \text{false}\rangle\}$ por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT del «*concepto*» establecido por la llave `read-sty` y del $\{\langle\text{argumento}\rangle\}$ 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`

$$\backslash\mathrm{spmside}\{\langle\text{punto}, \text{punto}\rangle\} \quad \backslash\mathrm{spmside}[\langle\text{key} = \text{val}\rangle]\{\langle\text{punto}, \text{punto}\rangle\}$$

El comando `\spmside` trabaja en *modo matemático* y recibe el *argumento obligatorio* $\{\langle\text{punto}, \text{punto}\rangle\}$ con prima(s) ‘ $\prime$ ’ o subíndice(s) ‘ $_{\dots}$ ’ 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* $[\langle\text{key} = \text{val}\rangle]$, $\backslash\mathrm{spmside}\{AB\}$ imprimirá ‘ AB ’ en la salida PDF y NVDA/MathCAT verbalizará «*medida del lado A B*».

Llaves para `\spmside`

`read-cmd` = $\{\langle\text{school} \mid \text{mathcat}\rangle\}$ por defecto: *school*

Establece la *forma* en la que NVDA/MathCAT verbalizará el argumento $\{\langle\text{punto}, \text{punto}\rangle\}$. Si se establece en *school*, NVDA/MathCAT verbaliza las letras del argumento sin el prefijo «mayúscula»: $\backslash\mathrm{spmside}\{AB\}$ 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` = $\{\langle\text{lado} \mid \text{segmento} \mid \text{trazo}\rangle\}$ 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 $\backslash\mathrm{spmside}[\text{read-sty}=\text{lado}]\{AB\}$, NVDA/MathCAT verbalizará «*medida del lado A B*»; si se utiliza $\backslash\mathrm{spmside}[\text{read-sty}=\text{segmento}]\{AB\}$, NVDA/MathCAT verbalizará «*medida del segmento A B*» y si se utiliza $\backslash\mathrm{spmside}[\text{read-sty}=\text{trazo}]\{AB\}$, NVDA/MathCAT verbalizará «*medida del trazo A B*».

`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 '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 {⟨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 '∠A' en la salida PDF y NVDA/MathCAT verbalizará «ángulo en A»; `$\spangle{\alpha}$` imprimirá '∠α' en la salida PDF y NVDA/MathCAT verbalizará «ángulo alfa»; `$\spangle{1}$` imprimirá '∠1' en la salida PDF y NVDA/MathCAT verbalizará «ángulo uno» y `$\spangle{AOB}$` imprimirá '∠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 '⊥A' en la salida PDF y NVDA/MathCAT verbalizará «ángulo recto en A», `$\spangle[recto=true]{AOB}$` imprimirá '⊥AOB' en la salida PDF y NVDA/MathCAT verbalizará «ángulo recto A O B»; si se establece en *false*, imprimirá usando `\angle` en la salida PDF y NVDA/MathCAT omitirá el término «recto».

`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 ‘ $\angle A$ ’ en la salida PDF y NVDA/MathCAT verbalizará «medida del ángulo en A»; `\spmangle{\alpha}` imprimirá ‘ $\angle \alpha$ ’ en la salida PDF y NVDA/MathCAT verbalizará «medida del ángulo alfa»; `\spmangle{1}` imprimirá ‘ $\angle 1$ ’ en la salida PDF y NVDA/MathCAT verbalizará «medida del ángulo uno» y `\spmangle{AOB}` imprimirá ‘ $\angle AOB$ ’ 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 ‘ $\angle A$ ’ en la salida PDF y NVDA/MathCAT verbalizará «medida del ángulo recto en A»; `\spmangle[recto=true]{AOB}` imprimirá ‘ $\angle AOB$ ’ 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».

El comando `\sparc`

<code>\sparc</code>	<code>\sparc{⟨punto, punto⟩}</code>	<code>\sparc{⟨punto, punto, punto⟩}</code>
	<code>\sparc[⟨key = val⟩]{⟨punto, punto⟩}</code>	<code>\sparc[⟨key = val⟩]{⟨punto, punto, punto⟩}</code>

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 ‘ $\widehat{AB}$ ’ 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`

<code>\spmarc</code>	<code>\spmarc{⟨punto, punto⟩}</code>	<code>\spmarc{⟨punto, punto, punto⟩}</code>
	<code>\spmarc[⟨key = val⟩]{⟨punto, punto⟩}</code>	<code>\spmarc[⟨key = val⟩]{⟨punto, punto, punto⟩}</code>

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 ‘ $m\widehat{AB}$ ’ 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* ‘ $\widehat{AB}$ ’; `\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.

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{⟨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* $((F|P|B|L|T))$, $\$ \backslash \text{spAfig}(F) \$$ imprimirá ' $\mathcal{A}_F$ ' en la salida PDF y NVDA/MathCAT verbalizará «área de la figura»; $\$ \backslash \text{spAfig}(F_{\{1\}}) \$$ 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*, $\$ \backslash \text{spAfig} \$$ imprimirá ' $\mathcal{A}$ ' en la salida PDF y NVDA/MathCAT verbalizará «área».

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

Llaves para $\backslash \text{spAfig}$

`read-cmd` = $\{ \langle school | mathcat \rangle \}$ 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»: $\$ \backslash \text{spAfig}(ABC) \$$ 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` = $\{ \langle prefijo \rangle \}$ 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` = $\{ \langle true | false \rangle \}$ 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` = $\{ \langle verbalización NVDA \rangle \}$ 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 $((punto, punto, punto, \dots))$. 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: $\$ \backslash \text{spAfig}[read-arg=\{trapecio\}](ABCD) \$$ 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 $((1|2|3|\dots|n))$.

`concept*` $\langle \text{valor prohibido} \rangle$ 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` = $\{ \langle true | false \rangle \}$ por defecto: *false*

Establece si se *antepone* la palabra «sub» al verbalizar los argumentos $((1|2|3|\dots|n))$ y $((F|P|B|L|T))$ para NVDA/MathCAT. Si se utiliza $((1|2|3|\dots|n))$ y se establece en *true*, $\$ \backslash \text{spAfig}[read-sub=true](1) \$$ 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*, $\$ \backslash \text{spAfig}[read-sub=true](F_{\{1\}}) \$$ 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` = $\{ \langle auto | triangle | square | none \rangle \}$ por defecto: *auto*

Establece si se *usará* $\backslash triangle$ o $\backslash mdlgwhtsquare$ en la impresión para los argumentos $((punto, punto, punto, \dots))$ y $((1|2|3|\dots|n))$ pasado al comando sin modificar la verbalización de este. Si se utiliza $((punto, punto, punto, \dots))$ se infiere el *símbolo* acorde al número de puntos; si se establece en *auto* se imprime y si se establece en *none* se omite: $\$ \backslash \text{spAfig}[print-sym=auto](ABC) \$$ y $\$ \backslash \text{spAfig}[print-sym=none](ABC) \$$ 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 $((1|2|3|\dots|n))$ los valores *auto* y *none* quedan sin efecto; si se establece en *triangle* o *square* anida el argumento bajo el *símbolo*: $\$ \backslash \text{spAfig}[print-sym=triangle](1) \$$ imprimirá ' $\mathcal{A}_{\triangle 1}$ ' en la salida PDF y NVDA/MathCAT verbalizará «área del triángulo uno»; $\$ \backslash \text{spAfig}[print-sym=square](1) \$$ 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 $((F|P|B|L|T))$.

`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`

<code>\spPfig</code>	<code>\spPfig((punto, punto, punto, ...))</code>
<code>\spPfig[(key = val)]</code>	<code>\spPfig[(key = val)]((punto, punto, punto, ...))</code>
<code>\spPfig((1 2 3 ... n))</code>	<code>\spPfig((F P B L T))</code>
<code>\spPfig[(key = val)]((1 2 3 ... n))</code>	<code>\spPfig[(key = val)]((F P B L T))</code>

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={trapezio}](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`.

`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`

<code>\sprad</code>	<code>\sprad</code>	<code>\sprad[key = val]</code>
<code>\spdiam</code>	<code>\spdiam</code>	<code>\spdiam[key = val]</code>

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 ' $\varnothing d$ ' 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 = {\langle true | false \rangle}` 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 = {\langle verbalización NVDA \rangle}` 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 = {\langle true | false \rangle}` 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`

<code>\spcirc</code>	<code>\spcirc</code>	<code>\spcirc[\langle key = val \rangle]</code>
<code>\spcirc(\langle centro \rangle)</code>	<code>\spcirc(\langle centro \rangle)</code>	<code>\spcirc[\langle key = val \rangle](\langle centro \rangle)</code>
<code>\spcirc(\langle centro, radio \rangle)</code>	<code>\spcirc(\langle centro, radio \rangle)</code>	<code>\spcirc[\langle key = val \rangle](\langle centro, radio \rangle)</code>

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

La forma en la que se procesa el `\langle radio \rangle` junto a su impresión y verbalización se establece de manera interna: si es un `\langle número \rangle` entero o decimal se usa `\spnum`; si es una `\langle medida \rangle` se usa `\spunit`; si es un segmento `\langle punto punto \rangle` 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* `[\langle key = val \rangle]` ni el *argumento opcional delimitado por paréntesis*, `\spcirc` imprimirá ‘ $\odot$ ’ en la salida PDF y NVDA/MathCAT verbalizará «circunferencia».

Si se proporciona el *argumento* `(\langle centro \rangle)`, `\spcirc(0)` imprimirá ‘ $\odot(O)$ ’ en la salida PDF y NVDA/MathCAT verbalizará «circunferencia de centro en O».

Si se proporciona el *argumento* `(\langle centro, radio \rangle)`, `\spcirc(0,3)` imprimirá ‘ $\odot(O,3)$ ’ en la salida PDF y NVDA/MathCAT verbalizará «circunferencia de centro en O y radio tres».

El argumento `(\langle centro \rangle)` y el argumento `\langle radio \rangle` cuando tiene la forma `\langle punto punto \rangle` 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 = {\langle school | mathcat \rangle}` por defecto: *school*

Establece la *forma* en la que NVDA/MathCAT verbalizará el argumento `(\langle centro \rangle)`. Si se establece en *school*, NVDA/MathCAT verbaliza el argumento `(\langle centro \rangle)` 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 `\langle radio \rangle` cuando tiene la forma `\langle punto punto \rangle`.

`prefix = {\langle prefijo \rangle}` 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 = {\langle usual | odot \rangle}` 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](0)` imprimirá ‘C(O)’ en la salida PDF; si se establece en *odot*, `\spcirc[print-sym=odot](0)` imprimirá ‘ $\odot(O)$ ’ en la salida PDF.

`read-sty = {\langle circun | circulo \rangle}` por defecto: *circun*

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`

<code>\spAcirc</code>	<code>\spAcirc</code>	<code>\spAcirc[key = val]</code>
	<code>\spAcirc(centro)</code>	<code>\spAcirc[key = val](centro)</code>
	<code>\spAcirc(1 2 3 ... n)</code>	<code>\spAcirc[key = val](1 2 3 ... n)</code>

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.

Llaves para `\spAcirc`

- `read-cmd` = $\{\langle school \mid mathcat \rangle\}$ por defecto: *school*
- Establece la *forma* en la que NVDA/MathCAT verbalizará el argumento ($\langle centro \rangle$). Si se establece en *school*, NVDA/MathCAT verbaliza el argumento ($\langle centro \rangle$) 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` = $\{\langle prefijo \rangle\}$ 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` = $\{\langle circun \mid circulo \rangle\}$ 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*, $\$ \backslash \text{spAcirc}[\text{read-sty}=\text{circun}](0) \$$ se verbalizará «área de la circunferencia con centro en O» en NVDA/MathCAT; si se establece en *circulo*, $\$ \backslash \text{spAcirc}[\text{read-sty}=\text{circulo}](0) \$$ se verbalizará «área del círculo con centro en O» en NVDA/MathCAT.
- `concept` = $\{\langle true \mid false \rangle\}$ 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 ($\langle centro \rangle$) o ($\langle 1 \mid 2 \mid 3 \mid \dots \mid n \rangle$); si se establece en *false*, NVDA/MathCAT solo verbalizará el argumento ($\langle centro \rangle$) acorde al valor establecido por la llave `read-cmd`.
- 🔊 Utilizar `concept=false` sin el argumento ($\langle centro \rangle$) o con el argumento ($\langle 1 \mid 2 \mid 3 \mid \dots \mid n \rangle$) es equivalente a usar `silent=true`.
- `read-arg` = $\{\langle verbalización NVDA \rangle\}$ 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: $\$ \backslash \text{spAcirc}[\text{read-arg}=\{\text{esfera}\}](0) \$$ se verbalizará «área de la esfera con centro en O» en NVDA/MathCAT; $\$ \backslash \text{spAcirc}[\text{concept}=\text{false}, \text{read-arg}=\{\text{esfera}\}](0) \$$ se verbalizará «esfera con centro en O» en NVDA/MathCAT.
- 🔊 Utilizar esta llave con `concept=false` y el argumento ($\langle 1 \mid 2 \mid 3 \mid \dots \mid n \rangle$) es equivalente a usar `silent=true`.
- `print-sym` = $\{\langle usual \mid odot \rangle\}$ 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*, $\$ \backslash \text{spAcirc}[\text{print-sym}=\text{usual}](0) \$$ imprimirá ' $\mathcal{A}_{C(O)}$ ' en la salida PDF; si se establece en *odot*, $\$ \backslash \text{spAcirc}[\text{print-sym}=\text{odot}](0) \$$ imprimirá ' $\mathcal{A}_{\odot(O)}$ ' en la salida PDF.
- `sub-index` = $\{\langle subíndice \rangle\}$ por defecto: *no usado*
- Establece un *subíndice* sobre `\bigodot` o la letra 'C' establecida por la llave `print-sym` cuando se utiliza el argumento ($\langle centro \rangle$). El valor de esta llave *siempre* debe pasarse entre llaves '{ }'. Por ejemplo: $\$ \backslash \text{spAcirc}[\text{sub-index}=\{1\}](0) \$$ 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` = $\{\langle true \mid false \rangle\}$ 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 ($\langle 1 \mid 2 \mid 3 \mid \dots \mid n \rangle$) al ejecutar el comando sin modificar la impresión de este. Si se establece en *true*, $\$ \backslash \text{spAcirc}[\text{read-sub}=\text{true}](1) \$$ 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` = $\{\langle true \mid false \rangle\}$ 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 ($\langle centro \rangle$) o ($\langle 1 \mid 2 \mid 3 \mid \dots \mid n \rangle$) 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`

<code>\spPcirc</code>	<code>\spPcirc</code>	<code>\spPcirc[\langle key = val \rangle]</code>
	<code>\spPcirc(\langle centro \rangle)</code>	<code>\spPcirc[\langle key = val \rangle](\langle centro \rangle)</code>
	<code>\spPcirc(\langle 1 \mid 2 \mid 3 \mid \dots \mid n \rangle)</code>	<code>\spPcirc[\langle key = val \rangle](\langle 1 \mid 2 \mid 3 \mid \dots \mid n \rangle)</code>

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

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

Si se proporciona el argumento (`\centro`), `\spPcirc(0)` 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 (`\centro{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](0)` se verbalizará «perímetro de la circunferencia con centro en O» en NVDA/MathCAT; si se establece en `\circulo`, `\spPcirc[read-sty=\circulo](0)` 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 (`\centro{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 (`\centro{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}](0)` se verbalizará «perímetro del disco con centro en O» en NVDA/MathCAT; `\spPcirc[concept=false,read-arg={disco}](0)` se verbalizará «disco con centro en O» en NVDA/MathCAT.

- Utilizar esta llave con `concept=false` y el argumento (`\centro{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](0)` imprimirá ' $\mathcal{P}_{C(O)}$ ' en la salida PDF; si se establece en `\odot`, `\spPcirc[print-sym=odot](0)` 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}](0)` 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 (`\centro{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 (`\centro{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 ‘ $\mathcal{V}$ ’; 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.

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 ' $-\infty, 3$ ' 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: ' $-\infty$ ' 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`, `\bigg` y `\biggr`, o `\Bigg` 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.

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

Con dos o más argumentos, $\$ \backslash \text{spfun}\{f(x,y)\}$ imprimirá ' $f(x,y)$ ' y NVDA/MathCAT verbalizará «*f de x coma y*». Un argumento puede ser una expresión: $\$ \backslash \text{spfun}\{f(x+1)\}$ 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: $\$ \backslash \text{spfun}\{f(g(x))\}$ imprimirá ' $f(g(x))$ ' y verbalizará «*f de g de x*»; $\$ \backslash \text{spfun}\{f(f^{-1}(y))\}$ imprimirá ' $f(f^{-1}(y))$ ' y verbalizará «*f de f inversa de y*»; $\$ \backslash \text{spfun}\{f(g(x), h(y))\}$ 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: $\$ \backslash \text{spfun}\{f(1\{, \}5)\}$ tiene un solo argumento.

El comando `\spfun` devuelve un “error” y se detiene la compilación del documento cuando el $\{ \langle función \rangle \}$ 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` = $\{ \langle school \mid mathcat \rangle \}$ 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»: $\$ \backslash \text{spfun}\{F'(x)\}$ 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` = $\{ \langle prefijo \rangle \}$ 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` = $\{ \langle true \mid false \rangle \}$ 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` = $\{ \langle true \mid false \rangle \}$ por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT de la $\{ \langle función \rangle \}$ 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`

<code>\spdom</code>	$\backslash \text{spdom}\{ \langle función \rangle \}$	$\backslash \text{spdom}[\langle key = val \rangle] \{ \langle función \rangle \}$
<code>\sprec</code>	$\backslash \text{sprec}\{ \langle función \rangle \}$	$\backslash \text{sprec}[\langle key = val \rangle] \{ \langle función \rangle \}$
<code>\spcod</code>	$\backslash \text{spcod}\{ \langle función \rangle \}$	$\backslash \text{spcod}[\langle key = val \rangle] \{ \langle función \rangle \}$

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

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

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

Con la inversa, $\$ \backslash \text{spdom}\{f^{-1}\}$ imprimirá ' $\text{Dom}(f^{-1})$ ' y verbalizará «*dominio de f inversa*»; $\$ \backslash \text{sprec}\{f^{-1}\}$ 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` = $\{ \langle prefijo \rangle \}$ 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 $\$ \backslash \text{spdom}[\text{prefix}=\{\text{el}\}]\{f\}$, NVDA/MathCAT verbalizará «*el dominio de f*»; con $\$ \backslash \text{sprec}[\text{prefix}=\{\text{el}\}]\{f\}$, «*el recorrido de f*»; con $\$ \backslash \text{spcod}[\text{prefix}=\{\text{el}\}]\{f\}$, «*el codominio de f*». La impresión no cambia.

`silent = {\textit{true} | \textit{false}}`

por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT de la $\{ \langle \textit{función} \rangle \}$ pasada al comando sin modificar la impresión de este. Si se establece en *true*, NVDA/MathCAT NO verbalizará la salida del comando: $\$ \backslash \text{spdom}[\text{silent}=\textit{true}]\{f\}$ 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 $\backslash \text{spfuncomp}$

$\backslash \text{spfuncomp}$ $\backslash \text{spfuncomp}\{\langle \textit{composición} \rangle\}$ $\backslash \text{spfuncomp}[\langle \textit{key} = \textit{val} \rangle]\{\langle \textit{composición} \rangle\}$

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

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

El comando $\backslash \text{spfuncomp}$ devuelve un “error” y se detiene la compilación del documento cuando hay menos de dos funciones, cuando algo entre los $\backslash \text{circ}$ 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 $f \backslash \text{circ} g(x)$, sin paréntesis alrededor de la composición, no es válida porque es ambigua.

Llaves para $\backslash \text{spfuncomp}$

`silent = {\textit{true} | \textit{false}}`

por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT de la $\{ \langle \textit{composición} \rangle \}$ 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 $\backslash \text{spabs}$

$\backslash \text{spabs}$ $\backslash \text{spabs}\{\langle x \rangle\}$ $\backslash \text{spabs}[\langle \textit{key} = \textit{val} \rangle]\{\langle x \rangle\}$

El comando $\backslash \text{spabs}$ trabaja en *modo matemático* y recibe el *argumento obligatorio* $\{ \langle x \rangle \}$: cualquier expresión matemática. Imprime el valor absoluto de $\{ \langle x \rangle \}$ entre las barras $\backslash \text{vert}$ y $\backslash \text{rvert}$.

Si no se proporciona el *argumento opcional* $[\langle \textit{key} = \textit{val} \rangle]$, $\$ \backslash \text{spabs}\{x\}$ imprimirá ‘ $|x|$ ’ en la salida PDF y NVDA/MathCAT verbalizará «*valor absoluto de x*»; $\$ \backslash \text{spabs}\{-5\}$ imprimirá ‘ $|-5|$ ’ y verbalizará «*valor absoluto de menos 5*»; $\$ \backslash \text{spabs}\{x-3\}$ 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: $\$ \backslash \text{spabs}\{x-3\} - 1$ verbalizará «*valor absoluto de x menos 3 fin valor absoluto menos 1*». La llave *close* controla este cierre.

El comando $\backslash \text{spabs}$ devuelve un “error” y se detiene la compilación del documento cuando el argumento está vacío.

Llaves para $\backslash \text{spabs}$

`close = {\textit{auto} | \textit{true} | \textit{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 $\{ \langle x \rangle \}$ tiene más de un símbolo, sin contar un signo + o - inicial: $\$ \backslash \text{spabs}\{x\}$ y $\$ \backslash \text{spabs}\{-5\}$ no lo llevan y $\$ \backslash \text{spabs}\{x-3\}$ sí. Con *true* siempre se verbaliza: $\$ \backslash \text{spabs}[\text{close}=\textit{true}]\{x\}$ verbalizará «*valor absoluto de x fin valor absoluto*». Con *false* nunca se verbaliza: $\$ \backslash \text{spabs}[\text{close}=\textit{false}]\{x-3\}$ verbalizará «*valor absoluto de x menos 3*». La llave no tiene efecto con *silent* ni con *read-arg*.

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».

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 una *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`

<code>\sprango</code>	<code>\sprango</code>	<code>\sprango[⟨key = val⟩]</code>
<code>\spriq</code>	<code>\spriq</code>	<code>\spriq[⟨key = val⟩]</code>
<code>\spmin</code>	<code>\spmin</code>	<code>\spmin[⟨key = val⟩]</code>
<code>\spmax</code>	<code>\spmax</code>	<code>\spmax[⟨key = val⟩]</code>

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`

<code>\spcuartil</code>	<code>\spcuartil{⟨k⟩}</code>	<code>\spcuartil[⟨key = val⟩]{⟨k⟩}</code>
<code>\spdecil</code>	<code>\spdecil{⟨k⟩}</code>	<code>\spdecil[⟨key = val⟩]{⟨k⟩}</code>
<code>\spPerc</code>	<code>\spPerc{⟨k⟩}</code>	<code>\spPerc[⟨key = val⟩]{⟨k⟩}</code>

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».

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*

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: $\$ \backslash \text{spcase}[\text{read-arg}=\{\text{casos que cumplen}\}]\{f\}$ imprimirá ‘ C_f ’ en la salida PDF y NVDA/MathCAT verbalizará «casos que cumplen».

`silent = {\langle true \mid false \rangle}`

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 $\backslash \text{spOmega}$

$\backslash \text{spOmega}$ $\backslash \text{spOmega}$ $\backslash \text{spOmega}[\langle \text{key} = \text{val} \rangle]$

El comando $\backslash \text{spOmega}$ trabaja en *modo matemático* y recibe el *argumento opcional* $[\langle \text{key} = \text{val} \rangle]$ 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*, $\$ \backslash \text{spOmega}$ imprimirá ‘ Ω ’ en la salida PDF y NVDA/MathCAT verbalizará «espacio muestral».

Llaves para $\backslash \text{spOmega}$

`card = {\langle true \mid false \rangle}`

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*, $\$ \backslash \text{spOmega}[\text{card}=\text{true}]$ imprimirá ‘ $\#\Omega$ ’ en la salida PDF y NVDA/MathCAT verbalizará «cardinalidad del espacio muestral».

`sub-index = {\langle subíndice \rangle}`

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

`read-arg = {\langle verbalización NVDA \rangle}`

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: $\$ \backslash \text{spOmega}[\text{read-arg}=\{\text{universo}\}]$ imprimirá ‘ Ω ’ en la salida PDF y NVDA/MathCAT verbalizará «universo».

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

`silent = {\langle true \mid false \rangle}`

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 $\backslash \text{spPval}$

$\backslash \text{spPval}$ $\backslash \text{spPval}\{\langle \text{casos} \rangle\}\{\langle \text{total} \rangle\}$ $\backslash \text{spPval}[\langle \text{key} = \text{val} \rangle]\{\langle \text{casos} \rangle\}\{\langle \text{total} \rangle\}$

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

Si no se proporciona el *argumento opcional* $[\langle \text{key} = \text{val} \rangle]$, $\$ \backslash \text{spPval}\{1\}\{5\}$ imprimirá ‘ $\frac{1}{5}$ ’ en la salida PDF y NVDA/MathCAT verbalizará «1 de 5».

🔹 Los argumentos $\{\langle \text{casos} \rangle\}$ y $\{\langle \text{total} \rangle\}$ se tipografían directamente, sin procesamiento semántico adicional.

Llaves para $\backslash \text{spPval}$

`silent = {\langle true \mid false \rangle}`

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 = {\langle true \mid false \rangle}`

por defecto: *false*

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

El comando `\spProb`

$$\backslash\mathrm{spProb}\{\langle\textit{evento}\rangle\} \qquad \backslash\mathrm{spProb}[\langle\textit{key} = \textit{val}\rangle][\{\langle\textit{evento}\rangle\}]$$

El comando `\spProb` trabaja en *modo matemático* y recibe el *argumento obligatorio* $\{\langle\textit{evento}\rangle\}$: 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 $\$ \backslash\mathrm{spProb}\{A \cap (B \cup C)\} \$$ es válido.

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

Llaves para `\spProb`

`read-arg = \{\langle\textit{verbalización NVDA}\rangle\}` 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: $\$ \backslash\mathrm{spProb}[read-arg=\{\textit{probabilidad del suceso}\}]\{A\} \$$ imprimirá ‘ $P(A)$ ’ en la salida PDF y NVDA/MathCAT verbalizará «probabilidad del suceso».

`silent = \{\langle\textit{true} | \textit{false}\rangle\}` 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 $\$ \backslash\mathrm{spProb}\{A \cap (B \cup C)\} \$$, también se verbalizan como «y», «o» y «dado», respectivamente.

El comando `\spcomp`

$$\backslash\mathrm{spcomp}\{\langle\textit{evento}\rangle\} \qquad \backslash\mathrm{spcomp}[\langle\textit{key} = \textit{val}\rangle][\{\langle\textit{evento}\rangle\}]$$

El comando `\spcomp` trabaja en *modo matemático* y recibe el *argumento obligatorio* $\{\langle\textit{evento}\rangle\}$: 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* $[\langle\textit{key} = \textit{val}\rangle]$, $\$ \backslash\mathrm{spcomp}\{A\} \$$ imprimirá ‘ A^c ’ en la salida PDF y NVDA/MathCAT verbalizará «complemento de A»; $\$ \backslash\mathrm{spProb}\{A \cap \backslash\mathrm{spcomp}\{B\}\} \$$ imprimirá ‘ $P(A \cap B^c)$ ’ y verbalizará «probabilidad de A y complemento de B»; $\$ \backslash\mathrm{spcomp}\{A \cup B\} \$$ imprimirá ‘ $(A \cup B)^c$ ’ y verbalizará «complemento de se abren paréntesis A o B se cierran paréntesis».

Llaves para `\spcomp`

`bar = \{\langle\textit{true} | \textit{false}\rangle\}` 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*, $\$ \backslash\mathrm{spcomp}[bar=true]\{A\} \$$ imprimirá ‘ $\overline{A}$ ’ en la salida PDF; la verbalización es la misma en ambos casos.

`wrap-parens = \{\langle\textit{auto} | \textit{true} | \textit{false}\rangle\}` 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, $\$ \backslash\mathrm{spcomp}[bar=true]\{A \cup B\} \$$ se verbaliza «complemento de la expresión A o B», para dejar claro que abarca todo el evento.

`read-arg = \{\langle\textit{verbalización NVDA}\rangle\}` 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: $\$ \backslash\mathrm{spcomp}[read-arg=\{\textit{no ocurre A}\}]\{A\} \$$ imprimirá ‘ A^c ’ en la salida PDF y NVDA/MathCAT verbalizará «no ocurre A».

`silent = \{\langle\textit{true} | \textit{false}\rangle\}` 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.

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: $\{ \langle n \rangle \}$, el total de elementos, y, después de la primera coma, $\{ \langle k \rangle \}$, 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 $\{ \langle n \rangle \}$.

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* $[\langle key = val \rangle]$, `\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 $\{ \langle k \rangle \}$ 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 $\{ \langle n \rangle \}$.

`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 $\{ \quad \}$.

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 , $\{ \langle k \rangle \}$ 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* $\{ \langle n \rangle \}$. Imprime el factorial de $\{ \langle n \rangle \}$. Si $\{ \langle n \rangle \}$ es una expresión, los paréntesis se escriben en el argumento: `\spfact{(n-1)}` imprimirá $(n-1)!$.

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 spmml-i-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 `spmml-i-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 spmml-i-tool

El script `spmml-i-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 <code>AFghost=true</code> , obtiene el archivo de <code>MathML</code> de <code>luamml</code> , lo limpia y escribe <code>archivo-mathml.html</code> . 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 <code>rnv-wrapp</code> y <code>veraPDF</code> 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 <code>-mathml.html</code> 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 <code>lua_latex-dev</code> si existe y, si no, <code>lua_latex</code> .
-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 `LualTeX`. 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 `LualTeX`. The attached files are intended as examples and, in a way, to save time for Matthew Bertucci (@mbertucci), who runs the tests on the `TeX` side, when he sees this package and adds it to The `TeX` 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`, `\sp0mega`, `\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 📄](#).

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 PDF/A, <https://pdfa.org/PDF-2-0-new-features-real-world-impact/>, 2026.
- [10] PDF ASSOCIATION. “PDF 2.0 Errata Collection 3”. Available from PDF/A, <https://pdfa.org/PDF-2-0-errata-collection-3-now-available/>, 2026.
- [11] PDF ASSOCIATION. “WTPDF - Well-Tagged PDF specification”. Available from PDF/A, <https://pdfa.org/wtpdf/>, 2024.
- [12] LATEX PROJECT LWG. “Best Practice Guide: Math in PDF”. Available from PDF/A, <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.

- [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 $\text{\LaTeX}$ Project. “ $\text{\LaTeX}$ for authors current version”. Available from CTAN, <https://ctan.org/pkg/latex-base>, 2026.
- [17] LEARNLATEX.ORG. “Learn $\text{\LaTeX}$ - Your first steps in $\text{\LaTeX}$, in short lessons”. Available from LEARNLATEX.ORG, <https://www.learnlatex.org/es/>, 2026.
- [18] BERRY, KARL. “ $\text{\LaTeX} 2_{\epsilon}$: An Unofficial Reference Manual”. Available from CTAN, <https://ctan.org/pkg/latex2e-help-texinfo>, 2026.
- [19] The $\text{\LaTeX}$ Project. “The $\text{\LaTeX} 2_{\epsilon}$ sources”. Available from CTAN, <https://ctan.org/tex-archive/macros/latex/base>, 2026.
- [20] The $\text{\LaTeX}$ Project. “latex-lab – $\text{\LaTeX}$ laboratory”. Available from CTAN, <https://www.ctan.org/pkg/latex-lab>, 2026.
- [21] KRUEGER, MARCEL. “LuaMML - Automatically generate MathML from Lua $\text{\TeX}$ math mode material”. Available from CTAN, <https://ctan.org/pkg/luamml>, 2026.
- [22] FISCHER, ULRIKE. “tagpdf – $\text{\LaTeX}$ 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 $\text{\LaTeX}$, Lua $\text{\LaTeX}$, X $\text{\LaTeX}$, and Plain $\text{\TeX}$ ”. Available from CTAN, <https://www.ctan.org/pkg/babel>, 2026.
- [25] The $\text{\LaTeX}$ Project. “fontspec - Advanced font selection for X $\text{\LaTeX}$ and Lua $\text{\TeX}$ ”. 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 $\text{\TeX}$ (-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 $\text{\LaTeX}$ Project. “Extensive support for hypertext in $\text{\LaTeX}$ ”. Available from CTAN, <https://www.ctan.org/pkg/hyperref>, 2026.
- [31] The $\text{\LaTeX}$ Project. “The expl3 package”. Available from CTAN, <https://www.ctan.org/pkg/l3kernel>, 2026.
- [32] The $\text{\LaTeX}$ Project. “The $\text{\LaTeX} 3$ Interfaces”. Available from CTAN, <https://www.ctan.org/pkg/l3kernel>, 2026.
- [33] The $\text{\LaTeX}$ Project. “ $\text{\LaTeX}$ 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.

13 Change history

- v1.4 (ctan), 2026-11-02**
 - The geometry commands put an invisible separator between consecutive points (AB, AOB, ABC...), so that **MathCAT** 0.7.7 reads each point on its own instead of merging the letters into a word.
- 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 `spmml-i-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.

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
\spmcn	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

prefix	35	silent	48
print-sym	35	Llaves de \spFrec provistas por spanish-mathml-intent:	
radio-sty	35	read-arg	44
read-arg	35	read-sub	44
read-cmd	34	silent	44
read-sty	35	sub-index	44
read-sub	35	Llaves de \spfuncomp provistas por spanish-mathml-intent:	
silent	35	silent	41
spmside	34	Llaves de \spfun provistas por spanish-mathml-intent:	
spside	34	concept	40
sub-index	35	prefix	40
Llaves de \spcomb provistas por spanish-mathml-intent:		read-cmd	40
circular	47	silent	40
kind	47	Llaves de \spinterval provistas por spanish-mathml-intent:	
print-sty	47	concept	39
read-arg	47	prefix	39
repeat	47	read-arg	39
silent	47	read-cmd	39
Llaves de \spcomp provistas por spanish-mathml-intent:		silent	40
bar	46	size	39
read-arg	46	Llaves de \spmangle provistas por spanish-mathml-intent:	
silent	46	concept	28
wrap-parens	46	prefix	28
Llaves de \spcoord provistas por spanish-mathml-intent:		print-sym	28
silent	23	read-cmd	28
Llaves de \spcuartil provistas por spanish-mathml-intent:		recto	28
read-arg	44	silent	28
silent	44	Llaves de \spmarc provistas por spanish-mathml-intent:	
Llaves de \spdecil provistas por spanish-mathml-intent:		concept	29
read-arg	44	prefix	29
silent	44	read-cmd	29
Llaves de \spdesv provistas por spanish-mathml-intent:		silent	30
read-arg	43	Llaves de \spmcd provistas por spanish-mathml-intent:	
read-sub	43	concept	20
sigma	43	prefix	20
silent	43	read-sty	20
sub-index	43	result	20
Llaves de \spdiam provistas por spanish-mathml-intent:		silent	20
concept	33	upper-case	20
print-sym	34	Llaves de \spmcm provistas por spanish-mathml-intent:	
read-arg	34	concept	20
read-cmd	33	prefix	21
read-sub	34	read-sty	20
silent	34	result	21
sub-index	34	silent	21
upper-case	33	upper-case	20
Llaves de \spdiv provistas por spanish-mathml-intent:		Llaves de \spmediana provistas por spanish-mathml-intent:	
cifras-sep	18	read-arg	43
notation-sym	18	read-sub	43
read-parens	18	silent	43
read-period-sep	18	sub-index	43
read-sign	18	Llaves de \spmedia provistas por spanish-mathml-intent:	
read-sym	18	read-arg	43
set-sym	18	read-sub	43
wrap-parens	18	silent	43
Llaves de \spdom provistas por spanish-mathml-intent:		sub-index	43
prefix	41	Llaves de \spmoda provistas por spanish-mathml-intent:	
silent	41	read-arg	43
Llaves de \spdsym provistas por spanish-mathml-intent:		read-sub	43
prefix	17	silent	43
read-sym	17	sub-index	43
set-sym	17	Llaves de \spmone provistas por spanish-mathml-intent:	
suffix	17	cifras-sep	11
Llaves de \spfact provistas por spanish-mathml-intent:		currency	11
read-arg	48		

money-math	11	read-sty	37
sub-units	11	read-sub	37
Llaves de \spmQ provistas por spanish-mathml-intent:		silent	38
join-word	21	sub-index	37
read-parens	22	Llaves de \spPerc provistas por spanish-mathml-intent:	
silent	22	read-arg	44
use-dfrac	22	silent	44
use-spñZ	21	Llaves de \spPfig provistas por spanish-mathml-intent:	
wrap-parens	22	concept*	33
Llaves de \spmside provistas por spanish-mathml-intent:		concept	32
concept	27	prefix	32
prefix	27	print-sym	33
print-sym	27	read-arg	32
read-arg	27	read-cmd	32
read-cmd	26	read-sub	33
read-sty	27	silent	33
silent	27	Llaves de \spPoint provistas por spanish-mathml-intent:	
Llaves de \spmsym provistas por spanish-mathml-intent:		concept	24
prefix	17	prefix	24
print-sym	17	read-cmd	24
read-sym	17	silent	24
set-sym	17	Llaves de \spPoly provistas por spanish-mathml-intent:	
suffix	17	concept	30
Llaves de \spmul provistas por spanish-mathml-intent:		prefix	30
cifras-sep	18	print-sym	30
notation-sym	18	read-cmd	30
read-parens	18	silent	30
read-period-sep	18	Llaves de \spProb provistas por spanish-mathml-intent:	
read-sign	18	read-arg	46
read-sym	18	silent	46
set-sym	18	Llaves de \spProm provistas por spanish-mathml-intent:	
wrap-parens	18	read-arg	42
Llaves de \spnD provistas por spanish-mathml-intent:		silent	42
concept	18	Llaves de \spProp provistas por spanish-mathml-intent:	
prefix	18	inline	23
result-num	19	silent	23
result	19	use-dfrac	23
silent	19	Llaves de \spPval provistas por spanish-mathml-intent:	
Llaves de \spnM provistas por spanish-mathml-intent:		silent	45
concept	19	use-dfrac	45
prefix	19	Llaves de \sprad provistas por spanish-mathml-intent:	
result-num	19	concept	33
result	19	read-arg	34
silent	19	read-cmd	33
Llaves de \spnum provistas por spanish-mathml-intent:		read-sub	34
cifras-sep	7	silent	34
notation-sym	8	sub-index	34
read-period-sep	7	upper-case	33
read-sign	7	Llaves de \sprayo provistas por spanish-mathml-intent:	
Llaves de \spñZ provistas por spanish-mathml-intent:		concept	25
cifras-sep	21	prefix	25
read-parens	21	read-cmd	25
read-sign	21	read-sty	25
wrap-parens	21	silent	25
Llaves de \sp0omega provistas por spanish-mathml-intent:		Llaves de \sprazon provistas por spanish-mathml-intent:	
card	45	inline	22
read-arg	45	silent	22
silent	45	use-dfrac	22
sub-index	45	Llaves de \spread provistas por spanish-mathml-intent:	
Llaves de \spPcirc provistas por spanish-mathml-intent:		read-arg	16
concept	37	silent	16
prefix	37	Llaves de \sprecta provistas por spanish-mathml-intent:	
print-sym	37	concept	25
read-arg	37	prefix	25
read-cmd	37	read-cmd	25

T
Herramienta provista por [spanish-mathml-intent](#):
spmmli-tool 48

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.

```
1 <*package>
```

Identify the internal prefix, following the L^AT_EX3 DocStrip convention, for `l3doc` class.

```
2 <@@=spintent>
```

15.4 Declaration of the package

First we will make sure we have a minimum release version of L^AT_EX to work correctly, otherwise, we will return an “error”.

```
3 \NeedsTeXFormat{LaTeX2e}[2026-11-01]
4 \IfFormatAtLeastF { 2026-11-01 }
5 {
6   \PackageError { spanish-mathml-intent }
7     { LaTeX~kernel~too~old~need~2026-11-01~or~later }
8   {
9     spanish-mathml-intent~requires~a~LaTeX2e~release~2026-11-01~or~
10    later.~Update~your~TeX~distribution.
11  }
12 }
```

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

```
13 \ProvidesExplPackage {spanish-mathml-intent} {2026-11-02} {1.4} {Spanish MathML intent}
14 \keys_define:nn { spintent }
15 {
16   mathcal .usage:n = load,
17   mathcal .bool_set:N = \l__spintent_mathcal_bool,
18   mathcal .initial:n = true,
19   mathcal .value_required:n = true,
20   AFghost .usage:n = load,
21   AFghost .bool_set:N = \l__spintent_debug_ghost_bool,
22   AFghost .initial:n = false,
23   AFghost .value_required:n = true,
24 }
25 \ProcessKeyOptions [ spintent ]
```

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”.

```

26 \IfDocumentMetadataTF
27 {
28   \sys_if_engine luatex:TF
29   {
30     \msg_new:nnn { spintent } { package-not-load }
31     {
32       The~'#1'~package~will~be~loaded~as~a~dependency.
33     }
34     \IfPackageLoadedF { unicode-math }
35     {
36       \IfPackageLoadedF { lua-unicode-math }
37       {
38         \msg_info:nnn { spintent } { package-not-load } { unicode-math }
39         \RequirePackage{unicode-math}
40       }
41     }
42     \msg_new:nnn { spintent } { env-success }
43     {
44       Everything~looks~correct,~spanish-mathml-intent~will~make~the~intent!
45     }
46     \msg_info:nn { spintent } { env-success }
47   }
48   {
49     \msg_new:nnnn { spintent } { fatal-engine-requires-luatex }
50     {
51       The~spanish-mathml-intent~package~requires~LuaLaTeX.
52     }
53     {
54       This~package~relies~on~Lua~module.~Compile~with~lualatex.
55     }
56     \msg_fatal:nn { spintent } { fatal-engine-requires-luatex }
57   }
58 }
59 {
60   \msg_new:nnnn { spintent } { fatal-metadata-missing }
61   {
62     '\token_to_str:N \DocumentMetadata'~is~missing.
63   }
64   {
65     The~spanish-mathml-intent~package~requires~tagged~PDF~active.\\
66     You~must~declare~'\token_to_str:N \DocumentMetadata { tagging=on }'.
67   }
68   \msg_fatal:nn { spintent } { fatal-metadata-missing }
69 }

```

15.5 Some kernel variants

```

\str_lowercase:V
\seq_set_from_clist:Ne
\int_set:NV

```

Non-standard kernel variants.

```

70 \cs_generate_variant:Nn \str_lowercase:n { V }
71 \cs_generate_variant:Nn \seq_set_from_clist:Nn { Ne }
72 \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 `\l_spintent_luafun_classic_mathcal:nn`, controlled by the key `mathcal`, the string variable `\l_spintent_luaset_sym_fam_ok_str` used by the function `\l_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.

```

73 \str_new:N \l_spintent_luaset_cal_fam_ok_str
74 \str_new:N \l_spintent_luaset_sym_fam_ok_str
75 \str_new:N \l_spintent_luaset_scope_attr_str

```

(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.

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

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

`\l__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 `spmml-tool` can remove them before generating the Associated~Files structure of the *tagged* PDF for `mathml-AF` mode.

```
77 \bool_if:NTF \l__spintent_debug_ghost_bool
78 { \l__spintent_luafun_set_debug_ghost:n { true } }
79 { \l__spintent_luafun_set_debug_ghost:n { false } }
```

(End of definition for `\l__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 `\l__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.

```
80 \int_new:N \l__spintent_scope_attr_int
81 \int_set:NV \l__spintent_scope_attr_int \l__spintent_luaset_scope_attr_str
```

(End of definition for `\l__spintent_scope_attr_int`.)

`\l__spintent_luafun_spunit_parse_product:nV` Variants generated for `\l__spintent_luafun_spunit_parse_product:nn`, used by `\spunit`;
`\l__spintent_luafun_spmoney_lookup_data:V` `\l__spintent_luafun_spmoney_lookup_data:n`, used by `\spmoney`; `\l__spintent_luafun_spmQ_scale_int:Vn` used by `\spmQ`; and `\l__spintent_luafun_spfun_parse_and_set:n`, used by `\spfun`.

```
82 \cs_generate_variant:Nn \l__spintent_luafun_spunit_parse_product:nn { nV }
83 \cs_generate_variant:Nn \l__spintent_luafun_spmoney_lookup_data:n { V }
84 \cs_generate_variant:Nn \l__spintent_luafun_spmQ_scale_int:nn { Vn }
85 \cs_generate_variant:Nn \l__spintent_luafun_spfun_parse_and_set:n { v }
```

(End of definition for `\l__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` First, we define two warnings in case our fallback fails: the “warning” `mathcal-fallback` is raised when
`right-angle-sqr-fallback` 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`.

```
86 \hook_gput_code:nnn { begindocument } { spintent }
87 {
88   \l__spintent_luafun_init_math_fallbacks:
89 }
90 \msg_new:nnn { spintent } { mathcal-fallback }
91 {
92   Using~classic~'\token_to_str:N \mathcal'~as~fallback~
93   (font~not~found~or~no~free~math~family~available)~\msg_line_context:.
94 }
95 \msg_new:nnn { spintent } { right-angle-sqr-fallback }
96 {
97   Using~'\token_to_str:N \rightangle'~as~fallback~
98   for~the~right-angle~glyph~\msg_line_context:.
99 }
```

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

`\__spinttent_mathcal:n` Now we define the internal function `\__spinttent_mathcal:n` for `\mathcal` which checks the state of the variable `\l__spinttent_mathcal_bool` set by the package key `mathcal`; if its state is true, we check the string variable `\l__spinttent_luaset_cal_fam_ok_str` and if the latter has the value true, we call the function `\__spinttent_luafun_classic_mathcal:n` and otherwise we return a warning message and use `\mathcal`. Otherwise, that is, `mathcal=false` we use `\mathcal`.

```

100 \cs_new_protected:Nn \__spinttent_mathcal:n
101 {
102   \bool_if:NTF \l__spinttent_mathcal_bool
103   {
104     \str_if_eq:VnTF \l__spinttent_luaset_cal_fam_ok_str { true }
105     { \__spinttent_luafun_classic_mathcal:n {#1} }
106     {
107       \msg_warning:nn { spinttent } { mathcal-fallback }
108       \mathcal {#1}
109     }
110   }
111   { \mathcal {#1} }
112 }

```

(End of definition for `\__spinttent_mathcal:n`.)

`\__spinttent_right_angle_sqr:` The internal function `\__spinttent_right_angle_sqr:` establishes the internal copy of the command `\rightanglesqr`. First, we check the string variable `\l__spinttent_luaset_sym_fam_ok_str`. If it is true, we call the function `\__spinttent_luafun_right_angle_sqr:`; otherwise we return a warning message and use `\rightangle` instead.

```

113 \cs_new_protected:Nn \__spinttent_right_angle_sqr:
114 {
115   \str_if_eq:VnTF \l__spinttent_luaset_sym_fam_ok_str { true }
116   { \__spinttent_luafun_right_angle_sqr: }
117   {
118     \msg_warning:nn { spinttent } { right-angle-sqr-fallback }
119     \rightangle
120   }
121 }

```

(End of definition for `\__spinttent_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 `spinttent.lua`.

`\__spinttent_luafun_mrow_intent_arg:nn` The internal function `\__spinttent_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 `\__spinttent_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.

`\__spinttent_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.

```

122 \cs_new_protected:Nn \__spinttent_invisible_sep:
123 {
124   \__spinttent_luafun_invisible_sep:
125 }

```

🟢 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.

```
126 \cs_new_protected:Nn \__spintent_inner_sep:
127 {
128     \__spintent_luafun_inner_sep:
129 }
```

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:`)

`\__spintent_point_sep:` Places an invisible separator (U+2063) between two consecutive points of a geometric command, written to MathML as `<mo intent=":silent" mathvariant="normal">`. It must be an `<mo>`: MathCAT merges consecutive single-letter `<mi>` elements into a word before reading the intent, even when they carry `arg`, and a silent `<mi>` between them does not prevent it.

```
130 \cs_new_protected:Nn \__spintent_point_sep:
131 {
132     \__spintent_luafun_point_sep:
133 }
```

(End of definition for `\__spintent_point_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`.

```
134 \hook_gput_code:nnn { begindocument } { spintent }
135 {
136     \Umathchardef \c__spintent_invisible_sep_int = "0 "0 "2063
137 }
```

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.

```
138 \cs_new_protected:Nn \__spintent_collapse_hyphens:N
139 {
140     \tl_if_in:NnT #1 { -- }
141     {
142         \tl_replace_all:Nnn #1 { -- } { - }
143         \__spintent_collapse_hyphens:N #1
144     }
145 }
```

(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 $\langle argument \rangle$ passed in ‘#2’ is *blank*. If so, it clears the target token list variable passed in ‘#1’. Otherwise, it converts the $\langle argument \rangle$ 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 \__spintent_spsomesym_prefix_tl{ foo bar baz }

```

The function will “store” the normalized string `foo-bar-baz` inside `\__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*.

```

146 \cs_new_protected:Nn \__spintent_sanitize_affix:Nn
147 {
148   \tl_if_blank:nTF {#2}
149   { \tl_clear:N #1 }
150   {
151     \tl_set:Ne #1 { \tl_to_str:e {#2} }
152     \tl_trim_spaces:N #1
153     \str_if_eq:VnTF #1 { _- }
154     { \tl_set:Nn #1 { - } }
155     {
156       \tl_replace_all:Nnn #1 { ~ } { - }
157       \tl_replace_all:Nee #1 { \tl_to_str:n { _ } } { - }
158       \__spintent_collapse_hyphens:N #1
159       \tl_put_left:Nn #1 { \q_mark }
160       \tl_put_right:Nn #1 { \q_stop }
161       \tl_replace_all:Nnn #1 { \q_mark - } { \q_mark }
162       \tl_replace_all:Nnn #1 { - \q_stop } { \q_stop }
163       \tl_remove_all:Nn #1 { \q_mark }
164       \tl_remove_all:Nn #1 { \q_stop }
165     }
166   }
167 }
168 \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 $\langle keys \rangle$ “globally” for the utilities provided by `spanish-mathml-intent`.

15.11.1 Definition of variables and error messages for `\setspintent`

```

\__spintent_setspintent_input_arg_cmd_tl
unknown-command-setup

```

The variable `\__spintent_setspintent_input_arg_cmd_tl` will *store* the command or command name passed to the “first” $\langle argument \rangle$ ‘#1’ of the command `\setspintent`.

```

169 \tl_new:N \__spintent_setspintent_input_arg_cmd_tl

```

The “error” `unknown-command-setup` is raised when the $\langle argument \rangle$ ‘#1’ passed to `\setspintent` is a “non-existent” command.

```

170 \msg_new:nnnn { spintent } { unknown-command-setup }
171 {
172   Cannot~configure~unknown~command~'\c_backslash_str #1'~\msg_line_context:.
173 }
174 {
175   The~command~'\c_backslash_str #1'~is~not~defined.~Check~the~spelling~of~the~command~
176   name.
177 }

```

(End of definition for `\__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 $\langle argument \rangle$ exists in ‘#1’ and set the $\langle keys \rangle$ passed as $\langle argument \rangle$ in ‘#2’ to the correct path; otherwise, it will return an error.

```

178 \cs_new_protected:Nn \__spintent_setspintent_set_keys:nn
179 {
180   \cs_if_exist:cTF {#1}
181   {
182     \keys_set:nn { spintent / #1 } {#2}
183   }
184   {
185     \msg_error:nnn { spintent } { unknown-command-setup } {#1}
186   }
187 }
188 \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`.

```

189 \cs_new_protected:Nn \__spintent_setspintent_parse_args:nn
190 {
191   \tl_if_single:nTF {#1}
192   {
193     \token_if_cs:NTF #1
194     {
195       \__spintent_setspintent_set_keys:en { \cs_to_str:N #1 } {#2}
196     }
197     {
198       \__spintent_setspintent_set_keys:nn {#1} {#2}
199     }
200   }
201   {
202     \__spintent_setspintent_set_keys:nn {#1} {#2}
203   }
204 }
205 \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`.

```

206 \NewDocumentCommand \setspintent { m m }
207 {
208   \tl_if_blank:nTF {#1}
209   {
210     \msg_error:nnn { spintent } { unknown-command-setup } { <empty> }
211   }
212   {
213     \tl_set:Nn \l__spintent_setspintent_input_arg_cmd_tl {#1}
214     \tl_trim_spaces:N \l__spintent_setspintent_input_arg_cmd_tl
215     \__spintent_setspintent_parse_args:Vn \l__spintent_setspintent_input_arg_cmd_tl
216     {#2}
217   }
218 }

```

(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 $\langle keys \rangle$ that use `.choice:n` in their implementation and the messages `cmd-key-unknown`, `cmd-key-value-unknown` used by the $\langle key \rangle$ `unknown` in commands that use $[\langle key=val \rangle]$.

```

219 \str_new:N \l__spintent_command_name_str
220 \msg_new:nnnn { spintent } { unknown-choice }
221 {

```

```

222     Invalid~value~'#3'~for~key~'#1';~use~one~of:~#2~\msg_line_context:.
223 }
224 {
225     The~key~'#1'~only~accepts~one~of:~#2.
226 }
227 \msg_new:nnnn { spintent } { cmd-key-unknown }
228 {
229     The~key~'#1'~is~unknown~by~command~
230     '\c_backslash_str \l__spintent_command_name_str'~and~is~being~ignored~
231     \msg_line_context:.
232 }
233 {
234     Check~the~spelling~of~the~key~name.
235 }
236 \msg_new:nnnn { spintent } { cmd-key-value-unknown }
237 {
238     The~key~'#1=#2'~is~unknown~by~command~
239     '\c_backslash_str \l__spintent_command_name_str'~and~is~being~ignored~
240     \msg_line_context:.
241 }
242 {
243     Check~the~spelling~of~the~key~name.
244 }

```

(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`.

```

245 \cs_new_protected:Nn \__spintent_unknown_keys_cmd:nn
246 {
247     \tl_if_blank:nTF {#2}
248     { \msg_error:nnn { spintent } { cmd-key-unknown } {#1} }
249     { \msg_error:nnnn { spintent } { cmd-key-value-unknown } {#1} {#2} }
250 }
251 \cs_generate_variant:Nn \__spintent_unknown_keys_cmd:nn { Vn }
252 \cs_new_protected:Nn \__spintent_unknown_keys_cmd:n
253 {
254     \__spintent_unknown_keys_cmd:Vn \l_keys_key_str {#1}
255 }

```

(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”.

```

256 \tl_new:N \l__spintent_luaset_sign_tl
257 \tl_new:N \l__spintent_luaset_part_int_tl

```

```

258 \tl_new:N \l__spintent_luaset_dec_sep_tl
259 \tl_new:N \l__spintent_luaset_part_dec_tl
260 \tl_new:N \l__spintent_luaset_part_period_tl
261 \str_new:N \l__spintent_luaset_has_integer_str

```

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

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_part_period_str` for the “*infinite periodic decimal part*”.

Formatted versions, `cifras-sep=true`, are *stored* in `\l__spintent_luaset_format_part_int_str` for the “*integer part*” and `\l__spintent_luaset_format_part_dec_str` for the “*finite decimal part*”.

```

262 \str_new:N \l__spintent_luaset_only_part_int_str
263 \str_new:N \l__spintent_luaset_only_part_dec_str
264 \str_new:N \l__spintent_luaset_only_part_period_str
265 \str_new:N \l__spintent_luaset_format_part_int_str
266 \str_new:N \l__spintent_luaset_format_part_dec_str

```

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

The flag variable `\l__spintent_luaset_millions_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_str`. 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`.

```

267 \str_new:N \l__spintent_luaset_millions_str
268 \str_new:N \l__spintent_luaset_decimal_period_str
269 \str_new:N \l__spintent_luaset_not_decimal_period_str
270 \str_new:N \l__spintent_luaset_not_decimal_not_period_str
271 \str_new:N \l__spintent_luaset_dec_is_all_zeros_str
272 \str_new:N \l__spintent_luaset_has_exponent_str
273 \tl_new:N \l__spintent_luaset_exponent_tl

```

(End of definition for `\l__spintent_luaset_millions_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 $\{\langle arguments \rangle\}$ 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`.

```

274 \str_new:N \l__spintent_luaset_has_units_str
275 \str_new:N \l__spintent_luaset_status_str
276 \tl_new:N \l__spintent_luaset_arg_above_tl
277 \tl_new:N \l__spintent_luaset_arg_below_tl
278 \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.

```

279 \msg_new:nnnn { spintent } { need-math-mode }
280 {
281     Math~mode~required~for~'\tl_trim_spaces:n {#1}'~\msg_line_context:.
282 }
283 {

```

```

284     Use~the~command~inside~'$...$'~or~in~a~math~environment.
285   }
286   \msg_new:nnnn { spintent } { need-text-mode }
287   {
288     Text~mode~required~for~'\tl_trim_spaces:n {#1}'~\msg_line_context:.
289   }
290   {
291     Use~the~command~outside~of~'$...$'~and~math~environments.
292   }

```

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 ‘x’ 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`.

```

293 \bool_new:N \l_spintent_spnum_read_terse_bool
294 \str_new:N \l_spintent_spnum_read_period_sep_str
295 \tl_new:N \l_spintent_spnum_notation_sym_tl
296 \str_new:N \l_spintent_spnum_intent_read_sign_str
297 \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.

```

298 \msg_new:nnnn { spintent } { spnum-has-units }
299   {
300     Units~are~not~allowed~in~a~numerical~value~('#1')~\msg_line_context:.
301   }
302   {
303     Use~'\token_to_str:N \spunit'~to~print~numbers~with~units.
304   }
305 \msg_new:nnnn { spintent } { spnum-no-digits }
306   {
307     Invalid~numerical~value~'#1'~\msg_line_context:.
308   }
309   {
310     A~numerical~value~must~contain~at~least~one~digit.
311   }

```

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

15.15.3 Definition of keys for \spnum

We define the *(keys)* `cifras-sep`, `read-sign`, `read-period-sep` and `notation-sym`.

```

cifras-sep
read-sign
read-period-sep
notation-sym
unknown
312 \keys_define:nn { spintent / spnum }
313 {
314   cifras-sep      .bool_set:N = \l__spintent_spnum_cifras_sep_bool,
315   cifras-sep      .initial:n = true,
316   cifras-sep      .value_required:n = true,
317   read-sign       .choices:nn =
318     { terse , clear }
319     {
320       \int_case:nn { \l_keys_choice_int }
321       {
322         {1} { \bool_set_true:N \l__spintent_spnum_read_terse_bool }
323         {2} { \bool_set_false:N \l__spintent_spnum_read_terse_bool }
324       }
325     },
326   read-sign       .initial:n = terse,
327   read-sign       .value_required:n = true,
328   read-sign / unknown .code:n =
329     \msg_error:nneee { spintent } { unknown-choice }
330     { read-sign } { terse , ~clear } { \exp_not:n {#1} },
331   read-period-sep .choices:nn =
332     { coma , punto }
333     {
334       \int_case:nn { \l_keys_choice_int }
335       {
336         {1} { \str_set:Nn \l__spintent_spnum_read_period_sep_str { coma } }
337         {2} { \str_set:Nn \l__spintent_spnum_read_period_sep_str { punto } }
338       }
339     },
340   read-period-sep .initial:n = coma,
341   read-period-sep .value_required:n = true,
342   read-period-sep / unknown .code:n =
343     \msg_error:nneee { spintent } { unknown-choice }
344     { read-period-sep } { coma , ~punto } { \exp_not:n {#1} },
345   notation-sym    .choices:nn =
346     { times , cdot }
347     {
348       \int_case:nn { \l_keys_choice_int }
349       {
350         {1} { \tl_set:Nn \l__spintent_spnum_notation_sym_tl { \times } }
351         {2} { \tl_set:Nn \l__spintent_spnum_notation_sym_tl { \cdot } }
352       }
353     },
354   notation-sym    .initial:n = cdot,
355   notation-sym    .value_required:n = true,
356   notation-sym / unknown .code:n =
357     \msg_error:nneee { spintent } { unknown-choice }
358     { notation-sym } { times , ~ cdot } { \exp_not:n {#1} },
359   unknown         .code:n =
360     {
361       \str_set:Nn \l__spintent_command_name_str { spnum }
362       \__spintent_unknown_keys_cmd:n {#1}
363     },
364 }

```

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

15.15.4 Internal functions for \spnum

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.

```

365 \cs_new_protected:Nn \__spintent_spnum_render_part:n
366 {
367   \MathMLintent { \str_use:c { \l__spintent_luaset_only_part_ #1 _str } :number }
368   {
369     \bool_if:NTF \l__spintent_spnum_cifras_sep_bool
370     { \str_use:c { \l__spintent_luaset_format_part_ #1 _str } }

```

```

371         { \tl_use:c { l__spintent_luaset_part_ #1 _tl } }
372     }
373 }
374 \cs_new_protected:Nn \__spintent_snum_render_int:
375 {
376     \__spintent_snum_render_part:n { int }
377 }
378 \cs_new_protected:Nn \__spintent_snum_render_dec:
379 {
380     \__spintent_snum_render_part:n { dec }
381 }

```

(End of definition for `\__spintent_snum_render_part:n`, `\__spintent_snum_render_int:`, and `\__spintent_snum_render_dec:`.)

```

\__spintent_snum_render_only_period:
\__spintent_snum_print_period_bar:
\__spintent_snum_render_period_sep:

```

The function `\__spintent_snum_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_snum_print_period_bar:`.

```

382 \cs_new_protected:Nn \__spintent_snum_render_only_period:
383 {
384     \MathMLintent { \l__spintent_luaset_only_part_period_str :number }
385     {
386         \l__spintent_luaset_part_period_tl
387     }
388 }

```

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

```

389 \cs_new_protected:Nn \__spintent_snum_print_period_bar:
390 {
391     \MathMLintent { _periódico:postfix($dp) }
392     {
393         {
394             \overline
395             {
396                 \MathMLarg { dp } { \__spintent_snum_render_only_period: }
397             }
398         }
399     }
400 }

```

The function `\__spintent_snum_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.

```

401 \cs_new_protected:Nn \__spintent_snum_render_period_sep:
402 {
403     \str_if_eq:VnTF \l__spintent_snum_read_period_sep_str { coma }
404     {
405         \MathMLintent { _coma:prefix($pp) }
406         {
407             \mathord{,}
408             \MathMLarg { pp } { \__spintent_snum_print_period_bar: }
409         }
410     }
411     {
412         \MathMLintent { _punto:prefix($pp) }
413         {
414             \mathord{.}
415             \MathMLarg { pp } { \__spintent_snum_print_period_bar: }
416         }
417     }
418 }

```

(End of definition for `\__spintent_snum_render_only_period:`, `\__spintent_snum_print_period_bar:`, and `\__spintent_snum_render_period_sep:`.)

```
\__spintent_snum_render_decimal_sep:
```

The function `\__spintent_snum_render_decimal_sep:` sets the variable `\l__spintent_snum_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_snum_render_dec:` to print the finite decimal part.

```

419 \cs_new_protected:Nn \__spintent_snum_render_decimal_sep:
420 {
421   \str_if_eq:VnTF \__spintent_luaset_dec_sep_tl { , }
422   { \str_set:Nn \__spintent_snum_intent_read_dec_sep_str { _coma } }
423   { \str_set:Nn \__spintent_snum_intent_read_dec_sep_str { _punto } }
424   \MathMLintent { \__spintent_snum_intent_read_dec_sep_str :prefix($dp) }
425   {
426     \mathord { \__spintent_luaset_dec_sep_tl }
427     \MathMLarg { dp }
428     {
429       % keep "dp" in mathml-AF
430       \MathMLintent { :silent } { \c__spintent_invisible_sep_int }
431       \__spintent_snum_render_dec:
432     }
433   }
434   % period part
435   \str_if_eq:VnTF \__spintent_luaset_decimal_period_str { true }
436   {
437     \str_if_eq:VnTF \__spintent_luaset_dec_is_all_zeros_str { true }
438     {
439       \__spintent_inner_sep:
440       \__spintent_snum_print_period_bar:
441     }
442     {
443       \MathMLintent { _con:prefix($pnum) }
444       {
445         \__spintent_inner_sep:
446         \MathMLarg { pnum }
447         {
448           \__spintent_snum_print_period_bar:
449         }
450       }
451     }
452   }
453 }

```

(End of definition for `\__spintent_snum_render_decimal_sep:`.)

`\__spintent_snum_print_part_dec:`
`\__spintent_snum_print_part_period:`

The function `\__spintent_snum_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_snum_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.

```

454 \cs_new_protected:Nn \__spintent_snum_print_part_dec:
455 {
456   \tl_if_empty:NF \__spintent_luaset_part_dec_tl
457   {
458     \__spintent_snum_render_decimal_sep:
459   }
460 }
461 \cs_new_protected:Nn \__spintent_snum_print_part_period:
462 {
463   \tl_if_empty:NF \__spintent_luaset_part_period_tl
464   {
465     \str_if_eq:VnTF \__spintent_luaset_not_decimal_period_str { true }
466     { \__spintent_snum_render_period_sep: }
467     {
468       \tl_if_empty:NT \__spintent_luaset_part_dec_tl
469       {
470         \__spintent_snum_render_period_sep:
471       }
472     }
473   }
474 }

```

(End of definition for `\__spintent_snum_print_part_dec:` and `\__spintent_snum_print_part_period:`.)

`\__spintent_snum_print_integer:`
`\__spintent_snum_print_num_start:`
`\__spintent_snum_mathml:`

The function `\__spintent_snum_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_snum_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_snum_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:`.

```

475 \cs_new_protected:Nn \__spintent_snum_print_integer:
476 {
477   \str_if_eq:VnTF \__spintent_luaset_not_decimal_not_period_str { true }
478   { \__spintent_snum_render_int: }
479   {
480     \MathMLintent { \__spintent_luaset_only_part_int_str :number:prefix($next) }
481     {
482       \bool_if:NTF \__spintent_snum_cifras_sep_bool
483       { \__spintent_luaset_format_part_int_str }
484       { \__spintent_luaset_part_int_tl }
485       \MathMLarg { next }
486       {
487         \tl_if_empty:NTF \__spintent_luaset_part_dec_tl
488         { \__spintent_snum_print_part_period: }
489         { \__spintent_snum_print_part_dec: }
490       }
491     }
492   }
493 }
494 \cs_new_protected:Nn \__spintent_snum_print_num_start:
495 {
496   \tl_if_empty:NTF \__spintent_luaset_sign_tl
497   {
498     \__spintent_invisible_sep: % need
499     \__spintent_snum_print_integer:
500   }
501   {
502     \bool_if:NTF \__spintent_snum_read_terse_bool
503     {
504       \str_if_eq:VnTF \__spintent_luaset_sign_tl { + }
505       { \str_set:Nn \__spintent_snum_intent_read_sign_str { _más:prefix($n) } }
506       {
507         \str_set:Nn \__spintent_snum_intent_read_sign_str { _menos:prefix($n) }
508       }
509     }
510     {
511       \str_if_eq:VnTF \__spintent_luaset_sign_tl { + }
512       {
513         \str_set:Nn \__spintent_snum_intent_read_sign_str
514         { _positivo:postfix($n) }
515       }
516       {
517         \str_set:Nn \__spintent_snum_intent_read_sign_str
518         { _negativo:postfix($n) }
519       }
520     }
521     \__spintent_invisible_sep: % need
522     \MathMLintent { \__spintent_snum_intent_read_sign_str }
523     {
524       \__spintent_luaset_sign_tl
525       \MathMLarg { n } { \__spintent_snum_print_integer: }
526     }
527   }
528 }
529 \cs_new_protected:Nn \__spintent_snum_mathml:
530 {
531   \__spintent_snum_print_num_start:
532   \str_if_eq:VnTF \__spintent_luaset_has_exponent_str { true }
533   {

```

```

534     \MathMLintent { _por } { \l__spintent_spnunotation_sym_tl }
535     10^{ \l__spintent_luaset_exponent_tl }
536   }
537 }

```

(End of definition for `\__spintent_spnunprint_integer:`, `\__spintent_spnunprint_num_start:`, and `\__spintent_spnunmathml:`.)

`\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_spnunmathml:` to render the final number, and finally closes the group.

```

538 \NewDocumentCommand \spnum { O{} m }
539 {
540   \mode_if_math:TF
541   {
542     \group_begin:
543     \keys_set:nn { spintent / spnum } {#1}
544     \__spintent_luafun_clean_split_and_set:n { \exp_not:n {#2} }
545     \str_if_eq:VnTF \l__spintent_luaset_has_units_str { true }
546     { \msg_error:nnn { spintent } { spnum-has-units } {#2} }
547     {
548       \tl_if_empty:NTF \l__spintent_luaset_part_int_tl
549       { \msg_error:nnn { spintent } { spnum-no-digits } {#2} }
550       { \__spintent_spnunmathml: }
551     }
552     \group_end:
553   }
554   { \msg_error:nnn { spintent } { need-math-mode } { \spnum } }
555 }

```

(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 SI fractions, products, and sexagesimal angles, are processed in the same way by the `spintent.lua`.

15.16.1 Definition of variables for `\spunit`

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`.

```

556 \str_new:N \l__spintent_spunit_luaset_above_n_str
557 \str_new:N \l__spintent_spunit_luaset_above_found_str
558 \str_new:N \l__spintent_spunit_luaset_above_sexa_str
559 \str_new:N \l__spintent_spunit_luaset_below_n_str
560 \str_new:N \l__spintent_spunit_luaset_below_found_str

```

(End of definition for `\l__spintent_spunit_luaset_above_n_str` and others.)

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.

```

561 \str_new:N \l__spintent_spunit_luaset_canonical_str
562 \str_new:N \l__spintent_spunit_luaset_read_str
563 \str_new:N \l__spintent_spunit_luaset_status_str

```

(End of definition for `\l__spinttent_spunit_luaset_canonical_str`, `\l__spinttent_spunit_luaset_read_str`, and `\l__spinttent_spunit_luaset_status_str`.)

`\l__spinttent_spunit_exponent_tl`
`\l__spinttent_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.

```
564 \tl_new:N \l__spinttent_spunit_exponent_tl
565 \str_new:N \l__spinttent_spunit_read_slash_str
```

(End of definition for `\l__spinttent_spunit_exponent_tl` and `\l__spinttent_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 `{\langle argument \rangle}`, 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.

```
566 \msg_new:nnnn { spinttent } { spunit-invalid-unit }
567 {
568   Invalid~unit~expression~'#1'~\msg_line_context:.
569 }
570 {
571   The~expression~contains~an~unknown~or~malformed~unit.~Check~the~spelling~or~verify~
572   that~the~unit~is~defined.
573 }
574 \msg_new:nnnn { spinttent } { spunit-no-units }
575 {
576   No~unit~found~in~'#1'~\msg_line_context:.
577 }
578 {
579   At~least~one~unit~is~required;~use~'\token_to_str:N \spnum'~for~pure~
580   numbers.
581 }
582 \msg_new:nnnn { spinttent } { spunit-inline-numeric-denominator }
583 {
584   Invalid~numeric~coefficient~in~the~inline~denominator~'#1'~\msg_line_context:.
585 }
586 {
587   Numbers~are~not~allowed~after~the~slash~'/'~in~the~inline~mode~of~units.
588 }
```

(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`.

```
589 \keys_define:nn { spinttent / spunit }
590 {
591   print-cdot      .bool_set:N = \l__spinttent_spunit_print_cdot_bool,
592   print-cdot      .initial:n = false,
593   print-cdot      .value_required:n = true,
594   read-cdot       .bool_set:N = \l__spinttent_spunit_read_cdot_bool,
595   read-cdot       .initial:n = false,
596   read-cdot       .value_required:n = true,
597   read-slash      .choices:nn =
598     { por , partido-por }
599   {
600     \int_case:nn { \l_keys_choice_int }
601     {
602       {1} { \str_set:Nn \l__spinttent_spunit_read_slash_str { _por } }
603       {2} { \str_set:Nn \l__spinttent_spunit_read_slash_str { _partido-por } }
604     }
605   },
606   read-slash      .initial:n = por,
607   read-slash      .value_required:n = true,
608   read-slash / unknown .code:n =
609     \msg_error:nneee { spinttent } { unknown-choice }
610     { read-slash } { por , ~partido-por } { \exp_not:n {#1} },
611   cifras-sep      .meta:nn = { spinttent / spnum } { cifras-sep = {#1} },
612   cifras-sep      .value_required:n = true,
613   read-period-sep .meta:nn = { spinttent / spnum } { read-period-sep = {#1} },
```

```

614     read-period-sep .value_required:n = true,
615     notation-sym    .meta:nn = { spintent / spnum } { notation-sym = {#1} },
616     notation-sym    .value_required:n = true,
617     unknown         .code:n =
618     {
619         \str_set:Nn \l__spintent_command_name_str { spunit }
620         \__spintent_unknown_keys_cmd:n {#1}
621     },
622 }

```

(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`.

```

623 \cs_new_protected:Nn \__spintent_spunit_format_canonical:
624 {
625     \str_case:VnF \l__spintent_spunit_luaset_canonical_str
626     {
627         { ℓ } { \ell }
628     }
629     { \mathrm { \l__spintent_spunit_luaset_canonical_str } }
630 }

```

(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.

```

631 \cs_new_protected:Nn \__spintent_spunit_load_term:nn
632 {
633     \str_set_eq:Nc \l__spintent_spunit_luaset_canonical_str
634     { \l__spintent_spunit_luaset_ #1 _ #2 _canonical_str }
635     \str_set_eq:Nc \l__spintent_spunit_luaset_read_str
636     { \l__spintent_spunit_luaset_ #1 _ #2 _read_str }
637     \tl_set_eq:Nc \l__spintent_spunit_exponent_tl
638     { \l__spintent_spunit_luaset_ #1 _ #2 _exp_tl }
639 }

```

(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.

```

640 \cs_new_protected:Nn \__spintent_spunit_print_term:nn
641 {
642     \int_compare:nNnT {#2} > { 1 }
643     {
644         \bool_if:NTF \l__spintent_spunit_print_cdot_bool
645         {
646             \bool_if:NTF \l__spintent_spunit_read_cdot_bool
647             { \MathMLintent { _por } { \cdot } }
648             { \MathMLintent { :silent } { \cdot } }
649         }
650         {
651             \,
652             \bool_if:NT \l__spintent_spunit_read_cdot_bool
653             { \MathMLintent { _por } { \mathopen { \c__spintent_invisible_sep_int } } }
654         }
655     }

```

Then it renders the base unit and, if it has one, its exponent.

```

656     \__spintent_spunit_load_term:nn {#1} {#2}
657     \tl_if_empty:NTF \l__spintent_spunit_exponent_tl
658     {
659         \MathMLintent { \l__spintent_spunit_luaset_read_str }
660         {
661             \__spintent_spunit_format_canonical:

```

```

662     }
663   }
664   {
665     {
666       \MathMLintent { \l__spintent_spunit_luaset_read_str }
667       {
668         \__spintent_spunit_format_canonical:
669       }
670     }
671     ^ { \l__spintent_spunit_exponent_tl }
672   }
673 }

```

(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.

```

674 \cs_new_protected:Nn \__spintent_spunit_print_product:n
675 {
676   \int_step_inline:nn { \str_use:c { \l__spintent_spunit_luaset_ #1 _n_str } }
677   { \__spintent_spunit_print_term:nn {#1} {##1} }
678 }

```

(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”.

```

679 \cs_new_protected:Nn \__spintent_print_spunit_hierarchy:
680 {
681   \tl_if_empty:NF \l__spintent_luaset_arg_above_tl
682   {
683     \__spintent_spunit_print_product:n { above }
684     \tl_if_empty:NF \l__spintent_luaset_arg_below_tl
685     {
686       \MathMLintent { \l__spintent_spunit_read_slash_str } { / }
687       \__spintent_spunit_print_product:n { below }
688     }
689   }
690 }

```

(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:.`

```

691 \cs_new_protected:Nn \__spintent_spunit_exec:n
692 {
693   \str_if_eq:VnTF \l__spintent_luaset_has_units_str { false }
694   { \msg_error:nnn { spintent } { spunit-no-units } {#1} }
695   {
696     \str_if_eq:VnTF \l__spintent_luaset_status_str { valid }
697     {
698       \str_if_eq:VnTF
699       \l__spintent_luaset_denom_has_number_str { true }
700       {
701         \msg_error:nnn { spintent }
702         { spunit-inline-numeric-denominator } {#1}
703       }
704     }
705     \__spintent_luafun_spunit_parse_product:nV { above }
706     \l__spintent_luaset_arg_above_tl
707     \tl_if_empty:NF \l__spintent_luaset_arg_below_tl
708     {
709       \__spintent_luafun_spunit_parse_product:nV { below }

```

```

710         \l__spintent_luaset_arg_below_tl
711     }
712     \bool_lazy_and:nnTF
713     {
714         \str_if_eq_p:ee { \l__spintent_spunit_luaset_above_found_str }
715         { true }
716     }
717     {
718         \bool_lazy_or_p:nn
719         { \tl_if_empty_p:N \l__spintent_luaset_arg_below_tl }
720         {
721             \str_if_eq_p:ee { \l__spintent_spunit_luaset_below_found_str }
722             { true }
723         }
724     }
725     {
726         \__spintent_spunit_mathml:
727     }
728     { \msg_error:nnn { spintent } { spunit-invalid-unit } {#1} }
729 }
730 { \msg_error:nnn { spintent } { spunit-invalid-unit } {#1} }
731 }
732 }
733 }

```

(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.

```

734 \cs_new_protected:Nn \__spintent_spunit_mathml:
735 {
736     \str_if_eq:VnTF \l__spintent_spunit_luaset_above_sexa_str { true }
737     {
738         \__spintent_spunit_load_term:nn { above } { 1 }
739         \tl_if_empty:NF \l__spintent_luaset_part_int_tl
740         {
741             \__spintent_spnum_mathml:
742         }
743         \bool_lazy_and:nnT
744         { \str_if_eq_p:ee { \l__spintent_luaset_part_int_tl } { 1 } }
745         { \tl_if_empty_p:N \l__spintent_luaset_part_dec_tl }
746         {
747             \str_set:Ne \l__spintent_spunit_luaset_read_str
748             {
749                 \str_case:Vn \l__spintent_spunit_luaset_canonical_str
750                 {
751                     { ° } { _grado:unit }
752                     { ' } { _minuto:unit }
753                     { " } { _segundo:unit }
754                 }
755             }
756         }
757         \MathMLintent { \l__spintent_spunit_luaset_read_str }
758         {
759             \mathord
760             {
761                 \str_case:Vn \l__spintent_spunit_luaset_canonical_str
762                 {
763                     { " } { \prime \! \prime }
764                     { ' } { \prime }
765                     { ° } { ° }
766                 }
767             }
768         }
769     }
770     {
771         \tl_if_empty:NF \l__spintent_luaset_part_int_tl

```

```

772     {
773         \__spintent_spnum_mathml: \,
774     }
775     \__spintent_print_spunit_hierarchy:
776 }
777 }

```

(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.

```

778 \NewDocumentCommand \spunit { 0{ } m }
779 {
780     \mode_if_math:TF
781     {
782         \group_begin:
783         \keys_set:nn { spintent / spunit } {#1}
784         \__spintent_luafun_clean_split_and_set:n { \exp_not:n {#2} }
785         \__spintent_spunit_exec:n {#2}
786         \group_end:
787     }
788     { \msg_error:nnn { spintent } { need-math-mode } { \spunit } }
789 }

```

(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 $\text{\TeX}$ 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.

```

790 \msg_new:nnnn { spintent } { spnewunit-duplicate }
791 {
792     The~unit~symbol~or~alias~'#1'~is~already~defined~\msg_line_context:.
793 }
794 {
795     Existing~units~and~aliases~cannot~be~redefined~with~
796     '\token_to_str:N \spnewunit'.
797 }
798 \msg_new:nnnn { spintent } { spunitalias-duplicate }
799 {
800     The~alias~'#1'~is~already~defined~or~reserved~\msg_line_context:.
801 }
802 {
803     Existing~units~and~aliases~cannot~be~redefined.
804 }
805 \msg_new:nnnn { spintent } { spunitalias-invalid-expr }
806 {
807     Invalid~expression~'#2'~for~alias~'#1'~\msg_line_context:.
808 }
809 {
810     The~expression~must~contain~only~defined~units,~exponents~and~one~'/'~.
811 }

```

(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.

```

812 \NewDocumentCommand \spnewunit { m m }
813 {
814   \mode_if_math:TF
815   {
816     \msg_error:nnn { spintent } { need-text-mode } { \spnewunit }
817   }
818   {
819     \__spintent_luafun_define_custom_unit:nn {#1} {#2}
820     \str_if_eq:VnT \l__spintent_spunit_luaset_status_str { duplicate }
821     {
822       \msg_error:nnn { spintent } { spnewunit-duplicate } {#1}
823     }
824   }
825 }

```

(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.

```

826 \NewDocumentCommand \spunitalias { m m }
827 {
828   \mode_if_math:TF
829   {
830     \msg_error:nnn { spintent } { need-text-mode } { \spunitalias }
831   }
832   {
833     \__spintent_luafun_define_spunit_alias:nn {#1} {#2}
834     \str_if_eq:VnT \l__spintent_spunit_luaset_status_str { duplicate }
835     {
836       \msg_error:nnn { spintent } { spunitalias-duplicate } {#1}
837     }
838     \str_if_eq:VnT \l__spintent_spunit_luaset_status_str { invalid-expr }
839     {
840       \msg_error:nnnn { spintent } { spunitalias-invalid-expr } {#1} {#2}
841     }
842   }
843 }

```

(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 $\langle \textit{argument} \rangle$ 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`

These variables are set by the Lua module `spintent.lua`. The variable `\l__spintent_spmoney_luaset_symbol_str` stores the currency symbol and `\l__spintent_spmoney_luaset_position_str` stores its position, pre or post.

```

844 \str_new:N \l__spintent_spmoney_luaset_symbol_str
845 \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`.)

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`

`\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.

```
846 \str_new:N \l__spintent_spmoney_luaset_sing_str
847 \str_new:N \l__spintent_spmoney_luaset_plur_str
848 \str_new:N \l__spintent_spmoney_luaset_conde_str
849 \str_new:N \l__spintent_spmoney_luaset_subsing_str
850 \str_new:N \l__spintent_spmoney_luaset_subplur_str
```

(End of definition for `\l__spintent_spmoney_luaset_sing_str` and others.)

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.

```
851 \str_new:N \l__spintent_spmoney_luaset_print_iso_str
852 \str_new:N \l__spintent_spmoney_luaset_currency_arg_str
853 \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`.)

The variables `\l__spintent_spmoney_luaset_extracted_curr_str` and `\l__spintent_spmoney_luaset_extracted_num_tl` store the currency and the number that the module extracts from the `{\argument}`.

```
854 \str_new:N \l__spintent_spmoney_luaset_extracted_curr_str
855 \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`.)

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_str` 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.

```
856 \str_new:N \l__spintent_spmoney_intent_money_str
857 \str_new:N \l__spintent_spmoney_intent_sign_str
858 \str_new:N \l__spintent_spmoney_intent_sub_unit_str
859 \str_new:N \l__spintent_spmoney_res_main_voice_str
860 \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.

```
861 \msg_new:nnnn { spintent } { spmoney-no-subunits }
862 {
863   The~option~'sub-units=true'~is~ignored~for~the~currency~'#1'~\msg_line_context:.
864 }
865 {
866   Currencies~like~'JPY'~or~'CLP'~have~no~subunits~in~the~Lua~module.
867 }
868 \msg_new:nnnn { spintent } { spmoney-invalid-currency }
869 {
870   Invalid~currency~'#1'~for~'\token_to_str:N \spmoney'~\msg_line_context:.
871 }
872 {
873   The~symbol,~alias~or~ISO~code~does~not~exist~in~the~Lua~module.
874 }
```

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

15.18.3 Definition of keys for \spmoney

currency We define the *(keys)* currency, sub-units, money-math and cifras-sep.
sub-units
money-math
cifras-sep
unknown

```

875 \keys_define:nn { spintent / spmoney }
876 {
877   currency .code:n = { \__spintent_spmoney_set_currency:n {#1} },
878   currency .value_required:n = true,
879   sub-units .bool_set:N = \__spintent_spmoney_sub_units_bool,
880   sub-units .initial:n = { false },
881   sub-units .value_required:n = true,
882   money-math .bool_set:N = \__spintent_spmoney_money_math_bool,
883   money-math .initial:n = { true },
884   money-math .value_required:n = true,
885   cifras-sep .meta:nn = { spintent / spnum } { cifras-sep = {#1} },
886   cifras-sep .value_required:n = true,
887   unknown .code:n =
888     {
889       \str_set:Nn \__spintent_command_name_str { spmoney }
890       \__spintent_unknown_keys_cmd:n {#1}
891     },
892 }
```

(End of definition for currency and others.)

15.18.4 Internal functions for \spmoney

__spintent_spmoney_set_currency:n 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 `\__spintent_spmoney_luaset_currency_arg_str`.

```

893 \cs_new_protected:Nn \__spintent_spmoney_set_currency:n
894 {
895   \__spintent_spmoney_normalize_key:n {#1}
896   \str_if_eq:VnTF \__spintent_spmoney_luaset_status_str { notfound }
897   {
898     \msg_error:nnn { spintent } { spmoney-invalid-currency } {#1}
899   }
900   { \str_set:Nn \__spintent_spmoney_luaset_currency_arg_str {#1} }
901 }
902 \cs_new_protected:Nn \__spintent_spmoney_normalize_key:n
903 {
904   \__spintent_luafun_spmoney_normalize_key:n { \exp_not:n {#1} }
905 }
906 \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.

```

907 \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 `\__spintent_luaset_has_units_str`. If a currency is detected, it extracts and normalizes the key from `\__spintent_luaset_arg_above_tl`. It throws an “error” if the currency is not found in the database. If valid, it updates `\__spintent_spmoney_luaset_currency_arg_str` and retrieves the full currency metadata from the `spintent.lua` via `\__spintent_luafun_spmoney_lookup_data:V`.

```

908 \cs_new_protected:Nn \__spintent_spmoney_validate_money:
909 {
910   \str_if_eq:VnT \__spintent_luaset_has_units_str { true }
911   {
912     \__spintent_spmoney_normalize_key:V \__spintent_luaset_arg_above_tl
913     \str_if_eq:VnTF \__spintent_spmoney_luaset_status_str { notfound }
914     {
915       \msg_error:nne { spintent } { spmoney-invalid-currency }
916       { \__spintent_luaset_arg_above_tl }
917     }
918   }
919   \str_set:NV
920     \__spintent_spmoney_luaset_currency_arg_str
921     \__spintent_luaset_arg_above_tl
```

```

922     }
923   }
924   \str_set:N\l__spinttent_spmoney_luaset_status_str { found }
925   \__spinttent_luafun_spmoney_lookup_data:V \l__spinttent_spmoney_luaset_currency_arg_str
926 }

```

(End of definition for `\__spinttent_spmoney_validate_money:`.)

`\__spinttent_spmoney_print_symbol:`

The function `\__spinttent_spmoney_print_symbol:` renders the “*currency symbol*”. If `\l__spinttent_spmoney_luaset_status_str` is `true`, it outputs the ISO code using `\mathrm`. Otherwise, it prints the “*standard symbol*”, directly in math mode when `\l__spinttent_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.

```

927 \cs_new_protected:Nn \__spinttent_spmoney_print_symbol:
928 {
929   \str_if_eq:NnTF \l__spinttent_spmoney_luaset_print_iso_str { true }
930   { \mathrm { \l__spinttent_spmoney_luaset_currency_arg_str } }
931   {
932     \str_if_eq:NnTF \l__spinttent_spmoney_luaset_symbol_str { $ }
933     {
934       \bool_if:NnTF \l__spinttent_spmoney_money_math_bool
935       { \$ } { \text { \$ } }
936     }
937     {
938       \bool_if:NnTF \l__spinttent_spmoney_money_math_bool
939       { \l__spinttent_spmoney_luaset_symbol_str }
940       { \text { \l__spinttent_spmoney_luaset_symbol_str } }
941     }
942   }
943 }

```

(End of definition for `\__spinttent_spmoney_print_symbol:`.)

`\__spinttent_spmoney_render_layout:`

The function `\__spinttent_spmoney_render_layout:` determines the layout order of the “*currency symbol*” relative to the numeric value. It evaluates `\l__spinttent_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*”.

```

944 \cs_new_protected:Nn \__spinttent_spmoney_render_layout:
945 {
946   \str_if_eq:NnTF \l__spinttent_spmoney_luaset_position_str { pre }
947   {
948     \__spinttent_spmoney_print_symbol:
949     \MathMLLarg { n } { \__spinttent_spmoney_render_number_node: }
950   }
951   {
952     \MathMLLarg { n } { \__spinttent_spmoney_render_number_node: }
953     \__spinttent_spmoney_print_symbol:
954   }
955 }

```

(End of definition for `\__spinttent_spmoney_render_layout:`.)

`\__spinttent_spmoney_render_number_node:`
`\__spinttent_spmoney_render_only_integer_node:`

The function `\__spinttent_spmoney_render_number_node:` renders the whole number, with the integer and the decimal part. The function `\__spinttent_spmoney_render_only_integer_node:` renders only the integer part.

```

956 \cs_new_protected:Nn \__spinttent_spmoney_render_number_node:
957 {
958   \__spinttent_spmoney_render_int:
959   \__spinttent_spmoney_print_part_dec:
960 }
961 \cs_new_protected:Nn \__spinttent_spmoney_render_only_integer_node:
962 {
963   \__spinttent_spmoney_render_int:
964 }

```

(End of definition for `\__spinttent_spmoney_render_number_node:` and `\__spinttent_spmoney_render_only_integer_node:`.)

`\__spinttent_spmoney_mathml:`

The function `\__spinttent_spmoney_mathml:` checks if there is an explicit sign in `\l__spinttent_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.

```

965 \cs_new_protected:Nn \__spinttent_spmoney_mathml:

```

```

966 {
967   \tl_if_empty:NTF \l__spintent_luaset_sign_tl
968   {
969     \bool_if:NTF \l__spintent_spmoney_sub_units_bool
970     { \l__spintent_spmoney_set_and_print_sub_units: }
971     { \l__spintent_spmoney_intent_int: }
972   }
973   {
974     \str_if_eq:VnTF \l__spintent_luaset_sign_tl { + }
975     {
976       \str_set:Nn \l__spintent_spmoney_intent_sign_str { _más:prefix ($n) }
977     }
978     {
979       \str_set:Nn \l__spintent_spmoney_intent_sign_str { _menos:prefix ($n) }
980     }
981     \l__spintent_invisible_sep:
982     \MathMLintent { \l__spintent_spmoney_intent_sign_str }
983     {
984       \l__spintent_luaset_sign_tl
985       \MathMLarg { n }
986       {
987         \bool_if:NTF \l__spintent_spmoney_sub_units_bool
988         { \l__spintent_spmoney_set_and_print_sub_units: }
989         { \l__spintent_spmoney_intent_int: }
990       }
991     }
992   }
993 }

```

(End of definition for `\l__spintent_spmoney_mathml:`)

`\l__spintent_spmoney_intent_int:` The function `\l__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 `\l__spintent_spmoney_render_layout:`.

```

994 \cs_new_protected:Nn \l__spintent_spmoney_intent_int:
995 {
996   \tl_if_empty:NTF \l__spintent_luaset_dec_sep_tl
997   {
998     \str_if_eq:VnTF \l__spintent_luaset_only_part_int_str { 1 }
999     {
1000       \str_set:NV \l__spintent_spmoney_intent_money_str
1001       \l__spintent_spmoney_luaset_sing_str
1002     }
1003     {
1004       \str_set:NV \l__spintent_spmoney_intent_money_str
1005       \l__spintent_spmoney_luaset_plur_str
1006     }
1007     \str_if_eq:VnT \l__spintent_luaset_millions_str { true }
1008     {
1009       \str_set:NV \l__spintent_spmoney_intent_money_str
1010       \l__spintent_spmoney_luaset_conde_str
1011     }
1012   }
1013   {
1014     \str_set:NV \l__spintent_spmoney_intent_money_str
1015     \l__spintent_spmoney_luaset_plur_str
1016     \str_if_eq:VnT \l__spintent_luaset_millions_str { true }
1017     {
1018       \str_if_eq:VnTF \l__spintent_luaset_dec_is_all_zeros_str { true }
1019       {
1020         \str_set:NV \l__spintent_spmoney_intent_money_str
1021         \l__spintent_spmoney_luaset_conde_str
1022       }
1023       {
1024         \str_set:NV \l__spintent_spmoney_intent_money_str
1025         \l__spintent_spmoney_luaset_plur_str
1026       }
1027     }
1028   }
1029   \l__spintent_invisible_sep:
1030   \MathMLintent { _ \l__spintent_spmoney_intent_money_str :postfix($n) }

```

```

1031     {
1032         \__spintent_spmoney_render_layout:
1033     }
1034 }

```

(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.

```

1035 \cs_new_protected:Nn \__spintent_spmoney_check_sub_units:
1036 {
1037     \str_if_empty:NT \__spintent_spmoney_luaset_subsing_str
1038     {
1039         \msg_warning:nne { spintent } { spmoney-no-subunits }
1040         { \__spintent_spmoney_luaset_currency_arg_str }
1041         \bool_set_false:N \__spintent_spmoney_sub_units_bool
1042     }
1043     \tl_if_empty:NT \__spintent_luaset_dec_sep_tl
1044     { \bool_set_false:N \__spintent_spmoney_sub_units_bool }
1045 }
1046 \cs_new_protected:Nn \__spintent_spmoney_set_and_print_sub_units:
1047 {
1048     \int_compare:nNnTF { \str_count:N \__spintent_luaset_only_part_dec_str } = { 1 }
1049     {
1050         \int_set:Nn \__spintent_spmoney_subnum_int
1051         { \__spintent_luaset_only_part_dec_str * 10 }
1052         \str_set:Ne \__spintent_luaset_only_part_dec_str
1053         { \__spintent_luaset_only_part_dec_str 0 }
1054     }
1055     {
1056         \int_set:Nn \__spintent_spmoney_subnum_int
1057         { \__spintent_luaset_only_part_dec_str }
1058     }
1059     \int_compare:nNnTF { \__spintent_spmoney_subnum_int } = { 1 }
1060     {
1061         \str_set:Ne \__spintent_spmoney_intent_sub_unit_str
1062         { _ \__spintent_spmoney_luaset_subsing_str :postfix($sub) }
1063     }
1064     {
1065         \str_set:Ne \__spintent_spmoney_intent_sub_unit_str
1066         { _ \__spintent_spmoney_luaset_subplur_str :postfix($sub) }
1067     }
1068     \str_if_eq:VnTF \__spintent_luaset_only_part_int_str { 1 }
1069     {
1070         \str_set:NV \__spintent_spmoney_res_main_voice_str
1071         \__spintent_spmoney_luaset_sing_str
1072     }
1073     {
1074         \str_set:NV \__spintent_spmoney_res_main_voice_str
1075         \__spintent_spmoney_luaset_plur_str
1076     }
1077     \str_if_eq:VnT \__spintent_luaset_millions_str { true }
1078     {
1079         \str_set:NV \__spintent_spmoney_res_main_voice_str
1080         \__spintent_spmoney_luaset_conde_str
1081     }
1082     \__spintent_spmoney_build_tree:
1083 }

```

(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.

```

1084 \cs_new_protected:Nn \__spintent_spmoney_build_tree:
1085 {
1086   \str_if_eq:VnTF \l__spintent_spmoney_luastr_position_str { pre }
1087   {
1088     \MathMLintent { _ \l__spintent_spmoney_res_main_voice_str :postfix($n) }
1089     {
1090       \__spintent_spmoney_print_symbol:
1091       \MathMLarg { n } { \__spintent_spmoney_render_only_integer_node: }
1092     }
1093     \__spintent_spmoney_print_sep_con:
1094     \__spintent_spmoney_print_dec_sub_unit:
1095   }
1096   {
1097     \MathMLintent { _ \l__spintent_spmoney_res_main_voice_str :postfix($n) }
1098     {
1099       \__spintent_inner_sep:
1100       \MathMLarg { n } { \__spintent_spmoney_render_only_integer_node: }
1101     }
1102     \__spintent_spmoney_print_sep_con:
1103     \__spintent_spmoney_print_dec_sub_unit:
1104     \MathMLintent { :silent } { \__spintent_spmoney_print_symbol: }
1105   }
1106 }
1107 \cs_new_protected:Nn \__spintent_spmoney_print_sep_con:
1108 {
1109   \MathMLintent { _con }
1110   {
1111     \mathord { \l__spintent_luastr_dec_sep_tl }
1112   }
1113 }
1114 \cs_new_protected:Nn \__spintent_spmoney_print_dec_sub_unit:
1115 {
1116   \MathMLintent { \l__spintent_spmoney_intent_sub_unit_str }
1117   {
1118     \__spintent_inner_sep:
1119     \MathMLarg { sub }
1120     {
1121       \__spintent_spmoney_render_dec:
1122     }
1123   }
1124 }

```

(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 $\langle \textit{argument} \rangle$ passed in ‘#1’ using Lua module to extract the numeric value and any “currency” identifier. If a “currency” is found in `\l__spintent_spmoney_luastr_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.

```

1125 \cs_new_protected:Nn \__spintent_spmoney_parse_and_route:n
1126 {
1127   \__spintent_luafun_spmoney_prepare_input:n { \exp_not:n {#1} }
1128   \__spintent_luafun_clean_split_and_set:n
1129   { \exp_not:V \l__spintent_spmoney_luastr_extracted_num_tl }
1130   \str_if_empty:NF \l__spintent_spmoney_luastr_extracted_curr_str
1131   {
1132     \str_set:Nc \l__spintent_luastr_has_units_str { true }
1133     \str_set:Nc \l__spintent_luastr_arg_above_tl
1134     \l__spintent_spmoney_luastr_extracted_curr_str
1135   }
1136   \__spintent_spmoney_validate_money:
1137   \bool_if:NT \l__spintent_spmoney_sub_units_bool
1138   { \__spintent_spmoney_check_sub_units: }
1139   \__spintent_spmoney_mathml:
1140 }

```

(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.

```

1141 \NewDocumentCommand \spmoney { 0{} m }
1142 {
1143   \mode_if_math:TF
1144   {
1145     \group_begin:
1146     \keys_set:nn { spintent / spmoney } {#1}
1147     \__spintent_spmoney_parse_and_route:n {#2}
1148     \group_end:
1149   }
1150   { \msg_error:nnn { spintent } { need-math-mode } { \spmoney } }
1151 }

```

(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`

`\__spintent_spshort_luaset_status_str` These variables store the values returned by the Lua module `spintent.lua`. The variable `\__spintent_spshort_luaset_status_str` stores success, fallback or *error*, and `\__spintent_spshort_luaset_layout_str` stores the way the abbreviation is typeset. The variable `\__spintent_spshort_luaset_expand_str` stores the text read, which is also the one used by the key `E`, and `\__spintent_spshort_luaset_output_str` stores the text that is printed. For ordinals, `\__spintent_spshort_luaset_base_str` stores the number and `\__spintent_spshort_luaset_suffix_str` stores the suffix.

```

1152 \str_new:N \__spintent_spshort_luaset_status_str
1153 \str_new:N \__spintent_spshort_luaset_layout_str
1154 \str_new:N \__spintent_spshort_luaset_expand_str
1155 \str_new:N \__spintent_spshort_luaset_output_str
1156 \str_new:N \__spintent_spshort_luaset_base_str
1157 \str_new:N \__spintent_spshort_luaset_suffix_str

```

(End of definition for `\__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.

```

1158 \msg_new:nnnn { spintent } { spshort-unknown-abbreviation }
1159 {
1160   The~abbreviation~or~ordinal~'#1'~is~invalid~or~not~recognized~\msg_line_context:.
1161 }
1162 {
1163   Check~the~spelling~or~the~grammar~rules~for~ordinals.
1164 }

```

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

15.19.3 Definition of keys for `\spshort`

`small-caps` We define the *(keys)* `small-caps` and `read-arg`.

```

1165 \keys_define:nn { spintent / spshort }
1166 {
1167   small-caps .bool_set:N = \__spintent_spshort_smallcaps_bool,
1168   small-caps .initial:n = false,
1169   small-caps .value_required:n = true,
1170   read-arg .str_set:N = \__spintent_spshort_read_arg_str,
1171   read-arg .initial:n = {},
1172   read-arg .value_required:n = true,

```

```

1173     unknown     .code:n =
1174     {
1175         \str_set:Nn \l__spintent_command_name_str { spshort }
1176         \__spintent_unknown_keys_cmd:n {#1}
1177     },
1178 }

```

(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 $\langle argument \rangle$ 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’.

```

1179 \cs_new_protected:Nn \__spintent_spshort_tag_span:nn
1180 {
1181     \tag_mc_end_push:
1182     \tag_struct_begin:n { tag=Span, E={#1} }
1183     \tag_mc_begin:n { tag=Span }
1184     \tag_suspend:n { \spshort }
1185     #2
1186     \tag_resume:n { \spshort }
1187     \tag_mc_end:
1188     \tag_struct_end:
1189     \tag_mc_begin_pop:n {}
1190 }
1191 \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 $\langle keys \rangle$ in ‘#1’. If that string variable is still empty, it calls `\__spintent_spshort_exec:n` with the $\langle argument \rangle$ in ‘#2’. Otherwise, it calls `\__spintent_spshort_render_bypass:n`.

```

1192 \cs_new_protected:Nn \__spintent_spshort_print:nn
1193 {
1194     \str_clear:N \l__spintent_spshort_read_arg_str
1195     \keys_set:nn { spintent / spshort } {#1}
1196     \str_if_empty:NTF \l__spintent_spshort_read_arg_str
1197     {
1198         \__spintent_spshort_exec:n {#2}
1199     }
1200     {
1201         \__spintent_spshort_render_bypass:n {#2}
1202     }
1203 }

```

(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**.

```

1204 \cs_new_protected:Nn \__spintent_spshort_exec:n
1205 {
1206     \__spintent_luafun_spshort_process:n {#1}
1207     \str_case:VnF \l__spintent_spshort_luaset_status_str
1208     {
1209         { success } { \__spintent_spshort_render_success: }
1210         { fallback } { \__spintent_spshort_render_fallback: }
1211         { error } { \__spintent_spshort_render_error:n {#1} }
1212     }
1213     { \__spintent_spshort_render_error:n {#1} }
1214 }

```

(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.

```

1215 \cs_new_protected:Nn \__spintent_spshort_render_success:
1216 {
1217   \__spintent_spshort_tag_span:en
1218   { \l__spintent_spshort_luaset_expand_str }
1219   { \l__spintent_spshort_execute_layout: }
1220 }
1221 \cs_new_protected:Nn \__spintent_spshort_render_fallback:
1222 {
1223   \__spintent_spshort_tag_span:en
1224   { \l__spintent_spshort_luaset_expand_str }
1225   {
1226     \bool_if:NTF \l__spintent_spshort_smallcaps_bool
1227     { \textsc{ \l__spintent_spshort_luaset_output_str } }
1228     { \l__spintent_spshort_luaset_output_str }
1229   }
1230 }
1231 \cs_new_protected:Nn \__spintent_spshort_render_bypass:n
1232 {
1233   \__spintent_spshort_tag_span:en { \l__spintent_spshort_read_arg_str } {#1}
1234 }
1235 \cs_new_protected:Nn \__spintent_spshort_render_error:n
1236 {
1237   \msg_error:nnn { spintent } { spshort-unknown-abbreviation } {#1}
1238 }

```

(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.

```

1239 \cs_new_protected:Nn \__spintent_spshort_execute_layout:
1240 {
1241   \str_case:VnF \l__spintent_spshort_luaset_layout_str
1242   {
1243     { linear_regular }
1244     {
1245       \bool_if:NTF \l__spintent_spshort_smallcaps_bool
1246       { \textsc{ \l__spintent_spshort_luaset_output_str } }
1247       { \l__spintent_spshort_luaset_output_str }
1248     }
1249     { linear_caps }
1250     { \l__spintent_spshort_luaset_output_str }
1251     { linear_mixed }
1252     { \l__spintent_spshort_luaset_output_str }
1253     { superscript }
1254     {
1255       \l__spintent_spshort_luaset_base_str
1256       \bool_if:NTF \l__spintent_spshort_smallcaps_bool
1257       {
1258         \textsuperscript{
1259           \textsc{
1260             \str_lowercase:V \l__spintent_spshort_luaset_suffix_str
1261           }
1262         }
1263       }
1264       {
1265         \textsuperscript{ \l__spintent_spshort_luaset_suffix_str }
1266       }
1267     }
1268     { small_caps_pure }
1269     {
1270       \bool_if:NTF \l__spintent_spshort_smallcaps_bool
1271       {
1272         \textsc{
1273           \str_lowercase:V \l__spintent_spshort_luaset_output_str
1274         }

```

```

1275         }
1276         { \l__spintent_spsshort_luaset_output_str }
1277     }
1278     { acronym_long }
1279     { \l__spintent_spsshort_luaset_output_str }
1280 }
1281 { \l__spintent_spsshort_luaset_output_str }
1282 }

```

(End of definition for \l__spintent_spsshort_execute_layout:.)

\spsshort The public command **\spsshort** 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 **\l__spintent_spsshort_print:nn**.

```

1283 \NewDocumentCommand \spsshort { 0{} m }
1284 {
1285     \mode_if_math:TF
1286     {
1287         \msg_error:nnn { spintent } { need-text-mode } { \spsshort }
1288     }
1289     {
1290         \mode_leave_vertical:
1291         \group_begin:
1292         \l__spintent_spsshort_print:nn {#1} {#2}
1293         \group_end:
1294     }
1295 }

```

(End of definition for \spsshort. 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 $\langle argument \rangle$ 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

The variable **\l__spintent_spsiglo_luaset_error_str** stores **true** or **false** if the Lua module **spintent.lua** could not read the $\langle argument \rangle$. 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_str** 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**.

```

1296 \str_new:N \l__spintent_spsiglo_luaset_error_str
1297 \str_new:N \l__spintent_spsiglo_luaset_arabic_str
1298 \str_new:N \l__spintent_spsiglo_luaset_roman_min_str
1299 \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 $\langle argument \rangle$ to an integer within the valid range of **1** to **4000**, preventing incorrect speech synthesis.

```

1300 \msg_new:nnnn { spintent } { spsiglo-invalid-arg }
1301 {
1302     Invalid~argument~'#2'~for~'\tl_trim_spaces:n {#1}'~\msg_line_context:.
1303 }
1304 {
1305     The~argument~must~be~an~Arabic~or~Roman~numeral~between~'1'~and~'4000'.
1306 }

```

(End of definition for spsiglo-invalid-arg.)

15.20.3 Definition of keys for \spsiglo

print-era We define the *(key)* **print-era**.

unknown

```

1307 \keys_define:nn { spintent / spsiglo }
1308 {

```

```

1309     print-era .choices:nn =
1310     { none , ac , dc }
1311     {
1312         \int_case:nn { \l_keys_choice_int }
1313         {
1314             {1} { \str_set:Nn \l__spintent_spsiglo_print_era_str { none } }
1315             {2} { \str_set:Nn \l__spintent_spsiglo_print_era_str { ac } }
1316             {3} { \str_set:Nn \l__spintent_spsiglo_print_era_str { dc } }
1317         }
1318     },
1319     print-era .initial:n = none,
1320     print-era .value_required:n = true,
1321     print-era / unknown .code:n =
1322         \msg_error:nnee { spintent } { unknown-choice }
1323         { print-era } { none, ~ac, ~dc } { \exp_not:n {#1} },
1324     unknown .code:n =
1325     {
1326         \str_set:Nn \l__spintent_command_name_str { spsiglo }
1327         \__spintent_unknown_keys_cmd:n {#1}
1328     },
1329 }

```

(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.

```

1330 \cs_new_protected:Nn \__spintent_spsiglo_render_simple:nn
1331 {
1332     \MathMLint { #1:number } { \text { \textsc {#2} } }
1333 }
1334 \cs_generate_variant:Nn \__spintent_spsiglo_render_simple:nn { VV }
1335 \cs_new_protected:Nn \__spintent_spsiglo_render_with_era:nnn
1336 {
1337     \str_if_eq:nnTF {#1} { ac }
1338     {
1339         \MathMLint { _antes-de-cristo:postfix($z) }
1340         {
1341             \__spintent_inner_sep:
1342             \MathMLarg { z }
1343             {
1344                 \MathMLint { #2:number }
1345                 {
1346                     \text { \textsc {#3}~a.~C. }
1347                 }
1348             }
1349         }
1350     }
1351     {
1352         \MathMLint { _después-de-cristo:postfix($z) }
1353         {
1354             \__spintent_inner_sep:
1355             \MathMLarg { z }
1356             {
1357                 \MathMLint { #2:number }
1358                 {
1359                     \text { \textsc {#3}~d.~C. }
1360                 }
1361             }
1362         }
1363     }
1364 }
1365 \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.

```

1366 \cs_new_protected:Nn \__spintent_spsiglo_tag_span:nn
1367 {
1368     \tag_mc_end_push:
1369     \tag_struct_begin:n { tag=Span, alt={#1} }
1370     \tag_mc_begin:n { tag=Span }
1371     \tag_suspend:n { \spsiglo }
1372     #2
1373     \tag_resume:n { \spsiglo }
1374     \tag_mc_end:
1375     \tag_struct_end:
1376     \tag_mc_begin_pop:n {}
1377 }
1378 \cs_generate_variant:Nn \__spintent_spsiglo_tag_span:nn { en }

```

(End of definition for __spintent_spsiglo_tag_span:nn.)

__spintent_spsiglo_exec:n 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 $\langle argument \rangle$ 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.

```

1379 \cs_new_protected:Nn \__spintent_spsiglo_exec:n
1380 {
1381     \mode_if_math:F { \mode_leave_vertical: }
1382     \str_if_eq:VnTF \l__spintent_spsiglo_luaset_error_str { true }
1383     {
1384         \msg_error:nnnn { spintent } { spsiglo-invalid-arg }
1385         { \spsiglo } { #1 }
1386     }
1387     {
1388         \mode_if_math:TF
1389         { \__spintent_spsiglo_mathml: }
1390         { \__spintent_spsiglo_text: }
1391     }
1392 }
1393 \cs_new_protected:Nn \__spintent_spsiglo_mathml:
1394 {
1395     \str_if_eq:VnTF \l__spintent_spsiglo_print_era_str { none }
1396     {
1397         \__spintent_spsiglo_render_simple:VV
1398         \l__spintent_spsiglo_luaset_arabic_str
1399         \l__spintent_spsiglo_luaset_roman_min_str
1400     }
1401     {
1402         \__spintent_spsiglo_render_with_era:VVV
1403         \l__spintent_spsiglo_print_era_str
1404         \l__spintent_spsiglo_luaset_arabic_str
1405         \l__spintent_spsiglo_luaset_roman_min_str
1406     }
1407 }
1408 \cs_new_protected:Nn \__spintent_spsiglo_text:
1409 {
1410     \str_case:Vn \l__spintent_spsiglo_print_era_str
1411     {
1412         { none }
1413         {
1414             \__spintent_spsiglo_tag_span:en
1415             { \l__spintent_spsiglo_luaset_arabic_str }
1416             { \textsc { \l__spintent_spsiglo_luaset_roman_min_str } }
1417         }
1418         { ac }
1419         {
1420             \__spintent_spsiglo_tag_span:en
1421             { \l__spintent_spsiglo_luaset_arabic_str ~ antes ~ de ~ cristo }
1422             { \textsc { \l__spintent_spsiglo_luaset_roman_min_str } ~ a.~C. }
1423         }
1424         { dc }

```

```

1425     {
1426         \__spintent_spsiglo_tag_span:en
1427         { \__spintent_spsiglo_luaset_arabic_str ~ después ~ de ~ cristo }
1428         { \textsc { \__spintent_spsiglo_luaset_roman_min_str } ~ d.~C. }
1429     }
1430 }
1431 }

```

(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*.

```

1432 \NewDocumentCommand \spsiglo { O{} m }
1433 {
1434     \group_begin:
1435     \keys_set:nn { spintent / spsiglo } {#1}
1436     \__spintent_luafun_spsiglo_parse:n {#2}
1437     \__spintent_spsiglo_exec:n {#2}
1438     \group_end:
1439 }

```

(End of definition for `\spsiglo`. This function is documented on page 15.)

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`

`\__spintent_sptime_luaset_seconds_str` These variables store the values returned by the Lua module `spintent.lua`. The variable `\__spintent_sptime_luaset_error_str` stores *true* or *false* if the *⟨argument⟩* is invalid. The variable `\__spintent_sptime_luaset_base_str` stores the hours and minutes, and `\__spintent_sptime_luaset_seconds_str` stores the seconds, which is empty if there are none. The variables `\__spintent_sptime_luaset_has_seconds_str` and `\__spintent_sptime_luaset_is_fraction_str` store *true* or *false* if the time has seconds and if the seconds have a decimal fraction.

```

1440 \str_new:N \__spintent_sptime_luaset_seconds_str
1441 \str_new:N \__spintent_sptime_luaset_has_seconds_str
1442 \str_new:N \__spintent_sptime_luaset_error_str
1443 \str_new:N \__spintent_sptime_luaset_is_fraction_str
1444 \str_new:N \__spintent_sptime_luaset_base_str

```

(End of definition for `\__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.

```

1445 \msg_new:nnnn { spintent } { sptime-invalid-arg }
1446 {
1447     Invalid~argument~'~#2'~for~'\tl_trim_spaces:n {#1}'~\msg_line_context:.
1448 }
1449 {
1450     Allowed~formats:~'HH:MM',~'HH:MM:SS'~or~'HH:MM:SS.ss'~.~
1451     Hours~must~be~'0-23'~and~minutes/seconds~'0-59'.
1452 }

```

(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.

```

1453 \cs_new_protected:Nn \__spintent_sptime_build_seconds:nn
1454 {

```

```

1455 \MathMLintent { #1:prefix($z) }
1456 {
1457   \text { : }
1458   \MathMLarg { z }
1459   {
1460     \MathMLintent { _segundos:postfix($m) }
1461     {
1462       \__spintent_inner_sep:
1463       \MathMLarg { m }
1464       {
1465         \text { #2 }
1466       }
1467     }
1468   }
1469 }
1470 }

```

(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 `\__spintent_sptime_luaset_has_seconds_str` and `\__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.

```

1471 \cs_new_protected:Nn \__spintent_sptime_mathml:
1472 {
1473   \str_if_eq:VnTF \__spintent_sptime_luaset_has_seconds_str { true }
1474   {
1475     \MathMLintent { :time }
1476     {
1477       \text { \__spintent_sptime_luaset_base_str }
1478     }
1479     \str_if_eq:VnTF \__spintent_sptime_luaset_is_fraction_str { true }
1480     {
1481       \__spintent_sptime_build_seconds:nn { con }
1482       { \__spintent_sptime_luaset_seconds_str }
1483     }
1484     {
1485       \__spintent_sptime_build_seconds:nn { _minutos-con }
1486       { \__spintent_sptime_luaset_seconds_str }
1487     }
1488   }
1489   {
1490     \MathMLintent { :time }
1491     {
1492       \text { \__spintent_sptime_luaset_base_str }
1493     }
1494     \str_if_eq:VnTF \__spintent_sptime_luaset_is_fraction_str { false }
1495     {
1496       \MathMLintent { _minutos }
1497       {
1498         \c__spintent_invisible_sep_int
1499       }
1500     }
1501   }
1502 }

```

(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 `\__spintent_sptime_luaset_error_str`. If an invalid format is detected, it issues the `sptime-invalid-arg` error; otherwise, it calls `\__spintent_sptime_mathml:` to assemble the **MathML** output.

```

1503 \NewDocumentCommand \sptime { m }
1504 {
1505   \mode_if_math:TF
1506   {
1507     \group_begin:

```

```

1508     \__spinttent_luafun_sptime_parse:n {#1}
1509     \str_if_eq:VnTF \l__spinttent_sptime_luaset_error_str { true }
1510     {
1511         \msg_error:nnnn { spinttent } { sptime-invalid-arg }
1512         { \sptime } {#1}
1513     }
1514     {
1515         \__spinttent_sptime_mathml:
1516     }
1517     \group_end:
1518 }
1519 { \msg_error:nnn { spinttent } { need-math-mode } { \sptime } }
1520 }

```

(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 `\__spinttent_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__spinttent_spdate_luaset_output_str`
`\l__spinttent_spdate_luaset_error_str`

These variables store the values returned by the Lua module `spinttent.lua`. The variable `\l__spinttent_spdate_luaset_output_str` stores the date already formatted, and `\l__spinttent_spdate_luaset_error_str` stores `true` or `false` if the $\langle argument \rangle$ is invalid.

```

1521 \str_new:N \l__spinttent_spdate_luaset_output_str
1522 \str_new:N \l__spinttent_spdate_luaset_error_str

```

(End of definition for `\l__spinttent_spdate_luaset_output_str` and `\l__spinttent_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.

```

1523 \msg_new:nnnn { spinttent } { spdate-invalid-arg }
1524 {
1525     Invalid~argument~'#2'~for~'\tl_trim_spaces:n {#1}'~'\msg_line_context:.
1526 }
1527 {
1528     Use~'DD/MM/YYYY',~'DD-MM-YYYY'~or~'YYYY-MM-DD',~with~a~valid~day~
1529     and~month.
1530 }

```

(End of definition for `spdate-invalid-arg`.)

15.22.3 Internal functions for `\spdate`

`\__spinttent_spdate_exec:n`

The internal function `\__spinttent_spdate_exec:n` parses the input using `\__spinttent_luafun_spdate_parse:n` and checks the error status in `\l__spinttent_spdate_luaset_error_str`. If an invalid format is detected, it issues the `spdate-invalid-arg` error. Otherwise, it calls `\__spinttent_spdate_mathml:`.

```

1531 \cs_new_protected:Nn \__spinttent_spdate_exec:n
1532 {
1533     \__spinttent_luafun_spdate_parse:n {#1}
1534     \str_if_eq:VnTF \l__spinttent_spdate_luaset_error_str { true }
1535     {
1536         \msg_error:nnnn { spinttent } { spdate-invalid-arg }
1537         { \spdate } {#1}
1538     }
1539     { \__spinttent_spdate_mathml: }
1540 }

```

(End of definition for `\__spinttent_spdate_exec:n`.)

`\__spinttent_spdate_mathml:`

The internal function `\__spinttent_spdate_mathml:` builds the **MathML intent** structure using the formatted date stored in `\l__spinttent_spdate_luaset_output_str`.

```

1541 \cs_new_protected:Nn \__spinttent_spdate_mathml:
1542 {
1543     \MathMLintent { :date }

```

```

1544     {
1545         \text { \l__spinttent_spdate_luaset_output_str }
1546     }
1547 }

```

(End of definition for `\l__spinttent_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 `\l__spinttent_spdate_exec:n` to process the input.

```

1548 \NewDocumentCommand \spdate { m }
1549 {
1550     \mode_if_math:TF
1551     {
1552         \group_begin:
1553         \l__spinttent_spdate_exec:n {#1}
1554         \group_end:
1555     }
1556     { \msg_error:nnn { spinttent } { need-math-mode } { \spdate } }
1557 }

```

(End of definition for `\spdate`. This function is documented on page 16.)

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`

The variable `\l__spinttent_spusep_arg_tl` stores the *{argument}*, the variable `\l__spinttent_spusep_left_tl` and `\l__spinttent_spusep_right_tl` stores the parentheses.

```

1558 \tl_new:N \l__spinttent_spusep_arg_tl
1559 \tl_new:N \l__spinttent_spusep_left_tl
1560 \tl_new:N \l__spinttent_spusep_right_tl

```

(End of definition for `\l__spinttent_spusep_arg_tl`, `\l__spinttent_spusep_left_tl`, and `\l__spinttent_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.

```

1561 \msg_new:nnnn { spinttent } { spusep-empty-arg }
1562 {
1563     Empty~argument~for~'\token_to_str:N \spusep'\~\msg_line_context:.
1564 }
1565 {
1566     A~non-empty~argument~is~required.
1567 }
1568 \msg_new:nnnn { spinttent } { spusep-invalid-arg }
1569 {
1570     Invalid~argument~'#1'~for~'\token_to_str:N \spusep'\~\msg_line_context:.
1571 }
1572 {
1573     Only~the~delimiters~listed~in~the~documentation~of~'\token_to_str:N \spusep'\~are~
1574     accepted.
1575 }

```

(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`.

```

1576 \keys_define:nn { spinttent / spusep }
1577 {
1578     size .choices:nn =
1579     { none , big , Big, bigg, Bigg }
1580     {
1581         \int_case:nn { \l_keys_choice_int }
1582         {

```

```

1583         { 1 }
1584         {
1585             \tl_clear:N \l__spinttent_spusep_left_tl
1586             \tl_clear:N \l__spinttent_spusep_right_tl
1587         }
1588         { 2 }
1589         {
1590             \tl_set:Nn \l__spinttent_spusep_left_tl { \bigl }
1591             \tl_set:Nn \l__spinttent_spusep_right_tl { \bigr }
1592         }
1593         { 3 }
1594         {
1595             \tl_set:Nn \l__spinttent_spusep_left_tl { \Bigl }
1596             \tl_set:Nn \l__spinttent_spusep_right_tl { \Bigr }
1597         }
1598         { 4 }
1599         {
1600             \tl_set:Nn \l__spinttent_spusep_left_tl { \biggl }
1601             \tl_set:Nn \l__spinttent_spusep_right_tl { \biggr }
1602         }
1603         { 5 }
1604         {
1605             \tl_set:Nn \l__spinttent_spusep_left_tl { \Biggl }
1606             \tl_set:Nn \l__spinttent_spusep_right_tl { \Biggr }
1607         }
1608     }
1609 },
1610 size .initial:n = none,
1611 size .value_required:n = true,
1612 size / unknown .code:n =
1613     \msg_error:nneee { spinttent } { unknown-choice }
1614     { size } { none,~big,~Big,~bigg,~Bigg } { \exp_not:n {#1} },
1615 silent .bool_set:N = \l__spinttent_spusep_silent_bool,
1616 silent .initial:n = false,
1617 silent .value_required:n = true,
1618 unknown .code:n =
1619     {
1620         \str_set:Nn \l__spinttent_command_name_str { spusep }
1621         \__spinttent_unknown_keys_cmd:n {#1}
1622     },
1623 }

```

(End of definition for `size`, `silent`, and `unknown`.)

15.23.4 Internal functions for `\spusep`

`\__spinttent_spusep_mathml:n`

The function `\__spinttent_spusep_mathml:n` will place the argument passed in ‘#2’ to the right of the variables `\l__spinttent_spusep_left_tl` and `\l__spinttent_spusep_right_tl` and then check the state of the variable `\l__spinttent_spusep_silent_bool` handled by the key `silent` to print.

```

1624 \cs_new_protected:Nn \__spinttent_spusep_mathml:n
1625 {
1626     \tl_put_right:cn { \l__spinttent_spusep_ #1 _tl } {#2}
1627     \bool_if:NTF \l__spinttent_spusep_silent_bool
1628     {
1629         \MathMLintent { :silent } { \tl_use:c { \l__spinttent_spusep_ #1 _tl } }
1630     }
1631     {
1632         \tl_use:c { \l__spinttent_spusep_ #1 _tl }
1633     }
1634 }

```

(End of definition for `\__spinttent_spusep_mathml:n`.)

`\__spinttent_spusep_exec:n`

The function `\__spinttent_spusep_exec:n` captures and stores the $\{\langle argument \rangle\}$ passed in ‘#1’ in the variable `\l__spinttent_spusep_arg_tl`, validates the cases by converting them to the macros provided by `unicode-math`, and passes them to the function `\__spinttent_spusep_mathml:n` for printing. If the input is not supported or is empty, we return an “error”.

```

1635 \cs_new_protected:Nn \__spinttent_spusep_exec:n
1636 {
1637     \tl_set:Ne \l__spinttent_spusep_arg_tl { \tl_trim_spaces:n {#1} }

```

```

1638 \tl_if_empty:NT \l__spinttent_spusep_arg_tl
1639 {
1640   \msg_error:nn { spinttent } { spusep-empty-arg }
1641 }
1642 \str_case:VnF \l__spinttent_spusep_arg_tl
1643 {
1644   { ( }
1645   {
1646     \__spinttent_spusep_mathml:nn { left } { \lparen }
1647   }
1648   { ) }
1649   {
1650     \__spinttent_spusep_mathml:nn { right } { \rparen }
1651   }
1652   { [ }
1653   {
1654     \__spinttent_spusep_mathml:nn { left } { \lbrack }
1655   }
1656   { ] }
1657   {
1658     \__spinttent_spusep_mathml:nn { right } { \rbrack }
1659   }
1660   { \{ }
1661   {
1662     \__spinttent_spusep_mathml:nn { left } { \lbrace }
1663   }
1664   { \} }
1665   {
1666     \__spinttent_spusep_mathml:nn { right } { \rbrace }
1667   }
1668 }
1669 {
1670   \tl_if_empty:NF \l__spinttent_spusep_arg_tl
1671   { \msg_error:nnn { spinttent } { spusep-invalid-arg } {#1} }
1672 }
1673 }

```

(End of definition for `\__spinttent_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 $\langle argument \rangle$ passed in ‘#2’ by means of the function `\__spinttent_spusep_exec:n`.

```

1674 \NewDocumentCommand \spusep { O{} m }
1675 {
1676   \mode_if_math:TF
1677   {
1678     \group_begin:
1679     \keys_set:nn { spinttent / spusep } {#1}
1680     \__spinttent_spusep_exec:n {#2}
1681     \group_end:
1682   }
1683   { \msg_error:nnn { spinttent } { need-math-mode } { \spusep } }
1684 }

```

(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`

The variables `\l__spinttent_spread_luaset_status_str` and `\l__spinttent_spread_luaset_reading_str` are set by `\__spinttent_luafun_spread_new:nn/\__spinttent_luafun_spread_lookup:n`. The variable `\l__spinttent_spread_read_arg_tl` stores `read-arg`.

```

1685 \str_new:N \l__spinttent_spread_luaset_status_str
1686 \str_new:N \l__spinttent_spread_luaset_reading_str
1687 \tl_new:N \l__spinttent_spread_read_arg_tl
1688 \tl_new:N \l__spinttent_spread_read_tmp_tl

```

(End of definition for `\l__spinttent_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`.

```

1689 \msg_new:nnnn { spintent } { spnewread-empty-args }
1690 {
1691   Empty~argument~for~'\token_to_str:N \spnewread'~\msg_line_context:.
1692 }
1693 {
1694   Two~non-empty~arguments~are~required.
1695 }
1696 \msg_new:nnnn { spintent } { spnewread-duplicate }
1697 {
1698   The~symbol~'#1'~is~already~defined~\msg_line_context:.
1699 }
1700 {
1701   Existing~symbols~cannot~be~redefined~with~
1702   '\token_to_str:N \spnewread'.
1703 }
1704 \msg_new:nnnn { spintent } { spread-empty-sym }
1705 {
1706   Empty~argument~for~'\token_to_str:N \spread'~\msg_line_context:.
1707 }
1708 {
1709   A~non-empty~symbol~is~required.
1710 }
1711 \msg_new:nnnn { spintent } { spread-unknown-sym }
1712 {
1713   The~symbol~'#1'~is~not~registered~and~no~'read-arg'~was~given~\msg_line_context:.
1714 }
1715 {
1716   Register~the~symbol~first~with~'\token_to_str:N \spnewread'~or~pass~'read-arg'.
1717 }

```

(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`.

```

1718 \keys_define:nn { spintent / spread }
1719 {
1720   read-arg .code:n =
1721   {
1722     \__spintent_sanitizet_affix:Nn \l__spintent_spread_read_arg_tl {#1}
1723   },
1724   read-arg .initial:n = ,
1725   read-arg .value_required:n = true,
1726   silent .bool_set:N = \l__spintent_spread_silent_bool,
1727   silent .initial:n = false,
1728   silent .value_required:n = true,
1729   unknown .code:n =
1730   {
1731     \str_set:Nn \l__spintent_command_name_str { spread }
1732     \__spintent_unknown_keys_cmd:n {#1}
1733   },
1734 }

```

(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.

```

1735 \cs_new_protected:Nn \__spintent_spread_exec:nn
1736 {

```

```

1737 \keys_set:nn { spintent / spread } {#1}
1738 \tl_if_blank:nTF {#2}
1739 { \msg_error:nn { spintent } { spread-empty-sym } }
1740 {
1741   \bool_if:NTF \l__spintent_spread_silent_bool
1742   { \__spintent_spread_mathml:nn {#2} { - } }
1743   {
1744     \tl_if_empty:NTF \l__spintent_spread_read_arg_tl
1745     {
1746       \__spintent_luafun_spread_lookup:n { \exp_not:n {#2} }
1747       \str_if_eq:VnTF \l__spintent_spread_luaset_status_str { found }
1748       {
1749         \__spintent_spread_mathml:nV {#2}
1750         \l__spintent_spread_luaset_reading_str
1751       }
1752       {
1753         \msg_error:nne { spintent } { spread-unknown-sym }
1754         { \exp_not:n {#2} }
1755       }
1756     }
1757     { \__spintent_spread_mathml:nV {#2} \l__spintent_spread_read_arg_tl }
1758   }
1759 }
1760 }

```

(End of definition for `\__spintent_spread_exec:nn`.)

`\__spintent_spread_mathml:nn` The function `\__spintent_spread_mathml:nn` receives the $\langle symbol \rangle$ 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 ‘_’.

```

1761 \cs_new_protected:Nn \__spintent_spread_mathml:nn
1762 {
1763   \__spintent_invisible_sep:
1764   \MathMLintent { _ #2 } { \__spintent_inner_sep: #1 }
1765 }
1766 \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`.

```

1767 \NewDocumentCommand \spread { O{} m }
1768 {
1769   \mode_if_math:TF
1770   {
1771     \group_begin:
1772     \__spintent_spread_exec:nn {#1} {#2}
1773     \group_end:
1774   }
1775   { \msg_error:nnn { spintent } { need-math-mode } { \spread } }
1776 }

```

(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.

```

1777 \NewDocumentCommand \spnewread { m m }
1778 {
1779   \mode_if_math:TF
1780   {
1781     \msg_error:nnn { spintent } { need-text-mode } { \spnewread }
1782   }
1783   {
1784     \__spintent_sanitize_affix:Nn \l__spintent_spread_read_tmp_tl {#2}
1785     \bool_lazy_or:nnTF

```

```

1786 { \tl_if_blank_p:n {#1} } { \tl_if_empty_p:N \l__spinttent_spread_read_tmp_tl }
1787 { \msg_error:nn { spinttent } { spnewread-empty-args } }
1788 {
1789   \l__spinttent_luafun_spread_new:nn { \exp_not:n {#1} }
1790   { \exp_not:V \l__spinttent_spread_read_tmp_tl }
1791   \str_if_eq:VnT \l__spinttent_spread_luaaset_status_str { duplicate }
1792   {
1793     \msg_error:nnn { spinttent } { spnewread-duplicate } {#1}
1794   }
1795 }
1796 }
1797 }

```

(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`

The variable `\l__spinttent_spdsym_symbol_tl` stores the symbol that is printed, chosen by the key `set-sym`, and `\l__spinttent_spdsym_read_sym_tl` stores the word used in the *intent*, chosen by the key `read-sym`. The variables `\l__spinttent_spdsym_prefix_tl` and `\l__spinttent_spdsym_suffix_tl` store the values of the keys `prefix` and `suffix`.

```

1798 \tl_new:N \l__spinttent_spdsym_symbol_tl
1799 \tl_new:N \l__spinttent_spdsym_read_sym_tl
1800 \tl_new:N \l__spinttent_spdsym_prefix_tl
1801 \tl_new:N \l__spinttent_spdsym_suffix_tl

```

(End of definition for `\l__spinttent_spdsym_symbol_tl` and others.)

15.25.2 Definition of keys for `\spdsym`

We define the *(keys)* `prefix`, `suffix`, `set-sym` and `read-sym` for `\spdsym`.

```

1802 \keys_define:nn { spinttent / spdsym }
1803 {
1804   prefix .code:n =
1805     { \l__spinttent_sanitizize_affix:Nn \l__spinttent_spdsym_prefix_tl {#1} },
1806   prefix .initial:n = ,
1807   prefix .value_required:n = true,
1808   suffix .code:n =
1809     { \l__spinttent_sanitizize_affix:Nn \l__spinttent_spdsym_suffix_tl {#1} },
1810   suffix .initial:n = ,
1811   suffix .value_required:n = true,
1812   set-sym .choices:nn =
1813     { old-style , two-dots , slash }
1814     {
1815       \int_case:nn { \l_keys_choice_int }
1816       {
1817         {1} { \tl_set:Nn \l__spinttent_spdsym_symbol_tl { \div } }
1818         {2} { \tl_set:Nn \l__spinttent_spdsym_symbol_tl { : } }
1819         {3} { \tl_set:Nn \l__spinttent_spdsym_symbol_tl { / } }
1820       }
1821     },
1822   set-sym .initial:n = two-dots,
1823   set-sym .value_required:n = true,
1824   set-sym / unknown .code:n =
1825     \msg_error:nneee { spinttent } { unknown-choice }
1826     { set-sym } { old-style, ~two-dots, ~slash } { \exp_not:n {#1} },
1827   read-sym .choices:nn =
1828     { dividido , dividido-por , dividido-entre }
1829     {
1830       \int_case:nn { \l_keys_choice_int }
1831       {
1832         {1} { \tl_set:Nn \l__spinttent_spdsym_read_sym_tl { dividido } }
1833         {2} { \tl_set:Nn \l__spinttent_spdsym_read_sym_tl { dividido-por } }
1834         {3} { \tl_set:Nn \l__spinttent_spdsym_read_sym_tl { dividido-entre } }
1835       }
1836     },
1837   read-sym .initial:n = dividido,

```

```

1838   read-sym .value_required:n = true,
1839   read-sym / unknown .code:n =
1840     \msg_error:nneee { spintent } { unknown-choice }
1841     { read-sym } { dividido, ~dividido-por, ~dividido-entre }
1842     { \exp_not:n {#1} },
1843   unknown .code:n =
1844     {
1845       \str_set:Nn \l__spintent_command_name_str { spdsym }
1846       \__spintent_unknown_keys_cmd:n {#1}
1847     },
1848 }

```

(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.

```

1849 \cs_new_protected:Nn \__spintent_spdsym_mathml:
1850 {
1851   \MathMLintent
1852   {
1853     -
1854     \tl_if_empty:NF \l__spintent_spdsym_prefix_tl
1855     { \l__spintent_spdsym_prefix_tl - }
1856     \l__spintent_spdsym_read_sym_tl
1857     \tl_if_empty:NF \l__spintent_spdsym_suffix_tl
1858     { - \l__spintent_spdsym_suffix_tl }
1859   }
1860   { \l__spintent_spdsym_symbol_tl }
1861 }

```

(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:`.

```

1862 \NewDocumentCommand \spdsym { O{} }
1863 {
1864   \mode_if_math:TF
1865   {
1866     \group_begin:
1867     \keys_set:nn { spintent / spdsym } {#1}
1868     \__spintent_spdsym_mathml:
1869     \group_end:
1870   }
1871   { \msg_error:nnn { spintent } { need-math-mode } { \spdsym } }
1872 }

```

(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` 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`.

```

1873 \tl_new:N \l__spintent_spmsym_symbol_tl
1874 \tl_new:N \l__spintent_spmsym_read_sym_tl
1875 \tl_new:N \l__spintent_spmsym_prefix_tl
1876 \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

We define the *(keys)* prefix, suffix, set-sym, read-sym and print-sym for \spmsym.

```

1877 \keys_define:nn { spintent / spmsym }
1878 {
1879   prefix      .code:n =
1880   { \__spintent_sanitizé_affix:Nn \l__spintent_spmsym_prefix_tl {#1} },
1881   prefix      .initial:n = ,
1882   prefix      .value_required:n = true,
1883   suffix      .code:n =
1884   { \__spintent_sanitizé_affix:Nn \l__spintent_spmsym_suffix_tl {#1} },
1885   suffix      .initial:n = ,
1886   suffix      .value_required:n = true,
1887   set-sym     .choices:nn =
1888   { cross , dot , asterisk }
1889   {
1890     \int_case:nn { \l_keys_choice_int }
1891     {
1892       {1} { \tl_set:Nn \l__spintent_spmsym_symbol_tl { \times } }
1893       {2} { \tl_set:Nn \l__spintent_spmsym_symbol_tl { \cdot } }
1894       {3} { \tl_set:Nn \l__spintent_spmsym_symbol_tl { \ast } }
1895     }
1896   },
1897   set-sym     .initial:n = dot,
1898   set-sym     .value_required:n = true,
1899   set-sym / unknown .code:n =
1900   \msg_error:nneee { spintent } { unknown-choice }
1901   { set-sym } { cross, ~dot, ~asterisk } { \exp_not:n {#1} },
1902   read-sym    .choices:nn =
1903   { por , multiplicado-por , veces }
1904   {
1905     \int_case:nn { \l_keys_choice_int }
1906     {
1907       {1} { \tl_set:Nn \l__spintent_spmsym_read_sym_tl { por } }
1908       {2} { \tl_set:Nn \l__spintent_spmsym_read_sym_tl { multiplicado-por } }
1909       {3} { \tl_set:Nn \l__spintent_spmsym_read_sym_tl { veces } }
1910     }
1911   },
1912   read-sym    .initial:n = por,
1913   read-sym    .value_required:n = true,
1914   read-sym / unknown .code:n =
1915   \msg_error:nneee { spintent } { unknown-choice }
1916   { read-sym } { por, ~multiplicado-por, ~veces } { \exp_not:n {#1} },
1917   print-sym   .bool_set:N = \l__spintent_spmsym_print_sym_bool,
1918   print-sym   .initial:n = true,
1919   print-sym   .value_required:n = true,
1920   unknown     .code:n =
1921   {
1922     \str_set:Nn \l__spintent_command_name_str { spmsym }
1923     \__spintent_unknown_keys_cmd:n {#1}
1924   },
1925 }
```

(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.

```

1926 \cs_new_protected:Nn \__spintent_spmsym_mathml:
1927 {
1928   \MathMLintent
1929   {
1930     -
1931     \tl_if_empty:NF \l__spintent_spmsym_prefix_tl
1932     { \l__spintent_spmsym_prefix_tl - }
1933     \l__spintent_spmsym_read_sym_tl
1934     \tl_if_empty:NF \l__spintent_spmsym_suffix_tl
1935     { - \l__spintent_spmsym_suffix_tl }
1936   }
```

```

1937     {
1938         \bool_if:NTF \l__spinttent_spmsym_print_sym_bool
1939         { \l__spinttent_spmsym_symbol_tl }
1940         {
1941             \mathopen { \c__spinttent_invisible_sep_int }
1942         }
1943     }
1944 }

```

(End of definition for `\__spinttent_spmsym_mathml:`.)

`\spmsym` Defines the `\spmsym` operator. Similar to `\spdsym`, it takes a single optional argument for local setup and delegates the output to `\__spinttent_spmsym_mathml:`.

```

1945 \NewDocumentCommand \spmsym { 0{} }
1946 {
1947     \mode_if_math:TF
1948     {
1949         \group_begin:
1950         \keys_set:nn { spinttent / spmsym } {#1}
1951         \__spinttent_spmsym_mathml:
1952         \group_end:
1953     }
1954     { \msg_error:nnn { spinttent } { need-math-mode } { \spmsym } }
1955 }

```

(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*.

```

1956 \msg_new:nnnn { spinttent } { spdiv-empty-arg }
1957 {
1958     Empty~argument~for~'\token_to_str:N \spdiv'~\msg_line_context:.
1959 }
1960 {
1961     A~dividend~and~a~divisor~are~required.
1962 }

```

(End of definition for `spdiv-empty-arg`.)

15.27.2 Definition of keys for `\spdiv`

`cifras-sep` We define the `\keys` `cifras-sep`, `read-sign`, `read-period-sep`, `notation-sym`, `set-sym`, `read-sym`, `wrap-parens` and `read-parens`.

```

1963 \keys_define:nn { spinttent / spdiv }
1964 {
1965     cifras-sep      .meta:nn = { spinttent / spnum } { cifras-sep      = {#1} },
1966     read-sign       .meta:nn = { spinttent / spnum } { read-sign       = {#1} },
1967     read-period-sep .meta:nn = { spinttent / spnum } { read-period-sep = {#1} },
1968     notation-sym    .meta:nn = { spinttent / spnum } { notation-sym    = {#1} },
1969     set-sym         .meta:nn = { spinttent / spdsym } { set-sym         = {#1} },
1970     read-sym        .meta:nn = { spinttent / spdsym } { read-sym        = {#1} },
1971     wrap-parens     .bool_set:N = \l__spinttent_spdiv_wrap_parens_bool,
1972     wrap-parens     .initial:n = false,
1973     wrap-parens     .value_required:n = true,
1974     read-parens     .bool_set:N = \l__spinttent_spdiv_read_parens_bool,
1975     read-parens     .initial:n = true,
1976     read-parens     .value_required:n = true,
1977     unknown         .code:n =
1978     {
1979         \str_set:Nn \l__spinttent_command_name_str { spdiv }
1980         \__spinttent_unknown_keys_cmd:n {#1}
1981     },
1982 }

```

(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 $\langle arguments \rangle$ through the `\spnum` command and embeds the semantic operator from `\spdsym`.

```

1983 \cs_new_protected:Nn \__spintent_spdiv_render_op:nn
1984 {
1985   \spnum {#1}
1986   \MathMLintent { _ \__spintent_spdsym_read_sym_tl }
1987   {
1988     \__spintent_spdsym_symbol_tl
1989   }
1990   \spnum {#2}
1991 }
```

(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`.

```

1992 \cs_new_protected:Nn \__spintent_spdiv_mathml:nn
1993 {
1994   \bool_if:NTF \__spintent_spdiv_wrap_parens_bool
1995   {
1996     \bool_if:NTF \__spintent_spdiv_read_parens_bool
1997     {
1998       \lparen \__spintent_spdiv_render_op:nn {#1} {#2} \rparen
1999     }
2000     {
2001       \MathMLintent { :silent } { \lparen }
2002       \__spintent_spdiv_render_op:nn {#1} {#2}
2003       \MathMLintent { :silent } { \rparen }
2004     }
2005   }
2006   { \__spintent_spdiv_render_op:nn {#1} {#2} }
2007 }
```

(End of definition for `\__spintent_spdiv_mathml:nn`.)

`\__spintent_spdiv_exec:nnn`

The function `\__spintent_spdiv_exec:nnn` sets $\langle keys \rangle$, validates $\langle arguments \rangle$, and calls `\__spintent_spdiv_mathml:nn`.

```

2008 \cs_new_protected:Nn \__spintent_spdiv_exec:nnn
2009 {
2010   \keys_set:nn { spintent / spdiv } {#1}
2011   \tl_if_blank:NTF {#2}
2012   { \msg_error:nn { spintent } { spdiv-empty-arg } }
2013   {
2014     \tl_if_blank:NTF {#3}
2015     { \msg_error:nn { spintent } { spdiv-empty-arg } }
2016     {
2017       \__spintent_spdiv_mathml:nn {#2} {#3}
2018     }
2019   }
2020 }
```

(End of definition for `\__spintent_spdiv_exec:nnn`.)

`\spdiv`

Defines the user command `\spdiv`. It requires *math mode*.

```

2021 \NewDocumentCommand \spdiv { 0{ } m m }
2022 {
2023   \mode_if_math:TF
2024   {
2025     \group_begin:
2026     \__spintent_spdiv_exec:nnn {#1} {#2} {#3}
2027     \group_end:
2028   }
2029   { \msg_error:nnn { spintent } { need-math-mode } { \spdiv } }
2030 }
```

(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 $\langle first\ factor \rangle$ and the $\langle second\ factor \rangle$. Both $\langle arguments \rangle$ are internally processed by the `\spnum` command to ensure proper digit separation. This command checks that neither the $\langle first\ factor \rangle$ nor the $\langle second\ factor \rangle$ is *blank*.

15.28.1 Definition of error messages for `\spmul`

The “error” `spmul-empty-arg` is raised when either the $\langle first\ factor \rangle$ or the $\langle second\ factor \rangle$ is missing or *blank*.

```

2031 \msg_new:nnnn { spintent } { spmul-empty-arg }
2032 {
2033   Empty~argument~for~'\token_to_str:N \spmul'\~\msg_line_context:.
2034 }
2035 {
2036   A~first~factor~and~a~second~factor~are~required.
2037 }

```

(End of definition for `spmul-empty-arg`.)

15.28.2 Definition of keys for `\spmul`

We define the $\langle keys \rangle$ `cifras-sep`, `read-sign`, `read-period-sep`, `notation-sym`, `set-sym`, `read-sym`, `wrap-parens` and `read-parens`.

```

cifras-sep
read-sign
read-period-sep
notation-sym
set-sym
read-sym
wrap-parens
read-parens
unknown
2038 \keys_define:nn { spintent / spmul }
2039 {
2040   cifras-sep      .meta:nn = { spintent / spnum } { cifras-sep      = {#1} },
2041   read-sign       .meta:nn = { spintent / spnum } { read-sign       = {#1} },
2042   read-period-sep .meta:nn = { spintent / spnum } { read-period-sep = {#1} },
2043   notation-sym    .meta:nn = { spintent / spnum } { notation-sym    = {#1} },
2044   set-sym         .meta:nn = { spintent / spmsym } { set-sym         = {#1} },
2045   read-sym        .meta:nn = { spintent / spmsym } { read-sym        = {#1} },
2046   wrap-parens     .bool_set:N = \l__spintent_spmul_wrap_parens_bool,
2047   wrap-parens     .initial:n = false,
2048   wrap-parens     .value_required:n = true,
2049   read-parens     .bool_set:N = \l__spintent_spmul_read_parens_bool,
2050   read-parens     .initial:n = true,
2051   read-parens     .value_required:n = true,
2052   unknown         .code:n =
2053   {
2054     \str_set:Nn \l__spintent_command_name_str { spmul }
2055     \__spintent_unknown_keys_cmd:n {#1}
2056   },
2057 }

```

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

15.28.3 Internal functions for `\spmul`

The function `\__spintent_spmul_render_op:nn` parses both $\langle arguments \rangle$ through the `\spnum` command and embeds the semantic operator from `\spmsym`.

```

2058 \cs_new_protected:Nn \__spintent_spmul_render_op:nn
2059 {
2060   \spnum {#1}
2061   \MathMLintent { _ \l__spintent_spmsym_read_sym_tl }
2062   {
2063     \l__spintent_spmsym_symbol_tl
2064   }
2065   \spnum {#2}
2066 }

```

(End of definition for `\__spintent_spmul_render_op: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`.

```

2067 \cs_new_protected:Nn \__spintent_spmul_mathml:nn
2068 {
2069   \bool_if:NTF \l__spintent_spmul_wrap_parens_bool

```

```

2070     {
2071         \bool_if:NTF \__spinttent_spmul_read_parens_bool
2072         {
2073             \lparen \__spinttent_spmul_render_op:nn {#1} {#2} \rparen
2074         }
2075         {
2076             \MathMLintent { :silent } { \lparen }
2077             \__spinttent_spmul_render_op:nn {#1} {#2}
2078             \MathMLintent { :silent } { \rparen }
2079         }
2080     }
2081     { \__spinttent_spmul_render_op:nn {#1} {#2} }
2082 }

```

(End of definition for `\__spinttent_spmul_mathml:nn`.)

`\__spinttent_spmul_exec:nnn` The function `\__spinttent_spmul_exec:nnn` sets `<keys>`, validates `<arguments>`, and calls `\__spinttent_spmul_mathml:nn`.

```

2083 \cs_new_protected:Nn \__spinttent_spmul_exec:nnn
2084 {
2085     \keys_set:nn { spinttent / spmul } {#1}
2086     \tl_if_blank:nTF {#2}
2087     { \msg_error:nn { spinttent } { spmul-empty-arg } }
2088     {
2089         \tl_if_blank:nTF {#3}
2090         { \msg_error:nn { spinttent } { spmul-empty-arg } }
2091         {
2092             \__spinttent_spmul_mathml:nn {#2} {#3}
2093         }
2094     }
2095 }

```

(End of definition for `\__spinttent_spmul_exec:nnn`.)

`\spmul` Defines the user command `\spmul`. It requires *math mode*.

```

2096 \NewDocumentCommand \spmul { O{} m m }
2097 {
2098     \mode_if_math:TF
2099     {
2100         \group_begin:
2101         \__spinttent_spmul_exec:nnn {#1} {#2} {#3}
2102         \group_end:
2103     }
2104     { \msg_error:nnn { spinttent } { need-math-mode } { \spmul } }
2105 }

```

(End of definition for `\spmul`. This function is documented on page 19.)

15.29 The commands `\spgcd` and `\spmcn`

The user-level commands `\spgcd` and `\spmcn` 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 `\spgcd` and `\spmcn`

```

\__spinttent_mc_luaset_types_clist
\__spinttent_mc_luaset_all_numeric_str

```

The variable `\__spinttent_mc_luaset_types_clist` stores the type of each argument, `number` or `other`, in the same order and separated by commas. The variable `\__spinttent_mc_luaset_all_numeric_str` stores `true` when all the arguments are natural numbers. Both are set by the Lua module `spintent.lua`.

```

2106 \clist_new:N \__spinttent_mc_luaset_types_clist
2107 \str_new:N \__spinttent_mc_luaset_all_numeric_str

```

(End of definition for `\__spinttent_mc_luaset_types_clist` and `\__spinttent_mc_luaset_all_numeric_str`.)

```

\__spinttent_mc_raw_args_seq
\__spinttent_mc_types_seq
\__spinttent_mc_args_seq
\__spinttent_mc_idx_int

```

The variable `\__spinttent_mc_raw_args_seq` stores the arguments as they were given and `\__spinttent_mc_types_seq` stores their types. The variable `\__spinttent_mc_args_seq` stores the arguments ready to be typeset, with `\spnum` applied to those of type `number`. The variable `\__spinttent_mc_idx_int` stores the position of the argument being processed.

```

2108 \seq_new:N \__spinttent_mc_raw_args_seq
2109 \seq_new:N \__spinttent_mc_types_seq
2110 \seq_new:N \__spinttent_mc_args_seq
2111 \int_new:N \__spinttent_mc_idx_int

```

(End of definition for `\l__spinttent_mc_raw_args_seq` and others.)

Each command has its own copy of these variables. The variables `\l__spinttent_spmcd_luaset_mcd_value_tl` and `\l__spinttent_spmcm_luaset_mcm_value_tl` store the result of the greatest common divisor and of the least common multiple, set by the Lua module `spinttent.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.

```

2112 \clist_map_inline:nn { mcd , mcm }
2113 {
2114   \tl_new:c { \l__spinttent_sp #1_luaset_#1_value_tl }
2115   \bool_new:c { \l__spinttent_sp #1_read_terse_bool }
2116   \tl_new:c { \l__spinttent_sp #1_prefix_tl }
2117 }

```

(End of definition for `\l__spinttent_spmcd_luaset_mcd_value_tl` and others.)

15.29.2 Definition of keys for `\spmcd` and `\spmcm`

We define the shared *⟨keys⟩* `result`, `prefix`, `read-sty`, `upper-case`, `concept` and `silent`.

```

result
prefix
read-sty
upper-case
concept
silent
unknown
2118 \clist_map_inline:nn { mcd , mcm }
2119 {
2120   \keys_define:nn { spinttent / sp #1 }
2121   {
2122     result      .bool_set:c = { \l__spinttent_sp #1_print_result_bool },
2123     result      .initial:n = false,
2124     result      .value_required:n = true,
2125     prefix      .code:n =
2126       { \l__spinttent_sanitiz_affe:cn { \l__spinttent_sp #1_prefix_tl } {##1} },
2127     prefix      .initial:n = ,
2128     prefix      .value_required:n = true,
2129     read-sty    .choices:nn =
2130       { terse , clear }
2131       {
2132         \int_case:nn { \l_keys_choice_int }
2133         {
2134           {1} { \bool_set_true:c { \l__spinttent_sp #1_read_terse_bool } }
2135           {2} { \bool_set_false:c { \l__spinttent_sp #1_read_terse_bool } }
2136         }
2137       },
2138     read-sty    .initial:n = clear,
2139     read-sty    .value_required:n = true,
2140     read-sty / unknown .code:n =
2141       \msg_error:nneee { spinttent } { unknown-choice }
2142       { read-sty } { terse , ~clear } { \exp_not:n {##1} },
2143     upper-case .bool_set:c = { \l__spinttent_sp #1_print_upper_bool },
2144     upper-case .initial:n = false,
2145     upper-case .value_required:n = true,
2146     concept    .bool_set:c = { \l__spinttent_sp #1_concept_bool },
2147     concept    .initial:n = true,
2148     concept    .value_required:n = true,
2149     silent     .bool_set:c = { \l__spinttent_sp #1_silent_bool },
2150     silent     .initial:n = false,
2151     silent     .value_required:n = true,
2152     unknown    .code:n =
2153       {
2154         \str_set:Nn \l__spinttent_command_name_str { sp #1 }
2155         \l__spinttent_unknown_keys_cmd:n {##1}
2156       },
2157   }
2158 }

```

(End of definition for `result` and others.)

15.29.3 Internal functions for `\spmcd` and `\spmcm`

`\l__spinttent_mc_process_args:nn` The function `\l__spinttent_mc_process_args:nn` parses the operation ‘#1’ and its raw argument list ‘#2’. Calls `\l__spinttent_luafun_mc_classify_args:n`, splits ‘#2’ into `\l__spinttent_mc_raw_args_seq` and `\l__spinttent_mc_luaset_types_clist` into `\l__spinttent_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`.

```

2159 \cs_new_protected:Nn \__spintent_mc_process_args:nn
2160 {
2161   \__spintent_luafun_mc_classify_args:n {#2}
2162   \seq_set_from_clist:Nn \l__spintent_mc_raw_args_seq {#2}
2163   \seq_set_from_clist:NN \l__spintent_mc_types_seq \l__spintent_mc_lua_set_types_clist
2164   \seq_clear:N \l__spintent_mc_args_seq
2165   \int_zero:N \l__spintent_mc_idx_int
2166   \seq_map_inline:Nn \l__spintent_mc_raw_args_seq
2167   {
2168     \int_incr:N \l__spintent_mc_idx_int
2169     \str_if_eq:eeTF
2170     { \seq_item:Nn \l__spintent_mc_types_seq { \l__spintent_mc_idx_int } }
2171     { number }
2172     {
2173       \seq_put_right:Nn \l__spintent_mc_args_seq { \spnum {##1} }
2174     }
2175     { \seq_put_right:Nn \l__spintent_mc_args_seq {##1} }
2176   }
2177   \__spintent_mc_mathml_args:n {#1}
2178 }

```

(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`.

```

2179 \cs_new_protected:Nn \__spintent_mc_mathml_args:n
2180 {
2181   \bool_if:cTF { l__spintent_sp #1 _silent_bool }
2182   {
2183     \seq_set_map:NNn \l__spintent_mc_args_seq \l__spintent_mc_args_seq
2184     {
2185       \MathMLintent { _- } { \__spintent_inner_sep: ##1 }
2186     }
2187     \MathMLintent { :silent } { \c__spintent_invisible_sep_int } % prevent "por"
2188     \MathMLintent { :silent } { \lparen }
2189     % prevent "intervalo"
2190     \MathMLintent { :silent } { \c__spintent_invisible_sep_int }
2191     \seq_use:Nnnn \l__spintent_mc_args_seq
2192     { \MathMLintent { :silent } { , } }
2193     { \MathMLintent { :silent } { , } }
2194     { \MathMLintent { :silent } { , } }
2195   }
2196   {
2197     \MathMLintent { :silent } { \c__spintent_invisible_sep_int } % prevent doble "de"
2198     \bool_if:cTF { l__spintent_sp #1 _concept_bool }
2199     {
2200       \MathMLintent { _entre } { \lparen }
2201     }
2202     {
2203       \MathMLintent { :silent } { \lparen }
2204     }
2205     % prevent "intervalo"
2206     \MathMLintent { :silent } { \c__spintent_invisible_sep_int }
2207     \seq_use:Nnnn \l__spintent_mc_args_seq
2208     {
2209       \MathMLintent { :silent } { , }
2210       \MathMLintent { _y- } { \c__spintent_invisible_sep_int } % "y" medio
2211     }
2212     {
2213       \MathMLintent { :silent } { , }
2214     }
2215     {
2216       \MathMLintent { :silent } { , }
2217       \MathMLintent { _y- } { \c__spintent_invisible_sep_int } % "y" final
2218     }
2219   }
2220   \MathMLintent { :silent } { \rparen }
2221 }

```

(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 $\langle keys \rangle$ in ‘#3’, and the optional arguments in ‘#4’.

After setting the $\langle keys \rangle$, 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.

```
2222 \cs_new_protected:Nn \__spintent_spmc_exec:nnnn
2223 {
2224   \keys_set:nn { spintent / sp #1 } {#3}
2225   \__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.

```
2226   \tl_if_novalue:nF {#4}
2227   {
2228     \seq_set_from_clist:Nn \l__spintent_mc_raw_args_seq {#4}
2229     \int_compare:nNnTF { \seq_count:N \l__spintent_mc_raw_args_seq } < { 2 }
2230     {
2231       \msg_error:nne { spintent } { mc-too-few-args } { \exp_not:c { sp #1 } }
2232     }
2233     {
2234       \__spintent_luafun_mc_classify_args:n {#4}
2235       \str_if_eq:VnT \l__spintent_mc_luaset_all_numeric_str { true }
2236       {
2237         \use:c { __spintent_luafun_calculate_ #1 :n } {#4}
2238       }
2239       \__spintent_mc_process_args:nn {#1} {#4}
2240       \__spintent_spmc_result_mathml:n {#1}
2241     }
2242   }
2243 }
```

(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.

```
2244 \cs_new_protected:Nn \__spintent_spmc_mathml:nn
2245 {
2246   \bool_if:cTF { l__spintent_sp #1 _silent_bool }
2247   {
2248     \MathMLintent { :silent }
2249     {
2250       \mathrm
2251       {
2252         \bool_if:cTF { l__spintent_sp #1 _print_upper_bool }
2253         { \text_uppercase:n {#1} } {#1}
2254       }
2255     }
2256   }
2257   {
2258     \bool_if:cTF { l__spintent_sp #1 _concept_bool }
2259     {
2260       \MathMLintent
2261       {
2262         -
2263         \tl_if_empty:cF { l__spintent_sp #1 _prefix_tl }
2264         {
```

```

2265         \tl_use:c { \__spinttent_sp #1 _prefix_tl } -
2266     }
2267     \bool_if:cTF { \__spinttent_sp #1 _read_terse_bool }
2268         {#1} {#2}
2269     }
2270     {
2271         \mathrm
2272         {
2273             \bool_if:cTF { \__spinttent_sp #1 _print_upper_bool }
2274                 { \text_uppercase:n {#1} } {#1}
2275             }
2276         }
2277     }
2278     {
2279         \MathMLintent { :silent }
2280         {
2281             \mathrm
2282             {
2283                 \bool_if:cTF { \__spinttent_sp #1 _print_upper_bool }
2284                     { \text_uppercase:n {#1} } {#1}
2285                 }
2286             }
2287         }
2288     }
2289 }

```

(End of definition for `\__spinttent_spmc_mathml:nn`.)

`\__spinttent_spmc_result_mathml:n`

The function `\__spinttent_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.

```

2290 \cs_new_protected:Nn \__spinttent_spmc_result_mathml:n
2291 {
2292     \str_if_eq:VnT \__spinttent_mc_luaset_all_numeric_str { true }
2293     {
2294         \bool_if:cT { \__spinttent_sp #1 _print_result_bool }
2295         {
2296             \bool_if:cTF { \__spinttent_sp #1 _silent_bool }
2297             {
2298                 \MathMLintent { :silent } { = }
2299                 \MathMLintent { _- }
2300                 {
2301                     \__spinttent_inner_sep:
2302                     \tl_use:c { \__spinttent_sp #1 _luaset_ #1 _value_tl }
2303                 }
2304             }
2305             {
2306                 \MathMLintent { _es-igual-a } { = }
2307                 \tl_use:c { \__spinttent_sp #1 _luaset_ #1 _value_tl }
2308             }
2309         }
2310     }
2311 }

```

(End of definition for `\__spinttent_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 `\__spinttent_spmc_exec:nnnn`, passing along the appropriate reading strings and arguments.

```

2312 \NewDocumentCommand \spmcd { 0{} d() }
2313 {
2314     \mode_if_math:TF
2315     {
2316         \group_begin:
2317         \__spinttent_spmc_exec:nnnn { mcd } { máximo-común-divisor } {#1} {#2}
2318         \group_end:
2319     }
2320     { \msg_error:nnn { spinttent } { need-math-mode } { \spmcd } }
2321 }
2322 \NewDocumentCommand \spmcm { 0{} d() }

```

```

2323 {
2324   \mode_if_math:TF
2325   {
2326     \group_begin:
2327     __spintent_spmc_exec:nnnn { mcm } { mínimo-común-múltiplo } {#1} {#2}
2328     \group_end:
2329   }
2330   { \msg_error:nnn { spintent } { need-math-mode } { \spmc } }
2331 }

```

(End of definition for `\spmc` and `\spmc`. 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.

15.30.1 Definition of variables for `\spnM` and `\spnD`

The variable `\__spintent_spnmd_luaset_is_number_str` stores `true` when the `{⟨argument⟩}` is a positive integer. The variable `\__spintent_spnM_spnD_luaset_error_str` stores `true` when the `{⟨argument⟩}` is zero. Both are set by the Lua module `spintent.lua`.

```

2332 \str_new:N \__spintent_spnM_spnD_luaset_error_str
2333 \str_new:N \__spintent_spnmd_luaset_is_number_str

```

(End of definition for `\__spintent_spnmd_luaset_is_number_str` and `\__spintent_spnM_spnD_luaset_error_str`.)

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.

```

2334 \clist_map_inline:nn { nM , nD }
2335 {
2336   \tl_new:c { \__spintent_sp #1 _luaset_result_tl }
2337   \str_new:c { \__spintent_sp #1 _luaset_is_truncated_str }
2338   \tl_new:c { \__spintent_sp #1 _prefix_tl }
2339 }

```

(End of definition for `\__spintent_spnM_luaset_result_tl` and others.)

The variable `\__spintent_md_out_seq` stores the results to be printed, one by one. The variable `\__spintent_spnM_spnD_intent_tl` holds the intent that `\__spintent_spnM_spnD_intent:nn` builds.

```

2340 \seq_new:N \__spintent_md_out_seq
2341 \tl_new:N \__spintent_spnM_spnD_intent_tl

```

(End of definition for `\__spintent_md_out_seq` and `\__spintent_spnM_spnD_intent_tl`.)

15.30.2 Definition of error messages for `\spnM` and `\spnD`

The “error” `md-invalid-arg` is raised when the argument for multiples or divisors is invalid.

`md-invalid-arg`
`mc-too-few-args`

```

2342 \msg_new:nnnn { spintent } { md-invalid-arg }
2343 {
2344   Invalid~argument~for~'\tl_trim_spaces:n {#1}'~\msg_line_context:.
2345 }
2346 {
2347   The~argument~must~be~a~single~positive~integer.
2348 }

```

The “error” `mc-too-few-args` is raised when fewer than two arguments are provided.

```

2349 \msg_new:nnnn { spintent } { mc-too-few-args }
2350 {
2351   Invalid~arguments~for~'\tl_trim_spaces:n {#1}'~\msg_line_context:.
2352 }
2353 {
2354   At~least~two~arguments~are~required.
2355 }

```

(End of definition for `md-invalid-arg` and `mc-too-few-args`.)

15.30.3 Definition of keys for \spnM and \spnD

We define the shared *(keys)* result, silent, result-num, prefix and concept.

```

2356 \clist_map_inline:nn { nM , nD }
2357 {
2358   \keys_define:nn { spintent / sp #1 }
2359   {
2360     result      .bool_set:c = { l__spintent_sp #1 _print_result_bool },
2361     result      .initial:n = false,
2362     result      .value_required:n = true,
2363     silent      .bool_set:c = { l__spintent_sp #1 _silent_bool },
2364     silent      .initial:n = false,
2365     silent      .value_required:n = true,
2366     result-num  .tl_set:c = { l__spintent_sp #1 _print_result_num_tl },
2367     result-num  .initial:n = ,
2368     result-num  .value_required:n = true,
2369     prefix      .code:n =
2370       { \__spintent_sanitizize_affix:cn { l__spintent_sp #1 _prefix_tl } {##1} },
2371     prefix      .initial:n = ,
2372     prefix      .value_required:n = true,
2373     concept     .bool_set:c = { l__spintent_sp #1 _concept_bool },
2374     concept     .initial:n = true,
2375     concept     .value_required:n = true,
2376     unknown     .code:n =
2377       {
2378         \str_set:Nn \l__spintent_command_name_str { sp #1 }
2379         \__spintent_unknown_keys_cmd:n {##1}
2380       },
2381   }
2382 }

```

(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`.

```

2383 \cs_new_protected:Nn \__spintent_spnM_spnD_intent:nn
2384 {
2385   \tl_set:Nc \l__spintent_spnM_spnD_intent_tl
2386   {
2387     -
2388     \tl_if_empty:cF { l__spintent_sp #1 _prefix_tl }
2389     { \tl_use:c { l__spintent_sp #1 _prefix_tl } - }
2390     \str_case:nnF {#2} { { múltiplos } { múltiplos } }
2391     {#2}
2392     -de
2393   }
2394 }

```

(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.

```

2395 \cs_new_protected:Nn \__spintent_spnM_spnD_silent:nn
2396 {
2397   \MathMLintent { :silent } { \operatorname {#1} }
2398   \MathMLintent { :silent } { \lparen }
2399   \MathMLintent { :silent } { \c__spintent_invisible_sep_int } % prevent doble "de"
2400   \MathMLintent { _- } { \__spintent_inner_sep: #2 }
2401   \MathMLintent { :silent } { \rparen }
2402 }

```

(End of definition for `\__spintent_spnM_spnD_silent:nn`.)

`\__spintnt_spnM_spnD_result:nn`

The internal function `\__spintnt_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.

```

2403 \cs_new_protected:Nn \__spintnt_spnM_spnD_result:nn
2404 {
2405   \use:c { __spintnt_luafun_calcular_ #1 :nn } {#2}
2406   {
2407     \tl_use:c { l__spintnt_sp #1 _print_result_num_tl }
2408   }
2409   \str_if_eq:VnTF \__spintnt_spnM_spnD_luaset_error_str { true }
2410   {
2411     \msg_error:nne { spintnt } { md-invalid-arg } { \exp_not:c { sp #1 } }
2412   }
2413   {
2414     \MathMLintent { :silent } { \c__spintnt_invisible_sep_int } % prevent "por"
2415     \MathMLintent { _son } { = \{ }
2416     \seq_set_from_clist:Ne
2417       \l__spintnt_md_out_seq { \tl_use:c { l__spintnt_sp #1 _luaset_result_tl } }
2418     \seq_use:Nnnn \l__spintnt_md_out_seq
2419     {
2420       \MathMLintent { :silent } { , }
2421       \str_if_eq:vnT { l__spintnt_sp #1 _luaset_is_truncated_str } { false }
2422       {
2423         \MathMLintent { _y- } { \c__spintnt_invisible_sep_int } % "y" (solo dos)
2424       }
2425     }
2426     { \MathMLintent { :silent } { , } }
2427     {
2428       \MathMLintent { :silent } { , }
2429       \str_if_eq:vnT { l__spintnt_sp #1 _luaset_is_truncated_str } { false }
2430       {
2431         \MathMLintent { _y- } { \c__spintnt_invisible_sep_int } % "y" final
2432       }
2433     }
2434     \str_if_eq:vnT { l__spintnt_sp #1 _luaset_is_truncated_str } { true }
2435     {
2436       \MathMLintent { :silent } { , } ~ \dots
2437     }
2438     \MathMLintent { :silent } { \} }
2439   }
2440 }

```

(End of definition for `\__spintnt_spnM_spnD_result:nn`.)

`\__spintnt_spnM_spnD_exec:nnnnn`

The internal function `\__spintnt_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, `\__spintnt_luafun_spnmd_classify:n` classifies it, and a zero is rejected with `md-invalid-arg`. Then it calls `\__spintnt_spnM_spnD_mathml:nnnn`.

```

2441 \cs_new_protected:Nn \__spintnt_spnM_spnD_exec:nnnnn
2442 {
2443   \keys_set:nn { spintnt / sp #1 } {#4}
2444   \tl_if_novalue:nF {#5}
2445   {
2446     \__spintnt_luafun_spnmd_classify:n { \exp_not:n {#5} }
2447     \str_if_eq:VnT \__spintnt_spnM_spnD_luaset_error_str { true }
2448     { \msg_error:nne { spintnt } { md-invalid-arg } { \exp_not:c { sp #1 } } }
2449   }
2450   \__spintnt_spnM_spnD_mathml:nnnn {#1} {#2} {#3} {#5}
2451 }

```

(End of definition for `\__spintnt_spnM_spnD_exec:nnnnn`.)

`\__spintnt_spnM_spnD_mathml:nnnn`

The function `\__spintnt_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__spintnt_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.

```

2452 \cs_new_protected:Nn \__spintent_spnM_spnD_mathml:nnnn
2453 {
2454   \tl_if_novalue:nTF {#4}
2455   {
2456     \bool_if:cTF { \__spintent_sp #1 _silent_bool }
2457     { \__spintent_spnM_spnD_silent:nn {#2} { n } }
2458     {
2459       \bool_if:cTF { \__spintent_sp #1 _concept_bool }
2460       {
2461         \__spintent_spnM_spnD_intent:nn {#1} {#3}
2462         \MathMLintent
2463         { \__spintent_spnM_spnD_intent_tl }
2464         { \operatorname {#2} }
2465       }
2466       { \MathMLintent { :silent } { \operatorname {#2} } }
2467       \MathMLintent { :silent } { \lparen }
2468       % prevent doble "de"
2469       \MathMLintent { :silent } { \c__spintent_invisible_sep_int }
2470       n
2471       \MathMLintent { :silent } { \rparen }
2472     }
2473   }
2474   {
2475     \bool_if:cTF { \__spintent_sp #1 _silent_bool }
2476     {
2477       \str_if_eq:VnTF \__spintent_spnmd_luaset_is_number_str { true }
2478       { \__spintent_spnM_spnD_silent:nn {#2} { \spnum {#4} } }
2479       { \__spintent_spnM_spnD_silent:nn {#2} {#4} }
2480     }
2481     {
2482       \bool_if:cTF { \__spintent_sp #1 _concept_bool }
2483       {
2484         \__spintent_spnM_spnD_intent:nn {#1} {#3}
2485         \MathMLintent
2486         { \__spintent_spnM_spnD_intent_tl }
2487         { \operatorname {#2} }
2488       }
2489       { \MathMLintent { :silent } { \operatorname {#2} } }
2490       \MathMLintent { :silent } { \lparen }
2491       % prevent doble "de"
2492       \MathMLintent { :silent } { \c__spintent_invisible_sep_int }
2493       \str_if_eq:VnTF \__spintent_spnmd_luaset_is_number_str { true }
2494       { \spnum {#4} }
2495       {#4}
2496       \MathMLintent { :silent } { \rparen }
2497     }
2498     \str_if_eq:VnT \__spintent_spnmd_luaset_is_number_str { true }
2499     {
2500       \bool_if:cT { \__spintent_sp #1 _print_result_bool }
2501       {
2502         \bool_if:cTF { \__spintent_sp #1 _silent_bool }
2503         {
2504           % prevent "por"
2505           \MathMLintent { :silent } { \c__spintent_invisible_sep_int }
2506           \MathMLintent { - }
2507           {
2508             \__spintent_inner_sep:
2509             \__spintent_spnM_spnD_result:nn {#1} {#4}
2510           }
2511         }
2512         {
2513           \__spintent_spnM_spnD_result:nn {#1} {#4}
2514         }
2515       }
2516     }
2517   }

```

```

2516     }
2517   }
2518 }

```

(End of definition for `\__spintent_spnM_spnD_mathml:nnnn`.)

`\spnM` 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’.

```

2519 \NewDocumentCommand \spnM { 0{} d() }
2520 {
2521   \mode_if_math:TF
2522   {
2523     \group_begin:
2524     \__spintent_spnM_spnD_exec:nnnnn { nM } { M } { multiplos } {#1} {#2}
2525     \group_end:
2526   }
2527   { \msg_error:nnn { spintent } { need-math-mode } { \spnM } }
2528 }
2529 \NewDocumentCommand \spnD { 0{} d() }
2530 {
2531   \mode_if_math:TF
2532   {
2533     \group_begin:
2534     \__spintent_spnM_spnD_exec:nnnnn { nD } { D } { divisores } {#1} {#2}
2535     \group_end:
2536   }
2537   { \msg_error:nnn { spintent } { need-math-mode } { \spnD } }
2538 }

```

(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`

`\__spintent_sprazon_cmd_str` The string variable `\__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`.

```

2539 \str_new:N \__spintent_sprazon_cmd_str
2540 \str_set:Nn \__spintent_sprazon_cmd_str { sprazon }

```

(End of definition for `\__spintent_sprazon_cmd_str`.)

15.31.2 Definition of keys for `\sprazon`

`silent` We define the *⟨keys⟩* `silent`, `use-dfrac` and `inline`.
`use-dfrac`
`inline`
`unknown`

```

2541 \keys_define:nn { spintent / sprazon }
2542 {
2543   silent      .bool_set:N = \__spintent_sprazon_silent_bool,
2544   silent      .initial:n = false,
2545   silent      .value_required:n = true,
2546   use-dfrac   .bool_set:N = \__spintent_sprazon_use_dfrac_bool,
2547   use-dfrac   .initial:n = false,
2548   use-dfrac   .value_required:n = true,
2549   inline      .bool_set:N = \__spintent_sprazon_inline_bool,
2550   inline      .initial:n = false,
2551   inline      .value_required:n = true,
2552   unknown     .code:n =
2553   {
2554     \str_set_eq:NN \__spintent_command_name_str \__spintent_sprazon_cmd_str
2555     \__spintent_unknown_keys_cmd:n {#1}
2556   },
2557 }

```

(End of definition for `silent` and others.)

15.31.3 Internal functions for `\sprazon`

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`.

```

2558 \cs_new_protected:Nn \__spintent_sprazon_exec:nnn
2559 {
2560   \keys_set:nn { spintent / sprazon } {#1}
2561   \__spintent_sprazon_mathml:nn {#2} {#3}
2562 }

```

(End of definition for `\__spintent_sprazon_exec:nnn`.)

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.

```

2563 \cs_new_protected:Nn \__spintent_sprazon_mathml:nn
2564 {
2565   \__spintent_invisible_sep:
2566   \bool_if:NTF \__spintent_sprazon_silent_bool
2567   {
2568     \MathMLintent { _- }
2569     {
2570       \__spintent_inner_sep:
2571       \bool_if:NTF \__spintent_sprazon_inline_bool
2572       { #1 : #2 }
2573       {
2574         \bool_if:NTF \__spintent_sprazon_use_dfrac_bool
2575         { \dfrac {#1} {#2} }
2576         { \frac {#1} {#2} }
2577       }
2578     }
2579   }
2580   {
2581     \bool_if:NTF \__spintent_sprazon_inline_bool
2582     {
2583       \MathMLintent { _es-a:infix($num,$denom) }
2584       {
2585         \MathMLarg { num } {#1}
2586         :
2587         \MathMLarg { denom } {#2}
2588       }
2589     }
2590     {
2591       \MathMLintent { _es-a:infix($num,$denom) }
2592       {
2593         \__spintent_inner_sep:
2594         \bool_if:NTF \__spintent_sprazon_use_dfrac_bool
2595         {
2596           \dfrac { \MathMLarg { num } {#1} } { \MathMLarg { denom } {#2} }
2597         }
2598         { \frac { \MathMLarg { num } {#1} } { \MathMLarg { denom } {#2} } }
2599       }
2600     }
2601   }
2602 }

```

(End of definition for `\__spintent_sprazon_mathml:nn`.)

`\sprazon` The public command `\sprazon` enforces *math mode* and delegates execution to `\__spintent_sprazon_exec:nnn`.

```

2603 \NewDocumentCommand \sprazon { 0{} m m }
2604 {
2605   \mode_if_math:TF
2606   {
2607     \group_begin:

```

```

2608     \__spintent_sprazon_exec:nnn {#1} {#2} {#3}
2609   \group_end:
2610 }
2611 { \msg_error:nnn { spintent } { need-math-mode } { \sprazon } }
2612 }

```

(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.

```

2613 \NewDocumentCommand \spProp { O{} m m m m }
2614 {
2615   \mode_if_math:TF
2616   {
2617     \group_begin:
2618     \str_set:Nn \__spintent_sprazon_cmd_str { spProp }
2619     \__spintent_sprazon_exec:nnn {#1} {#2} {#3}
2620     \bool_if:NTF \__spintent_sprazon_silent_bool
2621     {
2622       \MathMLintent { :silent } { = }
2623     }
2624     { \MathMLintent { _como } { = } }
2625     \__spintent_sprazon_exec:nnn {#1} {#4} {#5}
2626     \group_end:
2627   }
2628   { \msg_error:nnn { spintent } { need-math-mode } { \spProp } }
2629 }

```

(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`

```

\__spintent_spang_luaset_grado_str
\__spintent_spang_luaset_minuto_str
\__spintent_spang_luaset_segundo_str
\__spintent_spang_luaset_error_str

```

These variables store the values returned by the Lua module `spintent.lua`. The variables `\__spintent_spang_luaset_grado_str`, `\__spintent_spang_luaset_minuto_str` and `\__spintent_spang_luaset_segundo_str` store the degrees, the minutes and the seconds, and are empty if missing. The variable `\__spintent_spang_luaset_error_str` stores `true` or `false` if the `{argument}` is invalid.

```

2630 \str_new:N \__spintent_spang_luaset_grado_str
2631 \str_new:N \__spintent_spang_luaset_minuto_str
2632 \str_new:N \__spintent_spang_luaset_segundo_str
2633 \str_new:N \__spintent_spang_luaset_error_str

```

(End of definition for `\__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.

```

2634 \msg_new:nnnn { spintent } { spang-invalid-arg }
2635 {
2636   Invalid~argument~'~#2'~for~'\tl_trim_spaces:n {#1}'~'\msg_line_context:.
2637 }
2638 {
2639   The~argument~must~have~the~form~'degrees',~'degrees:minutes'~or~
2640   'degrees:minutes:seconds'.
2641 }

```

(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:`.

```

2642 \cs_new_protected:Nn \__spintent_spang_exec:n
2643 {
2644   \str_set:Nn \l__spintent_spang_luaset_error_str { false }
2645   \__spintent_luafun_spang_parse:n {#1}
2646   \str_if_eq:NnTF \l__spintent_spang_luaset_error_str { true }
2647   {
2648     \msg_error:nnnn { spintent } { spang-invalid-arg } { \spang } {#1}
2649   }
2650   { \__spintent_spang_mathml: }
2651 }
```

(End of definition for `\__spintent_spang_exec:n`.)

`\__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.

```

2652 \cs_new_protected:Nn \__spintent_spang_mathml:
2653 {
2654   \str_if_empty:NF \l__spintent_spang_luaset_grado_str
2655   {
2656     \exp_args:Ne \spunit { \l__spintent_spang_luaset_grado_str ° }
2657   }
2658   \str_if_empty:NF \l__spintent_spang_luaset_minuto_str
2659   {
2660     \MathMLintent { _minutos:postfix($m) }
2661     {
2662       \MathMLLarg { m }
2663       {
2664         \exp_args:Ne \spunit { \l__spintent_spang_luaset_minuto_str ' }
2665       }
2666     }
2667   }
2668   \str_if_empty:NF \l__spintent_spang_luaset_segundo_str
2669   {
2670     \MathMLintent { _segundos:postfix($s) }
2671     {
2672       \MathMLLarg { s }
2673       {
2674         \exp_args:Ne \spunit { \l__spintent_spang_luaset_segundo_str " }
2675       }
2676     }
2677   }
2678 }
```

(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)}`.

```

2679 \NewDocumentCommand \spang { m }
2680 {
2681   \mode_if_math:TF
2682   {
2683     \group_begin:
2684     \__spintent_spang_exec:n {#1}
2685     \group_end:
2686   }
2687   { \msg_error:nnn { spintent } { need-math-mode } { \spang } }
2688 }
```

(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 $\langle \textit{argument} \rangle$ 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 $\langle \textit{argument} \rangle$ contains decimal separators or attached units, which conflict with the constraints of a pure integer representation, or when it is empty.

```

2689 \msg_new:nnnn { spintent } { spnZ-not-integer }
2690 {
2691   Invalid~argument~'#1'~for~'\token_to_str:N \spnZ'~\msg_line_context:.
2692 }
2693 {
2694   Only~integers~are~accepted;~decimal~separators~and~units~are~not~allowed.
2695 }
```

(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 $\langle \textit{keys} \rangle$ `cifras-sep`, `read-sign`, `wrap-parens` and `read-parens`.

```

2696 \keys_define:nn { spintent / spnZ }
2697 {
2698   cifras-sep .meta:nn = { spintent / spnum } { cifras-sep = {#1} },
2699   read-sign .meta:nn = { spintent / spnum } { read-sign = {#1} },
2700   wrap-parens .bool_set:N = \__spintent_spnz_wrap_parens_bool,
2701   wrap-parens .initial:n = false,
2702   wrap-parens .value_required:n = true,
2703   read-parens .bool_set:N = \__spintent_spnz_read_parens_bool,
2704   read-parens .initial:n = true,
2705   read-parens .value_required:n = true,
2706   unknown .code:n =
2707     {
2708       \str_set:Nn \__spintent_command_name_str { spnZ }
2709       \__spintent_unknown_keys_cmd:n {#1}
2710     },
2711 }
```

(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 `\__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_spnnum_print_num_start:`.

```

2712 \cs_new_protected:Nn \__spintent_spnz_mathml:
2713 {
2714   \bool_if:NTF \__spintent_spnz_wrap_parens_bool
2715     {
2716       \bool_if:NTF \__spintent_spnz_read_parens_bool
2717         {
2718           \lparen \__spintent_spnnum_print_num_start: \rparen
2719         }
2720         {
2721           \MathMLintent { :silent } { \lparen }
2722           \__spintent_spnnum_print_num_start:
2723           \MathMLintent { :silent } { \rparen }
2724         }
2725     }
2726     { \__spintent_spnnum_print_num_start: }
2727 }
```

(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 `\__spintent_luaset_has_integer_str` is `true`. If not `true`, an “error” is raised.

If it is `true`, it calls `\__spintent_spnz_mathml:`.

```

2728 \cs_new_protected:Nn \__spintent_spnz_exec:nn
2729 {
2730   \keys_set:nn { spintent / spnZ } {#1}
2731   \__spintent_luafun_clean_split_and_set:n { \exp_not:n {#2} }
2732   \str_if_eq:VnTF \__spintent_luaset_has_integer_str { true }
2733     { \__spintent_spnz_mathml: }
2734     { \msg_error:nnn { spintent } { spnZ-not-integer } {#2} }
2735 }

```

(End of definition for `\__spintent_spnz_exec:nn`.)

`\spnZ` Defines the user command `\spnZ`, which is executed within a *group* and enforces *math mode*.

```

2736 \NewDocumentCommand \spnZ { O{} m }
2737 {
2738   \mode_if_math:TF
2739     {
2740       \group_begin:
2741         \__spintent_spnz_exec:nn {#1} {#2}
2742       \group_end:
2743     }
2744     { \msg_error:nnn { spintent } { need-math-mode } { \spnZ } }
2745 }

```

(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`

`\__spintent_spmQ_join_word_str` The variable `\__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 `\__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`.

```

2746 \str_new:N \__spintent_spmQ_join_word_str
2747 \str_new:N \__spintent_spmQ_intent_read_sign_str

```

(End of definition for `\__spintent_spmQ_join_word_str` and `\__spintent_spmQ_intent_read_sign_str`.)

`\__spintent_spmQ_wrap_parens_l_tl` The variables `\__spintent_spmQ_wrap_parens_l_tl` and `\__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`.

```

2748 \tl_new:N \__spintent_spmQ_wrap_parens_l_tl
2749 \tl_new:N \__spintent_spmQ_wrap_parens_r_tl

```

(End of definition for `\__spintent_spmQ_wrap_parens_l_tl` and `\__spintent_spmQ_wrap_parens_r_tl`.)

`\__spintent_spmQ_error_bool` The boolean variable `\__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.

```

2750 \bool_new:N \__spintent_spmQ_error_bool

```

(End of definition for `\__spintent_spmQ_error_bool`.)

`\__spintent_saved_luamml_flag_int` The integer variable `\__spintent_saved_luamml_flag_int` temporarily stores the current value of `\__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.

```

2751 \int_new:N \__spintent_saved_luamml_flag_int

```

(End of definition for `\__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.

```

2752 \msg_new:nnnn { spintent } { spmQ-zero-denominator }
2753 {
2754   Invalid~denominator~for~'\token_to_str:N \spmQ'\~\msg_line_context:.
2755 }
2756 {
2757   The~denominator~of~a~fraction~cannot~be~zero.
2758 }
2759 \msg_new:nnnn { spintent } { spmQ-not-integer }
2760 {
2761   Invalid~argument~'#1'~for~'\token_to_str:N \spmQ'\~\msg_line_context:.
2762 }
2763 {
2764   Only~integers~are~accepted.
2765 }
2766 \msg_new:nnnn { spintent } { spmQ-signed-part }
2767 {
2768   Invalid~argument~'#1'~for~'\token_to_str:N \spmQ'\~\msg_line_context:.
2769 }
2770 {
2771   The~numerator~and~the~denominator~must~be~integers~without~sign;~only~the~integer~
2772   part~may~have~one.
2773 }
```

(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`.

```

2774 \keys_define:nn { spintent / spmQ }
2775 {
2776   join-word .choices:nn =
2777     { entero , enteros , y }
2778   {
2779     \int_case:nn { \l_keys_choice_int }
2780     {
2781       {1} { \str_set:Nn \l__spintent_spmQ_join_word_str { entero } }
2782       {2} { \str_set:Nn \l__spintent_spmQ_join_word_str { enteros } }
2783       {3} { \str_set:Nn \l__spintent_spmQ_join_word_str { y } }
2784     }
2785   },
2786   join-word .initial:n = y,
2787   join-word .value_required:n = true,
2788   join-word / unknown .code:n =
2789     \msg_error:nneee { spintent } { unknown-choice }
2790     { join-word } { entero, ~enteros, ~y } { \exp_not:n {#1} },
2791   use-spnZ .bool_set:N = \l__spintent_spmQ_use_spnZ_bool,
2792   use-spnZ .initial:n = true,
2793   use-spnZ .value_required:n = true,
2794   use-dfrac .bool_set:N = \l__spintent_spmQ_use_dfrac_bool,
2795   use-dfrac .initial:n = false,
2796   use-dfrac .value_required:n = true,
2797   wrap-parens .bool_set:N = \l__spintent_spmQ_wrap_parens_bool,
2798   wrap-parens .initial:n = false,
2799   wrap-parens .value_required:n = true,
2800   read-parens .bool_set:N = \l__spintent_spmQ_read_parens_bool,
2801   read-parens .initial:n = true,
2802   read-parens .value_required:n = true,
2803   silent .bool_set:N = \l__spintent_spmQ_silent_bool,
2804   silent .initial:n = false,
2805   silent .value_required:n = true,
2806   unknown .code:n =
2807     {
2808       \str_set:Nn \l__spintent_command_name_str { spmQ }
2809       \__spintent_unknown_keys_cmd:n {#1}
2810     },

```

```
2811 }
```

(End of definition for `join-word` and others.)

15.35.4 Internal functions for `\spmQ`

```
\__spinttent_spmQ_set_wrap_parens_size:
```

The function `\__spinttent_spmQ_set_wrap_parens_size:` captures the current mathematical style using the primitive `\mathstyle` from LuaTeX and sets the variables `\l__spinttent_spmQ_wrap_parens_l_tl` and `\l__spinttent_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 `\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.

```
2812 \cs_new_protected:Nn \__spinttent_spmQ_set_wrap_parens_size:
2813 {
2814   \int_case:nnF { \mathstyle }
2815   {
2816     { 0 }
2817     {
2818       \tl_set:Nn \l__spinttent_spmQ_wrap_parens_l_tl { \Bigl \lparen }
2819       \tl_set:Nn \l__spinttent_spmQ_wrap_parens_r_tl { \Bigr \rparen }
2820     }
2821     { 1 }
2822     {
2823       \tl_set:Nn \l__spinttent_spmQ_wrap_parens_l_tl { \Bigl \lparen }
2824       \tl_set:Nn \l__spinttent_spmQ_wrap_parens_r_tl { \Bigr \rparen }
2825     }
2826     { 2 }
2827     {
2828       \tl_set:Nn \l__spinttent_spmQ_wrap_parens_l_tl { \bigl \lparen }
2829       \tl_set:Nn \l__spinttent_spmQ_wrap_parens_r_tl { \bigr \rparen }
2830     }
2831     { 3 }
2832     {
2833       \tl_set:Nn \l__spinttent_spmQ_wrap_parens_l_tl { \bigl \lparen }
2834       \tl_set:Nn \l__spinttent_spmQ_wrap_parens_r_tl { \bigr \rparen }
2835     }
2836   }
2837   {
2838     \tl_set:Nn \l__spinttent_spmQ_wrap_parens_l_tl { \lparen }
2839     \tl_set:Nn \l__spinttent_spmQ_wrap_parens_r_tl { \rparen }
2840   }
2841 }
```

If the key `use-dfrac` is active, we must reset the variables `\l__spinttent_spmQ_wrap_parens_l_tl` and `\l__spinttent_spmQ_wrap_parens_r_tl` to `\Bigl/\Bigr` unconditionally.

```
2841 \bool_if:NT \l__spinttent_spmQ_use_dfrac_bool
2842 {
2843   \tl_set:Nn \l__spinttent_spmQ_wrap_parens_l_tl { \Bigl \lparen }
2844   \tl_set:Nn \l__spinttent_spmQ_wrap_parens_r_tl { \Bigr \rparen }
2845 }
2846 }
```

(End of definition for `\__spinttent_spmQ_set_wrap_parens_size:`.)

```
\__spinttent_spmQ_start_wrap_parens:
```

```
\__spinttent_spmQ_stop_wrap_parens:
```

The functions `\__spinttent_spmQ_start_wrap_parens:` and `\__spinttent_spmQ_stop_wrap_parens:` open and close the parentheses around the “mixed number” when the key `wrap-parens` is active. They first call `\__spinttent_spmQ_set_wrap_parens_size:` to determine the correct size for the current mathematical style, then print the appropriate parenthesis stored in `\l__spinttent_spmQ_wrap_parens_l_tl` or `\l__spinttent_spmQ_wrap_parens_r_tl`.

If the variable `\l__spinttent_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.

```
2847 \cs_new_protected:Nn \__spinttent_spmQ_start_wrap_parens:
2848 {
2849   \bool_if:NT \l__spinttent_spmQ_wrap_parens_bool
2850   {
2851     \__spinttent_spmQ_set_wrap_parens_size:
2852     \bool_if:NTF \l__spinttent_spmQ_read_parens_bool
2853     { \l__spinttent_spmQ_wrap_parens_l_tl }
2854     { \MathMLintent { :silent } { \l__spinttent_spmQ_wrap_parens_l_tl } }
```

```

2855     }
2856   }
2857   \cs_new_protected:Nn \__spinttent_spmQ_stop_wrap_parens:
2858   {
2859     \bool_if:NT \__spinttent_spmQ_wrap_parens_bool
2860     {
2861       \__spinttent_spmQ_set_wrap_parens_size:
2862       \bool_if:NTF \__spinttent_spmQ_read_parens_bool
2863       { \__spinttent_spmQ_wrap_parens_r_tl }
2864       { \MathMLintent { :silent } { \__spinttent_spmQ_wrap_parens_r_tl } }
2865     }
2866   }

```

(End of definition for __spinttent_spmQ_start_wrap_parens: and __spinttent_spmQ_stop_wrap_parens:.)

`\__spinttent_spmQ_set_join_word:` The function `\__spinttent_spmQ_set_join_word:` sets the grammatical connector between the integer part and the fraction by branching on the value stored in `\__spinttent_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__spinttent_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.

```

2867   \cs_new_protected:Nn \__spinttent_spmQ_set_join_word:
2868   {
2869     \str_case:Vn \__spinttent_spmQ_join_word_str
2870     {
2871       { entero } { \MathMLintent { _entero } { \c__spinttent_invisible_sep_int } }
2872       { enteros } { \MathMLintent { _enteros } { \c__spinttent_invisible_sep_int } }
2873       { y } { \MathMLintent { _y- } { \c__spinttent_invisible_sep_int } }
2874     }
2875   }

```

(End of definition for __spinttent_spmQ_set_join_word:.)

`\__spinttent_spmQ_print_scale_int:` The function `\__spinttent_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 `\__spinttent_luafun_spmQ_scale_int:Vn`, it temporarily disables `luamml` by saving the current value of `\l__luamml_flag_int` into `\__spinttent_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 `\__spinttent_saved_luamml_flag_int`.

The Lua function receives the integer digits from `\__spinttent_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.

```

2876   \cs_new_protected:Nn \__spinttent_spmQ_print_scale_int:
2877   {
2878     \int_set_eq:Nc \__spinttent_saved_luamml_flag_int { \l__luamml_flag_int }
2879     \int_set:Nn \__luamml_flag_int { 0 }
2880     \bool_if:NTF \__spinttent_spmQ_use_dfrac_bool
2881     { \__spinttent_luafun_spmQ_scale_int:Vn \__spinttent_luaset_part_int_tl { dfraction } }
2882     { \__spinttent_luafun_spmQ_scale_int:Vn \__spinttent_luaset_part_int_tl { fraction } }
2883     \int_set_eq:cN { \__luamml_flag_int } \__spinttent_saved_luamml_flag_int
2884   }

```

(End of definition for __spinttent_spmQ_print_scale_int:.)

`\__spinttent_spmQ_print_scale_int:n` The function `\__spinttent_spmQ_print_scale_int:n` takes the integer part of the “mixed number” passed in ‘#1’. It first calls `\__spinttent_luafun_clean_split_and_set:n` on ‘#1’ to parse and set the shared variables from Lua, then checks the status of `\__spinttent_luaset_has_integer_str`.

If the status is `true`, it inspects `\__spinttent_luaset_sign_tl`. When the sign is *empty*, it delegates to `\__spinttent_spmQ_print_scale_int:` to render the scaled integer. When a sign is present, it sets `\__spinttent_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 `\__spinttent_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.

```

2885   \cs_new_protected:Nn \__spinttent_spmQ_print_scale_int:n

```

```

2886 {
2887   \__spintent_luafun_clean_split_and_set:n { \exp_not:n {#1} }
2888   \str_if_eq:VnTF \l__spintent_luaset_has_integer_str { true }
2889   {
2890     \tl_if_empty:NTF \l__spintent_luaset_sign_tl
2891     {
2892       \__spintent_spmQ_print_scale_int:
2893     }
2894     {
2895       \str_if_eq:VnTF \l__spintent_luaset_sign_tl { + }
2896       {
2897         \str_set:Nn
2898           \l__spintent_spmQ_intent_read_sign_str { _más:prefix($e) }
2899       }
2900       {
2901         \str_set:Nn
2902           \l__spintent_spmQ_intent_read_sign_str { _menos:prefix($e) }
2903       }
2904       \MathMLintent { \l__spintent_spmQ_intent_read_sign_str }
2905       {
2906         \l__spintent_luaset_sign_tl \c__spintent_invisible_sep_int
2907         \MathMLarg { e } { \__spintent_spmQ_print_scale_int: }
2908       }
2909     }
2910   }
2911   { \msg_error:nnn { spintent } { spmQ-not-integer } {#1} }
2912 }

```

(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.

```

2913 \cs_new_protected:Nn \__spintent_spmQ_print_spnZ_dfrac:nn
2914 {
2915   \bool_if:NTF \l__spintent_spmQ_use_spnZ_bool
2916   {
2917     \group_begin:
2918     \cs_set_protected:Nn \__spintent_invisible_sep:
2919       {
2920         \__spintent_luafun_invisible_sep_mark:
2921       }
2922     \use:c { \bool_if:NT \l__spintent_spmQ_use_dfrac_bool { d } frac }
2923     { \spnZ {#1} } { \spnZ {#2} }
2924   \group_end:
2925   }
2926   {
2927     \use:c { \bool_if:NT \l__spintent_spmQ_use_dfrac_bool { d } frac }
2928     {#1} {#2}
2929   }
2930 }

```

(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:`.

```

2931 \cs_new_protected:Nn \__spintent_spmQ_print:nnn

```

```

2932 {
2933   \__spintent_spmQ_start_wrap_parens:
2934   \__spintent_spmQ_print_scale_int:n {#1}
2935   \__spintent_spmQ_set_join_word:
2936   \__spintent_spmQ_print_spnZ_dfrac:nn {#2} {#3}
2937   \__spintent_spmQ_stop_wrap_parens:
2938 }

```

(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 `\__spintent_luaaset_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 `\__spintent_spmQ_error_bool` to `true`; the arguments are rejected whatever the value of the key `silent`.

```

2939 \cs_new_protected:Nn \__spintent_spmQ_check_part:nn
2940 {
2941   \__spintent_luafun_clean_split_and_set:n { \exp_not:n {#2} }
2942   \str_if_eq:VnTF \__spintent_luaaset_has_integer_str { true }
2943   {
2944     \bool_lazy_or:nnTF
2945     { \tl_if_empty_p:N \__spintent_luaaset_sign_tl }
2946     { \str_if_eq_p:nn {#1} { int } }
2947     {
2948       \str_if_eq:nnT {#1} { den }
2949       {
2950         \int_compare:nNnT { \__spintent_luaaset_part_int_tl } = { 0 }
2951         {
2952           \msg_error:nn { spintent } { spmQ-zero-denominator }
2953           \bool_set_true:N \__spintent_spmQ_error_bool
2954         }
2955       }
2956     }
2957     {
2958       \msg_error:nnn { spintent } { spmQ-signed-part } {#2}
2959       \bool_set_true:N \__spintent_spmQ_error_bool
2960     }
2961   }
2962   {
2963     \msg_error:nnn { spintent } { spmQ-not-integer } {#2}
2964     \bool_set_true:N \__spintent_spmQ_error_bool
2965   }
2966 }

```

(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`.

```

2967 \cs_new_protected:Nn \__spintent_spmQ_exec:nnnn
2968 {
2969   \keys_set:nn { spintent / spmQ } {#1}
2970   \bool_set_false:N \__spintent_spmQ_error_bool
2971   \__spintent_spmQ_check_part:nn { int } {#2}
2972   \__spintent_spmQ_check_part:nn { num } {#3}
2973   \__spintent_spmQ_check_part:nn { den } {#4}
2974   \bool_if:NF \__spintent_spmQ_error_bool
2975   { \__spintent_spmQ_mathml:nnn {#2} {#3} {#4} }
2976 }

```

(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.

```

2977 \cs_new_protected:Nn \__spintent_spmQ_mathml:nnn
2978 {
2979   \bool_if:NTF \__spintent_spmQ_silent_bool
2980   {
2981     \__spintent_invisible_sep:
2982     \MathMLintent { _- }
2983     {
2984       \__spintent_inner_sep:
2985       \__spintent_spmQ_print:nnn {#1} {#2} {#3}
2986     }
2987   }
2988   { \__spintent_spmQ_print:nnn {#1} {#2} {#3} }
2989 }

```

(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*.

```

2990 \NewDocumentCommand \spmQ { O{} m m m }
2991 {
2992   \mode_if_math:TF
2993   {
2994     \group_begin:
2995     \__spintent_spmQ_exec:nnnn {#1} {#2} {#3} {#4}
2996     \group_end:
2997   }
2998   { \msg_error:nnn { spintent } { need-math-mode } { \spmQ } }
2999 }

```

(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 `\__spintent_geo_luaset_error_str`, `\__spintent_geo_luaset_intent_str`, `\__spintent_geo_luaset_print_tl`, `\__spintent_geo_luaset_letra_str`, `\__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 `\__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

`\__spintent_geo_luaset_error_str` These variables are shared by all the commands of the family and are set by the Lua module `spintent.lua`.
`\__spintent_geo_luaset_intent_str` The variable `\__spintent_geo_luaset_error_str` stores `true` or `false` if the *argument* is invalid. The variable `\__spintent_geo_luaset_intent_str` stores the body of the intent and `\__spintent_geo_luaset_print_tl` stores what is printed.

```

3000 \str_new:N \__spintent_geo_luaset_error_str
3001 \str_new:N \__spintent_geo_luaset_intent_str
3002 \tl_new:N \__spintent_geo_luaset_print_tl

```

(End of definition for `\__spintent_geo_luaset_error_str`, `\__spintent_geo_luaset_intent_str`, and `\__spintent_geo_luaset_print_tl`.)

`\__spintent_geo_luaset_points_clist` The variable `\__spintent_geo_luaset_points_clist` stores the points of the *argument* separated by commas and not joined, such as A, B, C. The variable `\__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.
`\__spintent_geo_luaset_letra_str` When the *argument* is a single point or name, `\__spintent_geo_luaset_letra_str` stores its letter,
`\__spintent_geo_luaset_sub_str` `\__spintent_geo_luaset_sub_str` stores its subscript or prime and `\__spintent_geo_luaset_tipo_str` stores sub or prima, and is empty if there is none.

```

3003 \clist_new:N \__spintent_geo_luaset_points_clist
3004 \str_new:N \__spintent_geo_luaset_is_name_str
3005 \str_new:N \__spintent_geo_luaset_letra_str
3006 \str_new:N \__spintent_geo_luaset_sub_str
3007 \str_new:N \__spintent_geo_luaset_tipo_str

```

(End of definition for `\l__spinttent_geo_luaset_points_clist` and others.)

```
\l__spinttent_geo_luaset_poly_sym_str
\l__spinttent_geo_luaset_concept_str
```

The variable `\l__spinttent_geo_luaset_poly_sym_str` stores `triangle`, `square` or nothing, according to the number of vertices of the polygon. The variable `\l__spinttent_geo_luaset_concept_str` stores the name of the polygon or of the concept, such as `triángulo` or `área-del-triángulo`.

```
3008 \str_new:N \l__spinttent_geo_luaset_poly_sym_str
3009 \str_new:N \l__spinttent_geo_luaset_concept_str
```

(End of definition for `\l__spinttent_geo_luaset_poly_sym_str` and `\l__spinttent_geo_luaset_concept_str`.)

```
\l__spinttent_geo_luaset_angulo_case_str
\l__spinttent_geo_luaset_nombre_es_num_str
```

The variable `\l__spinttent_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__spinttent_geo_luaset_nombre_es_num` stores `true` when that name is a number.

```
3010 \str_new:N \l__spinttent_geo_luaset_angulo_case_str
3011 \str_new:N \l__spinttent_geo_luaset_nombre_es_num_str
```

(End of definition for `\l__spinttent_geo_luaset_angulo_case_str` and `\l__spinttent_geo_luaset_nombre_es_num_str`.)

```
\l__spinttent_geo_luaset_letra_word_str
\l__spinttent_geo_luaset_letra_conector_str
\l__spinttent_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__spinttent_geo_luaset_letra_word_str` stores its word, such as `figura` or `lateral`, and `\l__spinttent_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__spinttent_geo_luaset_sub_spoken_str` stores the subscript as it is spoken.

```
3012 \str_new:N \l__spinttent_geo_luaset_letra_word_str
3013 \str_new:N \l__spinttent_geo_luaset_letra_conector_str
3014 \str_new:N \l__spinttent_geo_luaset_sub_spoken_str
```

(End of definition for `\l__spinttent_geo_luaset_letra_word_str`, `\l__spinttent_geo_luaset_letra_conector_str`, and `\l__spinttent_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__spinttent_spcoord_luaset_error_str
\l__spinttent_spcoord_luaset_sep_str
```

These variables are set by the function `\__spinttent_luafun_spcoord_parse_and_set:n`. The variable `\l__spinttent_spcoord_luaset_error_str` stores `true` or `false` if the `{⟨argument⟩}` is invalid. The variable `\l__spinttent_spcoord_luaset_sep_str` stores `,` or `;`, whichever the argument actually used.

```
3015 \str_new:N \l__spinttent_spcoord_luaset_error_str
3016 \str_new:N \l__spinttent_spcoord_luaset_sep_str
```

(End of definition for `\l__spinttent_spcoord_luaset_error_str` and `\l__spinttent_spcoord_luaset_sep_str`.)

```
\l__spinttent_spcoord_luaset_x_sign_str
\l__spinttent_spcoord_luaset_x_type_str
\l__spinttent_spcoord_luaset_x_value_str
\l__spinttent_spcoord_luaset_x_has_frac_str
\l__spinttent_spcoord_luaset_x_composite_str
\l__spinttent_spcoord_luaset_y_sign_str
\l__spinttent_spcoord_luaset_y_type_str
\l__spinttent_spcoord_luaset_y_value_str
\l__spinttent_spcoord_luaset_y_has_frac_str
\l__spinttent_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.

```
3017 \clist_map_inline:nn { x , y }
3018 {
3019   \str_new:c { \l__spinttent_spcoord_luaset_ #1 _sign_str      }
3020   \str_new:c { \l__spinttent_spcoord_luaset_ #1 _type_str      }
3021   \str_new:c { \l__spinttent_spcoord_luaset_ #1 _value_str     }
3022   \str_new:c { \l__spinttent_spcoord_luaset_ #1 _has_frac_str  }
3023   \str_new:c { \l__spinttent_spcoord_luaset_ #1 _composite_str }
3024 }
```

(End of definition for `\l__spinttent_spcoord_luaset_x_sign_str` and others.)

```
\l__spinttent_spcoord_raw_arg_tl
\l__spinttent_spcoord_intent_str
\l__spinttent_spcoord_wrap_parens_l_tl
\l__spinttent_spcoord_wrap_parens_r_tl
```

The variable `\l__spinttent_spcoord_raw_arg_tl` holds the `{⟨argument⟩}` exactly as given. The variable `\l__spinttent_spcoord_intent_str` holds the `:infix` word for the separator `_coma:infix` or `_punto-y-coma:infix`. The variables `\l__spinttent_spcoord_wrap_parens_l_tl`/`\l__spinttent_spcoord_wrap_parens_r_tl` hold the opening/closing delimiter, sized by the function `\__spinttent_spcoord_set_wrap_parens_size:..`

```
3025 \tl_new:N \l__spinttent_spcoord_raw_arg_tl
3026 \str_new:N \l__spinttent_spcoord_intent_str
3027 \tl_new:N \l__spinttent_spcoord_wrap_parens_l_tl
3028 \tl_new:N \l__spinttent_spcoord_wrap_parens_r_tl
```

(End of definition for `\l__spinttent_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 $\langle argument \rangle$ does not have the form *abscissa,ordinate* or *abscissa;ordinate*.

```

3029 \msg_new:nnnn { spintent } { spcoord-invalid-arg }
3030 {
3031   Invalid~argument~'#1'~for~'\token_to_str:N \spcoord'~\msg_line_context:.
3032 }
3033 {
3034   The~argument~must~have~the~form~'abscissa,~ordinate'~or~'abscissa;~ordinate'.
3035 }

```

(End of definition for `spcoord-invalid-arg`.)

15.37.3 Definition of keys for \spcoord

silent
unknown

We define the $\langle key \rangle$ `silent`.

```

3036 \keys_define:nn { spintent / spcoord }
3037 {
3038   silent .bool_set:N = \l__spintent_spcoord_silent_bool,
3039   silent .initial:n = false,
3040   silent .value_required:n = true,
3041   unknown .code:n =
3042     {
3043       \str_set:Nn \l__spintent_command_name_str { spcoord }
3044       \__spintent_unknown_keys_cmd:n {#1}
3045     },
3046 }

```

(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.

```

3047 \cs_new_protected:Nn \__spintent_spcoord_set_wrap_parens_size:
3048 {
3049   \tl_set:Nn \l__spintent_spcoord_wrap_parens_l_tl { \lparen }
3050   \tl_set:Nn \l__spintent_spcoord_wrap_parens_r_tl { \rparen }
3051   \bool_if:nT
3052     {
3053       \bool_lazy_or_p:nn
3054         {
3055           \bool_not_p:n
3056             { \str_if_eq_p:Vn \l__spintent_spcoord_luaset_x_has_frac_str { none } }
3057         }
3058         {
3059           \bool_not_p:n
3060             { \str_if_eq_p:Vn \l__spintent_spcoord_luaset_y_has_frac_str { none } }
3061         }
3062     }
3063   {
3064     \int_case:nnF { \mathstyle }
3065     {
3066       { 0 }
3067       {
3068         \tl_set:Nn \l__spintent_spcoord_wrap_parens_l_tl { \Bigl \lparen }
3069         \tl_set:Nn \l__spintent_spcoord_wrap_parens_r_tl { \Bigr \rparen }
3070       }
3071       { 1 }
3072       {
3073         \tl_set:Nn \l__spintent_spcoord_wrap_parens_l_tl { \Bigl \lparen }
3074         \tl_set:Nn \l__spintent_spcoord_wrap_parens_r_tl { \Bigr \rparen }
3075       }
3076       { 2 }
3077       {
3078         \tl_set:Nn \l__spintent_spcoord_wrap_parens_l_tl { \bigl \lparen }
3079         \tl_set:Nn \l__spintent_spcoord_wrap_parens_r_tl { \bigr \rparen }

```

```

3080         }
3081     { 3 }
3082     {
3083         \tl_set:Nn \l__spinttent_spcoord_wrap_parens_l_tl { \bigl \lparen }
3084         \tl_set:Nn \l__spinttent_spcoord_wrap_parens_r_tl { \bigr \rparen }
3085     }
3086 }
3087 { }
3088 \bool_if:nT
3089 {
3090     \bool_lazy_or_p:nn
3091     { \str_if_eq_p:Vn \l__spinttent_spcoord_luaset_x_has_frac_str { dfraction } }
3092     { \str_if_eq_p:Vn \l__spinttent_spcoord_luaset_y_has_frac_str { dfraction } }
3093 }
3094 {
3095     \tl_set:Nn \l__spinttent_spcoord_wrap_parens_l_tl { \Bigl \lparen }
3096     \tl_set:Nn \l__spinttent_spcoord_wrap_parens_r_tl { \Bigr \rparen }
3097 }
3098 }
3099 }

```

(End of definition for `\__spinttent_spcoord_set_wrap_parens_size:`)

`\__spinttent_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’.

```

3100 \cs_new_protected:Nn \__spinttent_spcoord_value:n
3101 {
3102     \str_if_eq:eeTF
3103     { \str_use:c { \l__spinttent_spcoord_luaset_ #1 _type_str } } { number }
3104     {
3105         \exp_args:Ne \spnum { \str_use:c { \l__spinttent_spcoord_luaset_ #1 _value_str } }
3106     }
3107     { \str_use:c { \l__spinttent_spcoord_luaset_ #1 _value_str } }
3108 }

```

(End of definition for `\__spinttent_spcoord_value:n`)

`\__spinttent_spcoord_print:n`

With the variable `_sign_str` of the endpoint in ‘#1’ equal to `+` or `-`, it wraps the sign and `\__spinttent_spcoord_value:n` with `_más:prefix($vp)` or `_menos:prefix($vn)`, referencing the value with `With no sign`, it calls `\__spinttent_spcoord_value:n` directly. Receives `x` or `y` in ‘#1’.

```

3109 \cs_new_protected:Nn \__spinttent_spcoord_print:n
3110 {
3111     \str_case:enF { \str_use:c { \l__spinttent_spcoord_luaset_ #1 _sign_str } }
3112     {
3113         { + }
3114         {
3115             \MathMLintent { _más:prefix($vp) }
3116             {
3117                 \str_use:c { \l__spinttent_spcoord_luaset_ #1 _sign_str }
3118                 \MathMLarg{ vp } { \__spinttent_spcoord_value:n {#1} }
3119             }
3120         }
3121         { - }
3122         {
3123             \MathMLintent { _menos:prefix($vn) }
3124             {
3125                 \str_use:c { \l__spinttent_spcoord_luaset_ #1 _sign_str }
3126                 \MathMLarg{ vn } { \__spinttent_spcoord_value:n {#1} }
3127             }
3128         }
3129     }
3130     { \__spinttent_spcoord_value:n {#1} }
3131 }

```

(End of definition for `\__spinttent_spcoord_print:n`)

`\__spinttent_spcoord_inner_sep:n`

Receives `x` or `y` in ‘#1’. With the variable `_composite_str` of that endpoint equal to `true`, places `\__spinttent_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.

```

3132 \cs_new_protected:Nn \__spinttent_spcoord_inner_sep:n

```

```

3133 {
3134   \str_if_eq:eeT
3135   { \str_use:c { \l__spinttent_spcoord_luaset_ #1 _composite_str } } { true }
3136   { \l__spinttent_inner_sep: }
3137 }

```

(End of definition for `\l__spinttent_spcoord_inner_sep:n`.)

`\l__spinttent_spcoord_intent_infix:` Sets `\l__spinttent_spcoord_intent_str` to `_coma:infix` or `_punto-y-coma:infix`, according to `\l__spinttent_spcoord_luaset_sep_str`, and wraps both sides, each referenced with `with` it. The names `$x` and `$y` stand for the output of `\l__spinttent_spcoord_print:n`.

```

3138 \cs_new_protected:Nn \l__spinttent_spcoord_intent_infix:
3139 {
3140   \str_if_eq:VnTF \l__spinttent_spcoord_luaset_sep_str { ; }
3141   { \str_set:Nn \l__spinttent_spcoord_intent_str { _punto-y-coma:infix } }
3142   { \str_set:Nn \l__spinttent_spcoord_intent_str { _coma:infix } }
3143   \MathMLintent { \l__spinttent_spcoord_intent_str ($x, $y) }
3144   {
3145     \MathMLarg{x}
3146     { \l__spinttent_spcoord_inner_sep:n { x } \l__spinttent_spcoord_print:n { x } }
3147     \MathMLintent { :silent } { \l__spinttent_spcoord_luaset_sep_str }
3148     \MathMLarg{y}
3149     { \l__spinttent_spcoord_inner_sep:n { y } \l__spinttent_spcoord_print:n { y } }
3150   }
3151 }

```

(End of definition for `\l__spinttent_spcoord_intent_infix:.`)

`\l__spinttent_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__spinttent_spcoord_raw_arg_tl` is typeset as given. Otherwise, the parentheses are silenced individually and `\l__spinttent_spcoord_intent_infix:` provides the reading between them.

```

3152 \cs_new_protected:Nn \l__spinttent_spcoord_mathml:N
3153 {
3154   \bool_if:NTF #1
3155   {
3156     \MathMLintent { _- }
3157     {
3158       \l__spinttent_spcoord_wrap_parens_l_tl
3159       \l__spinttent_spcoord_raw_arg_tl
3160       \l__spinttent_spcoord_wrap_parens_r_tl
3161     }
3162   }
3163   {
3164     \MathMLintent { :silent } { \l__spinttent_spcoord_wrap_parens_l_tl }
3165     \l__spinttent_spcoord_intent_infix:
3166     \MathMLintent { :silent } { \l__spinttent_spcoord_wrap_parens_r_tl }
3167   }
3168 }

```

(End of definition for `\l__spinttent_spcoord_mathml:N`.)

`\l__spinttent_spcoord_exec:nn` Sets the *(keys)* in ‘`#1`’ and `\l__spinttent_spcoord_raw_arg_tl` to ‘`#2`’. Calls `\l__spinttent_luafun_spcoord_parse_and_set:n` and reports `spcoord-invalid-arg` on error, whatever the value of `silent`. Then it sizes the parentheses and, if there was no error, calls `\l__spinttent_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.

```

3169 \cs_new_protected:Nn \l__spinttent_spcoord_exec:nn
3170 {
3171   \keys_set:nn { spinttent / spcoord } { #1 }
3172   \tl_set:Nn \l__spinttent_spcoord_raw_arg_tl { #2 }
3173   \l__spinttent_luafun_spcoord_parse_and_set:n { \exp_not:n { #2 } }
3174   \str_if_eq:VnT \l__spinttent_spcoord_luaset_error_str { true }
3175   { \msg_error:nne { spinttent } { spcoord-invalid-arg } { \exp_not:n { #2 } } }
3176   \l__spinttent_spcoord_set_wrap_parens_size:
3177   \str_if_eq:VnF \l__spinttent_spcoord_luaset_error_str { true }
3178   {
3179     \mathord
3180     {

```

```

3181         % prevent "intervalo"
3182         \MathMLintent { :silent } { \c__spinttent_invisible_sep_int }
3183         \__spinttent_spcoord_mathml:N \l__spinttent_spcoord_silent_bool
3184     }
3185 }
3186 }

```

(End of definition for `\__spinttent_spcoord_exec:nn`.)

`\spcoord` The public command `\spcoord` enforces *math mode*, opens a local group, and delegates to `\__spinttent_spcoord_exec:nn`.

```

3187 \NewDocumentCommand \spcoord { 0{} m }
3188 {
3189     \mode_if_math:TF
3190     {
3191         \group_begin:
3192         \attribute \l__spinttent_scope_attr_int = 1
3193         \__spinttent_spcoord_exec:nn {#1} {#2}
3194         \group_end:
3195     }
3196     { \msg_error:nnn { spinttent } { need-math-mode } { \spcoord } }
3197 }

```

(End of definition for `\spcoord`. This function is documented on page 24.)

15.38 The command `\spinterval`

The user command `\spinterval` receives the $\{\langle argument \rangle\}$ 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`

These variables are set by the function `\__spinttent_luafun_spinterval_parse_and_set:n`. The variable `\l__spinttent_spinterval_luaset_error_str` stores *true* or *false* if the $\{\langle argument \rangle\}$ is invalid. The variable `\l__spinttent_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.

```

3198 \str_new:N \l__spinttent_spinterval_luaset_error_str
3199 \str_new:N \l__spinttent_spinterval_luaset_error_code_str

```

(End of definition for `\l__spinttent_spinterval_luaset_error_str` and `\l__spinttent_spinterval_luaset_error_code_str`.)

The variables `\l__spinttent_spinterval_luaset_left_sym_str` and `\l__spinttent_spinterval_luaset_right_sym_str` store the delimiters of the interval as they were written. The variable `\l__spinttent_spinterval_luaset_type_str` stores *open*, *closed*, *open-closed* or *closed-open*. The variable `\l__spinttent_spinterval_luaset_sep_str` stores the separator, *,* or *;*. The variable `\l__spinttent_spinterval_luaset_tall_str` stores *dfrac*, *frac* or *none*, according to the tallest fraction in the endpoints.

```

3200 \str_new:N \l__spinttent_spinterval_luaset_left_sym_str
3201 \str_new:N \l__spinttent_spinterval_luaset_right_sym_str
3202 \str_new:N \l__spinttent_spinterval_luaset_type_str
3203 \str_new:N \l__spinttent_spinterval_luaset_sep_str
3204 \str_new:N \l__spinttent_spinterval_luaset_tall_str

```

(End of definition for `\l__spinttent_spinterval_luaset_left_sym_str` and others.)

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.

```

3205 \clist_map_inline:nn { a , b }
3206 {
3207     \str_new:c { \l__spinttent_spinterval_luaset_ #1 _sign_str }
3208     \str_new:c { \l__spinttent_spinterval_luaset_ #1 _type_str }
3209     \str_new:c { \l__spinttent_spinterval_luaset_ #1 _value_str }
3210     \str_new:c { \l__spinttent_spinterval_luaset_ #1 _inf_str }

```

```

3211     \str_new:c { l__spinttent_spinterval_luaset_ #1 _tall_str      }
3212     \str_new:c { l__spinttent_spinterval_luaset_ #1 _composite_str }
3213     \str_new:c { l__spinttent_spinterval_luaset_ #1 _integer_str   }
3214     \str_new:c { l__spinttent_spinterval_luaset_ #1 _short_str     }
3215     \str_new:c { l__spinttent_spinterval_luaset_ #1 _braced_str    }
3216 }

```

(End of definition for `\l__spinttent_spinterval_luaset_a_sign_str` and others.)

```

\l__spinttent_spinterval_prefix_tl
\l__spinttent_spinterval_read_arg_tl
\l__spinttent_spinterval_mode_str
\l__spinttent_spinterval_size_str

```

The variables `\l__spinttent_spinterval_mode_str`, `\l__spinttent_spinterval_prefix_tl`, `\l__spinttent_spinterval_read_arg_tl` and `\l__spinttent_spinterval_size_str` store the values set by the keys `read-cmd`, `prefix`, `read-arg` and `size`.

```

3217 \tl_new:N \l__spinttent_spinterval_prefix_tl
3218 \tl_new:N \l__spinttent_spinterval_read_arg_tl
3219 \str_new:N \l__spinttent_spinterval_mode_str
3220 \str_new:N \l__spinttent_spinterval_size_str

```

(End of definition for `\l__spinttent_spinterval_prefix_tl` and others.)

```
\l__spinttent_spinterval_reason_str
```

The variable `\l__spinttent_spinterval_reason_str` stores the text of the error reason, set from the error code returned by the Lua module.

```
3221 \str_new:N \l__spinttent_spinterval_reason_str
```

(End of definition for `\l__spinttent_spinterval_reason_str`.)

```

\l__spinttent_spinterval_word_w_tl
\l__spinttent_spinterval_word_g_tl
\l__spinttent_spinterval_intent_arg_tl
\l__spinttent_spinterval_intent_group_tl

```

The variables `\l__spinttent_spinterval_word_w_tl` and `\l__spinttent_spinterval_word_g_tl` store the word that names the interval and the word that joins its endpoints. The variables `\l__spinttent_spinterval_intent_arg_tl` and `\l__spinttent_spinterval_intent_group_tl` store the intents of the whole interval and of the group of endpoints.

```

3222 \tl_new:N \l__spinttent_spinterval_word_w_tl
3223 \tl_new:N \l__spinttent_spinterval_word_g_tl
3224 \tl_new:N \l__spinttent_spinterval_intent_arg_tl
3225 \tl_new:N \l__spinttent_spinterval_intent_group_tl

```

(End of definition for `\l__spinttent_spinterval_word_w_tl` and others.)

```

\l__spinttent_spinterval_sym_l_tl
\l__spinttent_spinterval_sym_r_tl
\l__spinttent_spinterval_size_l_tl
\l__spinttent_spinterval_size_r_tl
\l__spinttent_spinterval_delim_l_tl
\l__spinttent_spinterval_delim_r_tl

```

The variables `\l__spinttent_spinterval_sym_l_tl` and `\l__spinttent_spinterval_sym_r_tl` store the delimiter symbols, the variables `\l__spinttent_spinterval_size_l_tl` and `\l__spinttent_spinterval_size_r_tl` store their size commands, and the variables `\l__spinttent_spinterval_delim_l_tl` and `\l__spinttent_spinterval_delim_r_tl` store the delimiters ready to be typeset.

```

3226 \tl_new:N \l__spinttent_spinterval_sym_l_tl
3227 \tl_new:N \l__spinttent_spinterval_sym_r_tl
3228 \tl_new:N \l__spinttent_spinterval_size_l_tl
3229 \tl_new:N \l__spinttent_spinterval_size_r_tl
3230 \tl_new:N \l__spinttent_spinterval_delim_l_tl
3231 \tl_new:N \l__spinttent_spinterval_delim_r_tl

```

(End of definition for `\l__spinttent_spinterval_sym_l_tl` and others.)

```
\l__spinttent_spinterval_tmp_tl
```

The variable `\l__spinttent_spinterval_tmp_tl` is a scratch variable.

```
3232 \tl_new:N \l__spinttent_spinterval_tmp_tl
```

(End of definition for `\l__spinttent_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 $\langle \textit{argument} \rangle$; it reports the argument and the reason set by `\__spinttent_spinterval_set_reason:`.

```

3233 \msg_new:nnnn { spinttent } { spinterval-invalid-arg }
3234 {
3235   Invalid~argument~'#1'~for~'\token_to_str:N \spinterval'~(#2)~
3236   \msg_line_context:.
3237 }
3238 {
3239   Use~'(a;~b)'~or~'(a,~b)',~with~'('~,~'['~or~']'~on~the~left~and~')',~']'~or~'[~
3240   on~the~right.
3241 }

```

(End of definition for `spinterval-invalid-arg`.)

15.38.3 Definition of keys for \spinterval

We define the *(keys)* read-cmd, prefix, read-arg, size, silent and concept.

```

3242 \keys_define:nn { spintent / spinterval }
3243 {
3244   read-cmd .choices:nn = { school , mathcat }
3245   {
3246     \int_case:nn { \l_keys_choice_int }
3247     {
3248       {1} { \str_set:Nn \l__spintent_spinterval_mode_str { school } }
3249       {2} { \str_set:Nn \l__spintent_spinterval_mode_str { mathcat } }
3250     }
3251   },
3252   read-cmd .initial:n = school,
3253   read-cmd .value_required:n = true,
3254   read-cmd / unknown .code:n =
3255     \msg_error:nneee { spintent } { unknown-choice }
3256     { read-cmd } { school,~mathcat } { \exp_not:n {#1} },
3257   prefix .code:n =
3258     { \__spintent_sanitizize_affix:Nn \l__spintent_spinterval_prefix_tl {#1} },
3259   prefix .initial:n = ,
3260   prefix .value_required:n = true,
3261   read-arg .code:n =
3262     { \__spintent_sanitizize_affix:Nn \l__spintent_spinterval_read_arg_tl {#1} },
3263   read-arg .initial:n = ,
3264   read-arg .value_required:n = true,
3265   size .choices:nn = { auto , none , big , Big , bigg , Bigg }
3266   {
3267     \int_case:nn { \l_keys_choice_int }
3268     {
3269       {1} { \str_set:Nn \l__spintent_spinterval_size_str { auto } }
3270       {2} { \str_set:Nn \l__spintent_spinterval_size_str { none } }
3271       {3} { \str_set:Nn \l__spintent_spinterval_size_str { big } }
3272       {4} { \str_set:Nn \l__spintent_spinterval_size_str { Big } }
3273       {5} { \str_set:Nn \l__spintent_spinterval_size_str { bigg } }
3274       {6} { \str_set:Nn \l__spintent_spinterval_size_str { Bigg } }
3275     }
3276   },
3277   size .initial:n = auto,
3278   size .value_required:n = true,
3279   size / unknown .code:n =
3280     \msg_error:nneee { spintent } { unknown-choice }
3281     { size } { auto,~none,~big,~Big,~bigg,~Bigg } { \exp_not:n {#1} },
3282   silent .bool_set:N = \l__spintent_spinterval_silent_bool,
3283   silent .initial:n = false,
3284   silent .value_required:n = true,
3285   concept .bool_set:N = \l__spintent_spinterval_concept_bool,
3286   concept .initial:n = true,
3287   concept .value_required:n = true,
3288   unknown .code:n =
3289     {
3290       \str_set:Nn \l__spintent_command_name_str { spinterval }
3291       \__spintent_unknown_keys_cmd:n {#1}
3292     },
3293 }
```

(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.

```

3294 \cs_new_protected:Nn \__spintent_spinterval_set_reason:
3295 {
3296   \str_case:VnF \l__spintent_spinterval_luaset_error_code_str
3297   {
3298     { delim }
3299     {
3300       \str_set:Nn \l__spintent_spinterval_reason_str
3301       { it~must~start~with~'(' ,~'['~or~'{'~and~end~with~')' ,~']'~or~'}' }

```

```

3302     }
3303     { vacio }
3304     { \str_set:Nn \l__spinttent_spinterval_reason_str { an~endpoint~is~empty } }
3305     { sep }
3306     {
3307         \str_set:Nn \l__spinttent_spinterval_reason_str
3308             { there~is~no~separator:~use~a~comma~or~a~semicolon }
3309     }
3310     { ambiguo }
3311     {
3312         \str_set:Nn \l__spinttent_spinterval_reason_str
3313             {
3314                 more~than~one~separator:~use~a~single~semicolon~and~the~commas~are~
3315                 decimal
3316             }
3317     }
3318     { llaves }
3319     { \str_set:Nn \l__spinttent_spinterval_reason_str { unbalanced~braces } }
3320     { inf-orden }
3321     {
3322         \str_set:Nn \l__spinttent_spinterval_reason_str
3323             { minus~infinity~goes~on~the~left~and~infinity~on~the~right }
3324     }
3325     { inf-cerrado }
3326     {
3327         \str_set:Nn \l__spinttent_spinterval_reason_str
3328             {
3329                 an~infinite~endpoint~cannot~be~included:~use~'('~or~')'~on~the~left,~
3330                 '~or~'['~on~the~right
3331             }
3332     }
3333 }
3334 { \str_set:Nn \l__spinttent_spinterval_reason_str { invalid~argument } }
3335 }

```

(End of definition for `\l__spinttent_spinterval_set_reason:`)

`\l__spinttent_spinterval_set_delims:`

The internal function `\l__spinttent_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.

```

3336 \cs_new_protected:Nn \l__spinttent_spinterval_set_delims:
3337 {
3338     \str_case:Nn \l__spinttent_spinterval_luaset_left_sym_str
3339     {
3340         { ( ) } { \tl_set:Nn \l__spinttent_spinterval_sym_l_tl { \lparen } }
3341         { [ ] } { \tl_set:Nn \l__spinttent_spinterval_sym_l_tl { \lbrack } }
3342         { } } { \tl_set:Nn \l__spinttent_spinterval_sym_l_tl { \rbrack } }
3343     }
3344     \str_case:Nn \l__spinttent_spinterval_luaset_right_sym_str
3345     {
3346         { ) } { \tl_set:Nn \l__spinttent_spinterval_sym_r_tl { \rparen } }
3347         { [ ] } { \tl_set:Nn \l__spinttent_spinterval_sym_r_tl { \lbrack } }
3348         { } } { \tl_set:Nn \l__spinttent_spinterval_sym_r_tl { \rbrack } }
3349     }
3350     \tl_clear:N \l__spinttent_spinterval_size_l_tl
3351     \tl_clear:N \l__spinttent_spinterval_size_r_tl
3352     \str_case:Nn \l__spinttent_spinterval_size_str
3353     {
3354         { big }
3355         {
3356             \tl_set:Nn \l__spinttent_spinterval_size_l_tl { \bigl }
3357             \tl_set:Nn \l__spinttent_spinterval_size_r_tl { \bigr }
3358         }
3359         { Big }
3360         {
3361             \tl_set:Nn \l__spinttent_spinterval_size_l_tl { \Bigl }
3362             \tl_set:Nn \l__spinttent_spinterval_size_r_tl { \Bigr }
3363         }
3364     }

```

```

3364 { bigg }
3365 {
3366   \tl_set:Nn \l__spinttent_spinterval_size_l_tl { \biggl }
3367   \tl_set:Nn \l__spinttent_spinterval_size_r_tl { \biggr }
3368 }
3369 { Bigg }
3370 {
3371   \tl_set:Nn \l__spinttent_spinterval_size_l_tl { \Biggl }
3372   \tl_set:Nn \l__spinttent_spinterval_size_r_tl { \Biggr }
3373 }
3374 { auto }
3375 {
3376   \str_if_eq:VnF \l__spinttent_spinterval_luaset_tall_str { none }
3377   {
3378     \int_case:nnF { \mathstyle }
3379     {
3380       { 0 }
3381       {
3382         \tl_set:Nn \l__spinttent_spinterval_size_l_tl { \Bigl }
3383         \tl_set:Nn \l__spinttent_spinterval_size_r_tl { \Bigr }
3384       }
3385       { 1 }
3386       {
3387         \tl_set:Nn \l__spinttent_spinterval_size_l_tl { \Bigl }
3388         \tl_set:Nn \l__spinttent_spinterval_size_r_tl { \Bigr }
3389       }
3390       { 2 }
3391       {
3392         \tl_set:Nn \l__spinttent_spinterval_size_l_tl { \bigl }
3393         \tl_set:Nn \l__spinttent_spinterval_size_r_tl { \bigr }
3394       }
3395       { 3 }
3396       {
3397         \tl_set:Nn \l__spinttent_spinterval_size_l_tl { \bigl }
3398         \tl_set:Nn \l__spinttent_spinterval_size_r_tl { \bigr }
3399       }
3400     }
3401     { }
3402     \str_if_eq:VnT \l__spinttent_spinterval_luaset_tall_str { dfrac }
3403     {
3404       \tl_set:Nn \l__spinttent_spinterval_size_l_tl { \Bigl }
3405       \tl_set:Nn \l__spinttent_spinterval_size_r_tl { \Bigr }
3406     }
3407   }
3408 }
3409 }
3410 \tl_if_empty:NTF \l__spinttent_spinterval_size_l_tl
3411 {
3412   \tl_set:Ne \l__spinttent_spinterval_delim_l_tl
3413   { \exp_not:N \mathopen { \exp_not:V \l__spinttent_spinterval_sym_l_tl } }
3414   \tl_set:Ne \l__spinttent_spinterval_delim_r_tl
3415   { \exp_not:N \mathclose { \exp_not:V \l__spinttent_spinterval_sym_r_tl } }
3416 }
3417 {
3418   \tl_set:Ne \l__spinttent_spinterval_delim_l_tl
3419   {
3420     \exp_not:V \l__spinttent_spinterval_size_l_tl \exp_not:V
3421     \l__spinttent_spinterval_sym_l_tl
3422   }
3423   \tl_set:Ne \l__spinttent_spinterval_delim_r_tl
3424   {
3425     \exp_not:V \l__spinttent_spinterval_size_r_tl \exp_not:V
3426     \l__spinttent_spinterval_sym_r_tl
3427   }
3428 }
3429 }

```

(End of definition for \l__spinttent_spinterval_set_delims:.)

\l__spinttent_spinterval_set_intents:

The internal function `\l__spinttent_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`.

```

3430 \cs_new_protected:Nn \__spintent_spinterval_set_intents:
3431 {
3432   \str_case:Nn \__spintent_spinterval_luaset_type_str
3433   {
3434     { open }
3435     {
3436       \tl_set:Nn \__spintent_spinterval_word_w_tl { abierto-de }
3437       \tl_set:Nn \__spintent_spinterval_word_g_tl { a }
3438     }
3439     { closed }
3440     {
3441       \tl_set:Nn \__spintent_spinterval_word_w_tl { cerrado-de }
3442       \tl_set:Nn \__spintent_spinterval_word_g_tl { a }
3443     }
3444     { open-closed }
3445     {
3446       \tl_set:Nn \__spintent_spinterval_word_w_tl { abierto-en }
3447       \tl_set:Nn \__spintent_spinterval_word_g_tl { y-cerrado-en }
3448     }
3449     { closed-open }
3450     {
3451       \tl_set:Nn \__spintent_spinterval_word_w_tl { cerrado-en }
3452       \tl_set:Nn \__spintent_spinterval_word_g_tl { y-abierto-en }
3453     }
3454   }
3455   \tl_set:Nc \__spintent_spinterval_intent_group_tl
3456   { _ \__spintent_spinterval_word_g_tl :infix($a,$b) }
3457   \bool_if:NTF \__spintent_spinterval_silent_bool
3458   { \tl_set:Nn \__spintent_spinterval_intent_arg_tl { _- } }
3459   {
3460     \tl_if_empty:NTF \__spintent_spinterval_read_arg_tl
3461     {
3462       \tl_set:Nc \__spintent_spinterval_intent_arg_tl
3463       {
3464         -
3465         \bool_if:NT \__spintent_spinterval_concept_bool
3466         {
3467           \tl_if_empty:NF \__spintent_spinterval_prefix_tl
3468           { \tl_use:N \__spintent_spinterval_prefix_tl - }
3469           intervalo-
3470         }
3471         \__spintent_spinterval_word_w_tl :prefix($ab)
3472       }
3473     }
3474     {
3475       \tl_set:Nc \__spintent_spinterval_intent_arg_tl
3476       { _ \__spintent_spinterval_read_arg_tl }
3477     }
3478   }
3479 }

```

(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.

```

3480 \cs_new_protected:Nn \__spintent_spinterval_inner_sep:n
3481 {
3482   \str_if_eq:eeT
3483   { \str_use:c { \__spintent_spinterval_luaset_ #1 _composite_str } } { true }
3484   { \__spintent_inner_sep: }
3485 }

```

(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 `\__spinttent_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.

```

3486 \cs_new_protected:Nn \__spinttent_spinterval_plain_number:n
3487 {
3488   \str_if_eq:eeTF { \str_use:c { l__spinttent_spinterval_luaset_ #1 _integer_str } }
3489   { true }
3490   % keep "a,b" apart in mathml-AF
3491   { \MathMLintent { :silent } { \c__spinttent_invisible_sep_int } }
3492   { \__spinttent_inner_sep: }
3493   \str_use:c { l__spinttent_spinterval_luaset_ #1 _braced_str }
3494 }

```

(End of definition for `\__spinttent_spinterval_plain_number:n`)

`\__spinttent_spinterval_value:n`

The internal function `\__spinttent_spinterval_value:n` typesets the value of the endpoint passed in ‘#1’ according to its type: `\infty` for infinity; for a number, `\__spinttent_inner_sep:` followed by `\spnum` when `read-cmd` is `school` and the value is not an integer of at most three digits, or `\__spinttent_spinterval_plain_number:n` otherwise, since `\spnum` reads such an integer as MathCAT does; and the value as written for any other type.

```

3495 \cs_new_protected:Nn \__spinttent_spinterval_value:n
3496 {
3497   \str_case:enF { \str_use:c { l__spinttent_spinterval_luaset_ #1 _type_str } }
3498   {
3499     { infinity } { \infty }
3500     { number }
3501     {
3502       \str_if_eq:VnTF \l__spinttent_spinterval_mode_str { school }
3503       {
3504         \tl_set:Nx \l__spinttent_spinterval_tmp_tl
3505         { \str_use:c { l__spinttent_spinterval_luaset_ #1 _value_str } }
3506         \str_if_eq:eeTF
3507         { \str_use:c { l__spinttent_spinterval_luaset_ #1 _short_str } }
3508         { true }
3509         { \__spinttent_spinterval_plain_number:n {#1} }
3510         {
3511           \__spinttent_inner_sep:
3512           \exp_args:NV \spnum \l__spinttent_spinterval_tmp_tl
3513         }
3514       }
3515       { \__spinttent_spinterval_plain_number:n {#1} }
3516     }
3517   }
3518   { \str_use:c { l__spinttent_spinterval_luaset_ #1 _value_str } }
3519 }

```

(End of definition for `\__spinttent_spinterval_value:n`)

`\__spinttent_spinterval_print:n`

The internal function `\__spinttent_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 `\__spinttent_spinterval_inner_sep:n`; an endpoint without sign is typeset through `\__spinttent_spinterval_value:n`. With `read-cmd=mathcat` the sign is typeset as is, followed by the value.

```

3520 \cs_new_protected:Nn \__spinttent_spinterval_print:n
3521 {
3522   \str_if_eq:VnTF \l__spinttent_spinterval_mode_str { school }
3523   {
3524     \str_case:enF { \str_use:c { l__spinttent_spinterval_luaset_ #1 _sign_str } }
3525     {
3526       { + }
3527       {
3528         \MathMLintent { _más:prefix($vp) }
3529         {
3530           +
3531           \MathMLarg { vp }
3532           {
3533             \__spinttent_spinterval_inner_sep:n {#1}
3534             \__spinttent_spinterval_value:n {#1}
3535           }
3536         }
3537       }
3538     }
3539   }

```

```

3536         }
3537     }
3538     { - }
3539     {
3540         \MathMLintent { _menos:prefix($vn) }
3541         {
3542             -
3543             \MathMLarg { vn }
3544             {
3545                 \__spintent_spinterval_inner_sep:n {#1}
3546                 \__spintent_spinterval_value:n {#1}
3547             }
3548         }
3549     }
3550 }
3551 { \__spintent_spinterval_value:n {#1} }
3552 }
3553 {
3554     \str_use:c { l__spintent_spinterval_luaset_ #1 _sign_str }
3555     \__spintent_spinterval_value:n {#1}
3556 }
3557 }

```

(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.

```

3558 \cs_new_protected:Nn \__spintent_spinterval_mathml:
3559 {
3560     \MathMLintent { \__spintent_spinterval_intent_arg_tl }
3561     {
3562         \MathMLintent { :silent } { \__spintent_spinterval_delim_l_tl }
3563         \MathMLarg { ab }
3564         {
3565             \__spintent_inner_sep:
3566             \MathMLintent { \__spintent_spinterval_intent_group_tl }
3567             {
3568                 \MathMLarg { a }
3569                 {
3570                     \__spintent_spinterval_inner_sep:n { a }
3571                     \__spintent_spinterval_print:n { a }
3572                 }
3573                 \MathMLintent { :silent } { \__spintent_spinterval_luaset_sep_str }
3574                 \MathMLarg { b }
3575                 {
3576                     \__spintent_spinterval_inner_sep:n { b }
3577                     \__spintent_spinterval_print:n { b }
3578                 }
3579             }
3580         }
3581         \MathMLintent { :silent } { \__spintent_spinterval_delim_r_tl }
3582     }
3583 }

```

(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:`.

```

3584 \cs_new_protected:Nn \__spintent_spinterval_exec:nn
3585 {
3586     \keys_set:nn { spintent / spinterval } {#1}
3587     \__spintent_luafun_spinterval_parse_and_set:n { \exp_not:n {#2} }
3588     \str_if_eq:VnTF \__spintent_spinterval_luaset_error_str { true }

```

```

3589     {
3590         \__spinttent_spinterval_set_reason:
3591         \msg_error:nnee { spinttent } { spinterval-invalid-arg }
3592         { \exp_not:n {#2} } { \l__spinttent_spinterval_reason_str }
3593     }
3594     {
3595         \__spinttent_spinterval_set_delims:
3596         \__spinttent_spinterval_set_intents:
3597         \__spinttent_invisible_sep:
3598         \__spinttent_spinterval_mathml:
3599     }
3600 }

```

(End of definition for `\__spinttent_spinterval_exec:nn`.)

`\spinterval` The public command `\spinterval` enforces *math mode*, opens a local group, and delegates to `\__spinttent_spinterval_exec:nn`.

```

3601 \NewDocumentCommand \spinterval { 0{} m }
3602 {
3603     \mode_if_math:TF
3604     {
3605         \group_begin:
3606         \attribute \l__spinttent_scope_attr_int = 1
3607         \__spinttent_spinterval_exec:nn {#1} {#2}
3608         \group_end:
3609     }
3610     { \msg_error:nnn { spinttent } { need-math-mode } { \spinterval } }
3611 }

```

(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 `\__spinttent_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__spinttent_geo_luaset_*` variables, and the list of arguments, cut at the top-level commas, each one stored in `\l__spinttent_spfun_luaset_arg_<i>_tl` together with a flag `\l__spinttent_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,$a2,...)`; the name is tagged `arg="f"` and each argument `arg="a<i>"`, while the parentheses and the commas are silenced. The name may carry the inverse exponent `^{\{-1\}}`, which sets the flag `\l__spinttent_spfun_luaset_inv_str`; in that case the literal `intent` of the name ends in `-inversa`, the exponent is typeset silenced and the parentheses may be omitted, with no arguments and `$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 `\__spinttent_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__spinttent_spfun_luaset_error_str
\l__spinttent_spfun_luaset_nargs_str
\l__spinttent_spfun_luaset_inv_str

```

These variables are set by the Lua module `spintent.lua`. The variable `\l__spinttent_spfun_luaset_error_str` stores `true` or `false` if the `{\argument}` is invalid. The variable `\l__spinttent_spfun_luaset_nargs_str` stores the number of arguments of the function and `\l__spinttent_spfun_luaset_inv_str` stores `true` when the name carries the inverse exponent `^{\{-1\}}`.

```

3612 \str_new:N \l__spinttent_spfun_luaset_error_str
3613 \str_new:N \l__spinttent_spfun_luaset_nargs_str
3614 \str_new:N \l__spinttent_spfun_luaset_inv_str

```

(End of definition for `\l__spinttent_spfun_luaset_error_str`, `\l__spinttent_spfun_luaset_nargs_str`, and `\l__spinttent_spfun_luaset_inv_str`.)

```

\l__spinttent_spfun_mode_str
\l__spinttent_spfun_prefix_tl
\l__spinttent_spfun_silent_bool
\l__spinttent_spfun_concept_bool

```

The variables `\l__spinttent_spfun_mode_str`, `\l__spinttent_spfun_prefix_tl`, `\l__spinttent_spfun_silent_bool` and `\l__spinttent_spfun_concept_bool` store the values set by the keys `read-cmd`, `prefix`, `silent` and `concept`. The last two are created by the keys.

```

3615 \str_new:N \l__spinttent_spfun_mode_str
3616 \tl_new:N \l__spinttent_spfun_prefix_tl

```

(End of definition for `\l__spinttent_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.

```
3617 \int_new:N \l__spintent_spfun_nargs_int
3618 \int_new:N \l__spintent_spfun_depth_int
3619 \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
\l__spintent_spfun_fname_tl
\l__spintent_spfun_aname_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`.

```
3620 \tl_new:N \l__spintent_spfun_refs_tl
3621 \tl_new:N \l__spintent_spfun_call_tl
3622 \tl_new:N \l__spintent_spfun_intent_arg_tl
3623 \tl_new:N \l__spintent_spfun_kv_tl
3624 \tl_new:N \l__spintent_spfun_fname_tl
3625 \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`.

```
3626 \msg_new:nnnn { spintent } { spfun-invalid-arg }
3627 {
3628   Invalid~argument~'#1'~for~'\token_to_str:N \spfun'~\msg_line_context:.
3629 }
3630 {
3631   The~argument~must~have~the~form~'f(x)'~or~'f(x,y,...)',~where~'f'~is~a~letter~with~
3632   optional~primes~or~subscript.
3633 }
```

(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`.

```
3634 \keys_define:nn { spintent / spfun }
3635 {
3636   read-cmd .choices:nn = { school , mathcat }
3637   {
3638     \int_case:nn { \l_keys_choice_int }
3639     {
3640       {1} { \str_set:Nn \l__spintent_spfun_mode_str { school } }
3641       {2} { \str_set:Nn \l__spintent_spfun_mode_str { mathcat } }
3642     }
3643   },
3644   read-cmd .initial:n = school,
3645   read-cmd .value_required:n = true,
3646   read-cmd / unknown .code:n =
3647     \msg_error:nnee { spintent } { unknown-choice }
3648     { read-cmd } { school,~mathcat } { \exp_not:n {#1} },
3649   prefix .code:n =
3650     { \__spintent_sanitize_affix:Nn \l__spintent_spfun_prefix_tl {#1} },
3651   prefix .initial:n = ,
3652   prefix .value_required:n = true,
3653   silent .bool_set:N = \l__spintent_spfun_silent_bool,
3654   silent .initial:n = false,
3655   silent .value_required:n = true,
3656   concept .bool_set:N = \l__spintent_spfun_concept_bool,
3657   concept .initial:n = false,
3658   concept .value_required:n = true,
```

```

3659     unknown .code:n =
3660     {
3661         \str_set:Nn \l__spinttent_command_name_str { spfun }
3662         \__spinttent_unknown_keys_cmd:n {#1}
3663     },
3664 }

```

(End of definition for `read-cmd` and others.)

15.39.4 Internal functions for `\spfun`

`\__spinttent_spfun_fname:` Names of the `arg` attributes at the current nesting depth `\l__spinttent_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__spinttent_spfun_fname_tl` and `\l__spinttent_spfun_aname_tl`.

```

3665 \cs_new_protected:Nn \__spinttent_spfun_fname:
3666 {
3667     \tl_set:Nc \l__spinttent_spfun_fname_tl
3668     {
3669         f
3670         \int_compare:nNf { \l__spinttent_spfun_depth_int } = { 1 }
3671         { \int_use:N \l__spinttent_spfun_depth_int }
3672     }
3673 }
3674 \cs_new_protected:Nn \__spinttent_spfun_aname:n
3675 {
3676     \tl_set:Nc \l__spinttent_spfun_aname_tl
3677     {
3678         a
3679         \int_compare:nNf { \l__spinttent_spfun_depth_int } = { 1 }
3680         { \int_use:N \l__spinttent_spfun_depth_int x }
3681         #1
3682     }
3683 }

```

(End of definition for `\__spinttent_spfun_fname:` and `\__spinttent_spfun_aname:n`.)

`\__spinttent_spfun_name_plain:` Typesets the name without any annotation: the letter, followed by the subscript or by the primes when there are any.

```

3684 \cs_new_protected:Nn \__spinttent_spfun_name_plain:
3685 {
3686     \l__spinttent_geo_luaset_letra_str
3687     \str_case:VnF \l__spinttent_geo_luaset_tipo_str
3688     {
3689         { sub } { \c_math_subscript_token { \l__spinttent_geo_luaset_sub_str } }
3690         { prima } { \l__spinttent_geo_luaset_sub_str }
3691     }
3692     { }
3693     \str_if_eq:VnT \l__spinttent_spfun_luaset_inv_str { true }
3694     { \c_math_superscript_token { -1 } }
3695 }

```

(End of definition for `\__spinttent_spfun_name_plain:`.)

`\__spinttent_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.

```

3696 \cs_new_protected:Nn \__spinttent_spfun_name_school:
3697 {
3698     \__spinttent_spfun_fname:
3699     \tl_set:Nc \l__spinttent_spfun_kv_tl
3700     {
3701         arg = " \l__spinttent_spfun_fname_tl ", intent =
3702         "_ \l__spinttent_geo_luaset_intent_str -"
3703     }
3704     \__spinttent_luafun_mrow_intent_arg:nn { } { \l__spinttent_spfun_kv_tl }
3705     \str_case:VnF \l__spinttent_geo_luaset_tipo_str
3706     {
3707         { sub }

```

```

3708     {
3709         \l__spintent_geo_luaset_letra_str
3710         \c_math_subscript_token
3711         { \MathMLintent { :silent } { \l__spintent_geo_luaset_sub_str } }
3712     }
3713     { prima }
3714     { \l__spintent_geo_luaset_letra_str \l__spintent_geo_luaset_sub_str }
3715 }
3716 { \l__spintent_geo_luaset_letra_str }
3717 \str_if_eq:VnT \l__spintent_spfun_luaset_inv_str { true }
3718 {
3719     \c_math_superscript_token
3720     { \MathMLintent { :silent } { -1 } }
3721 }
3722 }

```

(End of definition for \l__spintent_spfun_name_school:.)

\l__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>`.

```

3723 \cs_new_protected:Nn \l__spintent_spfun_name_mathcat:
3724 {
3725     \l__spintent_spfun_fname:
3726     \bool_lazy_and:nnTF
3727     { \str_if_eq_p:Vn \l__spintent_geo_luaset_tipo_str { } }
3728     { \str_if_eq_p:Vn \l__spintent_spfun_luaset_inv_str { false } }
3729     { \MathMLarg { \l__spintent_spfun_fname_tl } { \l__spintent_spfun_name_plain: } }
3730     {
3731         \MathMLarg { \l__spintent_spfun_fname_tl }
3732         { \l__spintent_inner_sep: \l__spintent_spfun_name_plain: }
3733     }
3734 }

```

(End of definition for \l__spintent_spfun_name_mathcat:.)

\l__spintent_spfun_arg:n

Tags argument number `#1` with `arg="a<i>"`, named by `\l__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 `\l__spintent_spfun_nested:n`.

```

3735 \cs_new_protected:Nn \l__spintent_spfun_arg:n
3736 {
3737     \l__spintent_spfun_aname:n {#1}
3738     \str_set_eq:Nc \l__spintent_spfun_tmp_str
3739     { \l__spintent_spfun_luaset_simple_ #1 _str }
3740     \str_case:VnF \l__spintent_spfun_tmp_str
3741     {
3742         { true }
3743         {
3744             \MathMLarg { \l__spintent_spfun_aname_tl }
3745             { \tl_use:c { \l__spintent_spfun_luaset_arg_ #1 _tl } }
3746         }
3747         { call }
3748         {
3749             \MathMLarg { \l__spintent_spfun_aname_tl }
3750             { \l__spintent_inner_sep: \l__spintent_spfun_nested:n {#1} }
3751         }
3752     }
3753     {
3754         \MathMLarg { \l__spintent_spfun_aname_tl }
3755         { \l__spintent_inner_sep: \tl_use:c { \l__spintent_spfun_luaset_arg_ #1 _tl } }
3756     }
3757 }

```

(End of definition for \l__spintent_spfun_arg:n.)

\l__spintent_spfun_args:

Walks the arguments, silencing the comma between them.

```

3758 \cs_new_protected:Nn \l__spintent_spfun_args:
3759 {
3760     \int_step_inline:nn { \l__spintent_spfun_nargs_int }

```

```

3761     {
3762       \int_compare:nNnF { ##1 } = { 1 }
3763       { \MathMLintent { :silent } { , } }
3764       \__spintent_spfun_arg:n {##1}
3765     }
3766   }

```

(End of definition for `\__spintent_spfun_args:`.)

`\__spintent_spfun_args_plain:` Typesets the arguments with no annotation for `silent`.

```

3767 \cs_new_protected:Nn \__spintent_spfun_args_plain:
3768 {
3769   \int_step_inline:nn { \__spintent_spfun_nargs_int }
3770   {
3771     \int_compare:nNnF { ##1 } = { 1 } { , }
3772     \tl_use:c { \__spintent_spfun_luaset_arg_ ##1 _tl }
3773   }
3774 }

```

(End of definition for `\__spintent_spfun_args_plain:`.)

`\__spintent_spfun_set_intent:` Builds the concept `intent` in `\__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`.

```

3775 \cs_new_protected:Nn \__spintent_spfun_set_intent:
3776 {
3777   \tl_clear:N \__spintent_spfun_refs_tl
3778   \int_step_inline:nn { \__spintent_spfun_nargs_int }
3779   {
3780     \__spintent_spfun_aname:n {##1}
3781     \tl_put_right:Ne \__spintent_spfun_refs_tl
3782     { \int_compare:nNnF { ##1 } = { 1 } { , } $ \__spintent_spfun_aname_tl }
3783   }
3784   \__spintent_spfun_fname:
3785   \int_compare:nNnTF { \__spintent_spfun_nargs_int } = { 0 }
3786   { \tl_set:Ne \__spintent_spfun_call_tl { $ \__spintent_spfun_fname_tl } }
3787   {
3788     \tl_set:Ne \__spintent_spfun_call_tl
3789     { $ \__spintent_spfun_fname_tl ( \__spintent_spfun_refs_tl ) }
3790   }
3791   \bool_if:NTF \__spintent_spfun_concept_bool
3792   {
3793     \tl_set:Ne \__spintent_spfun_intent_arg_tl
3794     {
3795       -
3796       \tl_if_empty:NF \__spintent_spfun_prefix_tl
3797       { \tl_use:N \__spintent_spfun_prefix_tl - }
3798       función:prefix( \__spintent_spfun_call_tl )
3799     }
3800   }
3801   {
3802     \tl_set_eq:NN \__spintent_spfun_intent_arg_tl \__spintent_spfun_call_tl
3803   }
3804 }

```

(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.

```

3805 \cs_new_protected:Nn \__spintent_spfun_mathml:
3806 {
3807   \__spintent_invisible_sep:
3808   \__spintent_spfun_body:
3809 }

```

(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.

```

3810 \cs_new_protected:Nn \__spintent_spfun_body:

```

```

3811 {
3812   \bool_if:NTF \l__spinttent_spfun_silent_bool
3813   {
3814     \MathMLintent { _- }
3815     {
3816       \__spinttent_inner_sep:
3817       \__spinttent_spfun_name_plain:
3818       \int_compare:nNnF { \l__spinttent_spfun_nargs_int } = { 0 }
3819       {
3820         \lparen
3821         \__spinttent_spfun_args_plain:
3822         \rparen
3823       }
3824     }
3825   }
3826   {
3827     \__spinttent_spfun_set_intent:
3828     \MathMLintent { \l__spinttent_spfun_intent_arg_tl }
3829     {
3830       \int_compare:nNnT { \l__spinttent_spfun_nargs_int } = { 0 }
3831       { \__spinttent_inner_sep: }
3832       \str_if_eq:VnTF \l__spinttent_spfun_mode_str { school }
3833       { \__spinttent_spfun_name_school: }
3834       { \__spinttent_spfun_name_mathcat: }
3835       \int_compare:nNnF { \l__spinttent_spfun_nargs_int } = { 0 }
3836       {
3837         \MathMLintent { :silent } { \lparen }
3838         \__spinttent_spfun_args:
3839         \MathMLintent { :silent } { \rparen }
3840       }
3841     }
3842   }
3843 }

```

(End of definition for `\__spinttent_spfun_body:`.)

`\__spinttent_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 `\__spinttent_spfun_body:` with no outer separator. It cannot fail to parse, since the first analysis already validated every level.

```

3844 \cs_new_protected:Nn \__spinttent_spfun_nested:n
3845 {
3846   \group_begin:
3847   \__spinttent_luafun_spfun_parse_and_set:v { \l__spinttent_spfun_luaaset_arg_ #1 _tl }
3848   \int_incr:N \l__spinttent_spfun_depth_int
3849   \bool_set_false:N \l__spinttent_spfun_concept_bool
3850   \int_set:NV \l__spinttent_spfun_nargs_int \l__spinttent_spfun_luaaset_nargs_str
3851   \__spinttent_spfun_body:
3852   \group_end:
3853 }

```

(End of definition for `\__spinttent_spfun_nested:n`.)

`\__spinttent_spfun_exec:nn` Receives the optional $\langle keys \rangle$ in ‘#1’ and the $\{\langle argument \rangle\}$ in ‘#2’. Parses first, reporting `spfun-invalid-arg` on error whatever `silent` is.

```

3854 \cs_new_protected:Nn \__spinttent_spfun_exec:nn
3855 {
3856   \keys_set:nn { spinttent / spfun } { #1 }
3857   \__spinttent_luafun_spfun_parse_and_set:n { \exp_not:n { #2 } }
3858   \str_if_eq:VnTF \l__spinttent_spfun_luaaset_error_str { true }
3859   {
3860     \msg_error:nne { spinttent } { spfun-invalid-arg } { \exp_not:n { #2 } }
3861   }
3862   {
3863     \int_set:NV \l__spinttent_spfun_nargs_int \l__spinttent_spfun_luaaset_nargs_str
3864     \int_set:Nn \l__spinttent_spfun_depth_int { 1 }
3865     \__spinttent_spfun_mathml:
3866   }
3867 }

```

(End of definition for `\__spinttent_spfun_exec:nn`.)

```

\spfun The public command \spfun enforces math mode, opens a local group, and delegates to
      \__spintent_spfun_exec:nn.

3868 \NewDocumentCommand \spfun { O{} m }
3869 {
3870   \mode_if_math:TF
3871   {
3872     \group_begin:
3873       \attribute \__spintent_scope_attr_int = 1
3874       \__spintent_spfun_exec:nn {#1} {#2}
3875     \group_end:
3876   }
3877   { \msg_error:nnn { spintent } { need-math-mode } { \spfun } }
3878 }

```

(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 $\langle argument \rangle$ 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`

These variables are set by the Lua module `spintent.lua`. The variable `\__spintent_spfuncomp_luaset_error_str` stores *true* or *false* if the $\langle argument \rangle$ is invalid. The variable `\__spintent_spfuncomp_luaset_n_str` stores the number of functions. The variables `\__spintent_spfuncomp_luaset_paren_str` and `\__spintent_spfuncomp_luaset_apply_str` store *true* when the composition is in parentheses and when it is applied to arguments.

```

3879 \str_new:N \__spintent_spfuncomp_luaset_error_str
3880 \str_new:N \__spintent_spfuncomp_luaset_n_str
3881 \str_new:N \__spintent_spfuncomp_luaset_paren_str
3882 \str_new:N \__spintent_spfuncomp_luaset_apply_str

```

(End of definition for `\__spintent_spfuncomp_luaset_error_str` and others.)

The variable `\__spintent_spfuncomp_n_int` stores the number of functions, copied from `\__spintent_spfuncomp_luaset_n_str`. The variable `\__spintent_spfuncomp_refs_tl` stores the references to the functions and `\__spintent_spfuncomp_intent_tl` stores the *intent* being built. The variable `\__spintent_spfuncomp_kv_tl` stores the key-value list with the attributes `arg` and *intent* of the function being typeset and `\__spintent_spfuncomp_tipo_str` stores the kind of its name.

```

3883 \int_new:N \__spintent_spfuncomp_n_int
3884 \tl_new:N \__spintent_spfuncomp_refs_tl
3885 \tl_new:N \__spintent_spfuncomp_intent_tl
3886 \tl_new:N \__spintent_spfuncomp_kv_tl
3887 \str_new:N \__spintent_spfuncomp_tipo_str

```

(End of definition for `\__spintent_spfuncomp_n_int` and others.)

15.40.2 Definition of error messages for `\spfuncomp`

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*.

```

3888 \msg_new:nnnn { spintent } { spfuncomp-invalid-arg }
3889 {
3890   Invalid~argument~'~#1'~for~'\token_to_str:N \spfuncomp'~\msg_line_context:.
3891 }
3892 {
3893   The~argument~must~be~two~or~more~functions~joined~by~'\token_to_str:N \circ'.
3894 }

```

(End of definition for `spfuncomp-invalid-arg`.)

15.40.3 Definition of keys for \spfuncomp

`silent` We define the *(key)* `silent`.

`unknown`

```

3895 \keys_define:nn { spintent / spfuncomp }
3896 {
3897   silent .bool_set:N = \l__spintent_spfuncomp_silent_bool,
3898   silent .initial:n = false,
3899   silent .value_required:n = true,
3900   unknown .code:n =
3901     {
3902       \str_set:Nn \l__spintent_command_name_str { spfuncomp }
3903       \__spintent_unknown_keys_cmd:n {#1}
3904     },
3905 }
```

(End of definition for `silent` and `unknown`.)

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.

```

3906 \cs_new_protected:Nn \__spintent_spfuncomp_fun:n
3907 {
3908   \tl_set:Nc \l__spintent_spfuncomp_kv_tl
3909     {
3910       arg = " k #1 ",
3911       intent = "_ \use:c { \__spintent_spfuncomp_luaset_intent_ #1 _str } -"
3912     }
3913   \__spintent_luafun_mrow_intent_arg:nn {} { \__spintent_spfuncomp_kv_tl }
3914   \str_set_eq:Nc \l__spintent_spfuncomp_tipo_str
3915     { \__spintent_spfuncomp_luaset_tipo_ #1 _str }
3916   \str_case:VnF \l__spintent_spfuncomp_tipo_str
3917     {
3918       { sub }
3919       {
3920         \use:c { \__spintent_spfuncomp_luaset_letra_ #1 _str }
3921         \c_math_subscript_token
3922         {
3923           \MathMLintent { :silent }
3924           { \use:c { \__spintent_spfuncomp_luaset_sub_ #1 _str } }
3925         }
3926       }
3927     { prima }
3928     {
3929       \use:c { \__spintent_spfuncomp_luaset_letra_ #1 _str }
3930       \use:c { \__spintent_spfuncomp_luaset_sub_ #1 _str }
3931     }
3932   }
3933   { \use:c { \__spintent_spfuncomp_luaset_letra_ #1 _str } }
3934   \str_set_eq:Nc \l__spintent_spfuncomp_tipo_str
3935     { \__spintent_spfuncomp_luaset_inv_ #1 _str }
3936   \str_if_eq:VnT \l__spintent_spfuncomp_tipo_str { true }
3937   {
3938     \c_math_superscript_token
3939     { \MathMLintent { :silent } { -1 } }
3940   }
3941 }
```

(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.

```

3942 \cs_new_protected:Nn \__spintent_spfuncomp_group:
3943 {
3944   \tl_clear:N \l__spintent_spfuncomp_refs_tl
3945   \int_step_inline:nn { \l__spintent_spfuncomp_n_int }
3946     {
3947       \tl_put_right:Nc \l__spintent_spfuncomp_refs_tl
3948         { \int_compare:nNnF { ##1 } = { 1 } { , } $k ##1 }
3949     }
```

```

3950 \tl_set:Nx \l__spinttent_sfuncomp_intent_tl
3951 { _compuesta-con:infix( \l__spinttent_sfuncomp_refs_tl ) }
3952 \MathMLintent { \l__spinttent_sfuncomp_intent_tl }
3953 {
3954   \str_if_eq:VnT \l__spinttent_sfuncomp_luaset_paren_str { true }
3955   { \MathMLintent { :silent } { \lparen } }
3956   \int_step_inline:nn { \l__spinttent_sfuncomp_n_int }
3957   {
3958     \int_compare:nNnF { ##1 } = { 1 }
3959     { \MathMLintent { :silent } { \circ } }
3960     \__spinttent_sfuncomp_fun:n {##1}
3961   }
3962   \str_if_eq:VnT \l__spinttent_sfuncomp_luaset_paren_str { true }
3963   { \MathMLintent { :silent } { \rparen } }
3964 }
3965 }

```

(End of definition for `\__spinttent_sfuncomp_group:`.)

`\__spinttent_sfuncomp_plain:` Typesets the whole with no annotation for `silent`.

```

3966 \cs_new_protected:Nn \__spinttent_sfuncomp_plain:
3967 {
3968   \str_if_eq:VnT \l__spinttent_sfuncomp_luaset_paren_str { true } { \lparen }
3969   \int_step_inline:nn { \l__spinttent_sfuncomp_n_int }
3970   {
3971     \int_compare:nNnF { ##1 } = { 1 } { \circ }
3972     \tl_use:c { l__spinttent_sfuncomp_luaset_print_ ##1 _tl }
3973   }
3974   \str_if_eq:VnT \l__spinttent_sfuncomp_luaset_paren_str { true } { \rparen }
3975   \str_if_eq:VnT \l__spinttent_sfuncomp_luaset_apply_str { true }
3976   {
3977     \lparen \__spinttent_sfun_args_plain: \rparen
3978   }
3979 }

```

(End of definition for `\__spinttent_sfuncomp_plain:`.)

`\__spinttent_sfuncomp_mathml:` Calls `\__spinttent_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 `\sfun`, including calls nested in them.

```

3980 \cs_new_protected:Nn \__spinttent_sfuncomp_mathml:
3981 {
3982   \__spinttent_invisible_sep:
3983   \str_if_eq:VnT \l__spinttent_sfuncomp_luaset_apply_str { true }
3984   {
3985     \int_set:Nv \l__spinttent_sfun_nargs_int \l__spinttent_sfun_luaset_nargs_str
3986   }
3987   \int_set:Nn \l__spinttent_sfun_depth_int { 1 }
3988   \str_set:Nn \l__spinttent_sfun_mode_str { school }
3989   \bool_set_false:N \l__spinttent_sfun_silent_bool
3990   \bool_set_false:N \l__spinttent_sfun_concept_bool
3991   \bool_if:NTF \l__spinttent_sfuncomp_silent_bool
3992   {
3993     \MathMLintent { - }
3994     { \__spinttent_inner_sep: \__spinttent_sfuncomp_plain: }
3995   }
3996   {
3997     \str_if_eq:VnTF \l__spinttent_sfuncomp_luaset_apply_str { true }
3998     {
3999       \tl_clear:N \l__spinttent_sfun_refs_tl
4000       \int_step_inline:nn { \l__spinttent_sfun_nargs_int }
4001       {
4002         \__spinttent_sfun_aname:n {##1}
4003         \tl_put_right:Ne \l__spinttent_sfun_refs_tl
4004         {
4005           \int_compare:nNnF { ##1 } = { 1 } { , }
4006           $ \l__spinttent_sfun_aname_tl
4007         }
4008       }
4009       \int_compare:nNnTF { \l__spinttent_sfun_nargs_int } > { 1 }

```

```

4010         {
4011             \tl_set:Nc \l__spinttent_spfun_call_tl { $c ( $l ) }
4012             \MathMLintent { \l__spinttent_spfun_call_tl }
4013             {
4014                 \MathMLarg { c }
4015                 { \__spinttent_inner_sep: \__spinttent_spfuncomp_group: }
4016                 \MathMLintent { :silent } { \lparen }
4017                 \MathMLarg { l }
4018                 {
4019                     \__spinttent_inner_sep:
4020                     \MathMLintent
4021                     { _coma:infix ( \l__spinttent_spfun_refs_tl ) }
4022                     { \__spinttent_spfun_args: }
4023                 }
4024                 \MathMLintent { :silent } { \rparen }
4025             }
4026         }
4027     {
4028         \tl_set:Nc \l__spinttent_spfun_call_tl
4029         { $c ( \l__spinttent_spfun_refs_tl ) }
4030         \MathMLintent { \l__spinttent_spfun_call_tl }
4031         {
4032             \MathMLarg { c }
4033             { \__spinttent_inner_sep: \__spinttent_spfuncomp_group: }
4034             \MathMLintent { :silent } { \lparen }
4035             \__spinttent_spfun_args:
4036             \MathMLintent { :silent } { \rparen }
4037         }
4038     }
4039 }
4040 { \__spinttent_spfuncomp_group: }
4041 }
4042 }

```

(End of definition for `\__spinttent_spfuncomp_mathml:`.)

`\__spinttent_spfuncomp_exec:nn` Receives the optional $\langle keys \rangle$ in ‘#1’ and the $\{\langle argument \rangle\}$ in ‘#2’. Parses first, reporting **spfuncomp-invalid-arg** on error whatever `silent` is.

```

4043 \cs_new_protected:Nn \__spinttent_spfuncomp_exec:nn
4044 {
4045     \keys_set:nn { spinttent / spfuncomp } {#1}
4046     \__spinttent_luafun_spfuncomp_parse_and_set:n { \exp_not:n {#2} }
4047     \str_if_eq:VnTF \l__spinttent_spfuncomp_luaerror_str { true }
4048     {
4049         \msg_error:nne { spinttent } { spfuncomp-invalid-arg }
4050         { \exp_not:n {#2} }
4051     }
4052     {
4053         \int_set:NV \l__spinttent_spfuncomp_n_int \l__spinttent_spfuncomp_luaerror_n_str
4054         \__spinttent_spfuncomp_mathml:
4055     }
4056 }

```

(End of definition for `\__spinttent_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 `\__spinttent_spfuncomp_exec:nn`.

```

4057 \NewDocumentCommand \spfuncomp { O{ } m }
4058 {
4059     \mode_if_math:TF
4060     {
4061         \group_begin:
4062         \attribute \l__spinttent_scope_attr_int = 1
4063         \__spinttent_spfuncomp_exec:nn {#1} {#2}
4064         \group_end:
4065     }
4066     { \msg_error:nnn { spinttent } { need-math-mode } { \spfuncomp } }
4067 }

```

(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:nnnn`, 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 `\__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`

Each public command of this family sets these variables before calling the common code. The variable `\__spintent_spdom_cmd_str` stores the name of the command, `\__spintent_spdom_op_tl` the operator typeset by it and `\__spintent_spdom_word_tl` the word used in the *intent*.

```
4968 \str_new:N \__spintent_spdom_cmd_str
4969 \tl_new:N \__spintent_spdom_op_tl
4970 \tl_new:N \__spintent_spdom_word_tl
```

(End of definition for `\__spintent_spdom_cmd_str`, `\__spintent_spdom_op_tl`, and `\__spintent_spdom_word_tl`.)

The variable `\__spintent_spdom_prefix_tl` stores the value set by the key `prefix` and `\__spintent_spdom_intent_arg_tl` stores the *intent* being built.

```
4971 \tl_new:N \__spintent_spdom_prefix_tl
4972 \tl_new:N \__spintent_spdom_intent_arg_tl
```

(End of definition for `\__spintent_spdom_prefix_tl` and `\__spintent_spdom_intent_arg_tl`.)

`\__spintent_spdom_luaset_error_str` This variable is set from Lua and stores `true` or `false` if the *{argument}* is not a function name.

```
4973 \str_new:N \__spintent_spdom_luaset_error_str
```

(End of definition for `\__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`.

```
4974 \msg_new:nnnn { spintent } { spdom-invalid-arg }
4975 {
4976   Invalid~argument~'#2'~for~'\c_backslash_str #1'~\msg_line_context:.
4977 }
4978 {
4979   The~argument~must~be~a~function~name.
4980 }
```

(End of definition for `spdom-invalid-arg`.)

15.41.3 Definition of keys for `\spdom`

`prefix` We define the *keys* `prefix` and `silent`.

`silent`

`unknown`

```
4981 \keys_define:nn { spintent / spdom }
4982 {
4983   prefix .code:n =
4984     { \__spintent_sanitize_affix:Nn \__spintent_spdom_prefix_tl {#1} },
4985   prefix .initial:n = ,
4986   prefix .value_required:n = true,
4987   silent .bool_set:N = \__spintent_spdom_silent_bool,
4988   silent .initial:n = false,
4989   silent .value_required:n = true,
4990   unknown .code:n =
4991     {
4992       \str_set_eq:NN \__spintent_command_name_str \__spintent_spdom_cmd_str
4993       \__spintent_unknown_keys_cmd:n {#1}
4994     },
4995 }
```

(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.

```

4096 \cs_new_protected:Nn \__spintent_spdom_name:
4097 {
4098   \MathMLarg { f }
4099   {
4100     \__spintent_inner_sep:
4101     \MathMLintent { _ \l__spintent_geo_luaset_intent_str - }
4102     { \__spintent_inner_sep: \__spintent_spfun_name_plain: }
4103   }
4104 }

```

(End of definition for `\__spintent_spdom_name:`.)

`\__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.

```

4105 \cs_new_protected:Nn \__spintent_spdom_mathml:
4106 {
4107   \__spintent_invisible_sep:
4108   \bool_if:NTF \l__spintent_spdom_silent_bool
4109   {
4110     \MathMLintent { _- }
4111     {
4112       \__spintent_inner_sep:
4113       \operatorname { \l__spintent_spdom_op_tl }
4114       \lparen \__spintent_spfun_name_plain: \rparen
4115     }
4116   }
4117   {
4118     \tl_set:Nc \l__spintent_spdom_intent_arg_tl
4119     {
4120       -
4121       \tl_if_empty:NF \l__spintent_spdom_prefix_tl
4122       { \tl_use:N \l__spintent_spdom_prefix_tl - }
4123       \l__spintent_spdom_word_tl -de:prefix( $f )
4124     }
4125     \MathMLintent { \l__spintent_spdom_intent_arg_tl }
4126     {
4127       \MathMLintent { :silent } { \operatorname { \l__spintent_spdom_op_tl } }
4128       \MathMLintent { :silent } { \lparen }
4129       \__spintent_spdom_name:
4130       \MathMLintent { :silent } { \rparen }
4131     }
4132   }
4133 }

```

(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.

```

4134 \cs_new_protected:Nn \__spintent_spdom_exec:nnnnn
4135 {
4136   \str_set:Nn \l__spintent_spdom_cmd_str {#1}
4137   \tl_set:Nn \l__spintent_spdom_op_tl {#2}
4138   \tl_set:Nn \l__spintent_spdom_word_tl {#3}
4139   \keys_set:nn { spintent / spdom } {#4}
4140   \__spintent_luafun_spdom_parse_and_set:n { \exp_not:n {#5} }
4141   \str_if_eq:VnTF \l__spintent_spdom_luaset_error_str { true }
4142   {
4143     \msg_error:nnee { spintent } { spdom-invalid-arg }
4144     {#1} { \exp_not:n {#5} }
4145   }
4146   { \__spintent_spdom_mathml: }
4147 }

```

(End of definition for `\__spintent_spdom_exec:nnnnn`.)

```

\spdom The public commands \spdom, \sprec and \spcod enforce math mode, open a local group, and delegate to
\sprec \__spintent_spdom_exec:nnnnn.
\spcod
4148 \NewDocumentCommand \spdom { O{} m }
4149 {
4150     \mode_if_math:TF
4151     {
4152         \group_begin:
4153         \__spintent_spdom_exec:nnnnn { \spdom } { Dom } { dominio } {#1} {#2}
4154         \group_end:
4155     }
4156     { \msg_error:nnn { spintent } { need-math-mode } { \spdom } }
4157 }
4158 \NewDocumentCommand \sprec { O{} m }
4159 {
4160     \mode_if_math:TF
4161     {
4162         \group_begin:
4163         \__spintent_spdom_exec:nnnnn { \sprec } { Rec } { recorrido } {#1} {#2}
4164         \group_end:
4165     }
4166     { \msg_error:nnn { spintent } { need-math-mode } { \sprec } }
4167 }
4168 \NewDocumentCommand \spcod { O{} m }
4169 {
4170     \mode_if_math:TF
4171     {
4172         \group_begin:
4173         \__spintent_spdom_exec:nnnnn { \spcod } { Cod } { codominio } {#1} {#2}
4174         \group_end:
4175     }
4176     { \msg_error:nnn { spintent } { need-math-mode } { \spcod } }
4177 }

```

(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`

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`.

```

4178 \tl_new:N \l__spintent_spabs_read_arg_tl
4179 \str_new:N \l__spintent_spabs_size_str
4180 \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`.)

The opening and closing bars, with their size commands.

```

4181 \tl_new:N \l__spintent_spabs_delim_l_tl
4182 \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`.)

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.

```

4183 \str_new:N \l__spintent_spabs_luaset_close_str
4184 \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*.

```

4185 \msg_new:nnnn { spintent } { spabs-empty-arg }
4186 {
4187   Empty~argument~for~'\token_to_str:N \spabs'~\msg_line_context:.
4188 }
4189 {
4190   The~expression~inside~the~absolute~value~is~required.
4191 }
```

(End of definition for `spabs-empty-arg`.)

15.42.3 Definition of keys for \spabs

silent We define the *(keys)* `silent`, `read-arg`, `size` and `close`.

```

4192 \keys_define:nn { spintent / spabs }
4193 {
4194   silent .bool_set:N = \l__spintent_spabs_silent_bool,
4195   silent .initial:n = false,
4196   silent .value_required:n = true,
4197   read-arg .code:n =
4198     { \__spintent_sanitize_affix:Nn \l__spintent_spabs_read_arg_tl {#1} },
4199   read-arg .initial:n = ,
4200   read-arg .value_required:n = true,
4201   size .choices:nn = { none , big , Big , bigg , Bigg }
4202     {
4203       \int_case:nn { \l_keys_choice_int }
4204       {
4205         {1} { \str_set:Nn \l__spintent_spabs_size_str { none } }
4206         {2} { \str_set:Nn \l__spintent_spabs_size_str { big } }
4207         {3} { \str_set:Nn \l__spintent_spabs_size_str { Big } }
4208         {4} { \str_set:Nn \l__spintent_spabs_size_str { bigg } }
4209         {5} { \str_set:Nn \l__spintent_spabs_size_str { Bigg } }
4210       }
4211     },
4212   size .initial:n = none,
4213   size .value_required:n = true,
4214   size / unknown .code:n =
4215     \msg_error:nneee { spintent } { unknown-choice }
4216     { size } { none,~big,~Big,~bigg,~Bigg } { \exp_not:n {#1} },
4217   close .choices:nn = { auto , true , false }
4218     {
4219       \int_case:nn { \l_keys_choice_int }
4220       {
4221         {1} { \str_set:Nn \l__spintent_spabs_close_mode_str { auto } }
4222         {2} { \str_set:Nn \l__spintent_spabs_close_mode_str { true } }
4223         {3} { \str_set:Nn \l__spintent_spabs_close_mode_str { false } }
4224       }
4225     },
4226   close .initial:n = auto,
4227   close .value_required:n = true,
4228   close / unknown .code:n =
4229     \msg_error:nneee { spintent } { unknown-choice }
4230     { close } { auto,~true,~false } { \exp_not:n {#1} },
4231   unknown .code:n =
4232     {
4233       \str_set:Nn \l__spintent_command_name_str { spabs }
4234       \__spintent_unknown_keys_cmd:n {#1}
4235     },
4236 }
```

(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.

```

4237 \cs_new_protected:Nn \__spintent_spabs_set_delims:
4238 {
4239   \str_case:Vn \l__spintent_spabs_size_str
```

```

4240 {
4241   { none }
4242   {
4243     \tl_set:Nn \l__spinttent_spabs_delim_l_tl { \lvert }
4244     \tl_set:Nn \l__spinttent_spabs_delim_r_tl { \rvert }
4245   }
4246   { big }
4247   {
4248     \tl_set:Nn \l__spinttent_spabs_delim_l_tl { \bigl \lvert }
4249     \tl_set:Nn \l__spinttent_spabs_delim_r_tl { \bigr \rvert }
4250   }
4251   { Big }
4252   {
4253     \tl_set:Nn \l__spinttent_spabs_delim_l_tl { \Bigl \lvert }
4254     \tl_set:Nn \l__spinttent_spabs_delim_r_tl { \Bigr \rvert }
4255   }
4256   { bigg }
4257   {
4258     \tl_set:Nn \l__spinttent_spabs_delim_l_tl { \biggl \lvert }
4259     \tl_set:Nn \l__spinttent_spabs_delim_r_tl { \biggr \rvert }
4260   }
4261   { Bigg }
4262   {
4263     \tl_set:Nn \l__spinttent_spabs_delim_l_tl { \Biggl \lvert }
4264     \tl_set:Nn \l__spinttent_spabs_delim_r_tl { \Biggr \rvert }
4265   }
4266 }
4267 }

```

(End of definition for `\__spinttent_spabs_set_delims:`.)

`\__spinttent_spabs_set_close:n`

The function `\__spinttent_spabs_set_close:n` sets `\l__spinttent_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.

```

4268 \cs_new_protected:Nn \__spinttent_spabs_set_close:n
4269 {
4270   \bool_set_false:N \l__spinttent_spabs_close_bool
4271   \str_case:Vn \l__spinttent_spabs_close_mode_str
4272   {
4273     { true } { \bool_set_true:N \l__spinttent_spabs_close_bool }
4274     { auto }
4275     {
4276       \__spinttent_luafun_spabs_close:n { \exp_not:n {#1} }
4277       \str_if_eq:VnT \l__spinttent_spabs_luaset_close_str { true }
4278       { \bool_set_true:N \l__spinttent_spabs_close_bool }
4279     }
4280   }
4281 }

```

(End of definition for `\__spinttent_spabs_set_close:n`.)

`\__spinttent_spabs_exec:nn`

The function `\__spinttent_spabs_exec:nn` sets the $\langle keys \rangle$ in ‘#1’, sets the delimiters with `\__spinttent_spabs_set_delims:` and calls `\__spinttent_spabs_mathml:n` with the argument in ‘#2’.

```

4282 \cs_new_protected:Nn \__spinttent_spabs_exec:nn
4283 {
4284   \keys_set:nn { spinttent / spabs } {#1}
4285   \__spinttent_spabs_set_delims:
4286   \__spinttent_spabs_mathml:n {#2}
4287 }

```

(End of definition for `\__spinttent_spabs_exec:nn`.)

`\__spinttent_spabs_mathml:n`

The function `\__spinttent_spabs_mathml:n` prints the absolute value of ‘#1’ inside one native `\MathMLintent`, preceded by the outer ghost `\__spinttent_invisible_sep:` and followed by the inner ghost `\__spinttent_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 `\__spinttent_spabs_set_close:n` sets the closing, a native `\MathMLintent` with the literal intent

`_fin-valor-absoluto` follows the bars as a sibling, holding `\__spinttent_inner_sep:` and an invisible character, and the outer ghost is omitted because the formula already has two top-level nodes.

```

4288 \cs_new_protected:Nn \__spinttent_spabs_mathml:n
4289 {
4290   \bool_if:NTF \__spinttent_spabs_silent_bool
4291   {
4292     \__spinttent_invisible_sep:
4293     \MathMLintent { _- }
4294     {
4295       \l__spinttent_spabs_delim_l_tl \__spinttent_inner_sep: #1
4296       \l__spinttent_spabs_delim_r_tl
4297     }
4298   }
4299   {
4300     \tl_if_empty:NTF \l__spinttent_spabs_read_arg_tl
4301     {
4302       \__spinttent_spabs_set_close:n {#1}
4303       \bool_if:NF \l__spinttent_spabs_close_bool { \__spinttent_invisible_sep: }
4304       \MathMLintent { _valor-absoluto($x) }
4305       {
4306         \l__spinttent_spabs_delim_l_tl
4307         \MathMLarg { x } { \__spinttent_inner_sep: #1 }
4308         \l__spinttent_spabs_delim_r_tl
4309       }
4310       \bool_if:NT \l__spinttent_spabs_close_bool
4311       {
4312         \MathMLintent { _fin-valor-absoluto }
4313         { \__spinttent_inner_sep: \c__spinttent_invisible_sep_int }
4314       }
4315     }
4316     {
4317       \__spinttent_invisible_sep:
4318       \MathMLintent { _ \tl_use:N \l__spinttent_spabs_read_arg_tl :nofix }
4319       {
4320         \l__spinttent_spabs_delim_l_tl \__spinttent_inner_sep: #1
4321         \l__spinttent_spabs_delim_r_tl
4322       }
4323     }
4324   }
4325 }

```

(End of definition for `\__spinttent_spabs_mathml:n`.)

`\spabs` The public command `\spabs` enforces *math mode*, opens a local group and delegates execution to `\__spinttent_spabs_exec:nn`.

```

4326 \NewDocumentCommand \spabs { O{} m }
4327 {
4328   \mode_if_math:TF
4329   {
4330     \group_begin:
4331     \attribute \l__spinttent_scope_attr_int = 1
4332     \tl_if_blank:nTF {#2}
4333     { \msg_error:nn { spinttent } { spabs-empty-arg } }
4334     { \__spinttent_spabs_exec:nn {#1} {#2} }
4335     \group_end:
4336   }
4337   { \msg_error:nnn { spinttent } { need-math-mode } { \spabs } }
4338 }

```

(End of definition for `\spabs`. This function is documented on page 42.)

15.43 The function `\__spinttent_mathmlarg_point:nnn`

`\l__spinttent_mathmlarg_point_ref_tl`
`\l__spinttent_mathmlarg_point_kv_tl`

These variables are scratch variables of `\__spinttent_mathmlarg_point:nnn`. The variable `\l__spinttent_mathmlarg_point_ref_tl` stores the reference name given in `#1` and `\l__spinttent_mathmlarg_point_kv_tl` stores the key-value list with the attributes of the point.

```

4339 \tl_new:N \l__spinttent_mathmlarg_point_ref_tl
4340 \tl_new:N \l__spinttent_mathmlarg_point_kv_tl

```

(End of definition for `\l__spinttent_mathmlarg_point_ref_tl` and `\l__spinttent_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.

```
4341 \str_new:N \l__spintent_mathmlarg_point_mode_str
4342 \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`.

```
4343 \cs_new_protected:Nn \__spintent_mathmlarg_point:nnn
4344 {
4345   \tl_set:Nc \l__spintent_mathmlarg_point_ref_tl {#1}
4346   \__spintent_luafun_geo_point_parse_and_set:n { \exp_not:n {#3} }
4347   \bool_set_eq:Nc \l__spintent_mathmlarg_point_silent_bool
4348     { \__spintent_#2_silent_bool }
4349   \str_set_eq:Nc \l__spintent_mathmlarg_point_mode_str { \__spintent_#2_mode_str }
4350   \__spintent_invisible_sep:
4351   \str_case:VnF \l__spintent_mathmlarg_point_ref_tl
4352     { { pt } { } { pt1 } { } { pts1 } { } }
4353     { \__spintent_point_sep: }
4354   \str_case:VnF \l__spintent_geo_luaset_tipo_str
4355     {
4356       { sub }
4357       {
4358         \str_if_eq:VnTF \l__spintent_mathmlarg_point_mode_str { school }
4359         {
4360           \tl_set:Nc \l__spintent_mathmlarg_point_kv_tl
4361             {
4362               arg = "\l__spintent_mathmlarg_point_ref_tl",
4363               intent = "_ \l__spintent_geo_luaset_intent_str -"
4364             }
4365           \__spintent_luafun_mrow_intent_arg:nn
4366             { } { \l__spintent_mathmlarg_point_kv_tl }
4367           \int_compare:nNnTF { \str_count:N \l__spintent_geo_luaset_sub_str } >
4368             { 1 }
4369             {
4370               \int_step_inline:nnnn { 1 } { 1 }
4371               { \str_count:N \l__spintent_geo_luaset_sub_str }
4372               {
4373                 \__spintent_luafun_mrow_intent_arg:nn
4374                   { } { intent = ":silent" }
4375               }
4376             {
4377               \l__spintent_geo_luaset_letra_str
4378               \c_math_subscript_token { \l__spintent_geo_luaset_sub_str }
4379             }
4380           }
4381           {
4382             \l__spintent_geo_luaset_letra_str
4383             \c_math_subscript_token
4384               { \MathMLintent { :silent } { \l__spintent_geo_luaset_sub_str } }
4385           }
4386         }
4387       }
4388     }
4389     {
4390       {
4391         \__spintent_inner_sep:
4392         \MathMLarg { \l__spintent_mathmlarg_point_ref_tl }
4393         {
4394           {
4395             \l__spintent_geo_luaset_letra_str
4396             \c_math_subscript_token { \l__spintent_geo_luaset_sub_str }
4397           }
4398         }
4399       }
4400     }
4401 }
```

```

4399         }
4400     }
4401 }

```

With `\__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`.

```

4402     { prima }
4403     {
4404         \str_if_eq:VnTF \__spintent_mathmlarg_point_mode_str { school }
4405         {
4406             \tl_set:Ne \__spintent_mathmlarg_point_kv_tl
4407             {
4408                 arg = "\__spintent_mathmlarg_point_ref_tl",
4409                 intent = "_ \__spintent_geo_luaset_intent_str -"
4410             }
4411             \__spintent_luafun_mrow_intent_arg:nn
4412             { } { \__spintent_mathmlarg_point_kv_tl }
4413             { \__spintent_geo_luaset_letra_str \__spintent_geo_luaset_sub_str }
4414         }
4415     }
4416     {
4417         {
4418             \__spintent_inner_sep:
4419             \MathMLarg { \__spintent_mathmlarg_point_ref_tl }
4420             {
4421                 {
4422                     \__spintent_geo_luaset_letra_str
4423                     \__spintent_geo_luaset_sub_str
4424                 }
4425             }
4426         }
4427     }
4428 }

```

Otherwise, the same distinction applies to the combined `\__spintent_geo_luaset_print_tl`: the shared queue for `read-cmd=school` or when `silent` is `true`, and a native `\MathMLarg` for `read-cmd=mathcat`. The two mechanisms are never mixed in the same formula.

```

4429     {
4430         \bool_if:NTF \__spintent_mathmlarg_point_silent_bool
4431         {
4432             \tl_set:Ne \__spintent_mathmlarg_point_kv_tl
4433             { arg = "\__spintent_mathmlarg_point_ref_tl", intent = "_-" }
4434             \__spintent_luafun_mrow_intent_arg:nn
4435             { } % must be empty
4436             { \__spintent_mathmlarg_point_kv_tl }
4437             { \__spintent_geo_luaset_print_tl }
4438         }
4439         {
4440             \str_if_eq:VnTF \__spintent_mathmlarg_point_mode_str { school }
4441             {
4442                 \tl_set:Ne \__spintent_mathmlarg_point_kv_tl
4443                 {
4444                     arg = "\__spintent_mathmlarg_point_ref_tl",
4445                     intent = "_ \__spintent_geo_luaset_intent_str -"
4446                 }
4447                 \__spintent_luafun_mrow_intent_arg:nn
4448                 { } % must be empty
4449                 { \__spintent_mathmlarg_point_kv_tl }
4450                 { \__spintent_geo_luaset_print_tl }
4451             }
4452             {
4453                 \MathMLarg { \__spintent_mathmlarg_point_ref_tl }
4454                 { \__spintent_geo_luaset_print_tl }
4455             }
4456         }
4457     }
4458 }
4459 \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`

`\__spintent_spPoint_mode_str`
`\__spintent_spPoint_prefix_tl`

The variables `\__spintent_spPoint_mode_str` and `\__spintent_spPoint_prefix_tl` store the values set by the keys `read-cmd` and `prefix`.

```
4460 \str_new:N \__spintent_spPoint_mode_str
4461 \tl_new:N \__spintent_spPoint_prefix_tl
```

(End of definition for `\__spintent_spPoint_mode_str` and `\__spintent_spPoint_prefix_tl`.)

`\__spintent_spPoint_intent_arg_tl`

The variable `\__spintent_spPoint_intent_arg_tl` stores the `intent` being built.

```
4462 \tl_new:N \__spintent_spPoint_intent_arg_tl
```

(End of definition for `\__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.

```
4463 \msg_new:nnnn { spintent } { spPoint-invalid-point }
4464 {
4465   Invalid~argument~'~#1'~for~'\token_to_str:N \spPoint'~\msg_line_context:.
4466 }
4467 {
4468   The~argument~must~be~a~single~uppercase~letter~with~an~optional~subscript~(\_{...})~
4469   or~prime~(').
4470 }
```

(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`.

```
4471 \keys_define:nn { spintent / spPoint }
4472 {
4473   read-cmd .choices:nn = { school , mathcat }
4474   {
4475     \int_case:nn { \l_keys_choice_int }
4476     {
4477       {1} { \str_set:Nn \__spintent_spPoint_mode_str { school } }
4478       {2} { \str_set:Nn \__spintent_spPoint_mode_str { mathcat } }
4479     }
4480   },
4481   read-cmd .initial:n = school,
4482   read-cmd .value_required:n = true,
4483   read-cmd / unknown .code:n =
4484     \msg_error:nneee { spintent } { unknown-choice }
4485     { read-cmd } { school,~mathcat } { \exp_not:n {#1} },
4486   prefix .code:n =
4487     { \__spintent_sanitizet_affix:Nn \__spintent_spPoint_prefix_tl {#1} },
4488   prefix .initial:n = ,
4489   prefix .value_required:n = true,
4490   silent .bool_set:N = \__spintent_spPoint_silent_bool,
4491   silent .initial:n = false,
4492   silent .value_required:n = true,
4493   concept .bool_set:N = \__spintent_spPoint_concept_bool,
4494   concept .initial:n = true,
4495   concept .value_required:n = true,
4496   unknown .code:n =
4497     {
4498       \str_set:Nn \__spintent_command_name_str { spPoint }
4499       \__spintent_unknown_keys_cmd:n {#1}
4500     },
4501 }
```

(End of definition for `read-cmd` and others.)

15.44.4 Internal functions for \spPoint

The internal function `\__spintent_spPoint_exec:nnn` processes the $\langle argument \rangle$ 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 `\__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 $\langle argument \rangle$, it sets the optional $\langle keys \rangle$ passed in ‘#1’.

```

4502 \cs_new_protected:Nn \__spintent_spPoint_exec:nnn
4503 {
4504   \keys_set:nn { spintent / spPoint } {#1}
4505   \__spintent_luafun_geo_point_parse_and_set:n { \exp_not:n {#2} }
4506   \str_if_eq:VnT \__spintent_geo_luaset_error_str { true }
4507   {
4508     \msg_error:nne { spintent } { spPoint-invalid-point } { \exp_not:n {#2} }
4509   }
4510   \bool_if:nTF
4511   {
4512     \bool_lazy_or_p:nn
4513     { \__spintent_spPoint_silent_bool }
4514     { ! \__spintent_spPoint_concept_bool }
4515   }
4516   {
4517     \tl_set:Nn \__spintent_spPoint_intent_arg_tl { _{ \$pt } }
4518     \bool_if:NT \__spintent_spPoint_silent_bool
4519     {
4520       \tl_set:Nn \__spintent_spPoint_intent_arg_tl { _{-} }
4521     }
4522   }
4523   {
4524     \tl_set:Nn \__spintent_spPoint_intent_arg_tl
4525     {
4526       -
4527       \tl_if_empty:NF \__spintent_spPoint_prefix_tl
4528       {
4529         \tl_use:N \__spintent_spPoint_prefix_tl -
4530       }
4531       punto:prefix(\$pt)
4532     }
4533   }
4534   \__spintent_spPoint_mathml:n {#2}
4535   \tl_if_novalue:nF {#3}
4536   {
4537     \MathMLintent { :silent } { \c__spintent_invisible_sep_int } % prevent "por"
4538     \tl_set:Nn \__spintent_spcoord_raw_arg_tl {#3}
4539     \__spintent_luafun_spcoord_parse_and_set:n { \exp_not:n {#3} }
4540     \str_if_eq:VnT \__spintent_spcoord_luaset_error_str { true }
4541     { \msg_error:nne { spintent } { spcoord-invalid-arg } { \exp_not:n {#3} } }
4542     \__spintent_spcoord_set_wrap_parens_size:
4543     \str_if_eq:VnF \__spintent_spcoord_luaset_error_str { true }
4544     { \__spintent_spcoord_mathml:N \__spintent_spPoint_silent_bool }
4545   }
4546 }

```

(End of definition for `\__spintent_spPoint_exec:nnn`.)

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 `\__spintent_spPoint_intent_arg_tl` as the concept intent. Inside that group it calls `\__spintent_inner_sep:` first.

```

4547 \cs_new_protected:Nn \__spintent_spPoint_mathml:n
4548 {
4549   \__spintent_invisible_sep:
4550   \MathMLintent { \__spintent_spPoint_intent_arg_tl }
4551   {
4552     \__spintent_inner_sep:
4553     \__spintent_mathmlarg_point:nnn { pt } { spPoint } {#1}
4554   }
4555 }

```

(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`.

```

4556 \NewDocumentCommand \spPoint { 0{} m d() }
4557 {
4558   \mode_if_math:TF
4559   {
4560     \group_begin:
4561       \attribute \__spintent_scope_attr_int = 1
4562       \__spintent_spPoint_exec:nnn {#1} {#2} {#3}
4563     \group_end:
4564   }
4565   { \msg_error:nnn { spintent } { need-math-mode } { \spPoint } }
4566 }

```

(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 $\langle argument \rangle$: 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 $\langle argument \rangle$.

15.45.1 Definition of variables for `\spvec`

`\__spintent_spvec_mode_str` The variables `\__spintent_spvec_mode_str` and `\__spintent_spvec_prefix_tl` store the values set by the keys `read-cmd` and `prefix`.

```

4567 \str_new:N \__spintent_spvec_mode_str
4568 \tl_new:N \__spintent_spvec_prefix_tl

```

(End of definition for `\__spintent_spvec_mode_str` and `\__spintent_spvec_prefix_tl`.)

`\__spintent_spvec_pt_seq` The variable `\__spintent_spvec_pt_seq` stores the points of the argument, split at the commas, and `\__spintent_spvec_pt_idx_int` counts the point being read. The variable `\__spintent_spvec_refs_tl` stores the reference list `$pts1, $pts2, \dots` built from them.

```

4569 \seq_new:N \__spintent_spvec_pt_seq
4570 \int_new:N \__spintent_spvec_pt_idx_int
4571 \tl_new:N \__spintent_spvec_refs_tl

```

(End of definition for `\__spintent_spvec_pt_seq`, `\__spintent_spvec_pt_idx_int`, and `\__spintent_spvec_refs_tl`.)

`\__spintent_spvec_intent_arg_tl` The variable `\__spintent_spvec_intent_arg_tl` stores the *intent* being built and `\__spintent_spvec_kv_tl` stores the key-value list with the attributes `arg` and `intent` of the points.

```

4572 \tl_new:N \__spintent_spvec_intent_arg_tl
4573 \tl_new:N \__spintent_spvec_kv_tl

```

(End of definition for `\__spintent_spvec_intent_arg_tl` and `\__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.

```

4574 \msg_new:nnnn { spintent } { spvec-invalid-arg }
4575 {
4576   Invalid~argument~'#1'~for~'\token_to_str:N \spvec'~\msg_line_context:.
4577 }
4578 {
4579   The~argument~must~be~a~lowercase~letter~or~two~points~separated~by~a~comma.
4580 }

```

(End of definition for `spvec-invalid-arg`.)

15.45.3 Definition of keys for `\spvec`

`silent` We define the $\langle keys \rangle$ `silent`, `concept`, `autofit`, `read-cmd` and `prefix`.

```

concept
autofit
read-cmd
prefix
unknown
4581 \keys_define:nn { spintent / spvec }
4582 {
4583   silent .bool_set:N = \__spintent_spvec_silent_bool,

```

```

4584   silent   .initial:n = false,
4585   silent   .value_required:n = true,
4586   concept  .bool_set:N = \l__spinttent_spvec_concept_bool,
4587   concept  .initial:n = true,
4588   concept  .value_required:n = true,
4589   autofit  .bool_set:N = \l__spinttent_spvec_autofit_bool,
4590   autofit  .initial:n = false,
4591   autofit  .value_required:n = true,
4592   read-cmd .choices:nn = { school , mathcat }
4593   {
4594     \int_case:nn { \l_keys_choice_int }
4595     {
4596       {1} { \str_set:Nn \l__spinttent_spvec_mode_str { school } }
4597       {2} { \str_set:Nn \l__spinttent_spvec_mode_str { mathcat } }
4598     }
4599   },
4600   read-cmd .initial:n = school,
4601   read-cmd .value_required:n = true,
4602   read-cmd / unknown .code:n =
4603     \msg_error:nneee { spinttent } { unknown-choice }
4604     { read-cmd } { school,~mathcat } { \exp_not:n {##1} },
4605   prefix   .code:n =
4606     { \__spinttent_sanitizet_affix:Nn \l__spinttent_spvec_prefix_tl {#1} },
4607   prefix   .initial:n = { },
4608   prefix   .value_required:n = true,
4609   unknown  .code:n =
4610     {
4611       \str_set:Nn \l__spinttent_command_name_str { spvec }
4612       \__spinttent_unknown_keys_cmd:n {#1}
4613     },
4614   }

```

(End of definition for *silent* and others.)

15.45.4 Internal functions for \spvec

`\__spinttent_spvec_exec:nnn` The internal function `\__spinttent_spvec_exec:nnn` processes the $\langle argument \rangle$ passed in ‘#2’. Calls `\__spinttent_luafun_geo_parse_and_set:n`, reporting **spvec-invalid-arg** on error, on a proper name with `\l__spinttent_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 `\__spinttent_spvec_mathml:n`.

Before processing the $\langle argument \rangle$, it sets the optional $\langle keys \rangle$ passed in ‘#1’.

```

4615 \cs_new_protected:Nn \__spinttent_spvec_exec:nnn
4616 {
4617   \keys_set:nn { spinttent / spvec } {#1}
4618   \__spinttent_luafun_geo_parse_and_set:n { \exp_not:n {#2} }
4619   \str_if_eq:VnT \l__spinttent_geo_luaset_is_name_str { true }
4620   {
4621     \str_if_eq:VnT \l__spinttent_geo_luaset_tipo_str { prima }
4622     { \str_set:Nn \l__spinttent_geo_luaset_error_str { true } }
4623   }
4624   \str_if_eq:VnT \l__spinttent_geo_luaset_is_name_str { false }
4625   {
4626     \int_compare:nNnF { \clist_count:N \l__spinttent_geo_luaset_points_clist } = { 2 }
4627     { \str_set:Nn \l__spinttent_geo_luaset_error_str { true } }
4628   }
4629   \str_if_eq:VnTF \l__spinttent_geo_luaset_error_str { true }
4630   { \msg_error:nne { spinttent } { spvec-invalid-arg } { \exp_not:n {#2} } }
4631   { \__spinttent_spvec_mathml:n {#2} }
4632   \tl_if_novalue:nF {#3}
4633   {
4634     \bool_if:NTF \l__spinttent_spvec_silent_bool
4635     { \MathMLintent { :silent } { \equal } }
4636     { \equal } % vector notation =( , )
4637     % prevent "intervalo"
4638     \MathMLintent { :silent } { \c__spinttent_invisible_sep_int }
4639     \tl_set:Nn \l__spinttent_spcoord_raw_arg_tl {#3}
4640     \__spinttent_luafun_spcoord_parse_and_set:n { \exp_not:n {#3} }
4641     \str_if_eq:VnT \l__spinttent_spcoord_luaset_error_str { true }
4642     { \msg_error:nne { spinttent } { spcoord-invalid-arg } { \exp_not:n {#3} } }
4643     \__spinttent_spcoord_set_wrap_parens_size:
4644     \str_if_eq:VnF \l__spinttent_spcoord_luaset_error_str { true }

```

```

4645     { \l__spinttent_spcoord_mathml:N \l__spinttent_spvec_silent_bool }
4646   }
4647 }

```

(End of definition for `\l__spinttent_spvec_exec:nnn`.)

`\l__spinttent_spvec_mathml:n`

The function `\l__spinttent_spvec_mathml:n` branches on `\l__spinttent_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, `_ $\textit{pt}$` when `concept` is `false`, and otherwise `_vector:prefix( $\textit{pt}$ )`, with `prefix` placed after the leading `_` when given. With `\l__spinttent_geo_luaset_tipo_str` equal to `sub`, `read-cmd=school` tags `\vec{letra}` with `\l__spinttent_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.

```

4648 \cs_new_protected:Nn \l__spinttent_spvec_mathml:n
4649 {
4650   \str_if_eq:VnTF \l__spinttent_geo_luaset_is_name_str { true }
4651   {
4652     \bool_if:nTF
4653     {
4654       \bool_lazy_or_p:nn
4655       { \l__spinttent_spvec_silent_bool }
4656       { ! \l__spinttent_spvec_concept_bool }
4657     }
4658     {
4659       \tl_set:Nn \l__spinttent_spvec_intent_arg_tl { _( $\textit{pt}$ ) }
4660       \bool_if:NT \l__spinttent_spvec_silent_bool
4661       { \tl_set:Nn \l__spinttent_spvec_intent_arg_tl { _- } }
4662     }
4663     {
4664       \tl_set:Ne \l__spinttent_spvec_intent_arg_tl
4665       {
4666         -
4667         \tl_if_empty:NF \l__spinttent_spvec_prefix_tl
4668         { \tl_use:N \l__spinttent_spvec_prefix_tl - }
4669         vector:prefix( $\textit{pt}$ )
4670       }
4671     }
4672   }
4673   \str_case:VnF \l__spinttent_geo_luaset_tipo_str
4674   {
4675     { sub }
4676     {
4677       \str_if_eq:VnTF \l__spinttent_spvec_mode_str { school }
4678       {
4679         \bool_if:NTF \l__spinttent_spvec_silent_bool
4680         {
4681           \tl_set:Nn \l__spinttent_spvec_kv_tl { arg = "pt", intent = "_-" }
4682         }
4683         {
4684           \tl_set:Ne \l__spinttent_spvec_kv_tl
4685           {
4686             arg = "pt", intent = "_ \l__spinttent_geo_luaset_intent_str -"
4687           }
4688         }
4689       }
4690       \__spinttent_invisible_sep:
4691       \MathMLintent { \l__spinttent_spvec_intent_arg_tl }
4692       {
4693         \__spinttent_inner_sep:
4694         \__spinttent_luafun_mrow_intent_arg:nn
4695         { } { \l__spinttent_spvec_kv_tl }
4696       }
4697       {
4698         \vec
4699         {
4700           \bool_if:NT \l__spinttent_spvec_autofit_bool
4701           { \vphantom{ b } }
4702           \l__spinttent_geo_luaset_letra_str
4703         }
4704       }
4705       \c_math_subscript_token
4706       {
4707         \MathMLintent { :silent }
4708       }
4709     }
4710   }
4711 }

```

```

4704             { \l__spintent_geo_luaset_sub_str }
4705         }
4706     }
4707 }
4708 }
4709 {
4710     \l__spintent_invisible_sep:
4711     \MathMLintent { \l__spintent_sptvec_intent_arg_tl }
4712     {
4713         \l__spintent_inner_sep:
4714         \vec
4715         {
4716             \bool_if:NT \l__spintent_sptvec_autofit_bool
4717             { \vphantom{ b } }
4718             \MathMLarg{ pt } { \l__spintent_geo_luaset_letra_str }
4719         }
4720         \bool_if:NT \l__spintent_sptvec_silent_bool
4721         { \c_math_subscript_token { \l__spintent_geo_luaset_sub_str } }
4722     }
4723     \bool_if:NF \l__spintent_sptvec_silent_bool
4724     { \c_math_subscript_token { \l__spintent_geo_luaset_sub_str } }
4725 }
4726 }
4727 }
4728 {
4729     \bool_if:NTF \l__spintent_sptvec_silent_bool
4730     { \tl_set:Nn \l__spintent_sptvec_kv_tl { arg = "pt", intent = "_-" } }
4731     {
4732         \tl_set:Nn \l__spintent_sptvec_kv_tl
4733         { arg = "pt", intent = "_ \l__spintent_geo_luaset_intent_str -" }
4734     }
4735     \l__spintent_invisible_sep:
4736     \MathMLintent { \l__spintent_sptvec_intent_arg_tl }
4737     {
4738         \l__spintent_inner_sep:
4739         \l__spintent_luafun_mrow_intent_arg:nn
4740         { } { \l__spintent_sptvec_kv_tl }
4741     }
4742     \vec
4743     {
4744         \bool_if:NT \l__spintent_sptvec_autofit_bool
4745         { \vphantom{ b } }
4746         \l__spintent_geo_luaset_print_tl
4747     }
4748 }
4749 }
4750 }
4751 }

```

For a pair of points, `\l__spintent_geo_luaset_points_clist` is split into `\l__spintent_sptvec_pt_seq` and walked to build `\l__spintent_sptvec_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 `\l__spintent_mathmlarg_point:nnn` once per point.

```

4752     {
4753         \seq_set_from_clist:NN \l__spintent_sptvec_pt_seq
4754         \l__spintent_geo_luaset_points_clist
4755         \tl_clear:N \l__spintent_sptvec_refs_tl
4756         \int_zero:N \l__spintent_sptvec_pt_idx_int
4757         \seq_map_inline:Nn \l__spintent_sptvec_pt_seq
4758         {
4759             \int_incr:N \l__spintent_sptvec_pt_idx_int
4760             \tl_if_empty:NF \l__spintent_sptvec_refs_tl
4761             { \tl_put_right:Nn \l__spintent_sptvec_refs_tl { , } }
4762             \tl_put_right:Ne \l__spintent_sptvec_refs_tl
4763             { $ pts \int_use:N \l__spintent_sptvec_pt_idx_int }
4764         }
4765         \bool_if:nTF
4766         {
4767             \bool_lazy_or_p:nn
4768             { \l__spintent_sptvec_silent_bool }

```

```

4769         { ! \l__spintent_svec_concept_bool }
4770     }
4771     {
4772         \tl_set:Nx \l__spintent_svec_intent_arg_tl
4773         { _ ( \l__spintent_svec_refs_tl ) }
4774         \bool_if:NT \l__spintent_svec_silent_bool
4775         { \tl_set:Nn \l__spintent_svec_intent_arg_tl { _- } }
4776     }
4777     {
4778         \tl_set:Nx \l__spintent_svec_intent_arg_tl
4779         {
4780             -
4781             \tl_if_empty:NF \l__spintent_svec_prefix_tl
4782             { \tl_use:N \l__spintent_svec_prefix_tl - }
4783             vector:prefix( \l__spintent_svec_refs_tl )
4784         }
4785     }
4786     \__spintent_invisible_sep:
4787     \MathMLintent { \l__spintent_svec_intent_arg_tl }
4788     {
4789         \__spintent_inner_sep:
4790         \overrightarrow
4791         {
4792             \int_zero:N \l__spintent_svec_pt_idx_int
4793             \seq_map_inline:Nn \l__spintent_svec_pt_seq
4794             {
4795                 \int_incr:N \l__spintent_svec_pt_idx_int
4796                 \__spintent_mathmlarg_point:nnn
4797                 { pts \int_use:N \l__spintent_svec_pt_idx_int }
4798                 { svec }
4799                 { ##1 }
4800             }
4801         }
4802     }
4803 }
4804 }

```

(End of definition for `\__spintent_svec_mathml:n`.)

\svec The public command `\svec` enforces *math mode*, opens a local group, and delegates execution to `\__spintent_svec_exec:nnn`.

```

4805 \NewDocumentCommand \svec { O{} m d() }
4806 {
4807     \mode_if_math:TF
4808     {
4809         \group_begin:
4810         \attribute \l__spintent_scope_attr_int = 1
4811         \__spintent_svec_exec:nnn {#1} {#2} {#3}
4812         \group_end:
4813     }
4814     { \msg_error:nnn { spintent } { need-math-mode } { \svec } }
4815 }

```

(End of definition for `\svec`. This function is documented on page 25.)

15.46 The command `\sprecta`

The user command `\sprecta` supports two types of notation for the input $\langle argument \rangle$: when the argument contains two points separated by an optional comma, `\overlefttrightarrow` 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`.

```

4816 \str_new:N \l__spintent_sprecta_mode_str
4817 \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` 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, \dots` built from them.

```
4818 \seq_new:N \l__spintent_sprecta_pt_seq
4819 \int_new:N \l__spintent_sprecta_pt_idx_int
4820 \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` 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.

```
4821 \tl_new:N \l__spintent_sprecta_intent_arg_tl
4822 \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.

```
4823 \msg_new:nnnn { spintent } { sprecta-invalid-points }
4824 {
4825   Invalid~argument~'#1'~for~'\token_to_str:N \sprecta'\~\msg_line_context:.
4826 }
4827 {
4828   The~argument~must~be~a~letter~or~two~points~separated~by~a~comma.
4829 }
```

(End of definition for `sprecta-invalid-points`.)

15.46.3 Definition of keys for `\sprecta`

`read-cmd` We define the `(keys)` `read-cmd`, `prefix`, `silent` and `concept`.

```
prefix
silent
concept
unknown
4830 \keys_define:nn { spintent / sprecta }
4831 {
4832   read-cmd .choices:nn = { school , mathcat }
4833   {
4834     \int_case:nn { \l_keys_choice_int }
4835     {
4836       {1} { \str_set:Nn \l__spintent_sprecta_mode_str { school } }
4837       {2} { \str_set:Nn \l__spintent_sprecta_mode_str { mathcat } }
4838     }
4839   },
4840   read-cmd .initial:n = school,
4841   read-cmd .value_required:n = true,
4842   read-cmd / unknown .code:n =
4843     \msg_error:nneee { spintent } { unknown-choice }
4844     { read-cmd } { school,~mathcat } { \exp_not:n {#1} },
4845   prefix .code:n =
4846     { \__spintent_sanitizize_affix:Nn \l__spintent_sprecta_prefix_tl {#1} },
4847   prefix .initial:n = ,
4848   prefix .value_required:n = true,
4849   silent .bool_set:N = \l__spintent_sprecta_silent_bool,
4850   silent .initial:n = false,
4851   silent .value_required:n = true,
4852   concept .bool_set:N = \l__spintent_sprecta_concept_bool,
4853   concept .initial:n = true,
4854   concept .value_required:n = true,
4855   unknown .code:n =
4856     {
4857       \str_set:Nn \l__spintent_command_name_str { sprecta }
4858       \__spintent_unknown_keys_cmd:n {#1}
4859     },
4860 }
```

(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 $\langle argument \rangle$ 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 $\langle argument \rangle$, it sets the optional $\langle keys \rangle$ passed in ‘#1’.

```

4861 \cs_new_protected:Nn \__spintent_sprecta_exec:nn
4862 {
4863   \keys_set:nn { spintent / sprecta } {#1}
4864   \__spintent_luafun_geo_parse_and_set:n { \exp_not:n {#2} }
4865   \str_if_eq:VnT \__spintent_geo_luaset_error_str { true }
4866   {
4867     \msg_error:nne { spintent } { sprecta-invalid-points } { \exp_not:n {#2} }
4868   }
4869   \__spintent_sprecta_mathml:n {#2}
4870 }

```

(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 `_ $\langle$pt\rangle$` when `silent` is `true` or `concept` is `false`, and `_recta:prefix( $\langle$pt\rangle$ )` otherwise, with `prefix` placed after the leading `_` when given; with `silent = true` and `read-cmd = school` 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.

```

4871 \cs_new_protected:Nn \__spintent_sprecta_mathml:n
4872 {
4873   \str_if_eq:VnTF \l__spintent_geo_luaset_is_name_str { true }
4874   {
4875     \__spintent_invisible_sep:
4876     \bool_if:nTF
4877     {
4878       \bool_lazy_or:nn
4879       { \l__spintent_sprecta_silent_bool }
4880       { ! \l__spintent_sprecta_concept_bool }
4881     }
4882     {
4883       \tl_set:Nn \l__spintent_sprecta_intent_arg_tl { _( $\langle$pt\rangle$ ) }
4884       \bool_if:NT \l__spintent_sprecta_silent_bool
4885       {
4886         \str_if_eq:VnF \l__spintent_sprecta_mode_str { school }
4887         { \tl_set:Nn \l__spintent_sprecta_intent_arg_tl { _- } }
4888       }
4889     }
4890     {
4891       \tl_set:Ne \l__spintent_sprecta_intent_arg_tl
4892       {
4893         -
4894         \tl_if_empty:NF \l__spintent_sprecta_prefix_tl
4895         { \tl_use:N \l__spintent_sprecta_prefix_tl - }
4896         recta:prefix( $\langle$pt\rangle$ )
4897       }
4898     }
4899     \str_case:VnF \l__spintent_geo_luaset_tipo_str
4900     {
4901       { sub }
4902       {
4903         \str_if_eq:VnTF \l__spintent_sprecta_mode_str { school }
4904         {
4905           \bool_if:NTF \l__spintent_sprecta_silent_bool
4906           {
4907             \tl_set:Nn \l__spintent_sprecta_kv_tl
4908             { arg = "pt", intent = "_-" }
4909           }
4910           {
4911             \tl_set:Ne \l__spintent_sprecta_kv_tl
4912             {

```

```

4913         arg = "pt", intent = "_ \l__spintent_geo_luaset_intent_str -"
4914     }
4915 }
4916 \MathMLintent { \l__spintent_sprecta_intent_arg_tl }
4917 {
4918     \__spintent_inner_sep:
4919     \__spintent_luafun_mrow_intent_arg:nn
4920     { } { \l__spintent_sprecta_kv_tl }
4921     {
4922         \l__spintent_geo_luaset_letra_str
4923         \c_math_subscript_token
4924         {
4925             \MathMLintent { :silent }
4926             { \l__spintent_geo_luaset_sub_str }
4927         }
4928     }
4929 }
4930 }

```

For `read-cmd=mathcat`, the same sub case tags the base and the subscript together with a native `\MathMLarg`, without using the shared queue.

```

4931     {
4932         \MathMLintent { \l__spintent_sprecta_intent_arg_tl }
4933         {
4934             {
4935                 \__spintent_inner_sep:
4936                 \MathMLarg { pt }
4937                 {
4938                     {
4939                         \l__spintent_geo_luaset_letra_str
4940                         \c_math_subscript_token
4941                         { \l__spintent_geo_luaset_sub_str }
4942                     }
4943                 }
4944             }
4945         }
4946     }
4947 }

```

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`.

```

4948     { prima }
4949     {
4950         \str_if_eq:VnTF \l__spintent_sprecta_mode_str { school }
4951         {
4952             \bool_if:NTF \l__spintent_sprecta_silent_bool
4953             {
4954                 \tl_set:Nn \l__spintent_sprecta_kv_tl
4955                 { arg = "pt", intent = "_-" }
4956             }
4957             {
4958                 \tl_set:Ne \l__spintent_sprecta_kv_tl
4959                 {
4960                     arg = "pt", intent = "_ \l__spintent_geo_luaset_intent_str -"
4961                 }
4962             }
4963             \MathMLintent { \l__spintent_sprecta_intent_arg_tl }
4964             {
4965                 \__spintent_inner_sep:
4966                 \__spintent_luafun_mrow_intent_arg:nn
4967                 { } { \l__spintent_sprecta_kv_tl }
4968                 {
4969                     \l__spintent_geo_luaset_letra_str
4970                     \l__spintent_geo_luaset_sub_str
4971                 }
4972             }
4973         }
4974     }
4975     \MathMLintent { \l__spintent_sprecta_intent_arg_tl }

```

```

4976         {
4977         {
4978             \__spintent_inner_sep:
4979             \MathMLarg { pt }
4980             {
4981             {
4982                 \l__spintent_geo_luaset_letra_str
4983                 \l__spintent_geo_luaset_sub_str
4984             }
4985             }
4986         }
4987     }
4988 }
4989 }
4990 }

```

Otherwise, the same kv logic wraps the combined `\l__spintent_geo_luaset_print_tl` as a whole: with the shared queue for `read-cmd=school` and with its own `arg` alone for `read-cmd=mathcat`.

```

4991 {
4992     \bool_if:NTF \l__spintent_sprecta_silent_bool
4993     {
4994         \tl_set:Nn \l__spintent_sprecta_kv_tl { arg = "pt", intent = "-" }
4995     }
4996     {
4997         \str_if_eq:VnTF \l__spintent_sprecta_mode_str { school }
4998         {
4999             \tl_set:Ne \l__spintent_sprecta_kv_tl
5000             { arg = "pt", intent = "-" \l__spintent_geo_luaset_intent_str -" }
5001         }
5002         {
5003             \tl_set:Nn \l__spintent_sprecta_kv_tl { arg = "pt" }
5004         }
5005     }
5006     \__spintent_luafun_mrow_intent_arg:nn
5007     { \l__spintent_sprecta_intent_arg_tl }
5008     { \l__spintent_sprecta_kv_tl }
5009     { \l__spintent_geo_luaset_print_tl }
5010 }
5011 }

```

For a list of points, `\l__spintent_geo_luaset_points_clist` is split into `\l__spintent_sprecta_pt_seq` and walked to build `\l__spintent_sprecta_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 `\overleftarrow`. The sequence is walked again, calling `\__spintent_mathmlarg_point:nnn` once per point.

```

5012 {
5013     \seq_set_from_clist:NN \l__spintent_sprecta_pt_seq
5014     \l__spintent_geo_luaset_points_clist
5015     \tl_clear:N \l__spintent_sprecta_refs_tl
5016     \int_zero:N \l__spintent_sprecta_pt_idx_int
5017     \seq_map_inline:Nn \l__spintent_sprecta_pt_seq
5018     {
5019         \int_incr:N \l__spintent_sprecta_pt_idx_int
5020         \tl_if_empty:NF \l__spintent_sprecta_refs_tl
5021         { \tl_put_right:Nn \l__spintent_sprecta_refs_tl { , } }
5022         \tl_put_right:Ne \l__spintent_sprecta_refs_tl
5023         { $ pt \int_use:N \l__spintent_sprecta_pt_idx_int }
5024     }
5025     \bool_if:nTF
5026     {
5027         \bool_lazy_or:p:nn
5028         { \l__spintent_sprecta_silent_bool }
5029         { ! \l__spintent_sprecta_concept_bool }
5030     }
5031     {
5032         \tl_set:Ne \l__spintent_sprecta_intent_arg_tl
5033         { _ ( \l__spintent_sprecta_refs_tl ) }
5034         \bool_if:NT \l__spintent_sprecta_silent_bool
5035         { \tl_set:Nn \l__spintent_sprecta_intent_arg_tl { _- } }
5036     }
5037 }

```

```

5038         \tl_set:Nx \l__spintent_sprecta_intent_arg_tl
5039         {
5040             -
5041             \tl_if_empty:NF \l__spintent_sprecta_prefix_tl
5042             { \tl_use:N \l__spintent_sprecta_prefix_tl - }
5043             recta:prefix( \l__spintent_sprecta_refs_tl )
5044         }
5045     }
5046     \__spintent_invisible_sep:
5047     \MathMLintent { \l__spintent_sprecta_intent_arg_tl }
5048     {
5049         \__spintent_inner_sep:
5050         \overlefttrightharpoon
5051         {
5052             \int_zero:N \l__spintent_sprecta_pt_idx_int
5053             \seq_map_inline:Nn \l__spintent_sprecta_pt_seq
5054             {
5055                 \int_incr:N \l__spintent_sprecta_pt_idx_int
5056                 \__spintent_mathmlarg_point:nnn
5057                 { pt \int_use:N \l__spintent_sprecta_pt_idx_int }
5058                 { sprecta }
5059                 { ##1 }
5060             }
5061         }
5062     }
5063 }
5064 }

```

(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`.

```

5065 \NewDocumentCommand \sprecta { 0{} m }
5066 {
5067     \mode_if_math:TF
5068     {
5069         \group_begin:
5070         \attribute \l__spintent_scope_attr_int = 1
5071         \__spintent_sprecta_exec:nn {##1} {##2}
5072         \group_end:
5073     }
5074     { \msg_error:nnn { spintent } { need-math-mode } { \sprecta } }
5075 }

```

(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`

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`.

```

5076 \str_new:N \l__spintent_sprayo_mode_str
5077 \tl_new:N \l__spintent_sprayo_prefix_tl
5078 \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`.)

The variable `\l__spintent_sprayo_pt_seq` stores the points taken from `\l__spintent_geo_luaset_points_cl` 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.

```

5079 \seq_new:N \l__spintent_sprayo_pt_seq
5080 \int_new:N \l__spintent_sprayo_pt_idx_int
5081 \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.

```
5082 \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.

```
5083 \msg_new:nnnn { spintent } { sprayo-invalid-points }
5084 {
5085   Invalid~argument~'#1'~for~'\token_to_str:N \sprayo'~\msg_line_context:.
5086 }
5087 {
5088   The~argument~must~be~two~points~separated~by~a~comma.
5089 }
```

(End of definition for `sprayo-invalid-points`.)

15.47.3 Definition of keys for `\sprayo`

`read-cmd`

We define the *(keys)* `read-cmd`, `prefix`, `silent`, `concept` and `read-sty`.

`prefix`

`silent`

`concept`

`read-sty`

`unknown`

```
5090 \keys_define:nn { spintent / sprayo }
5091 {
5092   read-cmd .choices:nn = { school , mathcat }
5093   {
5094     \int_case:nn { \l_keys_choice_int }
5095     {
5096       {1} { \str_set:Nn \l__spintent_sprayo_mode_str { school } }
5097       {2} { \str_set:Nn \l__spintent_sprayo_mode_str { mathcat } }
5098     }
5099   },
5100   read-cmd .initial:n = school,
5101   read-cmd .value_required:n = true,
5102   read-cmd / unknown .code:n =
5103     \msg_error:nneee { spintent } { unknown-choice }
5104     { read-cmd } { school,~mathcat } { \exp_not:n {#1} },
5105   prefix .code:n =
5106     { \__spintent_sanitizet_affix:Nn \l__spintent_sprayo_prefix_tl {#1} },
5107   prefix .initial:n = ,
5108   prefix .value_required:n = true,
5109   silent .bool_set:N = \l__spintent_sprayo_silent_bool,
5110   silent .initial:n = false,
5111   silent .value_required:n = true,
5112   concept .bool_set:N = \l__spintent_sprayo_concept_bool,
5113   concept .initial:n = true,
5114   concept .value_required:n = true,
5115   read-sty .choices:nn = { rayo , semirrecta }
5116   {
5117     \int_case:nn { \l_keys_choice_int }
5118     {
5119       {1} { \str_set:Nn \l__spintent_sprayo_read_sty_str { rayo } }
5120       {2} { \str_set:Nn \l__spintent_sprayo_read_sty_str { semirrecta } }
5121     }
5122   },
5123   read-sty .initial:n = rayo,
5124   read-sty .value_required:n = true,
5125   read-sty / unknown .code:n =
5126     \msg_error:nneee { spintent } { unknown-choice }
5127     { read-sty } { rayo,~semirrecta } { \exp_not:n {#1} },
5128   unknown .code:n =
5129     {
5130       \str_set:Nn \l__spintent_command_name_str { sprayo }
5131       \__spintent_unknown_keys_cmd:n {#1}
5132     },
5133 }
```

(End of definition for `read-cmd` and others.)

15.47.4 Internal functions for \spray

__spintent_sprayo_exec:nn

The internal function `\__spintent_sprayo_exec:nn` processes the $\langle argument \rangle$ passed in ‘#2’. It calls `\__spintent_luafun_geo_parse_and_set:n` to process the $\langle argument \rangle$, returning an “error” if the input is invalid, then calls `\__spintent_sprayo_mathml:`. The command `\spray` always takes a list of points, so no `intent` is built here.

Before processing the $\langle argument \rangle$, it sets the optional $\langle keys \rangle$ passed in ‘#1’.

```

5134 \cs_new_protected:Nn \__spintent_sprayo_exec:nn
5135 {
5136   \keys_set:nn { spintent / sprayo } {#1}
5137   \__spintent_luafun_geo_parse_and_set:n { \exp_not:n {#2} }
5138   \bool_if:nT
5139     {
5140       \str_if_eq_p:Vn \__spintent_geo_luaset_error_str { true } ||
5141       \str_if_eq_p:Vn \__spintent_geo_luaset_is_name_str { true }
5142     }
5143     {
5144       \msg_error:nne { spintent } { sprayo-invalid-points } { \exp_not:n {#2} }
5145     }
5146   \__spintent_sprayo_mathml:
5147 }
```

(End of definition for `\__spintent_sprayo_exec:nn`.)

__spintent_sprayo_mathml:

The function `\__spintent_sprayo_mathml:` splits `\__spintent_geo_luaset_points_clist`, which Lua sets with one raw point per element, into `\__spintent_sprayo_pt_seq` and walks it once to build `\__spintent_sprayo_refs_tl`, the comma-separated $\$pts1, \$pts2, \dots$ reference list, naming each point by position.

The concept `intent` is ‘-’ when `silent` is `true`, `_`(refs) when `concept` is `false`, and otherwise `\__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.

```

5148 \cs_new_protected:Nn \__spintent_sprayo_mathml:
5149 {
5150   \seq_set_from_clist:NN \__spintent_sprayo_pt_seq
5151     \__spintent_geo_luaset_points_clist
5152   \tl_clear:N \__spintent_sprayo_refs_tl
5153   \int_zero:N \__spintent_sprayo_pt_idx_int
5154   \seq_map_inline:Nn \__spintent_sprayo_pt_seq
5155     {
5156       \int_incr:N \__spintent_sprayo_pt_idx_int
5157       \tl_if_empty:NF \__spintent_sprayo_refs_tl
5158         { \tl_put_right:Nn \__spintent_sprayo_refs_tl { , } }
5159       \tl_put_right:Ne \__spintent_sprayo_refs_tl
5160         { $ pts \int_use:N \__spintent_sprayo_pt_idx_int }
5161     }
5162   \bool_if:nTF
5163     {
5164       \bool_lazy_or_p:nn
5165         { \__spintent_sprayo_silent_bool }
5166         { ! \__spintent_sprayo_concept_bool }
5167     }
5168     {
5169       \tl_set:Ne \__spintent_sprayo_intent_arg_tl
5170         { _ ( \__spintent_sprayo_refs_tl ) }
5171       \bool_if:NT \__spintent_sprayo_silent_bool
5172         { \tl_set:Nn \__spintent_sprayo_intent_arg_tl { _- } }
5173     }
5174     {
5175       \tl_set:Ne \__spintent_sprayo_intent_arg_tl
5176         {
5177           -
5178           \tl_if_empty:NF \__spintent_sprayo_prefix_tl
5179             { \tl_use:N \__spintent_sprayo_prefix_tl - }
5180           \__spintent_sprayo_read_sty_str :prefix( \__spintent_sprayo_refs_tl )
5181         }
5182     }
```

```

5182     }
5183     \__spintent_invisible_sep:
5184     \MathMLintent { \__spintent_sprayo_intent_arg_tl }
5185     {
5186         \__spintent_inner_sep:
5187         \overrightarrow
5188         {
5189             \int_zero:N \__spintent_sprayo_pt_idx_int
5190             \seq_map_inline:Nn \__spintent_sprayo_pt_seq
5191             {
5192                 \int_incr:N \__spintent_sprayo_pt_idx_int
5193                 \__spintent_mathmlarg_point:nnn
5194                 { pts \int_use:N \__spintent_sprayo_pt_idx_int }
5195                 { sprayo }
5196                 { ##1 }
5197             }
5198         }
5199     }
5200 }

```

(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`.

```

5201 \NewDocumentCommand \sprayo { O{} m }
5202 {
5203     \mode_if_math:TF
5204     {
5205         \group_begin:
5206         \attribute \__spintent_scope_attr_int = 1
5207         \__spintent_sprayo_exec:nn {#1} {#2}
5208         \group_end:
5209     }
5210     { \msg_error:nnn { spintent } { need-math-mode } { \sprayo } }
5211 }

```

(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`

Each pair of variables is declared once for both `\spside` and `\spmside` via `\clist_map_inline:nn`. The variables `\__spintent_spside_mode_str` and `\__spintent_spmside_mode_str` store the values set by the key `read-cmd`; the variables `\__spintent_spside_prefix_tl` and `\__spintent_spmside_prefix_tl` store the values set by the key `prefix`.

The variables `\__spintent_spside_read_sty_str` and `\__spintent_spmside_read_sty_str` store the value set by the key `read-sty`.

The variables `\__spintent_spside_read_arg_tl` and `\__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`.

```

5212 \clist_map_inline:nn { side , mside }
5213 {
5214     \str_new:c { \__spintent_sp #1 _mode_str }
5215     \tl_new:c { \__spintent_sp #1 _prefix_tl }
5216     \str_new:c { \__spintent_sp #1 _read_sty_str }
5217     \tl_new:c { \__spintent_sp #1 _read_arg_tl }
5218 }

```

(End of definition for `\__spintent_spside_mode_str` and others.)

`\__spintent_spside_intent_tl` The variable `\__spintent_spside_intent_tl` stores the concept word, taken from `read-arg` or `read-sty`. It is shared by `\spside` and `\spmside`.

```

5219 \tl_new:N \__spintent_spside_intent_tl

```

(End of definition for `\l__spintent_spside_intent_tl`.)

`\l__spintent_spside_pt_seq` The variable `\l__spintent_spside_pt_seq` stores the points taken from `\l__spintent_geo_luaset_points_cl` and `\l__spintent_spside_pt_idx_int` counts the point being read. The variable `\l__spintent_spside_refs_tl` stores the reference list `$pts1, $pts2, ...` built from them. They are shared by `\spside` and `\spmside`.

```
5220 \seq_new:N \l__spintent_spside_pt_seq
5221 \int_new:N \l__spintent_spside_pt_idx_int
5222 \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`.

```
5223 \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` The “error”s `spside-invalid-points` and `spmside-invalid-points` are raised when the user includes an invalid “*point*” in the $\langle argument \rangle$ passed in ‘#1’ to `\spside` or `\spmside`.

```
5224 \clist_map_inline:nn { side , mside }
5225 {
5226   \msg_new:nnnn { spintent } { sp #1 -invalid-points }
5227   {
5228     Invalid~argument~'##2'~for~'\tl_trim_spaces:n {##1}'~\msg_line_context:.
5229   }
5230   {
5231     The~argument~must~be~two~points~separated~by~a~comma.
5232   }
5233 }
```

(End of definition for `spside-invalid-points` and `spmside-invalid-points`.)

15.48.3 Definition of keys for `\spside` and `\spmside`

`read-cmd` We define the shared $\langle keys \rangle$ `read-cmd`, `prefix`, `silent`, `concept`, `read-sty`, `read-arg` and `print-sym`.

```
5234 \clist_map_inline:nn { side , mside }
5235 {
5236   \keys_define:nn { spintent / sp #1 }
5237   {
5238     read-cmd .choices:nn = { school , mathcat }
5239     {
5240       \int_case:nn { \l_keys_choice_int }
5241       {
5242         {1} { \str_set:cn { l__spintent_sp #1 _mode_str } { school } }
5243         {2} { \str_set:cn { l__spintent_sp #1 _mode_str } { mathcat } }
5244       }
5245     },
5246     read-cmd .initial:n = school,
5247     read-cmd .value_required:n = true,
5248     read-cmd / unknown .code:n =
5249       \msg_error:nneee { spintent } { unknown-choice }
5250       { read-cmd } { school,~mathcat } { \exp_not:n {##1} },
5251     prefix .code:n =
5252       { \l__spintent_sanitize_affix:cn { l__spintent_sp #1 _prefix_tl } {##1} },
5253     prefix .initial:n = ,
5254     prefix .value_required:n = true,
5255     silent .bool_set:c = { l__spintent_sp #1 _silent_bool },
5256     silent .initial:n = false,
5257     silent .value_required:n = true,
5258     concept .bool_set:c = { l__spintent_sp #1 _concept_bool },
5259     concept .initial:n = true,
5260     concept .value_required:n = true,
5261     read-sty .choices:nn = { lado , segmento , trazo }
5262     {
5263       \int_case:nn { \l_keys_choice_int }
5264       {
5265         {1} { \str_set:cn { l__spintent_sp #1 _read_sty_str } { lado } }
5266       }
5267     }
5268   }
5269 }
```

```

5266         {2} { \str_set:cn { \__spintent_sp #1 _read_sty_str } { segmento } }
5267         {3} { \str_set:cn { \__spintent_sp #1 _read_sty_str } { trazo } }
5268     }
5269 },
5270 read-sty .initial:n = lado,
5271 read-sty .value_required:n = true,
5272 read-sty / unknown .code:n =
5273     \msg_error:nneee { spintent } { unknown-choice }
5274     { read-sty } { lado,~segmento,~trazo } { \exp_not:n {##1} },
5275 read-arg .code:n =
5276     {
5277         \__spintent_sanitizate_affix:cn { \__spintent_sp #1 _read_arg_tl } {##1}
5278     },
5279 read-arg .initial:n = ,
5280 read-arg .value_required:n = true,
5281 print-sym .bool_set:c = { \__spintent_sp #1 _print_sym_bool },
5282 print-sym .initial:n = false,
5283 print-sym .value_required:n = true,
5284 unknown .code:n =
5285     {
5286         \str_set:Nn \__spintent_command_name_str { sp #1 }
5287         \__spintent_unknown_keys_cmd:n {##1}
5288     },
5289 }
5290 }

```

🌱 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 $\langle argument \rangle$ 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 $\langle argument \rangle$, it sets the optional $\langle keys \rangle$ passed in ‘#2’.

```

5291 \cs_new_protected:Nn \__spintent_spside_exec:nnn
5292 {
5293     \keys_set:nn { spintent / sp #1 } {#2}
5294     \__spintent_luafun_geo_parse_and_set:n { \exp_not:n {#3} }
5295     \bool_if:nT
5296     {
5297         \str_if_eq_p:Vn \__spintent_geo_luasets_error_str { true } ||
5298         \str_if_eq_p:Vn \__spintent_geo_luasets_is_name_str { true }
5299     }
5300     {
5301         \msg_error:nnee { spintent } { sp #1 -invalid-points }
5302         { \c_backslash_str sp #1 } { \exp_not:n {#3} }
5303     }
5304     \tl_if_empty:cTF { \__spintent_sp #1 _read_arg_tl }
5305     {
5306         \tl_set:Ne \l__spintent_spside_intent_tl
5307         { \str_use:c { \__spintent_sp #1 _read_sty_str } }
5308         \str_if_eq:eeT {#1} { mside }
5309         {
5310             \tl_put_left:Nn \l__spintent_spside_intent_tl { medida-del- }
5311         }
5312     }
5313     {
5314         \tl_set:Ne \l__spintent_spside_intent_tl
5315         { \tl_use:c { \__spintent_sp #1 _read_arg_tl } }
5316         \bool_if:cT { \__spintent_sp #1 _concept_bool }
5317         {
5318             \str_if_eq:eeT {#1} { mside }
5319             {
5320                 \tl_put_left:Nn \l__spintent_spside_intent_tl { medida-del- }

```

```

5321         }
5322     }
5323 }
5324 \__spintent_spside_mathml:n {#1}
5325 }

```

(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.

```

5326 \cs_new_protected:Nn \__spintent_spside_mathml:n
5327 {
5328     \seq_set_from_clist:NN \l__spintent_spside_pt_seq
5329     \l__spintent_geo_luaset_points_clist
5330     \tl_clear:N \l__spintent_spside_refs_tl
5331     \int_zero:N \l__spintent_spside_pt_idx_int
5332     \seq_map_inline:Nn \l__spintent_spside_pt_seq
5333     {
5334         \int_incr:N \l__spintent_spside_pt_idx_int
5335         \tl_if_empty:NF \l__spintent_spside_refs_tl
5336         { \tl_put_right:Nn \l__spintent_spside_refs_tl { , } }
5337         \tl_put_right:Ne \l__spintent_spside_refs_tl
5338         { $ pts \int_use:N \l__spintent_spside_pt_idx_int }
5339     }

```

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.

```

5340     \bool_if:cTF { \l__spintent_sp #1 _silent_bool }
5341     {
5342         \tl_set:Nn \l__spintent_spside_intent_arg_tl { _- }
5343     }
5344     {
5345         \tl_if_empty:cTF { \l__spintent_sp #1 _read_arg_tl }
5346         {
5347             \bool_if:cTF { \l__spintent_sp #1 _concept_bool }
5348             {
5349                 \tl_set:Ne \l__spintent_spside_intent_arg_tl
5350                 {
5351                     -
5352                     \tl_if_empty:cF { \l__spintent_sp #1 _prefix_tl }
5353                     { \tl_use:c { \l__spintent_sp #1 _prefix_tl } - }
5354                     \l__spintent_spside_intent_tl :prefix( \l__spintent_spside_refs_tl )
5355                 }
5356             }
5357             {
5358                 \tl_set:Ne \l__spintent_spside_intent_arg_tl
5359                 { _ ( \l__spintent_spside_refs_tl ) }
5360             }
5361         }
5362     }
5363     \tl_set:Ne \l__spintent_spside_intent_arg_tl
5364     {
5365         -
5366         \tl_if_empty:cF { \l__spintent_sp #1 _prefix_tl }
5367         { \tl_use:c { \l__spintent_sp #1 _prefix_tl } - }
5368         \l__spintent_spside_intent_tl :prefix( \l__spintent_spside_refs_tl )
5369     }
5370 }
5371 }

```

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 `ptsn` 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.

```

5372     \__spintent_invisible_sep: % (luamml issues #26)

```

```

5373 \MathMLintent { \l__spintent_spside_intent_arg_tl }
5374 {
5375   \__spintent_inner_sep:
5376   \str_if_eq:eeTF {#1} { side }
5377   {
5378     \overline { \__spintent_spside_points_loop:n {#1} }
5379   }
5380   {
5381     \bool_if:cTF { l__spintent_sp #1 _print_sym_bool }
5382     {
5383       \__spintent_luafun_mrow_intent_arg:nn
5384       { } { intent = ":silent" }
5385       { \operatorname { m } }
5386       \overline { \__spintent_spside_points_loop:n {#1} }
5387     }
5388     {
5389       \__spintent_spside_points_loop:n {#1}
5390     }
5391   }
5392 }
5393 }

```

(End of definition for `\__spintent_spside_mathml:n`.)

`\__spintent_spside_points_loop:n`

Walks `\__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.

```

5394 \cs_new_protected:Nn \__spintent_spside_points_loop:n
5395 {
5396   \int_zero:N \__spintent_spside_pt_idx_int
5397   \seq_map_inline:Nn \__spintent_spside_pt_seq
5398   {
5399     \int_incr:N \__spintent_spside_pt_idx_int
5400     \__spintent_mathmlarg_point:nnn
5401     { pts \int_use:N \__spintent_spside_pt_idx_int }
5402     { sp #1 }
5403     { ##1 }
5404   }
5405 }

```

(End of definition for `\__spintent_spside_points_loop:n`.)

`\spside` Both public commands `\spside` and `\spmside` enforces *math mode*, opens a local group, and delegates execution to `\__spintent_spside_exec:nnn`.

```

5406 \NewDocumentCommand \spside { 0{} m }
5407 {
5408   \mode_if_math:TF
5409   {
5410     \group_begin:
5411     \attribute \__spintent_scope_attr_int = 1
5412     \__spintent_spside_exec:nnn { side } {#1} {#2}
5413     \group_end:
5414   }
5415   { \msg_error:nnn { spintent } { need-math-mode } { \spside } }
5416 }
5417 \NewDocumentCommand \spmside { 0{} m }
5418 {
5419   \mode_if_math:TF
5420   {
5421     \group_begin:
5422     \attribute \__spintent_scope_attr_int = 1
5423     \__spintent_spside_exec:nnn { mside } {#1} {#2}
5424     \group_end:
5425   }
5426   { \msg_error:nnn { spintent } { need-math-mode } { \spmside } }
5427 }

```

(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`.

```
5428 \clist_map_inline:nn { angle , mangle }
5429 {
5430   \str_new:c { l__spintent_sp #1 _mode_str }
5431   \tl_new:c { l__spintent_sp #1 _prefix_tl }
5432 }
```

(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`.

```
5433 \tl_new:N \l__spintent_spangle_intent_tl
5434 \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_tl` stores the reference list built from them, in the tres-puntos and un-punto cases of `\__spintent_spangle_mathml:`.

```
5435 \seq_new:N \l__spintent_spangle_pt_seq
5436 \int_new:N \l__spintent_spangle_pt_idx_int
5437 \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`.

```
5438 \tl_new:N \l__spintent_spangle_intent_arg_tl
5439 \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`.

```
5440 \clist_map_inline:nn { angle , mangle }
5441 {
5442   \msg_new:nnnn { spintent } { sp #1 -invalid-arg }
5443   {
5444     Invalid~argument~'##2'~for~'\tl_trim_spaces:n {##1}'~\msg_line_context:.
5445   }
5446   {
5447     Use~a~point,~a~Greek~letter,~a~number~or~three~points~separated~by~
5448     commas.
5449   }
5450 }
```

(End of definition for `spangle-invalid-arg` and `spmangle-invalid-arg`.)

15.49.3 Definition of keys for `\spangle` and `\spmangle`

We define the shared *(keys)* `read-cmd`, `prefix`, `recto`, `print-sym`, `silent` and `concept`,
`unknown`

```

5451 \clist_map_inline:nn { angle , mangle }
5452 {
5453   \keys_define:nn { spintent / sp #1 }
5454   {
5455     read-cmd .choices:nn = { school , mathcat }
5456     {
5457       \int_case:nn { \l_keys_choice_int }
5458       {
5459         {1} { \str_set:cn { l__spintent_sp #1 _mode_str } { school } }
5460         {2} { \str_set:cn { l__spintent_sp #1 _mode_str } { mathcat } }
5461       }
5462     },
5463     read-cmd .initial:n = school,
5464     read-cmd .value_required:n = true,
5465     read-cmd / unknown .code:n =
5466       \msg_error:nneee { spintent } { unknown-choice }
5467       { read-cmd } { school,~mathcat } { \exp_not:n {##1} },
5468     prefix .code:n =
5469       { \__spintent_sanitize_affix:cn { l__spintent_sp #1 _prefix_tl } {##1} },
5470     prefix .initial:n = ,
5471     prefix .value_required:n = true,
5472     recto .bool_set:c = { l__spintent_sp #1 _recto_bool },
5473     recto .initial:n = false,
5474     recto .value_required:n = true,
5475     print-sym .bool_set:c = { l__spintent_sp #1 _print_sym_bool },
5476     print-sym .initial:n = false,
5477     print-sym .value_required:n = true,
5478     silent .bool_set:c = { l__spintent_sp #1 _silent_bool },
5479     silent .initial:n = false,
5480     silent .value_required:n = true,
5481     concept .bool_set:c = { l__spintent_sp #1 _concept_bool },
5482     concept .initial:n = true,
5483     concept .value_required:n = true,
5484     unknown .code:n =
5485       {
5486         \str_set:Nn \l__spintent_command_name_str { sp #1 }
5487         \__spintent_unknown_keys_cmd:n {##1}
5488       },
5489   }
5490 }
```

• 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`.

```

5491 \cs_new_protected:Nn \__spintent_spangle_sym:n
5492 {
5493   \str_if_eq:eeTF {#1} { mangle }
5494   {
5495     \bool_if:cTF { l__spintent_sp #1 _print_sym_bool }
5496     {
5497       \bool_if:cTF { l__spintent_sp #1 _recto_bool }
5498       { \__spintent_right_angle_sqr: } { \angle }
5499     }
5500     {
5501       \bool_if:cTF { l__spintent_sp #1 _recto_bool }
5502       { \measuredrightangle } { \measuredangle }
5503     }
5504   }
5505   {
5506     \bool_if:cTF { l__spintent_sp #1 _recto_bool }
5507     { \__spintent_right_angle_sqr: } { \angle }
5508   }
5509 }
```

(End of definition for `\__spinttent_spangle_sym:n`.)

`\__spinttent_spangle_exec:nnn` The internal function `\__spinttent_spangle_exec:nnn` receives the short name of the command, `angle` or `mangle`, in ‘#1’ and the $\langle argument \rangle$ in ‘#3’. It calls `\__spinttent_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__spinttent_spangle_intent_tl` to `ángulo`, or `medida-del-ángulo` for `\spmangle`, suffixed with `-recto` when `recto` is `true`, and calls `\__spinttent_spangle_mathml:n`.

Before processing the $\langle argument \rangle$, it sets the optional $\langle keys \rangle$ passed in ‘#2’.

```

5510 \cs_new_protected:Nn \__spinttent_spangle_exec:nnn
5511 {
5512   \keys_set:nn { spinttent / sp #1 } {#2}
5513   \__spinttent_luafun_geo_angulo_parse_and_set:n { \exp_not:n {#3} }
5514   \str_if_eq:VnT \l__spinttent_geo_luaset_error_str { true }
5515   {
5516     \msg_error:nnee { spinttent } { sp #1 -invalid-arg }
5517     { \c_backslash_str sp #1 } { \exp_not:n {#3} }
5518   }
5519   \str_if_eq:VnT \l__spinttent_geo_luaset_angulo_case_str { nombre }
5520   {
5521     \str_if_eq:VnF \l__spinttent_geo_luaset_nombre_es_num_str { true }
5522     {
5523       \int_compare:nNnT { \str_count:N \l__spinttent_geo_luaset_letra_str } = { 1 }
5524       {
5525         \str_if_eq:eeF { \l__spinttent_geo_luaset_letra_str }
5526         { \str_uppercase:n { \l__spinttent_geo_luaset_letra_str } }
5527         {
5528           \msg_error:nnee { spinttent } { sp #1 -invalid-arg }
5529           { \c_backslash_str sp #1 } { \exp_not:n {#3} }
5530         }
5531       }
5532     }
5533   }
5534   \str_if_eq:eeTF {#1} { mangle }
5535   {
5536     \tl_set:Nn \l__spinttent_spangle_intent_tl { medida-del-ángulo }
5537   }
5538   {
5539     \tl_set:Nn \l__spinttent_spangle_intent_tl { ángulo }
5540   }
5541   \bool_if:cT { l__spinttent_sp #1 _recto_bool }
5542   {
5543     \tl_put_right:Nn \l__spinttent_spangle_intent_tl { -recto }
5544   }
5545   \__spinttent_spangle_mathml:n {#1}
5546 }

```

(End of definition for `\__spinttent_spangle_exec:nnn`.)

`\__spinttent_spangle_mathml:n` The function `\__spinttent_spangle_mathml:n` branches on `\l__spinttent_geo_luaset_angulo_case_str`. For `nombre`, the symbol `\__spinttent_spangle_sym:n` is tagged with `\__spinttent_luafun_mrow_intent_arg:n` on its own, with `\l__spinttent_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.

```

5547 \cs_new_protected:Nn \__spinttent_spangle_mathml:n
5548 {
5549   \__spinttent_invisible_sep:
5550   \str_if_eq:VnTF \l__spinttent_geo_luaset_angulo_case_str { nombre }
5551   {
5552     \bool_if:cTF { l__spinttent_sp #1 _silent_bool }
5553     {
5554       \tl_set:Nn \l__spinttent_spangle_intent_arg_tl { _- }
5555     }
5556     {
5557       \bool_if:cTF { l__spinttent_sp #1 _concept_bool }
5558       {
5559         \tl_set:Ne \l__spinttent_spangle_intent_arg_tl
5560         {
5561           -

```

```

5562         \tl_if_empty:cF { l__spintent_sp #1 _prefix_tl }
5563         { \tl_use:c { l__spintent_sp #1 _prefix_tl } - }
5564         \l__spintent_spangle_intent_tl
5565     }
5566 }
5567 {
5568     \tl_set:Nn \l__spintent_spangle_intent_arg_tl { _- }
5569 }
5570 }
5571 \tl_set:Ne \l__spintent_spangle_kv_tl
5572 { intent = "\l__spintent_spangle_intent_arg_tl" }
5573 \__spintent_luafun_mrow_intent_arg:nn
5574 { } { \l__spintent_spangle_kv_tl }
5575 { \__spintent_spangle_sym:n {#1} }
5576 \bool_if:cTF { l__spintent_sp #1 _silent_bool }
5577 {
5578     \__spintent_luafun_mrow_intent_arg:nn { } { intent = "_-" }
5579     { \l__spintent_geo_luaset_print_tl }
5580 }

```

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`.

```

5581 {
5582     \str_if_eq:VnTF \l__spintent_geo_luaset_nombre_es_num_str { true }
5583     {
5584         \tl_set:Ne \l__spintent_spangle_kv_tl
5585         { intent = "_ \l__spintent_geo_luaset_intent_str -" }
5586         \__spintent_luafun_mrow_intent_arg:nn
5587         { } { \l__spintent_spangle_kv_tl }
5588         { \l__spintent_geo_luaset_print_tl }
5589     }

```

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.

```

5590 {
5591     \str_case:VnF \l__spintent_geo_luaset_tipo_str
5592     {
5593         { sub }
5594         {
5595             \str_if_eq:eeTF { \use:c { l__spintent_sp #1 _mode_str } }
5596             { school }
5597             {
5598                 \tl_set:Ne \l__spintent_spangle_kv_tl
5599                 { intent = "_ \l__spintent_geo_luaset_intent_str -" }
5600                 \__spintent_luafun_mrow_intent_arg:nn
5601                 { } { \l__spintent_spangle_kv_tl }
5602                 {
5603                     \l__spintent_geo_luaset_letra_str
5604                     \c_math_subscript_token
5605                     {
5606                         \MathMLintent { :silent }
5607                         { \l__spintent_geo_luaset_sub_str }
5608                     }
5609                 }
5610             }
5611             {
5612                 \l__spintent_geo_luaset_letra_str
5613                 \c_math_subscript_token { \l__spintent_geo_luaset_sub_str }
5614             }
5615         }

```

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`.

```

5616 { prima }
5617 {
5618     \str_if_eq:eeTF { \use:c { l__spintent_sp #1 _mode_str } }
5619     { school }
5620     {
5621         \tl_set:Ne \l__spintent_spangle_kv_tl

```

```

5622         { intent = "_" \l__spintent_geo_luaset_intent_str -" }
5623         \l__spintent_luafun_mrow_intent_arg:nn
5624         { } { \l__spintent_spangle_kv_tl }
5625         {
5626             \l__spintent_geo_luaset_letra_str
5627             \l__spintent_geo_luaset_sub_str
5628         }
5629     }
5630     {
5631         \l__spintent_geo_luaset_letra_str
5632         \l__spintent_geo_luaset_sub_str
5633     }
5634 }
5635 }
5636 { \l__spintent_geo_luaset_print_tl }
5637 }
5638 }
5639 }

```

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 `\l__spintent_spangle_sym:n` inside the same `\MathMLintent`. The sequence is walked again, calling `\l__spintent_mathmlarg_point:nnn` once per point.

```

5640 {
5641     \str_if_eq:VnT \l__spintent_geo_luaset_angulo_case_str { un-punto }
5642     { \tl_put_right:Nn \l__spintent_spangle_intent_tl { -en } }
5643     \seq_set_from_clist:NN \l__spintent_spangle_pt_seq
5644         \l__spintent_geo_luaset_points_clist
5645     \tl_clear:N \l__spintent_spangle_refs_tl
5646     \int_zero:N \l__spintent_spangle_pt_idx_int
5647     \seq_map_inline:Nn \l__spintent_spangle_pt_seq
5648     {
5649         \int_incr:N \l__spintent_spangle_pt_idx_int
5650         \tl_if_empty:NF \l__spintent_spangle_refs_tl
5651         { \tl_put_right:Nn \l__spintent_spangle_refs_tl { , } }
5652         \tl_put_right:Ne \l__spintent_spangle_refs_tl
5653         { $ pt \int_use:N \l__spintent_spangle_pt_idx_int }
5654     }
5655     \bool_if:cTF { l__spintent_sp #1 _silent_bool }
5656     {
5657         \tl_set:Nn \l__spintent_spangle_intent_arg_tl { _- }
5658     }
5659     {
5660         \bool_if:cTF { l__spintent_sp #1 _concept_bool }
5661         {
5662             \tl_set:Ne \l__spintent_spangle_intent_arg_tl
5663             {
5664                 -
5665                 \tl_if_empty:cF { l__spintent_sp #1 _prefix_tl }
5666                 { \tl_use:c { l__spintent_sp #1 _prefix_tl } - }
5667                 \l__spintent_spangle_intent_tl :prefix( \l__spintent_spangle_refs_tl
5668                 )
5669             }
5670         }
5671         {
5672             \tl_set:Ne \l__spintent_spangle_intent_arg_tl
5673             { _ ( \l__spintent_spangle_refs_tl ) }
5674         }
5675     }
5676     \bool_set_false:N \l__spintent_spangle_print_sym_now_bool
5677     \str_if_eq:eeT { #1 } { mangle }
5678     {
5679         \bool_if:cT { l__spintent_sp #1 _print_sym_bool }
5680         { \bool_set_true:N \l__spintent_spangle_print_sym_now_bool }
5681     }
5682     \MathMLintent { \l__spintent_spangle_intent_arg_tl }
5683     {

```

```

5684     \__spintent_inner_sep:
5685     \bool_if:NT \l__spintent_spangle_print_sym_now_bool
5686     {
5687         \__spintent_luafun_mrow_intent_arg:nn
5688         { } { intent = ":silent" }
5689         { \operatorname { m } }
5690     }
5691     \__spintent_luafun_mrow_intent_arg:nn
5692     { } { intent = ":silent" }
5693     { \__spintent_spangle_sym:n {#1} }
5694     \int_zero:N \l__spintent_spangle_pt_idx_int
5695     \seq_map_inline:Nn \l__spintent_spangle_pt_seq
5696     {
5697         \int_incr:N \l__spintent_spangle_pt_idx_int
5698         \__spintent_mathmlarg_point:nnn
5699         { pt \int_use:N \l__spintent_spangle_pt_idx_int
5700         { sp#1 }
5701         { ##1 }
5702     }
5703 }
5704 }
5705 }

```

(End of definition for `\__spintent_spangle_mathml:n`.)

`\spangle` Both public commands `\spangle` and `\spmangle` enforces *math mode*, opens a local group, and delegates execution to `\__spintent_spangle_exec:nnn`.

`\spmangle`

```

5706 \NewDocumentCommand \spangle { 0{} m }
5707 {
5708     \mode_if_math:TF
5709     {
5710         \group_begin:
5711         \attribute \l__spintent_scope_attr_int = 1
5712         \__spintent_spangle_exec:nnn { angle } {#1} {#2}
5713         \group_end:
5714     }
5715     { \msg_error:nnn { spintent } { need-math-mode } { \spangle } }
5716 }
5717 \NewDocumentCommand \spmangle { 0{} m }
5718 {
5719     \mode_if_math:TF
5720     {
5721         \group_begin:
5722         \attribute \l__spintent_scope_attr_int = 1
5723         \__spintent_spangle_exec:nnn { mangle } {#1} {#2}
5724         \group_end:
5725     }
5726     { \msg_error:nnn { spintent } { need-math-mode } { \spmangle } }
5727 }

```

(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 $m\widehat{AB}$ is the standard notation for the measure of an arc.

15.50.1 Definition of variables for `\sparc` and `\spmarc`

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`.

```

5728 \clist_map_inline:nn { arc , marc }
5729 {
5730     \str_new:c { l__spintent_sp #1 _mode_str }
5731     \tl_new:c { l__spintent_sp #1 _prefix_tl }
5732 }

```

(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`.

```
5733 \tl_new:N \l__spintent_sparc_intent_tl
```

(End of definition for `\l__spintent_sparc_intent_tl`.)

`\l__spintent_sparc_pt_seq` 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.

```
5734 \seq_new:N \l__spintent_sparc_pt_seq
5735 \int_new:N \l__spintent_sparc_pt_idx_int
5736 \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.

```
5737 \tl_new:N \l__spintent_sparc_intent_arg_tl
```

(End of definition for `\l__spintent_sparc_intent_arg_tl`.)

15.50.2 Definition of error messages for `\sparc` and `\spmarc`

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`.

```
5738 \clist_map_inline:nn { arc , marc }
5739 {
5740   \msg_new:nnnn { spintent } { sp #1 -invalid-points }
5741   {
5742     Invalid~argument~'##2'~for~'\tl_trim_spaces:n {##1}'~\msg_line_context:.
5743   }
5744   {
5745     The~argument~must~be~two~or~three~points~separated~by~commas.
5746   }
5747 }
```

(End of definition for `sparc-invalid-points` and `spmarc-invalid-points`.)

15.50.3 Definition of keys for `\sparc` and `\spmarc`

`read-cmd` We define the shared `⟨keys⟩` `read-cmd`, `prefix`, `silent` and `concept`.

```
prefix
silent
concept
unknown
5748 \clist_map_inline:nn { arc , marc }
5749 {
5750   \keys_define:nn { spintent / sp #1 }
5751   {
5752     read-cmd .choices:nn = { school , mathcat }
5753     {
5754       \int_case:nn { \l_keys_choice_int }
5755       {
5756         {1} { \str_set:cn { \l__spintent_sp #1 _mode_str } { school } }
5757         {2} { \str_set:cn { \l__spintent_sp #1 _mode_str } { mathcat } }
5758       }
5759     },
5760     read-cmd .initial:n = school,
5761     read-cmd .value_required:n = true,
5762     read-cmd / unknown .code:n =
5763     \msg_error:nnee { spintent } { unknown-choice }
5764     { read-cmd } { school,~mathcat } { \exp_not:n {##1} },
5765     prefix .code:n =
5766     { \l__spintent_sanitise_affix:cn { \l__spintent_sp #1 _prefix_tl } {##1} },
5767     prefix .initial:n = ,
5768     prefix .value_required:n = true,
5769     silent .bool_set:c = { \l__spintent_sp #1 _silent_bool },
5770     silent .initial:n = false,
5771     silent .value_required:n = true,
5772     concept .bool_set:c = { \l__spintent_sp #1 _concept_bool },
5773     concept .initial:n = true,
5774     concept .value_required:n = true,
5775     unknown .code:n =
5776     {
```

```

5777         \str_set:Nn \l__spintenc_command_name_str { sp #1 }
5778         \__spintenc_unknown_keys_cmd:n {##1}
5779     },
5780 }
5781 }

```

(End of definition for `read-cmd` and others.)

15.50.4 Internal functions for `\sparc` and `\spmarc`

`\__spintenc_sparc_wide_arc:n`

The function `\__spintenc_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](#).

```

5782 \cs_new_protected:Nn \__spintenc_sparc_wide_arc:n
5783 {
5784     \Umathaccent 7 \symoperators "023DC \scan_stop: {#1}
5785 }

```

(End of definition for `\__spintenc_sparc_wide_arc:n`.)

`\__spintenc_sparc_print_arc:n`

The function `\__spintenc_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 `\__spintenc_sparc_wide_arc:n`.

```

5786 \cs_new_protected:Nn \__spintenc_sparc_print_arc:n
5787 {
5788     \cs_if_exist:NTF \widearc
5789     { \widearc {#1} }
5790     { \__spintenc_sparc_wide_arc:n {#1} }
5791 }

```

(End of definition for `\__spintenc_sparc_print_arc:n`.)

`\__spintenc_sparc_exec:nnn`

The internal function `\__spintenc_sparc_exec:nnn` receives the short name of the command, `arc` or `marc`, in ‘#1’ and the $\langle argument \rangle$ in ‘#3’. It calls `\__spintenc_luafun_geo_parse_and_set:n` to process ‘#3’, returning an “error” if invalid, then sets `\l__spintenc_sparc_intent_tl` to `arco`, or `medida-del-arco` for `\spmarc`. No `intent` is built here: it needs the reference list, which `\__spintenc_sparc_mathml:n` builds after splitting the points.

Before processing the $\langle argument \rangle$, it sets the optional $\langle keys \rangle$ passed in ‘#2’.

```

5792 \cs_new_protected:Nn \__spintenc_sparc_exec:nnn
5793 {
5794     \keys_set:nn { spintenc / sp #1 } {#2}
5795     \__spintenc_luafun_geo_parse_and_set:n { \exp_not:n {#3} }
5796     \bool_if:nT
5797     {
5798         \str_if_eq_p:Vn \l__spintenc_geo_luaset_error_str { true } ||
5799         \str_if_eq_p:Vn \l__spintenc_geo_luaset_is_name_str { true }
5800     }
5801     {
5802         \msg_error:nnee { spintenc } { sp #1 -invalid-points }
5803         { \c_backslash_str sp #1 } { \exp_not:n {#3} }
5804     }
5805     \str_if_eq:eeTF {#1} { marc }
5806     { \tl_set:Nn \l__spintenc_sparc_intent_tl { medida-del-arco } }
5807     { \tl_set:Nn \l__spintenc_sparc_intent_tl { arco } }
5808     \__spintenc_sparc_mathml:n {#1}
5809 }

```

(End of definition for `\__spintenc_sparc_exec:nnn`.)

`\__spintenc_sparc_mathml:n`

The function `\__spintenc_sparc_mathml:n` receives the short command name in ‘#1’. It splits `\l__spintenc_geo_luaset_points_clist` into `\l__spintenc_sparc_pt_seq` and walks it once to build `\l__spintenc_sparc_refs_tl`, the comma-separated $\$pt1, \$pt2, \dots$ reference list, naming each point by position.

```

5810 \cs_new_protected:Nn \__spintenc_sparc_mathml:n
5811 {
5812     \seq_set_from_clist:NN \l__spintenc_sparc_pt_seq
5813     \l__spintenc_geo_luaset_points_clist
5814 }

```

```

5815 \tl_clear:N \l__spinttent_sparc_refs_tl
5816 \int_zero:N \l__spinttent_sparc_pt_idx_int
5817 \seq_map_inline:Nn \l__spinttent_sparc_pt_seq
5818 {
5819   \int_incr:N \l__spinttent_sparc_pt_idx_int
5820   \tl_if_empty:NF \l__spinttent_sparc_refs_tl
5821   { \tl_put_right:Nn \l__spinttent_sparc_refs_tl { , } }
5822   \tl_put_right:Ne \l__spinttent_sparc_refs_tl
5823   { $ pt \int_use:N \l__spinttent_sparc_pt_idx_int }
5824 }

```

The concept intent is ‘`-`’ when `silent` is `true`, ‘`_`’ (refs) when `concept` is `false`; otherwise ‘`_`’`\l__spinttent_sparc_intent_tl:prefix(refs)`, with `prefix` placed after the leading ‘`_`’ when given.

```

5825 \bool_if:cTF { l__spinttent_sp #1 _silent_bool }
5826 {
5827   \tl_set:Nn \l__spinttent_sparc_intent_arg_tl { _- }
5828 }
5829 {
5830   \bool_if:cTF { l__spinttent_sp #1 _concept_bool }
5831   {
5832     \tl_set:Ne \l__spinttent_sparc_intent_arg_tl
5833     {
5834       -
5835       \tl_if_empty:cF { l__spinttent_sp #1 _prefix_tl }
5836       { \tl_use:c { l__spinttent_sp #1 _prefix_tl } - }
5837       \l__spinttent_sparc_intent_tl :prefix( \l__spinttent_sparc_refs_tl )
5838     }
5839   }
5840   {
5841     \tl_set:Ne \l__spinttent_sparc_intent_arg_tl
5842     { _ ( \l__spinttent_sparc_refs_tl ) }
5843   }
5844 }

```

It is applied with a single native `\MathMLintent` wrapping `\__spinttent_sparc_print_arc:n` directly, with the same shape as `\__spinttent_spside_mathml:n`, preceded by `\__spinttent_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 `\__spinttent_mathmlarg_point:nnn` once per point with its `ptn` reference name, looked up under `sparc` or `spmarc`.

```

5845 \__spinttent_invisible_sep:
5846 \MathMLintent { \l__spinttent_sparc_intent_arg_tl }
5847 {
5848   \__spinttent_inner_sep:
5849   \str_if_eq:eeT {#1} { marc }
5850   {
5851     \__spinttent_luafun_mrow_intent_arg:nn
5852     { } { intent = ":silent" }
5853     { \operatorname { m } } }
5854   }
5855   \__spinttent_sparc_print_arc:n
5856   {
5857     \int_zero:N \l__spinttent_sparc_pt_idx_int
5858     \seq_map_inline:Nn \l__spinttent_sparc_pt_seq
5859     {
5860       \int_incr:N \l__spinttent_sparc_pt_idx_int
5861       \__spinttent_mathmlarg_point:nnn
5862       { pt \int_use:N \l__spinttent_sparc_pt_idx_int }
5863       { sp #1 }
5864       { ##1 }
5865     }
5866   }
5867 }
5868 }

```

(End of definition for `\__spinttent_sparc_mathml:n`.)

`\sparc` Both public commands `\sparc` and `\spmarc` enforces *math mode*, opens a local group, and delegates execution to `\__spinttent_sparc_exec:nnn`.

```

5869 \NewDocumentCommand \sparc { 0{} m }

```

```

5870 {
5871   \mode_if_math:TF
5872   {
5873     \group_begin:
5874       \attribute \l__spintent_scope_attr_int = 1
5875       \__spintent_sparc_exec:nnn { arc } {#1} {#2}
5876     \group_end:
5877   }
5878   { \msg_error:nnn { spintent } { need-math-mode } { \sparc } }
5879 }
5880 \NewDocumentCommand \spmarc { O{} m }
5881 {
5882   \mode_if_math:TF
5883   {
5884     \group_begin:
5885       \attribute \l__spintent_scope_attr_int = 1
5886       \__spintent_sparc_exec:nnn { marc } {#1} {#2}
5887     \group_end:
5888   }
5889   { \msg_error:nnn { spintent } { need-math-mode } { \spmarc } }
5890 }

```

(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` 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`.

```

5891 \str_new:N \l__spintent_spPoly_mode_str
5892 \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` 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.

```

5893 \seq_new:N \l__spintent_spPoly_pt_seq
5894 \tl_new:N \l__spintent_spPoly_refs_tl
5895 \tl_new:N \l__spintent_spPoly_intent_arg_tl
5896 \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.

```

5897 \msg_new:nnnn { spintent } { spPoly-invalid-points }
5898 {
5899   Invalid~argument~'#1'~for~'\token_to_str:N \spPoly'~\msg_line_context:.
5900 }
5901 {
5902   The~argument~must~be~a~list~of~at~least~three~points~separated~by~commas.
5903 }

```

(End of definition for `spPoly-invalid-points`.)

15.51.3 Definition of keys for `\spPoly`

`read-cmd` We define the `(keys)` `read-cmd`, `prefix`, `print-sym`, `silent` and `concept`.

```

prefix
print-sym
silent
concept
unknown
5904 \keys_define:nn { spintent / spPoly }
5905 {
5906   read-cmd .choices:nn = { school , mathcat }
5907   {
5908     \int_case:nn { \l_keys_choice_int }

```

```

5909         {
5910             {1} { \str_set:Nn \l__spinttent_spPoly_mode_str { school } }
5911             {2} { \str_set:Nn \l__spinttent_spPoly_mode_str { mathcat } }
5912         }
5913     },
5914     read-cmd .initial:n = school,
5915     read-cmd .value_required:n = true,
5916     read-cmd / unknown .code:n =
5917         \msg_error:nneee { spinttent } { unknown-choice }
5918         { read-cmd } { school,~mathcat } { \exp_not:n {#1} },
5919     prefix .code:n =
5920         { \__spinttent_sanitizize_affix:Nn \l__spinttent_spPoly_prefix_tl {#1} },
5921     prefix .initial:n = ,
5922     prefix .value_required:n = true,
5923     print-sym .bool_set:N = \l__spinttent_spPoly_print_sym_bool,
5924     print-sym .initial:n = true,
5925     print-sym .value_required:n = true,
5926     silent .bool_set:N = \l__spinttent_spPoly_silent_bool,
5927     silent .initial:n = false,
5928     silent .value_required:n = true,
5929     concept .bool_set:N = \l__spinttent_spPoly_concept_bool,
5930     concept .initial:n = true,
5931     concept .value_required:n = true,
5932     unknown .code:n =
5933         {
5934             \str_set:Nn \l__spinttent_command_name_str { spPoly }
5935             \__spinttent_unknown_keys_cmd:n {#1}
5936         },
5937 }

```

(End of definition for read-cmd and others.)

15.51.4 Internal functions for \spPoly

`\__spinttent_spPoly_sym:` Unchanged: reads `\l__spinttent_geo_luaset_poly_sym_str` and typesets the corresponding symbol.

```

5938 \cs_new_protected:Nn \__spinttent_spPoly_sym:
5939 {
5940     \str_case:Vn \l__spinttent_geo_luaset_poly_sym_str
5941     {
5942         { triangle } { \triangle }
5943         { square } { \mdlgwhtsquare }
5944     }
5945 }

```

(End of definition for `\__spinttent_spPoly_sym:`.)

`\__spinttent_spPoly_build_refs:` Splits `\l__spinttent_geo_luaset_points_clist` into `\l__spinttent_spPoly_pt_seq` and walks it once to build `\l__spinttent_spPoly_refs_tl`, the comma-separated `$pt1, $pt2, \dots` reference list, naming each vertex by position.

```

5946 \cs_new_protected:Nn \__spinttent_spPoly_build_refs:
5947 {
5948     \seq_set_from_clist:NN \l__spinttent_spPoly_pt_seq
5949     \l__spinttent_geo_luaset_points_clist
5950     \tl_clear:N \l__spinttent_spPoly_refs_tl
5951     \int_zero:N \l__spinttent_spPoly_pt_idx_int
5952     \seq_map_inline:Nn \l__spinttent_spPoly_pt_seq
5953     {
5954         \int_incr:N \l__spinttent_spPoly_pt_idx_int
5955         \tl_if_empty:NF \l__spinttent_spPoly_refs_tl
5956         { \tl_put_right:Nn \l__spinttent_spPoly_refs_tl { , } }
5957         \tl_put_right:Ne \l__spinttent_spPoly_refs_tl
5958         { $ pt \int_use:N \l__spinttent_spPoly_pt_idx_int }
5959     }
5960 }

```

(End of definition for `\__spinttent_spPoly_build_refs:`.)

`\__spinttent_spPoly_build_concept:` Builds `\l__spinttent_spPoly_intent_arg_tl`, independently of `print-sym: ‘_’` when `silent` is `true`, `_ (refs)` when `concept` is `false`; otherwise `\l__spinttent_geo_luaset_concept_str:prefix(refs)`, with `prefix` placed after the leading `_` when given. Must be called after `\__spinttent_spPoly_build_refs:`.

```

5961 \cs_new_protected:Nn \__spinttent_spPoly_build_concept:

```

```

5962 {
5963   \bool_if:NTF \l__spinttent_spPoly_silent_bool
5964   {
5965     \tl_set:Nn \l__spinttent_spPoly_intent_arg_tl { _- }
5966   }
5967   {
5968     \bool_if:NTF \l__spinttent_spPoly_concept_bool
5969     {
5970       \tl_set:Ne \l__spinttent_spPoly_intent_arg_tl
5971       {
5972         -
5973         \tl_if_empty:NF \l__spinttent_spPoly_prefix_tl
5974         { \tl_use:N \l__spinttent_spPoly_prefix_tl - }
5975         \l__spinttent_geo_luaset_concept_str :prefix( \l__spinttent_spPoly_refs_tl
5976         )
5977       }
5978     }
5979     {
5980       \tl_set:Ne \l__spinttent_spPoly_intent_arg_tl
5981       { _ ( \l__spinttent_spPoly_refs_tl ) }
5982     }
5983   }
5984 }

```

(End of definition for `\l__spinttent_spPoly_build_concept:`.)

`\l__spinttent_spPoly_points_loop:` It walks `\l__spinttent_spPoly_pt_seq` a second time, calling `\l__spinttent_mathmllarg_point:nnn` once per vertex with its `ptn` reference name, looked up under `spPoly`. This call decides `arg` or the literal `intent` per vertex.

```

5985 \cs_new_protected:Nn \l__spinttent_spPoly_points_loop:
5986 {
5987   \int_zero:N \l__spinttent_spPoly_pt_idx_int
5988   \seq_map_inline:Nn \l__spinttent_spPoly_pt_seq
5989   {
5990     \int_incr:N \l__spinttent_spPoly_pt_idx_int
5991     \l__spinttent_mathmllarg_point:nnn
5992     { pt \int_use:N \l__spinttent_spPoly_pt_idx_int }
5993     { spPoly }
5994     { ##1 }
5995   }
5996 }

```

(End of definition for `\l__spinttent_spPoly_points_loop:`.)

`\l__spinttent_spPoly_exec:nn` Calls `\l__spinttent_luafun_geo_poly_parse_and_set:n`, reports `spPoly-invalid-points` on failure, then delegates to `\l__spinttent_spPoly_mathml:`. No `intent` is built here: it needs the reference list, which `\l__spinttent_spPoly_mathml:` builds after splitting the vertices.

Before processing the $\langle \textit{argument} \rangle$, it sets the optional $\langle \textit{keys} \rangle$ passed in ‘#1’.

```

5997 \cs_new_protected:Nn \l__spinttent_spPoly_exec:nn
5998 {
5999   \keys_set:nn { spinttent / spPoly } {##1}
6000   \l__spinttent_luafun_geo_poly_parse_and_set:n { \exp_not:n {##2} }
6001   \str_if_eq:VnT \l__spinttent_geo_luaset_error_str { true }
6002   {
6003     \msg_error:nne { spinttent } { spPoly-invalid-points }
6004     { \exp_not:n {##2} }
6005   }
6006   \l__spinttent_spPoly_mathml:
6007 }

```

(End of definition for `\l__spinttent_spPoly_exec:nn`.)

`\l__spinttent_spPoly_mathml:` Builds the reference list with `\l__spinttent_spPoly_build_refs:` and the concept `intent` with `\l__spinttent_spPoly_build_concept:`, then wraps everything in a single native `\MathMLintent`, preceded by `\l__spinttent_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 `\l__spinttent_spPoly_sym:`, silenced with `\l__spinttent_luafun_mrow_intent_arg:nn`, is attempted, and the glyph is skipped when `\l__spinttent_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.

```

6008 \cs_new_protected:Nn \__spintent_spPoly_mathml:
6009 {
6010   \__spintent_invisible_sep:
6011   \__spintent_spPoly_build_refs:
6012   \__spintent_spPoly_build_concept:
6013   \MathMLintent { \__spintent_spPoly_intent_arg_tl }
6014   {
6015     \__spintent_inner_sep:
6016     \bool_if:NT \__spintent_spPoly_print_sym_bool
6017     {
6018       \str_if_empty:NF \__spintent_geo_luaset_poly_sym_str
6019       {
6020         \__spintent_luafun_mrow_intent_arg:nn
6021         { } { intent = ":silent" }
6022         { \__spintent_spPoly_sym: }
6023       }
6024     }
6025     \__spintent_spPoly_points_loop:
6026   }
6027 }

```

(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`.

```

6028 \NewDocumentCommand \spPoly { O{} m }
6029 {
6030   \mode_if_math:TF
6031   {
6032     \group_begin:
6033     \attribute \__spintent_scope_attr_int = 1
6034     \__spintent_spPoly_exec:nn {#1} {#2}
6035     \group_end:
6036   }
6037   { \msg_error:nnn { spintent } { need-math-mode } { \spPoly } }
6038 }

```

(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 `\__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`

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`.

```

6039 \clist_map_inline:nn { Afig , Pfig }
6040 {
6041   \str_new:c { \__spintent_sp #1 _mode_str }
6042   \str_new:c { \__spintent_sp #1 _print_sym_str }
6043   \tl_new:c { \__spintent_sp #1 _prefix_tl }
6044   \tl_new:c { \__spintent_sp #1 _read_arg_tl }
6045 }

```

(End of definition for `\__spintent_spAfig_mode_str` and others.)

These variables are used by `\__spintent_spfig_exec:nnn`. The variable `\__spintent_geo_final_print_tl` stores the final visual placed under the subscript of `\mathcal` and `\__spintent_spfig_symbol_tl` stores the symbol of the figure, `\triangle` or `\mdlgwhtsquare`, when it is typeset.

```

6046 \tl_new:N \__spintent_geo_final_print_tl
6047 \tl_new:N \__spintent_spfig_symbol_tl

```

(End of definition for `\__spintent_geo_final_print_tl` and `\__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.

```
6048 \str_new:N \l__spintent_geo_print_sym_str
```

(End of definition for `\l__spintent_geo_print_sym_str`.)

`\l__spintent_spfig_op_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`.

```
6049 \tl_new:N \l__spintent_spfig_op_tl
6050 \tl_new:N \l__spintent_spfig_intent_tl
6051 \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`.)

`\l__spintent_spfig_pt_seq`

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, \dots` built from them.

```
6052 \seq_new:N \l__spintent_spfig_pt_seq
6053 \int_new:N \l__spintent_spfig_pt_idx_int
6054 \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_tl` is a scratch copy of `\l__spintent_spfig_desc_tl` from which the connector is removed.

```
6055 \tl_new:N \l__spintent_spfig_intent_arg_tl
6056 \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-conflict` are raised when `print-sym` is forced to a value incompatible with the actual argument.

```
6057 \clist_map_inline:nn { Afig , Pfig }
6058 {
6059   \msg_new:nnnn { spintent } { sp #1 -invalid-points }
6060   {
6061     Invalid~argument~'##2'~for~'\tl_trim_spaces:n {##1}'~\msg_line_context:.
6062   }
6063   {
6064     The~argument~must~be~a~list~of~points~separated~by~commas.
6065   }
6066   \msg_new:nnnn { spintent } { sp #1 -invalid-arg }
6067   {
6068     Invalid~argument~'##2'~for~'\tl_trim_spaces:n {##1}'~\msg_line_context:.
6069   }
6070   {
6071     See~the~manual~for~the~accepted~argument~forms.
6072   }
6073   \msg_new:nnnn { spintent } { sp #1 -sym-letter-conflict }
6074   {
6075     The~key~'print-sym'~cannot~be~combined~with~a~lettered~argument~in~'##1'~\
6076     \msg_line_context:.
6077   }
6078   {
6079     See~the~documentation~of~'##1'~for~details~on~the~key~'print-sym'.
6080   }
6081   \msg_new:nnnn { spintent } { sp #1 -sym-cardinality-conflict }
6082   {
6083     The~key~'print-sym'~does~not~match~the~vertex~count~in~'##1'~\msg_line_context:.
6084   }
}
```

```

6085     {
6086       See~the~documentation~of~'##1'~for~details~on~the~key~'print-sym'.
6087     }
6088     \msg_new:nnnn { spintent } { sp #1 -read-arg-key }
6089     {
6090       The~key~'read-arg'~has~no~effect~with~a~single~argument~in~'##1'~
6091       \msg_line_context:.
6092     }
6093     {
6094       See~the~documentation~of~'##1'~for~details~on~the~key~'read-arg'.
6095     }
6096   }

```

(End of definition for `spAfig`-invalid-points and others.)

15.52.3 Definition of keys for `\spAfig` and `\spPfig`

We define the shared *keys* `read-cmd`, `prefix`, `print-sym`, `read-arg`, `read-sub`, `silent` and `concept`.

```

read-cmd  \clist_map_inline:nn { Afig , Pfig }
prefix    {
print-sym  \keys_define:nn { spintent / sp #1 }
read-arg   {
read-sub   {
  silent    read-cmd .choices:nn = { school , mathcat }
  concept   {
concept*    \int_case:nn { \l_keys_choice_int }
unknown     {
6105         {1} { \str_set:cn { l__spintent_sp #1 _mode_str } { school } }
6106         {2} { \str_set:cn { l__spintent_sp #1 _mode_str } { mathcat } }
6107       }
6108     },
6109     read-cmd .initial:n = school,
6110     read-cmd .value_required:n = true,
6111     read-cmd / unknown .code:n =
6112       \msg_error:nneee { spintent } { unknown-choice }
6113       { read-cmd } { school,~mathcat } { \exp_not:n {##1} },
6114     prefix .code:n =
6115       { \__spintent_sanitize_affix:cn { l__spintent_sp #1 _prefix_tl } {##1} },
6116     prefix .initial:n = ,
6117     prefix .value_required:n = true,
6118     print-sym .choices:nn = { auto , triangle , square , none }
6119     {
6120       \int_case:nn { \l_keys_choice_int }
6121       {
6122         {1} { \str_set:cn { l__spintent_sp #1 _print_sym_str } { auto } }
6123         {2} { \str_set:cn { l__spintent_sp #1 _print_sym_str } { triangle } }
6124         {3} { \str_set:cn { l__spintent_sp #1 _print_sym_str } { square } }
6125         {4} { \str_set:cn { l__spintent_sp #1 _print_sym_str } { none } }
6126       }
6127     },
6128     print-sym .initial:n = auto,
6129     print-sym .value_required:n = true,
6130     print-sym / unknown .code:n =
6131       \msg_error:nneee { spintent } { unknown-choice }
6132       { print-sym } { auto,~triangle,~square,~none } { \exp_not:n {##1} },
6133     read-arg .code:n =
6134       { \__spintent_sanitize_affix:cn { l__spintent_sp #1 _read_arg_tl } {##1} },
6135     read-arg .initial:n = ,
6136     read-arg .value_required:n = true,
6137     read-sub .bool_set:c = { l__spintent_sp #1 _read_sub_bool },
6138     read-sub .initial:n = false,
6139     read-sub .value_required:n = true,
6140     silent .bool_set:c = { l__spintent_sp #1 _silent_bool },
6141     silent .initial:n = false,
6142     silent .value_required:n = true,
6143     concept .bool_set:c = { l__spintent_sp #1 _concept_bool },
6144     concept .initial:n = true,
6145     concept .value_required:n = true,
6146     concept* .meta:n =
6147     {
6148       concept = false,
6149       read-arg = { \str_if_eq:eeTF {#1} { Afig } { _área } { _perímetro } },
6150     },

```

```

6151         concept* .value_forbidden:n = true,
6152         unknown .code:n =
6153         {
6154             \str_set:Nn \l__spinttent_command_name_str { sp #1 }
6155             \__spinttent_unknown_keys_cmd:n {##1}
6156         },
6157     }
6158 }

```

(End of definition for read-cmd and others.)

15.52.4 Internal functions for \spAfig and \spPfig

`\__spinttent_spfig_sym:` The function `\__spinttent_spfig_sym:` reads `\l__spinttent_geo_print_sym_str` and typesets the corresponding symbol: `\triangle` for the value `triangle`, `\mdlgwhtsquare` for the value `square`.

```

6159 \cs_new_protected:Nn \__spinttent_spfig_sym:
6160 {
6161     \str_case:Nn \l__spinttent_geo_print_sym_str
6162     {
6163         { triangle } { \triangle }
6164         { square } { \mdlgwhtsquare }
6165     }
6166 }

```

(End of definition for `\__spinttent_spfig_sym:`.)

`\__spinttent_spfig_exec:nnn` Receives the short command name, Afig or Pfig, in ‘#1’ and the $\langle argument \rangle$ in ‘#3’. Calls `\__spinttent_luafun_geo_afig_parse_and_set:n` or `\__spinttent_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__spinttent_spfig_op_tl` to «*área*»/«*perímetro*», then delegates to `\__spinttent_spfig_mathml:n`.

Before processing the $\langle argument \rangle$, it sets the optional $\langle keys \rangle$ passed in ‘#2’.

```

6167 \cs_new_protected:Nn \__spinttent_spfig_exec:nnn
6168 {
6169     \keys_set:nn { spinttent / sp #1 } {#2}
6170     \str_if_eq:eeTF {#1} { Afig }
6171     { \__spinttent_luafun_geo_afig_parse_and_set:n { \exp_not:n {#3} } }
6172     { \__spinttent_luafun_geo_pfig_parse_and_set:n { \exp_not:n {#3} } }
6173     \str_if_eq:NnT \l__spinttent_geo_luaset_error_str { true }
6174     {
6175         \msg_error:nnee { spinttent } { sp #1 -invalid-arg }
6176         { \c_backslash_str sp #1 } { \exp_not:n {#3} }
6177     }
6178     \clist_if_empty:NT \l__spinttent_geo_luaset_points_clist
6179     {
6180         \tl_if_empty:cF { \l__spinttent_sp #1 _read_arg_tl }
6181         {
6182             \msg_warning:nne { spinttent } { sp #1 -read-arg-key }
6183             { \c_backslash_str sp #1 }
6184         }
6185     }
6186     \str_if_eq:eeTF {#1} { Afig }
6187     { \tl_set:Nn \l__spinttent_spfig_op_tl { área } }
6188     { \tl_set:Nn \l__spinttent_spfig_op_tl { perímetro } }
6189     \str_set:Ne \l__spinttent_geo_print_sym_str
6190     { \str_use:c { \l__spinttent_sp #1 _print_sym_str } }
6191     \__spinttent_spfig_mathml:n {#1}
6192 }

```

(End of definition for `\__spinttent_spfig_exec:nnn`.)

`\__spinttent_spfig_build_simple:n` Handles the argument-less, table-letter and bare number/letter cases, none of which has vertices. Builds `\l__spinttent_geo_final_print_tl`, the visual placed under `\mathcal`’s subscript, and `\l__spinttent_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.

```

6193 \cs_new_protected:Nn \__spinttent_spfig_build_simple:n

```

```

6194 {
6195   \tl_if_empty:NTF \l__spintent_geo_luaset_print_tl
6196   {
6197     \str_case:VnF \l__spintent_geo_print_sym_str
6198     {
6199       { auto }
6200       {
6201         \tl_clear:N \l__spintent_geo_final_print_tl
6202         \tl_clear:N \l__spintent_spfig_desc_tl
6203       }
6204       { none }
6205       {
6206         \tl_clear:N \l__spintent_geo_final_print_tl
6207         \tl_clear:N \l__spintent_spfig_desc_tl
6208       }
6209     }
6210   {
6211     \tl_set:Nn \l__spintent_geo_final_print_tl { \__spintent_spfig_sym: }
6212     \tl_set:Ne \l__spintent_spfig_desc_tl
6213     {
6214       \str_if_eq:VnTF \l__spintent_geo_print_sym_str { triangle }
6215       { triángulo } { cuadrado }
6216     }
6217   }
6218 }

```

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`.

```

6219 {
6220   \str_if_empty:NTF \l__spintent_geo_luaset_letra_word_str
6221   {
6222     \str_case:VnF \l__spintent_geo_print_sym_str
6223     {
6224       { auto }
6225       {
6226         \tl_set:NV \l__spintent_geo_final_print_tl
6227         \l__spintent_geo_luaset_print_tl
6228       }
6229       { none }
6230       {
6231         \tl_set:NV \l__spintent_geo_final_print_tl
6232         \l__spintent_geo_luaset_print_tl
6233       }
6234     }
6235   {
6236     \tl_set:Nn \l__spintent_geo_final_print_tl
6237     {
6238       \__spintent_spfig_sym:
6239       \c_math_subscript_token { \l__spintent_geo_luaset_print_tl }
6240     }
6241   }
6242   \str_case:VnF \l__spintent_geo_print_sym_str
6243   {
6244     { auto }
6245     {
6246       \tl_set:Ne \l__spintent_spfig_desc_tl
6247       {
6248         \bool_if:cT { \l__spintent_sp #1 _read_sub_bool } { sub - }
6249         \l__spintent_geo_luaset_sub_spoken_str
6250       }
6251     }
6252     { none }
6253     {
6254       \tl_set:Ne \l__spintent_spfig_desc_tl
6255       {
6256         \bool_if:cT { \l__spintent_sp #1 _read_sub_bool } { sub - }
6257         \l__spintent_geo_luaset_sub_spoken_str
6258       }
6259     }

```

```

6260         }
6261     {
6262         \tl_set:Nc \l__spintent_spfig_desc_tl
6263         {
6264             del-
6265             \str_if_eq:VnTF \l__spintent_geo_print_sym_str { triangle }
6266             { triángulo } { cuadrado }
6267         }
6268         \bool_if:cT { \l__spintent_sp #1 _read_sub_bool } { sub - }
6269         \l__spintent_geo_luaset_sub_spoken_str
6270     }
6271 }
6272 }

```

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.

```

6273 {
6274     \str_case:VnF \l__spintent_geo_print_sym_str
6275     {
6276         { auto } { }
6277         { none } { }
6278     }
6279     {
6280         \msg_error:nne { spintent } { sp #1 -sym-letter-conflict }
6281         { \c_backslash_str sp #1 }
6282     }
6283     \tl_set:NV \l__spintent_geo_final_print_tl \l__spintent_geo_luaset_print_tl
6284     \tl_set:Nc \l__spintent_spfig_desc_tl
6285     {
6286         \str_if_empty:NF \l__spintent_geo_luaset_letra_conector_str
6287         { \l__spintent_geo_luaset_letra_conector_str - }
6288         \l__spintent_geo_luaset_letra_word_str
6289         \str_if_empty:NF \l__spintent_geo_luaset_sub_spoken_str
6290         {
6291             -
6292             \bool_if:cT { \l__spintent_sp #1 _read_sub_bool } { sub - }
6293             \l__spintent_geo_luaset_sub_spoken_str
6294         }
6295     }
6296 }
6297 }

```

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`.

```

6298     \tl_set:Nc \l__spintent_spfig_intent_tl
6299     {
6300         \tl_if_empty:NTF \l__spintent_spfig_desc_tl
6301         { \l__spintent_spfig_op_tl - }
6302         { \l__spintent_spfig_op_tl - \l__spintent_spfig_desc_tl - }
6303     }
6304     \bool_if:cTF { \l__spintent_sp #1 _silent_bool }
6305     { \tl_set:Nn \l__spintent_spfig_intent_arg_tl { _- } }
6306     {
6307         \bool_if:cTF { \l__spintent_sp #1 _concept_bool }
6308         {
6309             \tl_set:Nc \l__spintent_spfig_intent_arg_tl
6310             {
6311                 -
6312                 \tl_if_empty:cF { \l__spintent_sp #1 _prefix_tl }
6313                 { \tl_use:c { \l__spintent_sp #1 _prefix_tl } - }
6314                 \l__spintent_spfig_intent_tl
6315             }
6316         }

```

```

6317     {
6318         \tl_set_eq:NN \l__spinttent_geo_temp_tl \l__spinttent_spfig_desc_tl
6319         \str_if_empty:NF \l__spinttent_geo_luaset_letra_conector_str
6320         {
6321             \tl_replace_once:Nee \l__spinttent_geo_temp_tl
6322             { \l__spinttent_geo_luaset_letra_conector_str - } { }
6323         }
6324         \tl_set:Ne \l__spinttent_spfig_intent_arg_tl { _ \l__spinttent_geo_temp_tl - }
6325     }
6326 }
6327 }

```

(End of definition for `\__spinttent_spfig_build_simple:n`.)

`\__spinttent_spfig_build_refs:` Splits `\l__spinttent_geo_luaset_points_clist` into `\l__spinttent_spfig_pt_seq` and walks it once to build `\l__spinttent_spfig_refs_tl`, the comma-separated `$pt1, $pt2, ...` reference list, naming each vertex by position.

```

6328 \cs_new_protected:Nn \__spinttent_spfig_build_refs:
6329 {
6330     \seq_set_from_clist:NN \l__spinttent_spfig_pt_seq
6331     \l__spinttent_geo_luaset_points_clist
6332     \tl_clear:N \l__spinttent_spfig_refs_tl
6333     \int_zero:N \l__spinttent_spfig_pt_idx_int
6334     \seq_map_inline:Nn \l__spinttent_spfig_pt_seq
6335     {
6336         \int_incr:N \l__spinttent_spfig_pt_idx_int
6337         \tl_if_empty:NF \l__spinttent_spfig_refs_tl
6338         { \tl_put_right:Nn \l__spinttent_spfig_refs_tl { , } }
6339         \tl_put_right:Ne \l__spinttent_spfig_refs_tl
6340         { $ pt \int_use:N \l__spinttent_spfig_pt_idx_int }
6341     }
6342 }

```

(End of definition for `\__spinttent_spfig_build_refs:.`)

`\__spinttent_spfig_build_concept_vertices:n` Handles the vertices case. Receives the short command name in ‘`#1`’. Must be called after `\__spinttent_spfig_build_refs:.`

The key `read-arg`, when given, overrides the auto-detected figure name in `\l__spinttent_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__spinttent_spfig_intent_tl` is `\l__spinttent_geo_luaset_concept_str`.

```

6343 \cs_new_protected:Nn \__spinttent_spfig_build_concept_vertices:n
6344 {
6345     \tl_if_empty:cTF { l__spinttent_sp #1 _read_arg_tl }
6346     {
6347         \tl_set:Ne \l__spinttent_spfig_intent_tl { \l__spinttent_geo_luaset_concept_str }
6348     }
6349     {
6350         \bool_if:cTF { l__spinttent_sp #1 _concept_bool }
6351         {
6352             \tl_set:Ne \l__spinttent_spfig_intent_tl
6353             {
6354                 \l__spinttent_spfig_op_tl -del-
6355                 \tl_use:c { l__spinttent_sp #1 _read_arg_tl }
6356             }
6357         }
6358         {
6359             \tl_set:Ne \l__spinttent_spfig_intent_tl
6360             { \tl_use:c { l__spinttent_sp #1 _read_arg_tl } }
6361         }
6362     }

```

Determines `\l__spinttent_spfig_symbol_tl`, which is empty when none applies: for `print-sym=auto`, the symbol matching `\l__spinttent_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.

```

6363 \str_case:VnF \l__spinttent_geo_print_sym_str
6364 {
6365     { auto }

```

```

6366     {
6367         \str_case:Vn \l__spinttent_geo_luaset_poly_sym_str
6368         {
6369             { triangle }
6370             { \tl_set:Nn \l__spinttent_spfig_symbol_tl { \triangle } }
6371             { square }
6372             { \tl_set:Nn \l__spinttent_spfig_symbol_tl { \mdlgwhtsquare } }
6373         }
6374         \str_if_empty:NT \l__spinttent_geo_luaset_poly_sym_str
6375         { \tl_clear:N \l__spinttent_spfig_symbol_tl }
6376     }
6377 { none }
6378 { \tl_clear:N \l__spinttent_spfig_symbol_tl }
6379 }
6380 {
6381     \str_if_eq:VVTf \l__spinttent_geo_print_sym_str
6382     \l__spinttent_geo_luaset_poly_sym_str
6383     { \tl_set:Nn \l__spinttent_spfig_symbol_tl { \__spinttent_spfig_sym: } }
6384     {
6385         \msg_error:nne { spinttent } { sp #1 -sym-cardinality-conflict }
6386         { \c_backslash_str sp #1 }
6387     }
6388 }

```

The key `silent` sets `\l__spinttent_spfig_intent_arg_tl` to ‘`-`’. Otherwise, with `read-arg` empty, `concept` chooses between `\l__spinttent_spfig_intent_tl:prefix(refs)`, with `prefix` placed after the leading `_` when given, and `_(refs)` alone; with `read-arg` given, it is always `\l__spinttent_spfig_intent_tl:prefix(refs)`.

```

6389     \bool_if:cTF { l__spinttent_sp #1 _silent_bool }
6390     {
6391         \tl_set:Nn \l__spinttent_spfig_intent_arg_tl { _- }
6392     }
6393     {
6394         \tl_if_empty:cTF { l__spinttent_sp #1 _read_arg_tl }
6395         {
6396             \bool_if:cTF { l__spinttent_sp #1 _concept_bool }
6397             {
6398                 \tl_set:Ne \l__spinttent_spfig_intent_arg_tl
6399                 {
6400                     -
6401                     \tl_if_empty:cF { l__spinttent_sp #1 _prefix_tl }
6402                     { \tl_use:c { l__spinttent_sp #1 _prefix_tl } - }
6403                     \l__spinttent_spfig_intent_tl :prefix( \l__spinttent_spfig_refs_tl )
6404                 }
6405             }
6406             {
6407                 \tl_set:Ne \l__spinttent_spfig_intent_arg_tl
6408                 { _ ( \l__spinttent_spfig_refs_tl ) }
6409             }
6410         }
6411         {
6412             \tl_set:Ne \l__spinttent_spfig_intent_arg_tl
6413             {
6414                 -
6415                 \tl_if_empty:cF { l__spinttent_sp #1 _prefix_tl }
6416                 { \tl_use:c { l__spinttent_sp #1 _prefix_tl } - }
6417                 \l__spinttent_spfig_intent_tl :prefix( \l__spinttent_spfig_refs_tl )
6418             }
6419         }
6420     }
6421 }

```

(End of definition for `\__spinttent_spfig_build_concept_vertices:n`)

`\__spinttent_spfig_points_loop:n`

Walks `\l__spinttent_spfig_pt_seq` a second time, calling `\__spinttent_mathmlarg_point:nnn` once per vertex with its `ptn` reference name, looked up under `spAfig` or `spPfig`. Receives the short command name in ‘`#1`’.

```

6422 \cs_new_protected:Nn \__spinttent_spfig_points_loop:n
6423 {
6424     \int_zero:N \l__spinttent_spfig_pt_idx_int

```

```

6425 \seq_map_inline:Nn \l__spinttent_spfig_pt_seq
6426 {
6427   \int_incr:N \l__spinttent_spfig_pt_idx_int
6428   \__spinttent_mathmlarg_point:nnn
6429   { pt \int_use:N \l__spinttent_spfig_pt_idx_int }
6430   { sp #1 }
6431   { ##1 }
6432 }
6433 }

```

(End of definition for `\__spinttent_spfig_points_loop:n`.)

`\__spinttent_spfig_mathml:n`

Receives the short command name in ‘#1’. Branches on whether `\l__spinttent_geo_luaseta_points_clist` is empty.

Without vertices, `\__spinttent_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, `\__spinttent_spfig_build_refs:` builds the reference list and `\__spinttent_spfig_build_concept` builds the concept `intent`, applied with a single native `\MathMLintent`. Inside, `\__spinttent_mathcal:n{⟨A | P⟩}` is silenced with `\__spinttent_luafun_mrow_intent_arg:nn`; the same is done with `\l__spinttent_spfig_symbol_tl` when it is non-empty; the sequence is then walked by `\__spinttent_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 node.

The function `\__spinttent_invisible_sep:` precedes both branches. It adds a second sibling so that the sole `<mrow>` does not collapse into `<math>`.

```

6434 \cs_new_protected:Nn \__spinttent_spfig_mathml:n
6435 {
6436   \__spinttent_invisible_sep:
6437   \clist_if_empty:NTF \l__spinttent_geo_luaseta_points_clist
6438   {
6439     \__spinttent_spfig_build_simple:n {#1}
6440     \MathMLintent { \l__spinttent_spfig_intent_arg_tl }
6441     {
6442       \__spinttent_inner_sep:
6443       \str_if_eq:eeTF {#1} { Afig } { \__spinttent_mathcal:n { A } }
6444       { \__spinttent_mathcal:n { P } }
6445       \tl_if_empty:NF \l__spinttent_geo_final_print_tl
6446       {
6447         \c_math_subscript_token { \l__spinttent_geo_final_print_tl }
6448       }
6449     }
6450   }
6451   {
6452     \__spinttent_spfig_build_refs:
6453     \__spinttent_spfig_build_concept_vertices:n {#1}
6454     \MathMLintent { \l__spinttent_spfig_intent_arg_tl }
6455     {
6456       \__spinttent_inner_sep:
6457       \__spinttent_luafun_mrow_intent_arg:nn
6458       { } { intent = ":silent" }
6459       {
6460         \str_if_eq:eeTF {#1} { Afig } { \__spinttent_mathcal:n { A } }
6461         { \__spinttent_mathcal:n { P } }
6462       }
6463       \c_math_subscript_token
6464       {
6465         \tl_if_empty:NF \l__spinttent_spfig_symbol_tl
6466         {
6467           \__spinttent_luafun_mrow_intent_arg:nn
6468           { } { intent = ":silent" }
6469           { \tl_use:N \l__spinttent_spfig_symbol_tl }
6470         }
6471         \__spinttent_spfig_points_loop:n {#1}
6472       }
6473     }
6474   }
6475 }

```

(End of definition for `\__spinttent_spfig_mathml:n`.)

`\spAfig` Both public commands `\spAfig` and `\spPfig` enforce math mode, open a local group, and delegate execution to `\__spintent_spfig_exec:nnn`.

`\spPfig`

```

6476 \NewDocumentCommand \spAfig { 0{} d() }
6477 {
6478   \mode_if_math:TF
6479   {
6480     \group_begin:
6481       \attribute \__spintent_scope_attr_int = 1
6482       \tl_if_novalue:nTF {#2}
6483       { \__spintent_spfig_exec:nnn { Afig } {#1} {} }
6484       { \__spintent_spfig_exec:nnn { Afig } {#1} {#2} }
6485     \group_end:
6486   }
6487   { \msg_error:nnn { spintent } { need-math-mode } { \spAfig } }
6488 }
6489 \NewDocumentCommand \spPfig { 0{} d() }
6490 {
6491   \mode_if_math:TF
6492   {
6493     \group_begin:
6494       \attribute \__spintent_scope_attr_int = 1
6495       \tl_if_novalue:nTF {#2}
6496       { \__spintent_spfig_exec:nnn { Pfig } {#1} {} }
6497       { \__spintent_spfig_exec:nnn { Pfig } {#1} {#2} }
6498     \group_end:
6499   }
6500   { \msg_error:nnn { spintent } { need-math-mode } { \spPfig } }
6501 }

```

(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`

`\__spintent_sp_rad_diam_letra_tl` The variables `\__spintent_sp_rad_diam_letra_tl` and `\__spintent_sp_rad_diam_word_tl` are scratch variables used by `\__spintent_sp_rad_diam_mathml:n`. `\__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.

`\__spintent_sp_rad_diam_word_tl`

`\__spintent_sp_rad_diam_read_arg_full_tl`

```

6502 \tl_new:N \__spintent_sp_rad_diam_letra_tl
6503 \tl_new:N \__spintent_sp_rad_diam_word_tl
6504 \tl_new:N \__spintent_sp_rad_diam_read_arg_full_tl

```

(End of definition for `\__spintent_sp_rad_diam_letra_tl`, `\__spintent_sp_rad_diam_word_tl`, and `\__spintent_sp_rad_diam_read_arg_full_tl`.)

`\__spintent_sprad_mode_str` 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`.

`\__spintent_sprad_sub_index_str`

`\__spintent_sprad_read_arg_tl`

`\__spintent_spdiam_mode_str`

`\__spintent_spdiam_sub_index_str`

`\__spintent_spdiam_read_arg_tl`

```

6505 \clist_map_inline:nn { rad , diam }
6506 {
6507   \str_new:c { \__spintent_sp #1 _mode_str }
6508   \str_new:c { \__spintent_sp #1 _sub_index_str }
6509   \tl_new:c { \__spintent_sp #1 _read_arg_tl }
6510 }

```

(End of definition for `\__spintent_sprad_mode_str` and others.)

15.53.2 Definition of keys for `\sprad` and `\spdiam`

`read-cmd` We define the shared `\keys` `read-cmd`, upper-case, sub-index, read-sub, read-arg, concept, silent and `print-sym`.

`upper-case`

`sub-index`

`read-sub`

`read-arg`

`concept`

`silent`

`print-sym`

`unknown`

```

6511 \clist_map_inline:nn { rad , diam }
6512 {
6513   \keys_define:nn { spintent / sp #1 }
6514   {
6515     read-cmd .choices:nn = { school , mathcat }
6516   }

```

```

6517         \int_case:nn { \l_keys_choice_int }
6518         {
6519             {1} { \str_set:cn { \__spintent_sp #1 _mode_str } { school } }
6520             {2} { \str_set:cn { \__spintent_sp #1 _mode_str } { mathcat } }
6521         }
6522     },
6523     read-cmd .initial:n = school,
6524     read-cmd .value_required:n = true,
6525     read-cmd / unknown .code:n =
6526         \msg_error:nneee { spintent } { unknown-choice }
6527         { read-cmd } { school,~mathcat } { \exp_not:n {##1} },
6528     upper-case .bool_set:c = { \__spintent_sp #1 _upper_case_bool },
6529     upper-case .initial:n = false,
6530     upper-case .value_required:n = true,
6531     sub-index .code:n =
6532         { \str_set:cn { \__spintent_sp #1 _sub_index_str } {##1} },
6533     sub-index .initial:n = ,
6534     sub-index .value_required:n = true,
6535     read-sub .bool_set:c = { \__spintent_sp #1 _read_sub_bool },
6536     read-sub .initial:n = false,
6537     read-sub .value_required:n = true,
6538     read-arg .code:n =
6539         { \__spintent_sanitizize_affix:cn { \__spintent_sp #1 _read_arg_tl } {##1} },
6540     read-arg .initial:n = ,
6541     read-arg .value_required:n = true,
6542     concept .bool_set:c = { \__spintent_sp #1 _concept_bool },
6543     concept .initial:n = true,
6544     concept .value_required:n = true,
6545     silent .bool_set:c = { \__spintent_sp #1 _silent_bool },
6546     silent .initial:n = false,
6547     silent .value_required:n = true,
6548     print-sym .bool_set:c = { \__spintent_sp #1 _print_sym_bool },
6549     print-sym .initial:n = false,
6550     print-sym .value_required:n = true,
6551     unknown .code:n =
6552         {
6553             \str_set:Nn \__spintent_command_name_str { sp #1 }
6554             \__spintent_unknown_keys_cmd:n {##1}
6555         },
6556     }
6557 }

```

🔑 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 `\__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`.

```

6558 \cs_new_protected:Nn \__spintent_sp_rad_diam_mathml:n
6559 {
6560     \bool_if:cTF { \__spintent_sp #1 _upper_case_bool }
6561     {
6562         \str_if_eq:eeTF {#1} { rad }
6563         { \tl_set:Nn \__spintent_sp_rad_diam_letra_tl { R } }
6564         { \tl_set:Nn \__spintent_sp_rad_diam_letra_tl { D } }
6565     }
6566     {
6567         \str_if_eq:eeTF {#1} { rad }
6568         { \tl_set:Nn \__spintent_sp_rad_diam_letra_tl { r } }
6569         { \tl_set:Nn \__spintent_sp_rad_diam_letra_tl { d } }
6570     }
6571     \__spintent_invisible_sep:
6572     \str_if_eq:eeT {#1} { diam }
6573     {
6574         \bool_if:NT \__spintent_spdiam_print_sym_bool
6575         { \MathMLintent { :silent } { \diameter } }
6576     }

```

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.

```

6577 \bool_if:cTF { l__spintent_sp #1 _silent_bool }
6578 {
6579   \str_if_empty:cTF { l__spintent_sp #1 _sub_index_str }
6580   {
6581     \__spintent_luafun_mrow_intent_arg:nn
6582     { } { intent = "- " }
6583     { \l__spintent_sp_rad_diam_letra_tl }
6584   }
6585   {
6586     \MathMLintent { - }
6587     {
6588       \__spintent_inner_sep:
6589       \l__spintent_sp_rad_diam_letra_tl
6590       \c_math_subscript_token
6591       {
6592         \MathMLintent { :silent }
6593         { \str_use:c { l__spintent_sp #1 _sub_index_str } }
6594       }
6595     }
6596   }
6597 }

```

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- $\langle value \rangle$ or $\langle value \rangle$ 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.

```

6598 {
6599   \tl_if_empty:cTF { l__spintent_sp #1 _read_arg_tl }
6600   {
6601     \bool_if:cTF { l__spintent_sp #1 _concept_bool }
6602     {
6603       \str_if_eq:eeTF {#1} { rad }
6604       { \tl_set:Nn \l__spintent_sp_rad_diam_word_tl { radio } }
6605       { \tl_set:Nn \l__spintent_sp_rad_diam_word_tl { diámetro } }
6606       \str_if_empty:cTF { l__spintent_sp #1 _sub_index_str }
6607       {
6608         \__spintent_luafun_mrow_intent_arg:nn
6609         { } { intent = "- \l__spintent_sp_rad_diam_word_tl :nofix" }
6610         { \l__spintent_sp_rad_diam_letra_tl }
6611       }
6612       {
6613         \str_if_eq:eeTF
6614         { \str_use:c { l__spintent_sp #1 _mode_str } } { school }
6615         {
6616           \MathMLintent
6617           {
6618             - \l__spintent_sp_rad_diam_word_tl -
6619             \bool_if:cT { l__spintent_sp #1 _read_sub_bool } { sub- }
6620             \str_use:c { l__spintent_sp #1 _sub_index_str }
6621           }
6622           {
6623             \__spintent_inner_sep:
6624             \l__spintent_sp_rad_diam_letra_tl
6625             \c_math_subscript_token
6626             {
6627               \MathMLintent { :silent }
6628               { \str_use:c { l__spintent_sp #1 _sub_index_str } }
6629             }
6630           }
6631         }
6632         {
6633           \__spintent_luafun_mrow_intent_arg:nn
6634           { } { intent = "- \l__spintent_sp_rad_diam_word_tl :nofix" }
6635           { \l__spintent_sp_rad_diam_letra_tl }
6636           \c_math_subscript_token
6637           { \str_use:c { l__spintent_sp #1 _sub_index_str } }
6638         }

```

```

6639         }
6640     }

```

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.

```

6641     {
6642         \str_if_empty:cTF { \__spintent_sp #1 _sub_index_str }
6643         {
6644             \str_if_eq:eeTF
6645             { \str_use:c { \__spintent_sp #1 _mode_str } } { school }
6646             {
6647                 \__spintent_luafun_mrow_intent_arg:nn
6648                 { } { intent = "_" \__spintent_sp_rad_diam_letra_tl -" }
6649                 { \__spintent_sp_rad_diam_letra_tl }
6650             }
6651             { \__spintent_sp_rad_diam_letra_tl }
6652         }
6653         {
6654             \str_if_eq:eeTF
6655             { \str_use:c { \__spintent_sp #1 _mode_str } } { school }
6656             {
6657                 \MathMLintent
6658                 {
6659                     _ \__spintent_sp_rad_diam_letra_tl -
6660                     \bool_if:cT { \__spintent_sp #1 _read_sub_bool } { sub- }
6661                     \str_use:c { \__spintent_sp #1 _sub_index_str }
6662                 }
6663                 {
6664                     \__spintent_inner_sep:
6665                     \__spintent_sp_rad_diam_letra_tl
6666                     \c_math_subscript_token
6667                     {
6668                         \MathMLintent { :silent }
6669                         { \str_use:c { \__spintent_sp #1 _sub_index_str } }
6670                     }
6671                 }
6672             }
6673             {
6674                 \__spintent_sp_rad_diam_letra_tl
6675                 \c_math_subscript_token
6676                 { \str_use:c { \__spintent_sp #1 _sub_index_str } }
6677             }
6678         }
6679     }
6680 }
6681 { \__spintent_sp_rad_diam_read_arg_print:n {#1} }
6682 }
6683 }

```

(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, `\__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`.

```

6684 \cs_new_protected:Nn \__spintent_sp_rad_diam_read_arg_print:n
6685 {
6686     \bool_if:cTF { \__spintent_sp #1 _concept_bool }
6687     {
6688         \str_if_eq:eeTF {#1} { rad }
6689         {
6690             \tl_set:Ne \__spintent_sp_rad_diam_read_arg_full_tl
6691             { radio- \tl_use:c { \__spintent_sp #1 _read_arg_tl } }
6692         }
6693         {
6694             \tl_set:Ne \__spintent_sp_rad_diam_read_arg_full_tl
6695             { diámetro- \tl_use:c { \__spintent_sp #1 _read_arg_tl } }

```

```

6696     }
6697   }
6698   {
6699     \tl_set_eq:Nc \l__spintent_sp_rad_diam_read_arg_full_tl
6700     { \l__spintent_sp #1 _read_arg_tl }
6701   }
6702   \str_if_empty:cTF { \l__spintent_sp #1 _sub_index_str }
6703   {
6704     \__spintent_luafun_mrow_intent_arg:nn
6705     { } { intent = "_ \l__spintent_sp_rad_diam_read_arg_full_tl :nofix" }
6706     { \l__spintent_sp_rad_diam_letra_tl }
6707   }
6708   {
6709     \MathMLintent
6710     { _ \l__spintent_sp_rad_diam_read_arg_full_tl :nofix }
6711     {
6712       \__spintent_inner_sep:
6713       \l__spintent_sp_rad_diam_letra_tl
6714       \c_math_subscript_token
6715       {
6716         \MathMLintent { :silent }
6717         { \str_use:c { \l__spintent_sp #1 _sub_index_str } }
6718       }
6719     }
6720   }
6721 }

```

(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`.

```

6722 \cs_new_protected:Nn \__spintent_sp_rad_diam_exec:nn
6723 {
6724   \keys_set:nn { spintent / sp #1 } {#2}
6725   \__spintent_sp_rad_diam_mathml:n {#1}
6726 }

```

(End of definition for `\__spintent_sp_rad_diam_exec:nn`.)

`\sprad` Both public commands `\sprad` and `\spdiam` enforces *math mode*, opens a local group, and delegates execution to `\__spintent_sp_rad_diam_exec:nn`.

```

6727 \NewDocumentCommand \sprad { 0{} }
6728 {
6729   \mode_if_math:TF
6730   {
6731     \group_begin:
6732     \attribute \l__spintent_scope_attr_int = 1
6733     \__spintent_sp_rad_diam_exec:nn { rad } {#1}
6734     \group_end:
6735   }
6736   { \msg_error:nnn { spintent } { need-math-mode } { \sprad } }
6737 }
6738 \NewDocumentCommand \spdiam { 0{} }
6739 {
6740   \mode_if_math:TF
6741   {
6742     \group_begin:
6743     \attribute \l__spintent_scope_attr_int = 1
6744     \__spintent_sp_rad_diam_exec:nn { diam } {#1}
6745     \group_end:
6746   }
6747   { \msg_error:nnn { spintent } { need-math-mode } { \spdiam } }
6748 }

```

(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`

`\__spinttent_geo_luaset_circ_radio_type_str`
`\__spinttent_geo_luaset_circ_radio_raw_str`

Shared outputs of `\__spinttent_luafun_geo_circ_parse_and_set:n`. The variable `\__spinttent_geo_luaset_circ_radio_type_str` holds one of `number`, `measure`, `label`, `segment`, or an empty string when no radius was given. `\__spinttent_geo_luaset_circ_radio_raw_str` holds the corresponding raw text, ready to be forwarded to the subcommand that renders it.

```
6749 \str_new:N \__spinttent_geo_luaset_circ_radio_type_str
6750 \str_new:N \__spinttent_geo_luaset_circ_radio_raw_str
```

(End of definition for `\__spinttent_geo_luaset_circ_radio_type_str` and `\__spinttent_geo_luaset_circ_radio_raw_str`.)

`\__spinttent_spcirc_mode_str`
`\__spinttent_spcirc_prefix_tl`
`\__spinttent_spcirc_print_sym_str`
`\__spinttent_spcirc_read_sty_str`
`\__spinttent_spcirc_radio_sty_str`
`\__spinttent_spcirc_sub_index_str`
`\__spinttent_spcirc_read_arg_tl`

These variables store the values set by the keys. The variable `\__spinttent_spcirc_mode_str` stores `read-cmd` and `\__spinttent_spcirc_prefix_tl` stores `prefix`. The variable `\__spinttent_spcirc_print_sym_str` stores `print-sym`, either `usual` or `odot`. The variable `\__spinttent_spcirc_read_sty_str` stores `read-sty`, either `circun` or `circulo`. The variable `\__spinttent_spcirc_radio_sty_str` stores `radio-sty`, only meaningful when the radius is a segment. The variable `\__spinttent_spcirc_sub_index_str` stores `sub-index` and `\__spinttent_spcirc_read_arg_tl` stores `read-arg`.

```
6751 \str_new:N \__spinttent_spcirc_mode_str
6752 \tl_new:N \__spinttent_spcirc_prefix_tl
6753 \str_new:N \__spinttent_spcirc_print_sym_str
6754 \str_new:N \__spinttent_spcirc_read_sty_str
6755 \str_new:N \__spinttent_spcirc_radio_sty_str
6756 \str_new:N \__spinttent_spcirc_sub_index_str
6757 \tl_new:N \__spinttent_spcirc_read_arg_tl
```

(End of definition for `\__spinttent_spcirc_mode_str` and others.)

`\__spinttent_spcirc_articulo_str`
`\__spinttent_spcirc_word_tl`

The variable `\__spinttent_spcirc_articulo_str` is set by `\__spinttent_spcirc_word_and_article:` to `la` or `el`, matching the gender of the chosen word. The variable `\__spinttent_spcirc_word_tl` stores that word, `circunferencia` or `círculo`, or the `read-arg` override; it is used by `\__spinttent_spcirc_exec:nn`.

```
6758 \str_new:N \__spinttent_spcirc_articulo_str
6759 \tl_new:N \__spinttent_spcirc_word_tl
```

(End of definition for `\__spinttent_spcirc_articulo_str` and `\__spinttent_spcirc_word_tl`.)

`\__spinttent_spcirc_has_arg_bool`
`\__spinttent_spcirc_intent_tl`
`\__spinttent_spcirc_intent_arg_tl`
`\__spinttent_spcirc_connector_tl`
`\__spinttent_spcirc_connector_arg_tl`
`\__spinttent_spcirc_radio_arg_tl`

The variable `\__spinttent_spcirc_has_arg_bool` stores whether the optional argument `d()` was given. The variable `\__spinttent_spcirc_intent_tl` stores the word `circunferencia` or `círculo`, followed by the `sub-index` suffix if present. The variable `\__spinttent_spcirc_connector_tl` stores `de-centro-en`. The variables `\__spinttent_spcirc_intent_arg_tl` and `\__spinttent_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 `\__spinttent_spcirc_radio_arg_tl` stores the `intent` of the radius, spoken on the delimiter `,`.

```
6760 \bool_new:N \__spinttent_spcirc_has_arg_bool
6761 \tl_new:N \__spinttent_spcirc_intent_tl
6762 \tl_new:N \__spinttent_spcirc_intent_arg_tl
6763 \tl_new:N \__spinttent_spcirc_connector_tl
6764 \tl_new:N \__spinttent_spcirc_connector_arg_tl
6765 \tl_new:N \__spinttent_spcirc_radio_arg_tl
```

(End of definition for `\__spinttent_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.

```
6766 \msg_new:nnnn { spinttent } { spcirc-invalid-arg }
6767 {
6768   Invalid~argument~'#1'~for~'\token_to_str:N \spcirc'~\msg_line_context:.
6769 }
6770 {
6771   The~argument~must~have~the~form~'(center)'~or~'(center,~radius)'.
6772 }
```

(End of definition for `spcirc-invalid-arg`.)

15.54.3 Definition of keys for \spcirc

We define the *(keys)* read-cmd, prefix, print-sym, read-sty, diameter, radio-sty, sub-index, read-sub, read-arg, silent and concept.

```

6773 \keys_define:nn { spintent / spcirc }
6774 {
6775   read-cmd .choices:nn = { school , mathcat }
6776   {
6777     \int_case:nn { \l_keys_choice_int }
6778     {
6779       {1}
6780       {
6781         \str_set:Nn \l__spintent_spcirc_mode_str { school }
6782         \keys_set:nn { spintent / spside } { read-cmd = school }
6783         \keys_set:nn { spintent / spmside } { read-cmd = school }
6784       }
6785       {2}
6786       {
6787         \str_set:Nn \l__spintent_spcirc_mode_str { mathcat }
6788         \keys_set:nn { spintent / spside } { read-cmd = mathcat }
6789         \keys_set:nn { spintent / spmside } { read-cmd = mathcat }
6790       }
6791     }
6792   },
6793   read-cmd .initial:n = school,
6794   read-cmd .value_required:n = true,
6795   read-cmd / unknown .code:n =
6796     \msg_error:nnee { spintent } { unknown-choice }
6797     { read-cmd } { school,~mathcat } { \exp_not:n {#1} },
6798   prefix .code:n =
6799     { \__spintent_sanitizet_affix:Nn \l__spintent_spcirc_prefix_tl {#1} },
6800   prefix .initial:n = ,
6801   prefix .value_required:n = true,
6802   print-sym .choices:nn = { usual , odot }
6803   {
6804     \int_case:nn { \l_keys_choice_int }
6805     {
6806       {1} { \str_set:Nn \l__spintent_spcirc_print_sym_str { usual } }
6807       {2} { \str_set:Nn \l__spintent_spcirc_print_sym_str { odot } }
6808     }
6809   },
6810   print-sym .initial:n = odot,
6811   print-sym .value_required:n = true,
6812   print-sym / unknown .code:n =
6813     \msg_error:nnee { spintent } { unknown-choice }
6814     { print-sym } { usual,~odot } { \exp_not:n {#1} },
6815   read-sty .choices:nn = { circun , circulo }
6816   {
6817     \int_case:nn { \l_keys_choice_int }
6818     {
6819       {1} { \str_set:Nn \l__spintent_spcirc_read_sty_str { circun } }
6820       {2} { \str_set:Nn \l__spintent_spcirc_read_sty_str { circulo } }
6821     }
6822   },
6823   read-sty .initial:n = circun,
6824   read-sty .value_required:n = true,
6825   read-sty / unknown .code:n =
6826     \msg_error:nnee { spintent } { unknown-choice }
6827     { read-sty } { circun,~circulo } { \exp_not:n {#1} },
6828   diameter .bool_set:N = \l__spintent_spcirc_diameter_bool,
6829   diameter .initial:n = false,
6830   diameter .value_required:n = true,
6831   radio-sty .choices:nn = { side , mside }
6832   {
6833     \int_case:nn { \l_keys_choice_int }
6834     {
6835       {1} { \str_set:Nn \l__spintent_spcirc_radio_sty_str { side } }
6836       {2} { \str_set:Nn \l__spintent_spcirc_radio_sty_str { mside } }
6837     }
6838   },
6839   radio-sty .initial:n = mside,

```

```

6840     radio-sty .value_required:n = true,
6841     radio-sty / unknown .code:n =
6842       \msg_error:nneee { spintent } { unknown-choice }
6843       { radio-sty } { side,~mside } { \exp_not:n {#1} },
6844     sub-index .code:n =
6845       { \str_set:Nn \l__spintent_spcirc_sub_index_str {#1} },
6846     sub-index .initial:n = ,
6847     sub-index .value_required:n = true,
6848     read-sub .bool_set:N = \l__spintent_spcirc_read_sub_bool,
6849     read-sub .initial:n = false,
6850     read-sub .value_required:n = true,
6851     read-arg .code:n =
6852       { \__spintent_sanitize_affix:Nn \l__spintent_spcirc_read_arg_tl {#1} },
6853     read-arg .initial:n = ,
6854     read-arg .value_required:n = true,
6855     silent .bool_set:N = \l__spintent_spcirc_silent_bool,
6856     silent .initial:n = false,
6857     silent .value_required:n = true,
6858     concept .bool_set:N = \l__spintent_spcirc_concept_bool,
6859     concept .initial:n = true,
6860     concept .value_required:n = true,
6861     spside .meta:nn = { spintent / spside } {#1},
6862     spside .value_required:n = true,
6863     spmside .meta:nn = { spintent / spmside } {#1},
6864     spmside .value_required:n = true,
6865     unknown .code:n =
6866       {
6867         \str_set:Nn \l__spintent_command_name_str { spcirc }
6868         \__spintent_unknown_keys_cmd:n {#1}
6869       },
6870   }

```

(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`.

```

6871 \cs_new_protected:Nn \__spintent_spcirc_sym:
6872 {
6873   \str_if_eq:VnTF \l__spintent_spcirc_print_sym_str { odot }
6874   {
6875     \group_begin:
6876     \textstyle\bigodot
6877     \group_end:
6878   }
6879   { C }
6880 }

```

(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.

```

6881 \cs_new_protected:Nn \__spintent_spcirc_word_and_article:
6882 {
6883   \tl_if_empty:NTF \l__spintent_spcirc_read_arg_tl
6884   {
6885     \str_if_eq:VnTF \l__spintent_spcirc_read_sty_str { circun }
6886     {
6887       \tl_set:Nn \l__spintent_spcirc_word_tl { circunferencia }
6888       \str_set:Nn \l__spintent_spcirc_articulo_str { la }
6889     }
6890     {
6891       \tl_set:Nn \l__spintent_spcirc_word_tl { círculo }
6892       \str_set:Nn \l__spintent_spcirc_articulo_str { el }
6893     }
6894   }
6895   {
6896     \tl_set:NV \l__spintent_spcirc_word_tl \l__spintent_spcirc_read_arg_tl
6897     \str_if_eq:VnTF \l__spintent_spcirc_read_sty_str { circun }
6898     { \str_set:Nn \l__spintent_spcirc_articulo_str { la } }

```

```

6899         { \str_set:Nn \l__spinttent_spcirc_articulo_str { el } }
6900     }
6901 }

```

(End of definition for `\__spinttent_spcirc_word_and_article:`)

`\__spinttent_spcirc_radio_print:` Branches on `\l__spinttent_geo_luaset_circ_radio_type_str`, reusing `\spnum/\spunit/\spside/\spmside` directly rather than reimplementing any parsing.

```

6902 \cs_new_protected:Nn \__spinttent_spcirc_radio_print:
6903 {
6904     \str_case:Nn \l__spinttent_geo_luaset_circ_radio_type_str
6905     {
6906         { number }
6907         { \exp_args:Ne \spnum { \l__spinttent_geo_luaset_circ_radio_raw_str } }
6908         { measure }
6909         { \exp_args:Ne \spunit { \l__spinttent_geo_luaset_circ_radio_raw_str } }
6910         { label } { \l__spinttent_geo_luaset_circ_radio_raw_str }
6911         { segment }
6912         {
6913             \str_if_eq:NnTF \l__spinttent_spcirc_radio_sty_str { side }
6914             { \exp_args:Ne \spside { \l__spinttent_geo_luaset_circ_radio_raw_str } }
6915             { \exp_args:Ne \spmside { \l__spinttent_geo_luaset_circ_radio_raw_str } }
6916         }
6917     }
6918 }

```

(End of definition for `\__spinttent_spcirc_radio_print:`)

`\__spinttent_spcirc_build_concept:` Builds the words `\l__spinttent_spcirc_intent_tl`, with `\__spinttent_spcirc_word_and_article:` and an optional `sub-index` suffix, and `\l__spinttent_spcirc_connector_tl`, de-centro-en. Both are built even when `\l__spinttent_spcirc_has_arg_bool` is `false`, although `\__spinttent_spcirc_mathml:` does not use them in that case. The center is not concatenated into them: it is read by reference with `\__spinttent_mathmlarg_point:nnn`.

```

6919 \cs_new_protected:Nn \__spinttent_spcirc_build_concept:
6920 {
6921     \__spinttent_spcirc_word_and_article:
6922     \tl_set:Nn \l__spinttent_spcirc_intent_tl { \l__spinttent_spcirc_word_tl }
6923     \str_if_empty:NF \l__spinttent_spcirc_sub_index_str
6924     {
6925         \tl_put_right:Nn \l__spinttent_spcirc_intent_tl
6926         {
6927             -
6928             \bool_if:NT \l__spinttent_spcirc_read_sub_bool { sub - }
6929             \l__spinttent_spcirc_sub_index_str
6930         }
6931     }
6932     \tl_set:Nn \l__spinttent_spcirc_connector_tl { de-centro-en }

```

The key `silent`, when `true`, overrides `\l__spinttent_spcirc_intent_arg_tl`, `\l__spinttent_spcirc_connector_arg_tl` and `\l__spinttent_spcirc_radio_arg_tl` with ‘`_-`’, regardless of `read-arg`.

```

6933     \bool_if:NTF \l__spinttent_spcirc_silent_bool
6934     {
6935         \tl_set:Nn \l__spinttent_spcirc_intent_arg_tl { _- }
6936         \tl_set:Nn \l__spinttent_spcirc_connector_arg_tl { _- }
6937         \tl_set:Nn \l__spinttent_spcirc_radio_arg_tl { _- }
6938     }

```

With `concept=true`, `\l__spinttent_spcirc_intent_arg_tl` is `_l__spinttent_spcirc_intent_tl` alone, with `prefix` placed after the leading `_` when given; `\l__spinttent_spcirc_connector_arg_tl` is `_l__spinttent_spcirc_connector_tl:prefix($pt)`; and `\l__spinttent_spcirc_radio_arg_tl` is the literal `_y-radio-`, or `_y-diámetro-` when `diameter` is `true`, spoken on the delimiter `,` before the reading of `\__spinttent_spcirc_radio_print:`.

```

6939     {
6940         \bool_if:NTF \l__spinttent_spcirc_concept_bool
6941         {
6942             \tl_set:Nn \l__spinttent_spcirc_intent_arg_tl
6943             {
6944                 -
6945                 \tl_if_empty:NF \l__spinttent_spcirc_prefix_tl

```

```

6946         { \tl_use:N \l__spinttent_spcirc_prefix_tl - }
6947         \l__spinttent_spcirc_intent_tl
6948     }
6949     \tl_set:Nc \l__spinttent_spcirc_connector_arg_tl
6950     { _ \l__spinttent_spcirc_connector_tl :prefix( $ pt ) }
6951     \bool_if:NTF \l__spinttent_spcirc_diameter_bool
6952     { \tl_set:Nn \l__spinttent_spcirc_radio_arg_tl { _y-diámetro- } }
6953     { \tl_set:Nn \l__spinttent_spcirc_radio_arg_tl { _y-radio- } }
6954 }

```

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.

```

6955     {
6956         \tl_if_empty:NTF \l__spinttent_spcirc_read_arg_tl
6957         {
6958             \tl_set:Nn \l__spinttent_spcirc_intent_arg_tl { _- }
6959             \tl_set:Nc \l__spinttent_spcirc_connector_arg_tl { _ ( $ pt ) }
6960             \tl_set:Nn \l__spinttent_spcirc_radio_arg_tl { _- }
6961         }
6962         {
6963             \tl_set:Nc \l__spinttent_spcirc_intent_arg_tl
6964             { _ \l__spinttent_spcirc_intent_tl }
6965             \tl_set:Nc \l__spinttent_spcirc_connector_arg_tl
6966             { _ \l__spinttent_spcirc_connector_tl :prefix( $ pt ) }
6967             \bool_if:NTF \l__spinttent_spcirc_diameter_bool
6968             { \tl_set:Nn \l__spinttent_spcirc_radio_arg_tl { _y-diámetro- } }
6969             { \tl_set:Nn \l__spinttent_spcirc_radio_arg_tl { _y-radio- } }
6970         }
6971     }
6972 }
6973 }

```

(End of definition for `\__spinttent_spcirc_build_concept:`.)

`\__spinttent_spcirc_exec:nn` Receives the `d()` argument in ‘`#2`’, which is an explicit empty group when omitted because `\spcirc` converts `-NoValue-` to `{ }` before calling `\__spinttent_spcirc_exec:nn`. When the group is empty, the function sets `\l__spinttent_spcirc_has_arg_bool` to `false` and does not call `\__spinttent_luafun_geo_circ_parse_and_set:n`. Otherwise it calls that function, reports `spcirc-invalid-arg` on failure and delegates to `\__spinttent_spcirc_mathml:`, which calls `\__spinttent_spcirc_build_concept:`.

Before processing the `{⟨argument⟩}`, it sets the optional `⟨keys⟩` passed in ‘`#1`’.

```

6974 \cs_new_protected:Nn \__spinttent_spcirc_exec:nn
6975 {
6976     \keys_set:nn { spinttent / spcirc } {#1}
6977     \tl_if_empty:nTF {#2}
6978     {
6979         \bool_set_false:N \l__spinttent_spcirc_has_arg_bool
6980     }
6981     {
6982         \bool_set_true:N \l__spinttent_spcirc_has_arg_bool
6983         \__spinttent_luafun_geo_circ_parse_and_set:n { \exp_not:n {#2} }
6984         \str_if_eq:VnT \l__spinttent_geo_luaset_error_str { true }
6985         {
6986             \msg_error:nne { spinttent } { spcirc-invalid-arg } { \exp_not:n {#2} }
6987         }
6988     }
6989     \__spinttent_spcirc_mathml:
6990 }

```

(End of definition for `\__spinttent_spcirc_exec:nn`.)

`\__spinttent_spcirc_mathml:` Calls `\__spinttent_spcirc_build_concept:` and wraps the symbol `\l__spinttent_spcirc_intent_arg_tl` in its own native `\MathMLintent`. This is the whole result when `\l__spinttent_spcirc_has_arg_bool` is `false`, that is, for `\spcirc` with no `(...)`.

Otherwise, the group `(center)`, held in `\l__spinttent_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__spinttent_spcirc_radio_arg_tl` and `\__spinttent_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`.

```

6991 \cs_new_protected:Nn \__spintent_spcirc_mathml:
6992 {
6993   \__spintent_invisible_sep:
6994   \__spintent_spcirc_build_concept:
6995   \MathMLintent { \__spintent_spcirc_intent_arg_tl }
6996   {
6997     \__spintent_inner_sep:
6998     \__spintent_luafun_mrow_intent_arg:nn
6999     { } { intent = ":silent" }
7000     { \__spintent_spcirc_sym: }
7001     \str_if_empty:NF \__spintent_spcirc_sub_index_str
7002     {
7003       \c_math_subscript_token
7004       {
7005         \__spintent_luafun_mrow_intent_arg:nn
7006         { } { intent = ":silent" }
7007         { \__spintent_spcirc_sub_index_str }
7008       }
7009     }
7010   }
7011   \bool_if:NT \__spintent_spcirc_has_arg_bool
7012   {
7013     \str_if_empty:NTF \__spintent_geo_luaset_circ_radio_type_str
7014     {
7015       \MathMLintent { \__spintent_spcirc_connector_arg_tl }
7016       {
7017         \__spintent_inner_sep:
7018         \__spintent_luafun_mrow_intent_arg:nn { } { intent = ":silent" } \lparen
7019         \__spintent_mathmlarg_point:nnV { pt } { spcirc }
7020         \__spintent_geo_luaset_points_clist
7021         \__spintent_luafun_mrow_intent_arg:nn { } { intent = ":silent" } \rparen
7022       }
7023     }
7024     {
7025       \MathMLintent { \__spintent_spcirc_connector_arg_tl }
7026       {
7027         \__spintent_inner_sep:
7028         \__spintent_luafun_mrow_intent_arg:nn { } { intent = ":silent" } \lparen
7029         \__spintent_mathmlarg_point:nnV { pt } { spcirc }
7030         \__spintent_geo_luaset_points_clist
7031       }
7032       \MathMLintent { \__spintent_spcirc_radio_arg_tl } { , }
7033       \bool_if:NTF \__spintent_spcirc_silent_bool
7034       {
7035         \MathMLintent { _- }
7036         {
7037           \__spintent_inner_sep:
7038           \__spintent_spcirc_radio_print:
7039         }
7040       }
7041       { \__spintent_spcirc_radio_print: }
7042       \MathMLintent { :silent } { \rparen }
7043     }
7044   }
7045 }

```

(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.

```

7046 \NewDocumentCommand \spcirc { 0{ } d() }

```

```

7047 {
7048   \mode_if_math:TF
7049   {
7050     \group_begin:
7051       \attribute \l__spinttent_scope_attr_int = 1
7052       \tl_if_novalue:nTF {#2}
7053         { \__spinttent_spcirc_exec:nn {#1} { } }
7054         { \__spinttent_spcirc_exec:nn {#1} {#2} }
7055     \group_end:
7056   }
7057   { \msg_error:nnn { spinttent } { need-math-mode } { \spcirc } }
7058 }

```

(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`

These variables store the value set by each corresponding key: `read-cmd`, `print-sym`, `read-sty`, `sub-index`, `prefix` and `read-arg`, respectively.

```

7059 \clist_map_inline:nn { Acirc , Pcirc }
7060 {
7061   \str_new:c { l__spinttent_sp #1 _mode_str }
7062   \str_new:c { l__spinttent_sp #1 _print_sym_str }
7063   \str_new:c { l__spinttent_sp #1 _read_sty_str }
7064   \str_new:c { l__spinttent_sp #1 _sub_index_str }
7065   \tl_new:c { l__spinttent_sp #1 _prefix_tl }
7066   \tl_new:c { l__spinttent_sp #1 _read_arg_tl }
7067 }

```

(End of definition for `l__spinttent_spAcirc_mode_str` and others.)

The variable `l__spinttent_spcircfig_luaset_comma_str` is set by `l__spinttent_luafun_spcircfig_comma:` and stores `true` when the argument has a comma.

```

7068 \str_new:N l__spinttent_spcircfig_luaset_comma_str

```

(End of definition for `l__spinttent_spcircfig_luaset_comma_str`.)

The variable `l__spinttent_spcircfig_short_bool` stores whether the argument was resolved as the short number or letter form, with `l__spinttent_luafun_geo_afig_parse_and_set:n` or `l__spinttent_luafun_geo_pfig_parse_and_set:n`, and not as a circle.

```

7069 \bool_new:N l__spinttent_spcircfig_short_bool

```

(End of definition for `l__spinttent_spcircfig_short_bool`.)

The variable `l__spinttent_spcircfig_op_tl` stores «área» or «perímetro». The variable `l__spinttent_spcircfig_del_word_tl`, set by `l__spinttent_spcircfig_del_word:n`, stores the word that follows the operator. The variable `l__spinttent_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 `l__spinttent_mathmlarg_point:nnn`. The variable `l__spinttent_spcircfig_intent_arg_tl` stores the intent being built.

```

7070 \tl_new:N l__spinttent_spcircfig_op_tl
7071 \tl_new:N l__spinttent_spcircfig_del_word_tl
7072 \tl_new:N l__spinttent_spcircfig_intent_tl
7073 \tl_new:N l__spinttent_spcircfig_intent_arg_tl

```

(End of definition for `l__spinttent_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.

```

7074 \clist_map_inline:nn { Acirc , Pcirc }
7075 {
7076   \msg_new:nnnn { spintent } { sp #1 -invalid-arg }
7077   {
7078     Invalid~argument~'##2'~for~'\tl_trim_spaces:n {##1}'~\msg_line_context:.
7079   }
7080   {
7081     See~the~manual~for~the~accepted~argument~forms.
7082   }

```

(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
 unknown

We define the shared $\langle$ keys $\rangle$ read-cmd, prefix, print-sym, read-sty, concept, sub-index, read-sub, read-arg and silent.

```

7083 \keys_define:nn { spintent / sp #1 }
7084 {
7085   read-cmd .choices:nn = { school , mathcat }
7086   {
7087     \int_case:nn { \l_keys_choice_int }
7088     {
7089       {1} { \str_set:cn { l__spintent_sp #1 _mode_str } { school } }
7090       {2} { \str_set:cn { l__spintent_sp #1 _mode_str } { mathcat } }
7091     }
7092   },
7093   read-cmd .initial:n = school,
7094   read-cmd .value_required:n = true,
7095   read-cmd / unknown .code:n =
7096     \msg_error:nneee { spintent } { unknown-choice }
7097     { read-cmd } { school,~mathcat } { \exp_not:n {##1} },
7098   prefix .code:n =
7099     { \__spintent_sanitizе_affix:cn { l__spintent_sp #1 _prefix_tl } {##1} },
7100   prefix .initial:n = ,
7101   prefix .value_required:n = true,
7102   print-sym .choices:nn = { usual , odot }
7103   {
7104     \int_case:nn { \l_keys_choice_int }
7105     {
7106       {1} { \str_set:cn { l__spintent_sp #1 _print_sym_str } { usual } }
7107       {2} { \str_set:cn { l__spintent_sp #1 _print_sym_str } { odot } }
7108     }
7109   },
7110   print-sym .initial:n = odot,
7111   print-sym .value_required:n = true,
7112   print-sym / unknown .code:n =
7113     \msg_error:nneee { spintent } { unknown-choice }
7114     { print-sym } { usual,~odot } { \exp_not:n {##1} },
7115   read-sty .choices:nn = { circun , circulo }
7116   {
7117     \int_case:nn { \l_keys_choice_int }
7118     {
7119       {1} { \str_set:cn { l__spintent_sp #1 _read_sty_str } { circun } }
7120       {2} { \str_set:cn { l__spintent_sp #1 _read_sty_str } { circulo } }
7121     }
7122   },
7123   read-sty .initial:n = circun,
7124   read-sty .value_required:n = true,
7125   read-sty / unknown .code:n =
7126     \msg_error:nneee { spintent } { unknown-choice }
7127     { read-sty } { circun,~circulo } { \exp_not:n {##1} },
7128   concept .bool_set:c = { l__spintent_sp #1 _concept_bool },
7129   concept .initial:n = true,
7130   concept .value_required:n = true,
7131   sub-index .code:n =
7132     { \str_set:cn { l__spintent_sp #1 _sub_index_str } {##1} },

```

```

7133     sub-index .initial:n = ,
7134     sub-index .value_required:n = true,
7135     read-sub .bool_set:c = { l__spinttent_sp #1 _read_sub_bool },
7136     read-sub .initial:n = false,
7137     read-sub .value_required:n = true,
7138     read-arg .code:n =
7139       { l__spinttent_sanitizate_affix:cn { l__spinttent_sp #1 _read_arg_tl } {##1} },
7140     read-arg .initial:n = ,
7141     read-arg .value_required:n = true,
7142     silent .bool_set:c = { l__spinttent_sp #1 _silent_bool },
7143     silent .initial:n = false,
7144     silent .value_required:n = true,
7145     unknown .code:n =
7146       {
7147         \str_set:Nn \l__spinttent_command_name_str { sp #1 }
7148         \__spinttent_unknown_keys_cmd:n {##1}
7149       },
7150   }
7151 }

```

(End of definition for `read-cmd` and others.)

15.55.4 Internal functions for `\spAcirc` and `\spPcirc`

`\__spinttent_spcircfig_sym:n` Typesets `\bigodot` or the plain letter C, according to `print-sym`. Receives the short command name in ‘#1’.

```

7152 \cs_new_protected:Nn \__spinttent_spcircfig_sym:n
7153 {
7154   \str_if_eq:vnTF { l__spinttent_sp #1 _print_sym_str } { odot }
7155   { \bigodot }
7156   { C }
7157 }

```

(End of definition for `\__spinttent_spcircfig_sym:n`.)

`\__spinttent_spcircfig_word_and_article:n` Sets `\l__spinttent_spcirc_word_tl` to `read-arg` or `circunferencia/círculo` from `read-sty`, and `\l__spinttent_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’.

```

7158 \cs_new_protected:Nn \__spinttent_spcircfig_word_and_article:n
7159 {
7160   \str_if_eq:vnTF { l__spinttent_sp #1 _read_sty_str } { circun }
7161   { \str_set:Nn \l__spinttent_spcirc_articulo_str { la } }
7162   { \str_set:Nn \l__spinttent_spcirc_articulo_str { el } }
7163   \tl_if_empty:cTF { l__spinttent_sp #1 _read_arg_tl }
7164   {
7165     \str_if_eq:vnTF { l__spinttent_sp #1 _read_sty_str } { circun }
7166     { \tl_set:Nn \l__spinttent_spcirc_word_tl { circunferencia } }
7167     { \tl_set:Nn \l__spinttent_spcirc_word_tl { círculo } }
7168   }
7169   {
7170     \tl_set:Ne \l__spinttent_spcirc_word_tl
7171     {
7172       \tl_use:c { l__spinttent_sp #1 _read_arg_tl }
7173     }
7174   }
7175 }

```

(End of definition for `\__spinttent_spcircfig_word_and_article:n`.)

`\__spinttent_spcircfig_del_word:n` Sets `\l__spinttent_spcircfig_del_word_tl` to the word of `\__spinttent_spcircfig_word_and_article:n`, prefixed with `del-` or `de-la-`. Receives the short command name in ‘#1’.

```

7176 \cs_new_protected:Nn \__spinttent_spcircfig_del_word:n
7177 {
7178   \__spinttent_spcircfig_word_and_article:n {#1}
7179   \str_if_eq:VnTF \l__spinttent_spcirc_articulo_str { el }
7180   {
7181     \tl_set:Ne \l__spinttent_spcircfig_del_word_tl
7182     { del- \l__spinttent_spcirc_word_tl }
7183   }
7184   {
7185     \tl_set:Ne \l__spinttent_spcircfig_del_word_tl

```

```

7186         { de-la- \l__spintent_spcirc_word_tl }
7187     }
7188 }

```

(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'`.

```

7189 \cs_new_protected:Nn \__spintent_spcircfig_build_short:n
7190 {
7191     \bool_if:cTF { l__spintent_sp #1 _silent_bool }
7192     {
7193         \tl_set:Nn \l__spintent_spcircfig_intent_arg_tl { _- }
7194     }
7195     {
7196         \bool_if:cTF { l__spintent_sp #1 _concept_bool }
7197         {
7198             \__spintent_spcircfig_del_word:n {#1}
7199             \tl_set:Ne \l__spintent_spcircfig_intent_arg_tl
7200             {
7201                 -
7202                 \tl_if_empty:cF { l__spintent_sp #1 _prefix_tl }
7203                 { \tl_use:c { l__spintent_sp #1 _prefix_tl } - }
7204                 \l__spintent_spcircfig_op_tl - \l__spintent_spcircfig_del_word_tl -
7205                 \bool_if:cT { l__spintent_sp #1 _read_sub_bool } { sub- }
7206                 \l__spintent_geo_luaset_sub_spoken_str
7207             }
7208         }
7209         {
7210             \tl_set:Nn \l__spintent_spcircfig_intent_arg_tl { _- }
7211         }
7212     }
7213 }

```

(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'`.

```

7214 \cs_new_protected:Nn \__spintent_spcircfig_finish_intent:nn
7215 {
7216     \str_if_empty:cF { l__spintent_sp #1 _sub_index_str }
7217     {
7218         \tl_put_right:Ne \l__spintent_spcircfig_intent_tl
7219         {
7220             -
7221             \bool_if:cT { l__spintent_sp #1 _read_sub_bool } { sub- }
7222             \str_use:c { l__spintent_sp #1 _sub_index_str }
7223         }
7224     }
7225     \tl_if_empty:nF {#2}
7226     {
7227         \tl_put_right:Nn \l__spintent_spcircfig_intent_tl { -con-centro-en }
7228     }
7229 }

```

(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_word_tl` 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'`.

```

7230 \cs_new_protected:Nn \__spintent_spcircfig_build_concept:nn

```

```

7231 {
7232   \bool_if:cTF { l__spintent_sp #1 _silent_bool }
7233   {
7234     \tl_set:Nn \l__spintent_spcircfig_intent_arg_tl { _- }
7235   }
7236   {
7237     \bool_if:cTF { l__spintent_sp #1 _concept_bool }
7238     {
7239       \__spintent_spcircfig_del_word:n {#1}
7240       \tl_set:Ne \l__spintent_spcircfig_intent_tl
7241       {
7242         \l__spintent_spcircfig_op_tl - \l__spintent_spcircfig_del_word_tl
7243       }
7244       \__spintent_spcircfig_finish_intent:nn {#1} {#2}
7245       \tl_set:Ne \l__spintent_spcircfig_intent_arg_tl
7246       {
7247         -
7248         \tl_if_empty:cF { l__spintent_sp #1 _prefix_tl }
7249         {
7250           \tl_use:c { l__spintent_sp #1 _prefix_tl } -
7251         }
7252         \l__spintent_spcircfig_intent_tl
7253         \tl_if_empty:nF {#2} { :prefix ( $ pt ) }
7254       }
7255     }
7256     {
7257       \tl_if_empty:cTF { l__spintent_sp #1 _read_arg_tl }
7258       {
7259         \tl_if_empty:nTF {#2}
7260         { \tl_set:Nn \l__spintent_spcircfig_intent_arg_tl { _- } }
7261         { \tl_set:Ne \l__spintent_spcircfig_intent_arg_tl { _ ( $ pt ) } }
7262       }
7263       {
7264         \tl_set:Ne \l__spintent_spcircfig_intent_tl
7265         { \tl_use:c { l__spintent_sp #1 _read_arg_tl } }
7266         \__spintent_spcircfig_finish_intent:nn {#1} {#2}
7267         \tl_set:Ne \l__spintent_spcircfig_intent_arg_tl
7268         {
7269           -
7270           \l__spintent_spcircfig_intent_tl
7271           \tl_if_empty:nF {#2} { :prefix ( $ pt ) }
7272         }
7273       }
7274     }
7275   }
7276 }

```

(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 $\langle argument \rangle$, it sets the optional $\langle keys \rangle$ passed in ‘#2’.

```

7277 \cs_new_protected:Nn \__spintent_spcircfig_exec:nnn
7278 {
7279   \keys_set:nn { spintent / sp #1 } {#2}
7280   \str_if_eq:eeTF {#1} { Acirc }
7281   { \tl_set:Nn \l__spintent_spcircfig_op_tl { área } }
7282   { \tl_set:Nn \l__spintent_spcircfig_op_tl { perímetro } }
7283   \tl_if_empty:nTF {#3}
7284   {
7285     \bool_set_false:N \l__spintent_spcircfig_short_bool

```

```

7286   }
7287   {
7288     \bool_set_false:N \l__spintent_spcircfig_short_bool
7289     \__spintent_luafun_spcircfig_comma:n { \exp_not:n {#3} }
7290     \str_if_eq:VnTF \l__spintent_spcircfig_luaset_comma_str { true }
7291     {
7292       \msg_error:nnee { spintent } { sp #1 -invalid-arg }
7293       { \c_backslash_str sp #1 } { \exp_not:n {#3} }
7294     }
7295     {
7296       \__spintent_luafun_geo_circ_parse_and_set:n { \exp_not:n {#3} }
7297       \str_if_eq:VnT \l__spintent_geo_luaset_error_str { true }
7298       {
7299         \str_if_eq:eeTF {#1} { Acirc }
7300         { \__spintent_luafun_geo_afig_parse_and_set:n { \exp_not:n {#3} } }
7301         { \__spintent_luafun_geo_pfig_parse_and_set:n { \exp_not:n {#3} } }
7302         \str_if_eq:VnF \l__spintent_geo_luaset_error_str { true }
7303         {
7304           \str_if_empty:NTF \l__spintent_geo_luaset_letra_word_str
7305           { \bool_set_true:N \l__spintent_spcircfig_short_bool }
7306           { \str_set:Nn \l__spintent_geo_luaset_error_str { true } }
7307         }
7308       }
7309       \bool_if:NF \l__spintent_spcircfig_short_bool
7310       {
7311         \str_if_eq:VnT \l__spintent_geo_luaset_error_str { true }
7312         {
7313           \msg_error:nnee { spintent } { sp #1 -invalid-arg }
7314           { \c_backslash_str sp #1 } { \exp_not:n {#3} }
7315         }
7316       }
7317     }
7318   }
7319   \bool_if:NTF \l__spintent_spcircfig_short_bool
7320   { \__spintent_spcircfig_build_short:n {#1} }
7321   { \__spintent_spcircfig_build_concept:nn {#1} {#3} }
7322   \__spintent_spcircfig_mathml:nn {#1} {#3}
7323 }

```

(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`’.

```

7324 \cs_new_protected:Nn \__spintent_spcircfig_mathml:nn
7325 {
7326   \__spintent_invisible_sep:
7327   \MathMLintent { \l__spintent_spcircfig_intent_arg_tl }
7328   {
7329     \__spintent_inner_sep:
7330     \__spintent_luafun_mrow_intent_arg:nn
7331     { } { intent = ":silent" }
7332     {
7333       \str_if_eq:eeTF {#1} { Acirc } { \__spintent_mathcal:n { A } }
7334       { \__spintent_mathcal:n { P } }
7335     }
7336     \bool_if:NTF \l__spintent_spcircfig_short_bool
7337     {
7338       \c_math_subscript_token
7339       {
7340         \__spintent_spcircfig_sym:n {#1}
7341         \c_math_subscript_token { \l__spintent_geo_luaset_print_tl }
7342       }
7343     }
7344     {
7345       \c_math_subscript_token
7346       {
7347         \MathMLintent { :silent } { \__spintent_spcircfig_sym:n {#1} }

```

```

7348         \str_if_empty:cF { \l__spinttent_sp #1 _sub_index_str }
7349         {
7350             \c_math_subscript_token
7351             {
7352                 \MathMLintent { :silent }
7353                 { \str_use:c { \l__spinttent_sp #1 _sub_index_str } }
7354             }
7355         }
7356         \tl_if_empty:nF {#2}
7357         {
7358             \MathMLintent { :silent } { \lparen }
7359             \__spinttent_mathmlarg_point:nnV { pt } { sp #1 }
7360             \l__spinttent_geo_luaset_points_clist
7361             \MathMLintent { :silent } { \rparen }
7362         }
7363     }
7364 }
7365 }
7366 }

```

(End of definition for `\__spinttent_spcircfig_mathml:nn`.)

`\spAcirc` Both public commands `\spAcirc` and `\spPcirc` enforce math mode, open a local group, and delegate execution to `\__spinttent_spcircfig_exec:nnn`.

`\spPcirc`

```

7367 \NewDocumentCommand \spAcirc { 0{} d() }
7368 {
7369     \mode_if_math:TF
7370     {
7371         \group_begin:
7372         \attribute \l__spinttent_scope_attr_int = 1
7373         \tl_if_novalue:nTF {#2}
7374         { \__spinttent_spcircfig_exec:nnn { Acirc } {#1} { } }
7375         { \__spinttent_spcircfig_exec:nnn { Acirc } {#1} {#2} }
7376         \group_end:
7377     }
7378     { \msg_error:nnn { spinttent } { need-math-mode } { \spAcirc } }
7379 }
7380 \NewDocumentCommand \spPcirc { 0{} d() }
7381 {
7382     \mode_if_math:TF
7383     {
7384         \group_begin:
7385         \attribute \l__spinttent_scope_attr_int = 1
7386         \tl_if_novalue:nTF {#2}
7387         { \__spinttent_spcircfig_exec:nnn { Pcirc } {#1} { } }
7388         { \__spinttent_spcircfig_exec:nnn { Pcirc } {#1} {#2} }
7389         \group_end:
7390     }
7391     { \msg_error:nnn { spinttent } { need-math-mode } { \spPcirc } }
7392 }

```

(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__spinttent_spVol_mode_str` The variable `\l__spinttent_spVol_mode_str` stores the value of the key `read-cmd`, `school` or `mathcat`. The variable `\l__spinttent_spVol_sub_index_str` stores the raw text of `sub-index` and `\l__spinttent_spVol_read_arg_tl` the sanitized text of `read-arg`.

```

7393 \str_new:N \l__spinttent_spVol_mode_str
7394 \str_new:N \l__spinttent_spVol_sub_index_str
7395 \tl_new:N \l__spinttent_spVol_read_arg_tl

```

(End of definition for `\l__spinttent_spVol_mode_str`, `\l__spinttent_spVol_sub_index_str`, and `\l__spinttent_spVol_read_arg_tl`.)

`\l__spinttent_spVol_word_tl`
`\l__spinttent_spVol_suffix_tl`
`\l__spinttent_spVol_native_bool`

The variable `\l__spinttent_spVol_word_tl` stores the word to voice, «*volumen*», and `\l__spinttent_spVol_suffix_tl` the text attached after it. The variable `\l__spinttent_spVol_native_bool` records whether that text is combined natively with the subscript, silenced, or left for MathCAT to read on its own.

```
7396 \tl_new:N \l__spinttent_spVol_word_tl
7397 \tl_new:N \l__spinttent_spVol_suffix_tl
7398 \bool_new:N \l__spinttent_spVol_native_bool
```

(End of definition for `\l__spinttent_spVol_word_tl`, `\l__spinttent_spVol_suffix_tl`, and `\l__spinttent_spVol_native_bool`.)

`\l__spinttent_spVol_read_arg_full_tl`

The variable `\l__spinttent_spVol_read_arg_full_tl` stores the text to voice when `read-arg` is given.

```
7399 \tl_new:N \l__spinttent_spVol_read_arg_full_tl
```

(End of definition for `\l__spinttent_spVol_read_arg_full_tl`.)

15.56.2 Definition of keys for `\spVol`

We define the *(keys)* `read-cmd`, `sub-index`, `read-sub`, `read-arg`, `concept` and `silent`.

```
7400 \keys_define:nn { spinttent / spVol }
7401 {
7402   read-cmd .choices:nn = { school , mathcat }
7403   {
7404     \int_case:nn { \l_keys_choice_int }
7405     {
7406       {1} { \str_set:Nn \l__spinttent_spVol_mode_str { school } }
7407       {2} { \str_set:Nn \l__spinttent_spVol_mode_str { mathcat } }
7408     }
7409   },
7410   read-cmd .initial:n = school,
7411   read-cmd .value_required:n = true,
7412   read-cmd / unknown .code:n =
7413     \msg_error:nneee { spinttent } { unknown-choice }
7414     { read-cmd } { school,~mathcat } { \exp_not:n {#1} },
7415   sub-index .code:n =
7416     { \str_set:Nn \l__spinttent_spVol_sub_index_str {#1} },
7417   sub-index .initial:n = ,
7418   sub-index .value_required:n = true,
7419   read-sub .bool_set:N = \l__spinttent_spVol_read_sub_bool,
7420   read-sub .initial:n = false,
7421   read-sub .value_required:n = true,
7422   read-arg .code:n =
7423     { \__spinttent_sanitizate_affix:Nn \l__spinttent_spVol_read_arg_tl {#1} },
7424   read-arg .initial:n = ,
7425   read-arg .value_required:n = true,
7426   concept .bool_set:N = \l__spinttent_spVol_concept_bool,
7427   concept .initial:n = true,
7428   concept .value_required:n = true,
7429   silent .bool_set:N = \l__spinttent_spVol_silent_bool,
7430   silent .initial:n = false,
7431   silent .value_required:n = true,
7432   unknown .code:n =
7433     {
7434       \str_set:Nn \l__spinttent_command_name_str { spVol }
7435       \__spinttent_unknown_keys_cmd:n {#1}
7436     },
7437 }
```

(End of definition for `read-cmd` and others.)

15.56.3 Internal functions for `\spVol`

`\__spinttent_spVol_mathml:`

The function `\__spinttent_spVol_mathml:` calls `\__spinttent_spVol_silent:` when `silent` is `true`, or `\__spinttent_spVol_verbal:` otherwise.

```
7438 \cs_new_protected:Nn \__spinttent_spVol_mathml:
7439 {
7440   \__spinttent_invisible_sep:
7441   \bool_if:NTF \l__spinttent_spVol_silent_bool
7442     { \__spinttent_spVol_silent: }
7443     { \__spinttent_spVol_verbal: }
7444 }
```

(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.

```

7445 \cs_new_protected:Nn \__spintent_spVol_silent:
7446 {
7447   \str_if_empty:NTF \l__spintent_spVol_sub_index_str
7448   {
7449     \__spintent_luafun_mrow_intent_arg:nn
7450     { } { intent = "- " }
7451     { \mathcal{V} }
7452   }
7453   {
7454     \MathMLintent { _- }
7455     {
7456       \__spintent_inner_sep:
7457       \mathcal{V}
7458       \c_math_subscript_token
7459       {
7460         \MathMLintent { :silent }
7461         { \l__spintent_spVol_sub_index_str }
7462       }
7463     }
7464   }
7465 }

```

(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:`.

```

7466 \cs_new_protected:Nn \__spintent_spVol_verbal:
7467 {
7468   \tl_if_empty:NTF \l__spintent_spVol_read_arg_tl
7469   {
7470     \bool_if:NTF \l__spintent_spVol_concept_bool
7471     {
7472       \tl_set:Nn \l__spintent_spVol_word_tl { volumen }
7473       \str_if_eq:VnTF \l__spintent_spVol_mode_str { school }
7474       { \bool_set_true:N \l__spintent_spVol_native_bool }
7475       { \bool_set_false:N \l__spintent_spVol_native_bool }
7476       \tl_set:Ne \l__spintent_spVol_suffix_tl
7477       {
7478         \bool_if:NT \l__spintent_spVol_read_sub_bool { sub- }
7479         \l__spintent_spVol_sub_index_str
7480       }
7481       \__spintent_spVol_attach_subscript:
7482     }
7483     { \__spintent_spVol_symbol_print: }
7484   }
7485   { \__spintent_spVol_read_arg_print: }
7486 }

```

(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.

```

7487 \cs_new_protected:Nn \__spintent_spVol_symbol_print:
7488 {
7489   \str_if_empty:NTF \l__spintent_spVol_sub_index_str
7490   {
7491     \str_if_eq:VnTF \l__spintent_spVol_mode_str { school }
7492     {

```

```

7493     \__spintent_luafun_mrow_intent_arg:nn
7494     { } { intent = "_V-" }
7495     { \mathcal{V} }
7496   }
7497   { \mathcal{V} }
7498 }
7499 {
7500   \str_if_eq:VnTF \l__spintent_spVol_mode_str { school }
7501   {
7502     \MathMLintent
7503     {
7504       _V-
7505       \bool_if:NT \l__spintent_spVol_read_sub_bool { sub- }
7506       \l__spintent_spVol_sub_index_str
7507     }
7508     {
7509       \__spintent_inner_sep:
7510       \mathcal{V}
7511       \c_math_subscript_token
7512       {
7513         \MathMLintent { :silent }
7514         { \l__spintent_spVol_sub_index_str }
7515       }
7516     }
7517   }
7518   {
7519     \mathcal{V}
7520     \c_math_subscript_token
7521     { \l__spintent_spVol_sub_index_str }
7522   }
7523 }
7524 }

```

(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`.

```

7525 \cs_new_protected:Nn \__spintent_spVol_attach_subscript:
7526 {
7527   \str_if_empty:NTF \l__spintent_spVol_sub_index_str
7528   {
7529     \__spintent_luafun_mrow_intent_arg:nn
7530     { } { intent = "_ \l__spintent_spVol_word_tl :nofix" }
7531     { \mathcal{V} }
7532   }
7533   {
7534     \bool_if:NTF \l__spintent_spVol_native_bool
7535     {
7536       \MathMLintent
7537       { _ \l__spintent_spVol_word_tl - \l__spintent_spVol_suffix_tl :nofix }
7538       {
7539         \__spintent_inner_sep:
7540         \mathcal{V}
7541         \c_math_subscript_token
7542         {
7543           \MathMLintent { :silent }
7544           { \l__spintent_spVol_sub_index_str }
7545         }
7546       }
7547     }
7548     {
7549       \__spintent_luafun_mrow_intent_arg:nn
7550       { } { intent = "_ \l__spintent_spVol_word_tl :nofix" }
7551       { \mathcal{V} }
7552       \c_math_subscript_token
7553       { \l__spintent_spVol_sub_index_str }
7554     }
7555   }
7556 }

```

```

7555     }
7556 }

```

(End of definition for `\__spinttent_spVol_attach_subscript:`)

`\__spinttent_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`, `\__spinttent_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`.

```

7557 \cs_new_protected:Nn \__spinttent_spVol_read_arg_print:
7558 {
7559   \bool_if:NTF \__spinttent_spVol_concept_bool
7560   {
7561     \tl_set:Nx \__spinttent_spVol_read_arg_full_tl
7562     { volumen- \tl_use:N \__spinttent_spVol_read_arg_tl }
7563   }
7564   {
7565     \tl_set_eq:NN \__spinttent_spVol_read_arg_full_tl \__spinttent_spVol_read_arg_tl
7566   }
7567   \str_if_empty:NTF \__spinttent_spVol_sub_index_str
7568   {
7569     \__spinttent_luafun_mrow_intent_arg:nn
7570     { } { intent = "_ \__spinttent_spVol_read_arg_full_tl :nofix" }
7571     { \mathcal{V} }
7572   }
7573   {
7574     \MathMLintent
7575     { _ \__spinttent_spVol_read_arg_full_tl :nofix }
7576     {
7577       \__spinttent_inner_sep:
7578       \mathcal{V}
7579       \c_math_subscript_token
7580       {
7581         \MathMLintent { :silent }
7582         { \__spinttent_spVol_sub_index_str }
7583       }
7584     }
7585   }
7586 }

```

(End of definition for `\__spinttent_spVol_read_arg_print:`)

`\spVol` The public command `\spVol` enforces *math mode*, opens a local group, and delegates execution to `\__spinttent_spVol_mathml:`.

```

7587 \NewDocumentCommand \spVol { 0{} }
7588 {
7589   \mode_if_math:TF
7590   {
7591     \group_begin:
7592     \attribute \__spinttent_scope_attr_int = 1
7593     \keys_set:nn { spinttent / spVol } {#1}
7594     \__spinttent_spVol_mathml:
7595     \group_end:
7596   }
7597   { \msg_error:nnn { spinttent } { need-math-mode } { \spVol } }
7598 }

```

(End of definition for `\spVol`. This function is documented on page 39.)

15.57 The command `\spProm`

The user command `\spProm` typesets its $\langle \langle \textit{argument} \rangle \rangle$ 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`

`\__spinttent_spProm_read_arg_tl`
`\__spinttent_spProm_intent_tl`

The variable `\__spinttent_spProm_read_arg_tl` holds the normalized text of the `read-arg` key, and `\__spinttent_spProm_intent_tl` holds the name of the intent used by `\__spinttent_spProm_exec:nn`.

```

7599 \tl_new:N \__spinttent_spProm_read_arg_tl
7600 \tl_new:N \__spinttent_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` We define the *(keys)* `silent` and `read-arg`.

`read-arg`

`unknown`

```

7601 \keys_define:nn { spintent / spProm }
7602 {
7603   silent .bool_set:N = \l__spintent_spProm_silent_bool,
7604   silent .initial:n = false,
7605   silent .value_required:n = true,
7606   read-arg .code:n =
7607     { \l__spintent_sanitizize_affix:Nn \l__spintent_spProm_read_arg_tl {#1} },
7608   read-arg .initial:n = ,
7609   read-arg .value_required:n = true,
7610   unknown .code:n =
7611     {
7612       \str_set:Nn \l__spintent_command_name_str { spProm }
7613       \l__spintent_unknown_keys_cmd:n {#1}
7614     },
7615 }
```

(End of definition for `silent`, `read-arg`, and `unknown`.)

15.57.3 Internal functions for `\spProm`

`\l__spintent_spProm_exec:nn`

The function `\l__spintent_spProm_exec:nn` receives the optional *(keys)* in ‘#1’ and the *{(argument)}* in ‘#2’. It sets the *(keys)* and calls `\l__spintent_spProm_mathml:n`.

```

7616 \cs_new_protected:Nn \l__spintent_spProm_exec:nn
7617 {
7618   \keys_set:nn { spintent / spProm } {#1}
7619   \l__spintent_spProm_mathml:n {#2}
7620 }
```

(End of definition for `\l__spintent_spProm_exec:nn`.)

`\l__spintent_spProm_mathml:n`

The function `\l__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 `\l__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.

```

7621 \cs_new_protected:Nn \l__spintent_spProm_mathml:n
7622 {
7623   \l__spintent_invisible_sep:
7624   \bool_if:NTF \l__spintent_spProm_silent_bool
7625   {
7626     \MathMLintent { _- }
7627     {
7628       \l__spintent_inner_sep:
7629       \bar {#1}
7630     }
7631   }
7632   {
7633     \tl_if_empty:NTF \l__spintent_spProm_read_arg_tl
7634     { \tl_set:Nn \l__spintent_spProm_intent_tl { promedio-de } }
7635     {
7636       \tl_set_eq:NN \l__spintent_spProm_intent_tl \l__spintent_spProm_read_arg_tl
7637     }
7638     \MathMLintent { _ \l__spintent_spProm_intent_tl :prefix($a) }
7639     {
7640       \l__spintent_inner_sep:
7641       \bar { \MathMLarg { a } { \l__spintent_inner_sep: #1 } }
7642     }
7643   }
7644 }
```

(End of definition for `\l__spintent_spProm_mathml:n`.)

`\spProm` The public command `\spProm` enforces *math mode*, opens a local group, and delegates execution to `\__spintent_spProm_exec:nn`.

```

7645 \NewDocumentCommand \spProm { O{} m }
7646 {
7647   \mode_if_math:TF
7648   {
7649     \group_begin:
7650     \attribute \__spintent_scope_attr_int = 1
7651     \__spintent_spProm_exec:nn {#1} {#2}
7652     \group_end:
7653   }
7654   { \msg_error:nnn { spintent } { need-math-mode } { \spProm } }
7655 }

```

(End of definition for `\spProm`. This function is documented on page 43.)

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

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.

```

7656 \clist_map_inline:nn
7657 {
7658   spmedia , spmediana , spmoda , spdesv , spvarz , sprango , spriq ,
7659   spmin , spmax
7660 }
7661 {
7662   \str_new:c { \__spintent_ #1 _sub_index_str }
7663   \tl_new:c { \__spintent_ #1 _read_arg_tl }
7664 }

```

(End of definition for `\__spintent_spmedia_sub_index_str` and others.)

The variable `\__spintent_stat_intent_tl` holds the name of the intent that `\__spintent_stat_intent:nnnn` builds, until `\__spintent_stat_mathml:nnnn` uses it.

```

7665 \tl_new:N \__spintent_stat_intent_tl

```

(End of definition for `\__spintent_stat_intent_tl`.)

15.58.2 Definition of keys for the stat family

We define the *keys* `silent`, `sub-index`, `read-sub` and `read-arg` for the seven commands `\spmedia`, `\spmediana`, `\spmoda`, `\spdesv`, `\spvarz`, `\sprango` and `\spriq`.

```

7666 \clist_map_inline:nn
7667 {
7668   spmedia , spmediana , spmoda , spdesv , spvarz , sprango , spriq
7669 }
7670 {
7671   \keys_define:nn { spintent / #1 }
7672   {
7673     silent .bool_set:c = { \__spintent_ #1 _silent_bool },
7674     silent .initial:n = false,
7675     silent .value_required:n = true,
7676     sub-index .code:n =
7677       { \str_set:cn { \__spintent_ #1 _sub_index_str } {##1} },
7678     sub-index .initial:n = ,
7679     sub-index .value_required:n = true,
7680     read-sub .bool_set:c = { \__spintent_ #1 _read_sub_bool },
7681     read-sub .initial:n = false,
7682     read-sub .value_required:n = true,

```

```

7683     read-arg .code:n =
7684     { \__spintent_sanitizetize_affix:cn { l__spintent_ #1 _read_arg_tl } {##1} },
7685     read-arg .initial:n = ,
7686     read-arg .value_required:n = true,
7687     unknown .code:n =
7688     {
7689         \str_set:Nn \__spintent_command_name_str {#1}
7690         \__spintent_unknown_keys_cmd:n {##1}
7691     },
7692 }
7693 }

```

(End of definition for *silent* and others.)

`sigma` The key `sigma` exists only for `\spdesv` and `\spvarz`, and it starts in `false`.

```

7694 \clist_map_inline:nn { spdesv , spvarz }
7695 {
7696     \keys_define:nn { spintent / #1 }
7697     {
7698         sigma .bool_set:c = { l__spintent_ #1 _sigma_bool },
7699         sigma .initial:n = false,
7700         sigma .value_required:n = true,
7701         read-sub .initial:n = true,
7702     }
7703 }

```

- 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` Since the symbol of `\spmin` and `\spmax` already carries a subscript, we define for them only the *(keys)* `silent` and `read-arg`.

```

7704 \clist_map_inline:nn { spmin , spmax }
7705 {
7706     \keys_define:nn { spintent / #1 }
7707     {
7708         silent .bool_set:c = { l__spintent_ #1 _silent_bool },
7709         silent .initial:n = false,
7710         silent .value_required:n = true,
7711         read-arg .code:n =
7712         { \__spintent_sanitizetize_affix:cn { l__spintent_ #1 _read_arg_tl } {##1} },
7713         read-arg .initial:n = ,
7714         read-arg .value_required:n = true,
7715         unknown .code:n =
7716         {
7717             \str_set:Nn \__spintent_command_name_str {#1}
7718             \__spintent_unknown_keys_cmd:n {##1}
7719         },
7720     }
7721 }

```

(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.

```

7722 \cs_new_protected:Nn \__spintent_stat_typeset:nnn
7723 {
7724     #2
7725     \str_if_empty:cF { l__spintent_ #1 _sub_index_str }
7726     {
7727         \c_math_subscript_token
7728         {
7729             \MathMLintent { :silent }
7730             { \str_use:c { l__spintent_ #1 _sub_index_str } }

```

```

7731     }
7732   }
7733   \tl_if_blank:nF {#3}
7734   { \c_math_superscript_token {#3} }
7735 }

```

(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⟩, 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.

```

7736 \cs_new_protected:Nn \__spintent_stat_intent:nnnn
7737 {
7738   \tl_set:Nc \l__spintent_stat_intent_tl
7739   {
7740     \bool_if:cTF { \l__spintent_ #1 _silent_bool }
7741     { - }
7742     {
7743       \tl_if_empty:cTF { \l__spintent_ #1 _read_arg_tl }
7744       {
7745         - #2
7746         \tl_if_blank:nF {#3} { - #3 }
7747         \str_if_empty:cTF { \l__spintent_ #1 _sub_index_str }
7748         {
7749           \tl_if_blank:nF {#4} { - #4 }
7750           :nofix
7751         }
7752         {
7753           -
7754           \bool_if:cTF { \l__spintent_ #1 _read_sub_bool } { sub } { de }
7755           - \str_use:c { \l__spintent_ #1 _sub_index_str }
7756           \tl_if_blank:nF {#4} { - #4 }
7757         }
7758       }
7759       { - \tl_use:c { \l__spintent_ #1 _read_arg_tl } :nofix }
7760     }
7761   }
7762 }
7763 \cs_new_protected:Nn \__spintent_stat_intent:nnn
7764 {
7765   \__spintent_stat_intent:nnnn {#1} {#2} {#3} { }
7766 }
7767 \cs_new_protected:Nn \__spintent_stat_intent:nn
7768 {
7769   \__spintent_stat_intent:nnnn {#1} {#2} { } { }
7770 }

```

(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`.

```

7771 \cs_new_protected:Nn \__spintent_stat_mathml:nnnn
7772 {
7773   \__spintent_invisible_sep:
7774   \MathMLintent {#4}

```

```

7775     {
7776         \__spinttent_inner_sep:
7777         \__spinttent_stat_typeset:nnn {#1} {#2} {#3}
7778     }
7779 }

```

(End of definition for `\__spinttent_stat_mathml:nnnn`.)

```

\__spinttent_stat_exec:nnn
\__spinttent_stat_simple:nnn
\__spinttent_stat_letter:nnnn

```

The function `\__spinttent_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 `\__spinttent_stat_simple:nnn` is the short form of `\__spinttent_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’.

The function `\__spinttent_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’.

```

7780 \cs_new_protected:Nn \__spinttent_stat_exec:nnn
7781 {
7782     \mode_if_math:TF
7783     {
7784         \group_begin:
7785         \attribute \__spinttent_scope_attr_int = 1
7786         \keys_set:nn { spinttent / #1 } {#2}
7787         #3
7788         \group_end:
7789     }
7790     { \msg_error:nne { spinttent } { need-math-mode } { \exp_not:c {#1} } }
7791 }
7792 \cs_new_protected:Nn \__spinttent_stat_simple:nnn
7793 {
7794     \__spinttent_stat_intent:nn {#1} {#3}
7795     \__spinttent_stat_mathml:nnnn {#1} {#2} { } { \__spinttent_stat_intent_tl }
7796 }
7797 \cs_new_protected:Nn \__spinttent_stat_letter:nnnn
7798 {
7799     \__spinttent_stat_intent:nnn {#1} {#3} {#4}
7800     \__spinttent_stat_mathml:nnnn {#1} {#2} { } { \__spinttent_stat_intent_tl }
7801 }

```

(End of definition for `\__spinttent_stat_exec:nnn`, `\__spinttent_stat_simple:nnn`, and `\__spinttent_stat_letter:nnnn`.)

`\spmedia` The public commands `\spmedia`, `\spmediana` and `\spmoda` delegate execution to `\__spinttent_stat_exec:nnn`, which prints the symbols MA, MD and MO through `\__spinttent_stat_simple:nnn` with the concepts `media-aritmética`, `mediana` and `moda`.

```

7802 \NewDocumentCommand \spmedia { O{} }
7803 {
7804     \__spinttent_stat_exec:nnn { spmedia } {#1}
7805     { \__spinttent_stat_simple:nnn { spmedia } { \mathrm{MA} } { media-aritmética } }
7806 }
7807 \NewDocumentCommand \spmediana { O{} }
7808 {
7809     \__spinttent_stat_exec:nnn { spmediana } {#1}
7810     { \__spinttent_stat_simple:nnn { spmediana } { \mathrm{MD} } { mediana } }
7811 }
7812 \NewDocumentCommand \spmoda { O{} }
7813 {
7814     \__spinttent_stat_exec:nnn { spmoda } {#1}
7815     { \__spinttent_stat_simple:nnn { spmoda } { \mathrm{MO} } { moda } }
7816 }

```

(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 `\__spinttent_stat_letter:nnnn` and `\spvarz` calls `\__spinttent_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*.

```

7817 \NewDocumentCommand \spdesv { O{} }

```

```

7818 {
7819   \__spintent_stat_exec:nnn { spdesv } {#1}
7820   {
7821     \bool_if:NTF \__spintent_spdesv_sigma_bool
7822     {
7823       \__spintent_stat_letter:nnnn { spdesv } { \sigma }
7824       { desviación-estándar } { sigma }
7825     }
7826     {
7827       \__spintent_stat_letter:nnnn { spdesv } { s }
7828       { desviación-estándar } { s }
7829     }
7830   }
7831 }
7832 \NewDocumentCommand \spvarz { 0{ } }
7833 {
7834   \__spintent_stat_exec:nnn { spvarz } {#1}
7835   {
7836     \bool_if:NTF \__spintent_spvarz_sigma_bool
7837     {
7838       \__spintent_stat_intent:nnnn { spvarz } { varianza } { sigma }
7839       { al-cuadrado }
7840       \__spintent_stat_mathml:nnnn { spvarz } { \sigma } { 2 }
7841       { \__spintent_stat_intent_tl }
7842     }
7843     {
7844       \__spintent_stat_intent:nnnn { spvarz } { varianza } { s }
7845       { al-cuadrado }
7846       \__spintent_stat_mathml:nnnn { spvarz } { s } { 2 }
7847       { \__spintent_stat_intent_tl }
7848     }
7849   }
7850 }

```

(End of definition for `\spdesv` and `\spvarz`. These functions are documented on page 43.)

`\sprango` 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.

```

\spriq
\spmin
\spmax
7851 \NewDocumentCommand \sprango { 0{ } }
7852 {
7853   \__spintent_stat_exec:nnn { sprango } {#1}
7854   { \__spintent_stat_simple:nnn { sprango } { R } { rango } }
7855 }
7856 \NewDocumentCommand \spriq { 0{ } }
7857 {
7858   \__spintent_stat_exec:nnn { spriq } {#1}
7859   { \__spintent_stat_simple:nnn { spriq } { \mathrm{RIQ} } { rango-intercuartil } }
7860 }
7861 \NewDocumentCommand \spmin { 0{ } }
7862 {
7863   \__spintent_stat_exec:nnn { spmin } {#1}
7864   {
7865     \__spintent_stat_simple:nnn { spmin }
7866     { x \c_math_subscript_token { \min } } { mínimo }
7867   }
7868 }
7869 \NewDocumentCommand \spmax { 0{ } }
7870 {
7871   \__spintent_stat_exec:nnn { spmax } {#1}
7872   {
7873     \__spintent_stat_simple:nnn { spmax }
7874     { x \c_math_subscript_token { \max } } { máximo }
7875   }
7876 }

```

(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 $\langle argument \rangle$: 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

`\__spintent_spcuartil_read_arg_tl`
`\__spintent_spdecil_read_arg_tl`
`\__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`.

```
7877 \clist_map_inline:nn { spcuartil , spdecil , spPerc }
7878 {
7879   \tl_new:c { \__spintent_ #1 _read_arg_tl }
7880 }
```

(End of definition for `\__spintent_spcuartil_read_arg_tl`, `\__spintent_spdecil_read_arg_tl`, and `\__spintent_spPerc_read_arg_tl`.)

`\__spintent_stat_pos_error_bool`
`\__spintent_stat_pos_value_int`

The variable `\__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 `\__spintent_stat_pos_value_int` holds the accepted index without leading zeros.

```
7881 \bool_new:N \__spintent_stat_pos_error_bool
7882 \int_new:N \__spintent_stat_pos_value_int
```

(End of definition for `\__spintent_stat_pos_error_bool` and `\__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.

```
7883 \msg_new:nnnn { spintent } { stat-pos-invalid-index }
7884 {
7885   Invalid~index~'#2'~for~'\tl_trim_spaces:n {#1}'~\msg_line_context:.
7886 }
7887 {
7888   Only~an~integer~without~sign~between~'1'~and~'#3'~is~accepted.
7889 }
```

(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.

```
7890 \clist_map_inline:nn { spcuartil , spdecil , spPerc }
7891 {
7892   \keys_define:nn { spintent / #1 }
7893   {
7894     silent .bool_set:c = { \__spintent_ #1 _silent_bool },
7895     silent .initial:n = false,
7896     silent .value_required:n = true,
7897     read-arg .code:n =
7898       { \__spintent_sanitize_affix:cn { \__spintent_ #1 _read_arg_tl } {##1} },
7899     read-arg .initial:n = ,
7900     read-arg .value_required:n = true,
7901     unknown .code:n =
7902       {
7903         \str_set:Nn \__spintent_command_name_str {#1}
7904         \__spintent_unknown_keys_cmd:n {##1}
7905       },
7906   }
7907 }
```

(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 `\__spintent_stat_pos_error_bool` to `true`.

```
7908 \cs_new_protected:Nn \__spintent_stat_pos_check:nnn
7909 {
```

```

7910   \__spinttent_luafun_clean_split_and_set:n { \exp_not:n {#3} }
7911   \bool_set_true:N \l__spinttent_stat_pos_error_bool
7912   \bool_lazy_and:nnT
7913     { \str_if_eq_p:Vn \l__spinttent_luaset_has_integer_str { true } }
7914     { \tl_if_empty_p:N \l__spinttent_luaset_sign_tl }
7915     {
7916       \bool_lazy_and:nnT
7917         { \int_compare_p:nNn { \l__spinttent_luaset_only_part_int_str } > { 0 } }
7918         { \int_compare_p:nNn { \l__spinttent_luaset_only_part_int_str } < { #2 + 1 } }
7919         { \bool_set_false:N \l__spinttent_stat_pos_error_bool }
7920     }
7921   \bool_if:NT \l__spinttent_stat_pos_error_bool
7922   {
7923     \msg_error:nneee { spinttent } { stat-pos-invalid-index }
7924     { \token_to_str:c {#1} } { \exp_not:n {#3} } {#2}
7925   }
7926 }

```

(End of definition for `\__spinttent_stat_pos_check:nnn`.)

`\__spinttent_stat_pos_typeset:nn`

The function `\__spinttent_stat_pos_typeset:nn` typesets the symbol in ‘#1’ with the index in ‘#2’ as a subscript, with its own `\MathMLintent` set to `:silent`.

```

7927 \cs_new_protected:Nn \__spinttent_stat_pos_typeset:nn
7928 {
7929   #1
7930   \c_math_subscript_token { \MathMLintent { :silent } {#2} }
7931 }

```

(End of definition for `\__spinttent_stat_pos_typeset:nn`.)

`\__spinttent_stat_pos_mathml:nnnn`

`\__spinttent_stat_pos_mathml:nnnV`

The function `\__spinttent_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 `\__spinttent_stat_pos_typeset:nn` inside one native `\MathMLintent`, preceded by the inner ghost `\__spinttent_inner_sep:`.

With `silent=true`, the intent is ‘`_-`’. With `read-arg` given, the intent is `_ $\langle read-arg \rangle$ :nofix`. Otherwise the spoken content is the name of the measure followed by the index, as in `_cuartil-1:nofix`.

```

7932 \cs_new_protected:Nn \__spinttent_stat_pos_mathml:nnnn
7933 {
7934   \__spinttent_invisible_sep:
7935   \bool_if:cTF { l__spinttent_ #1 _silent_bool }
7936   {
7937     \MathMLintent { _- }
7938     {
7939       \__spinttent_inner_sep:
7940       \__spinttent_stat_pos_typeset:nn {#2} {#4}
7941     }
7942   }
7943   {
7944     \tl_if_empty:cTF { l__spinttent_ #1 _read_arg_tl }
7945     {
7946       \MathMLintent { _ #3 - #4 :nofix }
7947       {
7948         \__spinttent_inner_sep:
7949         \__spinttent_stat_pos_typeset:nn {#2} {#4}
7950       }
7951     }
7952     {
7953       \MathMLintent { _ \tl_use:c { l__spinttent_ #1 _read_arg_tl } :nofix }
7954       {
7955         \__spinttent_inner_sep:
7956         \__spinttent_stat_pos_typeset:nn {#2} {#4}
7957       }
7958     }
7959   }
7960 }
7961 \cs_generate_variant:Nn \__spinttent_stat_pos_mathml:nnnn { nnnV }

```

(End of definition for `\__spinttent_stat_pos_mathml:nnnn`.)

`\__spintent_stat_pos_exec:nnnn`

The function `\__spintent_stat_pos_exec:nnnn` 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`.

```

7962 \cs_new_protected:Nn \__spintent_stat_pos_exec:nnnnn
7963 {
7964   \__spintent_stat_pos_check:nnn {#1} {#4} {#5}
7965   \bool_if:NF \l__spintent_stat_pos_error_bool
7966   {
7967     \int_set:NV \l__spintent_stat_pos_value_int \l__spintent_luaset_only_part_int_str
7968     \__spintent_stat_pos_mathml:nnnV {#1} {#2} {#3} \l__spintent_stat_pos_value_int
7969   }
7970 }
```

(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`.

```

7971 \NewDocumentCommand \spcuartil { O{} m }
7972 {
7973   \mode_if_math:TF
7974   {
7975     \group_begin:
7976     \attribute \l__spintent_scope_attr_int = 1
7977     \keys_set:nn { spintent / spcuartil } {#1}
7978     \__spintent_stat_pos_exec:nnnnn { spcuartil } { Q } { cuartil } { 3 } {#2}
7979     \group_end:
7980   }
7981   { \msg_error:nnn { spintent } { need-math-mode } { \spcuartil } }
7982 }
7983 \NewDocumentCommand \spdecil { O{} m }
7984 {
7985   \mode_if_math:TF
7986   {
7987     \group_begin:
7988     \attribute \l__spintent_scope_attr_int = 1
7989     \keys_set:nn { spintent / spdecil } {#1}
7990     \__spintent_stat_pos_exec:nnnnn { spdecil } { D } { decil } { 9 } {#2}
7991     \group_end:
7992   }
7993   { \msg_error:nnn { spintent } { need-math-mode } { \spdecil } }
7994 }
7995 \NewDocumentCommand \spPerc { O{} m }
7996 {
7997   \mode_if_math:TF
7998   {
7999     \group_begin:
8000     \attribute \l__spintent_scope_attr_int = 1
8001     \keys_set:nn { spintent / spPerc } {#1}
8002     \__spintent_stat_pos_exec:nnnnn { spPerc } { P } { percentil } { 99 } {#2}
8003     \group_end:
8004   }
8005   { \msg_error:nnn { spintent } { need-math-mode } { \spPerc } }
8006 }
```

(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`

`\__spintent_spFrec_sub_index_str`
`\__spintent_spFrec_read_arg_tl`

The string variable `\__spintent_spFrec_sub_index_str` and the token list variable `\__spintent_spFrec_read_arg_tl` hold the values of the keys `sub-index` and `read-arg`.

```
8007 \str_new:N \__spintent_spFrec_sub_index_str
8008 \tl_new:N \__spintent_spFrec_read_arg_tl
```

(End of definition for `\__spintent_spFrec_sub_index_str` and `\__spintent_spFrec_read_arg_tl`.)

`\__spintent_spFrec_symbol_str`
`\__spintent_spFrec_concept_tl`

The string variable `\__spintent_spFrec_symbol_str` holds the $\langle \textit{argument} \rangle$ without spaces, and the token list variable `\__spintent_spFrec_concept_tl` the name of the frequency, empty while the symbol is not in the table.

```
8009 \str_new:N \__spintent_spFrec_symbol_str
8010 \tl_new:N \__spintent_spFrec_concept_tl
```

(End of definition for `\__spintent_spFrec_symbol_str` and `\__spintent_spFrec_concept_tl`.)

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`.

```
8011 \msg_new:nnnn { spintent } { spFrec-invalid-symbol }
8012 {
8013   Invalid~symbol~'#1'~for~'\token_to_str:N \spFrec'\msg_line_context:.
8014 }
8015 {
8016   Use~'f',~'F',~'f_{r}',~'F_{r}',~'f_{r}\iow_char:N \%'\or~'F_{r}\iow_char:N \%','or~
8017   give~the~key~'read-arg'.
8018 }
8019 \msg_new:nnnn { spintent } { spFrec-invalid-sub-index }
8020 {
8021   Invalid~key~'sub-index'~for~the~symbol~'#1'~\msg_line_context:.
8022 }
8023 {
8024   Only~the~symbols~'f'~and~'F'~accept~the~key~'sub-index'.
8025 }
```

(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 $\langle \textit{keys} \rangle$ `silent`, `sub-index`, `read-sub` and `read-arg`, the third one starting in `true`.

```
8026 \keys_define:nn { spintent / spFrec }
8027 {
8028   silent      .bool_set:N = \__spintent_spFrec_silent_bool,
8029   silent      .initial:n = false,
8030   silent      .value_required:n = true,
8031   sub-index   .code:n =
8032     { \str_set:Nn \__spintent_spFrec_sub_index_str {#1} },
8033   sub-index   .initial:n = ,
8034   sub-index   .value_required:n = true,
8035   read-sub    .bool_set:N = \__spintent_spFrec_read_sub_bool,
8036   read-sub    .initial:n = true,
8037   read-sub    .value_required:n = true,
8038   read-arg    .code:n =
8039     { \__spintent_sanitizize_affix:Nn \__spintent_spFrec_read_arg_tl {#1} },
8040   read-arg    .initial:n = ,
8041   read-arg    .value_required:n = true,
8042   unknown     .code:n =
8043     {
8044       \str_set:Nn \__spintent_command_name_str { spFrec }
8045       \__spintent_unknown_keys_cmd:n {#1}
8046     },
8047 }
```

(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`.

```

8048 \cs_new_protected:Nn \__spintent_spFrec_exec:n
8049 {
8050   \str_set:Nx \l__spintent_spFrec_symbol_str { \tl_to_str:n {#1} }
8051   \str_remove_all:Nn \l__spintent_spFrec_symbol_str { ~ }
8052   \tl_clear:N \l__spintent_spFrec_concept_tl
8053   \str_case:Vn \l__spintent_spFrec_symbol_str
8054   {
8055     { f }
8056     { \tl_set:Nn \l__spintent_spFrec_concept_tl { frecuencia-absoluta } }
8057     { F }
8058     { \tl_set:Nn \l__spintent_spFrec_concept_tl { frecuencia-absoluta-acumulada } }
8059     { f_{r} }
8060     { \tl_set:Nn \l__spintent_spFrec_concept_tl { frecuencia-relativa } }
8061     { F_{r} }
8062     { \tl_set:Nn \l__spintent_spFrec_concept_tl { frecuencia-relativa-acumulada } }
8063     { f_{r}\% }
8064     { \tl_set:Nn \l__spintent_spFrec_concept_tl { frecuencia-porcentual } }
8065     { F_{r}\% }
8066     {
8067       \tl_set:Nn \l__spintent_spFrec_concept_tl { frecuencia-porcentual-acumulada }
8068     }
8069   }
8070   \bool_lazy_and:nnTF
8071   { \tl_if_empty_p:N \l__spintent_spFrec_concept_tl }
8072   { \tl_if_empty_p:N \l__spintent_spFrec_read_arg_tl }
8073   {
8074     \msg_error:nnn { spintent } { spFrec-invalid-symbol } {#1}
8075   }
8076   {
8077     \bool_lazy_and:nnTF
8078     { \bool_not_p:n { \str_if_empty_p:N \l__spintent_spFrec_sub_index_str } }
8079     {
8080       \bool_not_p:n
8081       {
8082         \bool_lazy_or_p:nn
8083         { \str_if_eq_p:Vn \l__spintent_spFrec_symbol_str { f } }
8084         { \str_if_eq_p:Vn \l__spintent_spFrec_symbol_str { F } }
8085       }
8086     }
8087     {
8088       \msg_error:nnn { spintent } { spFrec-invalid-sub-index } {#1}
8089     }
8090     {
8091       \__spintent_stat_simple:nnn { spFrec } {#1}
8092       { \l__spintent_spFrec_concept_tl }
8093     }
8094   }
8095 }

```

(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`.

```

8096 \NewDocumentCommand \spFrec { 0{ } m }
8097 {
8098   \mode_if_math:TF
8099   {
8100     \group_begin:
8101     \attribute \l__spintent_scope_attr_int = 1

```

```

8102         \keys_set:nn { spintent / spFrec } {#1}
8103         \__spintent_spFrec_exec:n {#2}
8104     \group_end:
8105 }
8106 { \msg_error:nnn { spintent } { need-math-mode } { \spFrec } }
8107 }

```

(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 $\langle \textit{argument} \rangle$, and the symbol is read with the name of the cases, for example, «casos favorables».

15.61.1 Definition of variables for `\spcase`

The token list variable `\__spintent_spcase_read_arg_tl` holds the value of the key `read-arg`. This command has no `sub-index` key, so the string variable `\__spintent_spcase_sub_index_str` exists only because `\__spintent_stat_intent:nnnn` reads it, and always keeps its initial value.

```

8108 \str_new:N \__spintent_spcase_sub_index_str
8109 \tl_new:N \__spintent_spcase_read_arg_tl

```

(End of definition for `\__spintent_spcase_sub_index_str` and `\__spintent_spcase_read_arg_tl`.)

The string variable `\__spintent_spcase_letter_str` holds the $\langle \textit{argument} \rangle$ without spaces, and the token list variable `\__spintent_spcase_concept_tl` the name of the cases, empty while the letter is not in the table.

```

8110 \str_new:N \__spintent_spcase_letter_str
8111 \tl_new:N \__spintent_spcase_concept_tl

```

(End of definition for `\__spintent_spcase_letter_str` and `\__spintent_spcase_concept_tl`.)

15.61.2 Definition of error messages for `\spcase`

The “error” `spcase-invalid-letter` is raised when the $\langle \textit{argument} \rangle$ is not one of the letters f, p or t.

```

8112 \msg_new:nnnn { spintent } { spcase-invalid-letter }
8113 {
8114     Invalid~letter~'#1'~for~'\token_to_str:N \spcase'~'\msg_line_context:.
8115 }
8116 {
8117     The~letter~must~be~'f',~'p'~or~'t'.
8118 }

```

(End of definition for `spcase-invalid-letter`.)

15.61.3 Definition of keys for `\spcase`

We define the $\langle \textit{keys} \rangle$ `silent` and `read-arg`.

`read-arg`
`unknown`

```

8119 \keys_define:nn { spintent / spcase }
8120 {
8121     silent .bool_set:N = \__spintent_spcase_silent_bool,
8122     silent .initial:n = false,
8123     silent .value_required:n = true,
8124     read-arg .code:n =
8125     { \__spintent_sanitizize_affix:Nn \__spintent_spcase_read_arg_tl {#1} },
8126     read-arg .initial:n = ,
8127     read-arg .value_required:n = true,
8128     unknown .code:n =
8129     {
8130         \str_set:Nn \__spintent_command_name_str { spcase }
8131         \__spintent_unknown_keys_cmd:n {#1}
8132     },
8133 }

```

(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-possibles` 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`.

```

8134 \cs_new_protected:Nn \__spintent_spcase_exec:n
8135 {
8136   \str_set:Nc \l__spintent_spcase_letter_str { \tl_to_str:n {#1} }
8137   \str_remove_all:Nn \l__spintent_spcase_letter_str { ~ }
8138   \tl_clear:N \l__spintent_spcase_concept_tl
8139   \str_case:Vn \l__spintent_spcase_letter_str
8140   {
8141     { f } { \tl_set:Nn \l__spintent_spcase_concept_tl { casos-favorables } }
8142     { p } { \tl_set:Nn \l__spintent_spcase_concept_tl { casos-possibles } }
8143     { t } { \tl_set:Nn \l__spintent_spcase_concept_tl { casos-totales } }
8144   }
8145   \tl_if_empty:NTF \l__spintent_spcase_concept_tl
8146   { \msg_error:nnn { spintent } { spcase-invalid-letter } {#1} }
8147   {
8148     \__spintent_stat_simple:nnn { spcase }
8149     { C \c_math_subscript_token { \str_use:N \l__spintent_spcase_letter_str } }
8150     { \l__spintent_spcase_concept_tl }
8151   }
8152 }

```

(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`.

```

8153 \NewDocumentCommand \spcase { O{} m }
8154 {
8155   \mode_if_math:TF
8156   {
8157     \group_begin:
8158     \attribute \l__spintent_scope_attr_int = 1
8159     \keys_set:nn { spintent / spcase } {#1}
8160     \__spintent_spcase_exec:n {#2}
8161     \group_end:
8162   }
8163   { \msg_error:nnn { spintent } { need-math-mode } { \spcase } }
8164 }

```

(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_read_arg_tl` 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*».

```

8165 \str_new:N \l__spintent_spOmega_sub_index_str
8166 \tl_new:N \l__spintent_spOmega_read_arg_tl
8167 \bool_new:N \l__spintent_spOmega_read_sub_bool
8168 \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`.

```

8169 \keys_define:nn { spintent / spOmega }
8170 {

```

```

8171   silent      .bool_set:N = \l__spinttent_spOmega_silent_bool,
8172   silent      .initial:n = false,
8173   silent      .value_required:n = true,
8174   sub-index   .code:n =
8175     { \str_set:Nn \l__spinttent_spOmega_sub_index_str {#1} },
8176   sub-index   .initial:n = ,
8177   sub-index   .value_required:n = true,
8178   read-arg    .code:n =
8179     { \__spinttent_sanitizize_affix:Nn \l__spinttent_spOmega_read_arg_tl {#1} },
8180   read-arg    .initial:n = ,
8181   read-arg    .value_required:n = true,
8182   card        .bool_set:N = \l__spinttent_spOmega_card_bool,
8183   card        .initial:n = false,
8184   card        .value_required:n = true,
8185   unknown     .code:n =
8186     {
8187       \str_set:Nn \l__spinttent_command_name_str { spOmega }
8188       \__spinttent_unknown_keys_cmd:n {#1}
8189     },
8190   }

```

(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 `\__spinttent_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*.

```

8191 \NewDocumentCommand \spOmega { 0{} }
8192 {
8193   \mode_if_math:TF
8194   {
8195     \group_begin:
8196     \attribute \l__spinttent_scope_attr_int = 1
8197     \keys_set:nn { spinttent / spOmega } {#1}
8198     \bool_if:NTF \l__spinttent_spOmega_card_bool
8199     {
8200       \__spinttent_stat_simple:nnn { spOmega } { \# \Omega }
8201       { cardinalidad-del-espacio-muestral }
8202     }
8203     {
8204       \__spinttent_stat_simple:nnn { spOmega } { \Omega }
8205       { espacio-muestral }
8206     }
8207   }
8208   \group_end:
8209   { \msg_error:nnn { spinttent } { need-math-mode } { \spOmega } }
8210 }

```

(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 We define the *(keys)* *silent* and *use-dfrac*.
use-dfrac
unknown

```

8211 \keys_define:nn { spinttent / spPval }
8212 {
8213   silent      .bool_set:N = \l__spinttent_spPval_silent_bool,
8214   silent      .initial:n = false,
8215   silent      .value_required:n = true,
8216   use-dfrac   .bool_set:N = \l__spinttent_spPval_use_dfrac_bool,
8217   use-dfrac   .initial:n = false,
8218   use-dfrac   .value_required:n = true,
8219   unknown     .code:n =
8220     {
8221       \str_set:Nn \l__spinttent_command_name_str { spPval }
8222       \__spinttent_unknown_keys_cmd:n {#1}
8223     },
8224   }

```

(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 $\langle keys \rangle$ in ‘#1’, the numerator in ‘#2’ and the denominator in ‘#3’. It sets the $\langle keys \rangle$ and calls `\__spintent_spPval_mathml:nn`.

```

8225 \cs_new_protected:Nn \__spintent_spPval_exec:nnn
8226 {
8227   \keys_set:nn { spintent / spPval } {#1}
8228   \__spintent_spPval_mathml:nn {#2} {#3}
8229 }

```

(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)`.

```

8230 \cs_new_protected:Nn \__spintent_spPval_mathml:nn
8231 {
8232   \__spintent_invisible_sep:
8233   \bool_if:NTF \l__spintent_spPval_silent_bool
8234   {
8235     \MathMLintent { _- }
8236     {
8237       \__spintent_inner_sep:
8238       \bool_if:NTF \l__spintent_spPval_use_dfrac_bool
8239       { \dfrac {#1} {#2} }
8240       { \frac {#1} {#2} }
8241     }
8242   }
8243   {
8244     \MathMLintent { _de:infix($num,$denom) }
8245     {
8246       \__spintent_inner_sep:
8247       \bool_if:NTF \l__spintent_spPval_use_dfrac_bool
8248       {
8249         \dfrac { \MathMLarg { num } {#1} } { \MathMLarg { denom } {#2} }
8250       }
8251       { \frac { \MathMLarg { num } {#1} } { \MathMLarg { denom } {#2} } }
8252     }
8253   }
8254 }

```

(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`.

```

8255 \NewDocumentCommand \spPval { O{} m m }
8256 {
8257   \mode_if_math:TF
8258   {
8259     \group_begin:
8260     \__spintent_spPval_exec:nnn {#1} {#2} {#3}
8261     \group_end:
8262   }
8263   { \msg_error:nnn { spintent } { need-math-mode } { \spPval } }
8264 }

```

(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 $P(\langle event \rangle)$. The $\langle argument \rangle$ 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 $\langle argument \rangle$ 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.

```
8265 \str_new:N \l__spintent_spProb_luaset_error_str
8266 \str_new:N \l__spintent_spProb_luaset_op_str
8267 \str_new:N \l__spintent_spProb_luaset_a_str
8268 \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`.

```
8269 \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.

```
8270 \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 $\langle argument \rangle$ is empty, has unbalanced braces or parentheses, has an empty event, or has more than one operator.

```
8271 \msg_new:nnnn { spintent } { spProb-invalid-arg }
8272 {
8273   Invalid~argument~'#1'~for~'\token_to_str:N \spProb'~\msg_line_context:.
8274 }
8275 {
8276   Use~one~event,~or~two~joined~by~one~of~'\token_to_str:N \cap',~
8277   '\token_to_str:N \cup',~'\token_to_str:N \mid'~or~'|'~(outside~braces~
8278   and~parentheses).
8279 }
```

(End of definition for `spProb-invalid-arg`.)

15.64.3 Definition of keys for \spProb

silent
 read-arg
 unknown

We define the $\langle keys \rangle$ `silent` and `read-arg`.

```
8280 \keys_define:nn { spintent / spProb }
8281 {
8282   silent .bool_set:N = \l__spintent_spProb_silent_bool,
8283   silent .initial:n = false,
8284   silent .value_required:n = true,
8285   read-arg .code:n =
8286     { \__spintent_sanitize_affix:Nn \l__spintent_spProb_read_arg_tl {#1} },
8287   read-arg .initial:n = ,
8288   read-arg .value_required:n = true,
8289   unknown .code:n =
8290     {
8291       \str_set:Nn \l__spintent_command_name_str { spProb }
8292       \__spintent_unknown_keys_cmd:n {#1}
8293     },
8294 }
```

(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:`.

```

8295 \cs_new_protected:Nn \__spintent_spProb_cap:
8296 {
8297   \mathbin { \MathMLintent { _y- } { \__spintent_spProb_cap_orig: } }
8298 }
8299 \cs_new_protected:Nn \__spintent_spProb_cup:
8300 {
8301   \mathbin { \MathMLintent { _o } { \__spintent_spProb_cup_orig: } }
8302 }
8303 \cs_new_protected:Nn \__spintent_spProb_mid:
8304 {
8305   \mathrel { \MathMLintent { _dado } { \__spintent_spProb_mid_orig: } }
8306 }
8307 \cs_new_protected:Nn \__spintent_spProb_nested_ops:
8308 {
8309   \bool_if:NF \__spintent_spProb_nested_bool
8310   {
8311     \bool_set_true:N \__spintent_spProb_nested_bool
8312     \cs_set_eq:NN \__spintent_spProb_cap_orig: \cap
8313     \cs_set_eq:NN \__spintent_spProb_cup_orig: \cup
8314     \cs_set_eq:NN \__spintent_spProb_mid_orig: \mid
8315     \cs_set_eq:NN \cap \__spintent_spProb_cap:
8316     \cs_set_eq:NN \cup \__spintent_spProb_cup:
8317     \cs_set_eq:NN \mid \__spintent_spProb_mid:
8318   }
8319 }
8320 \cs_new_protected:Nn \__spintent_spProb_event:N
8321 {
8322   \group_begin:
8323     \__spintent_spProb_nested_ops:
8324     \str_use:N #1
8325   \group_end:
8326 }

```

(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.

```

8327 \cs_new_protected:Nn \__spintent_spProb_content:
8328 {
8329   \str_use:N \l__spintent_spProb_luaset_a_str
8330   \str_case:Vn \l__spintent_spProb_luaset_op_str
8331   {
8332     { cap } { \cap \str_use:N \l__spintent_spProb_luaset_b_str }
8333     { cup } { \cup \str_use:N \l__spintent_spProb_luaset_b_str }
8334     { mid } { \mid \str_use:N \l__spintent_spProb_luaset_b_str }
8335   }
8336 }

```

(End of definition for `\__spintent_spProb_content:`)

`\__spintent_spProb_mathml:` The function `\__spintent_spProb_mathml:` prints $P(\langle content \rangle)$ 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 `\_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.

```

8337 \cs_new_protected:Nn \__spintent_spProb_mathml:
8338 {
8339   \__spintent_invisible_sep:
8340   \bool_if:NTF \__spintent_spProb_silent_bool
8341   {
8342     \MathMLintent { _- }
8343     {
8344       \__spintent_inner_sep:
8345       P \lparen \__spintent_spProb_content: \rparen
8346     }

```

```

8347     }
8348   {
8349     \tl_if_empty:NTF \l__spintent_spProb_read_arg_tl
8350     {
8351       \str_if_eq:VnTF \l__spintent_spProb_luaset_op_str { none }
8352       {
8353         \MathMLintent { _probabilidad-de:prefix($a) }
8354         {
8355           \__spintent_inner_sep:
8356           P
8357           \lparen
8358             \MathMLarg { a }
8359             {
8360               \__spintent_inner_sep: \__spintent_spProb_event:N
8361               \l__spintent_spProb_luaset_a_str
8362             }
8363           \rparen
8364         }
8365       }
8366     {
8367       \MathMLintent { _probabilidad-de:prefix($p) }
8368       {
8369         \__spintent_inner_sep:
8370         P
8371         \lparen
8372           \MathMLarg { p }
8373           {
8374             \MathMLintent
8375             {
8376               -
8377               \str_case:Vn \l__spintent_spProb_luaset_op_str
8378               { { cap } { y- } { cup } { o } { mid } { dado } }
8379               :infix($a,$b)
8380             }
8381             {
8382               \__spintent_inner_sep:
8383               \MathMLarg { a }
8384               {
8385                 \__spintent_inner_sep: \__spintent_spProb_event:N
8386                 \l__spintent_spProb_luaset_a_str
8387               }
8388               \str_case:Vn \l__spintent_spProb_luaset_op_str
8389               { { cap } { \cap } { cup } { \cup } { mid } { \mid } }
8390               \MathMLarg { b }
8391               {
8392                 \__spintent_inner_sep: \__spintent_spProb_event:N
8393                 \l__spintent_spProb_luaset_b_str
8394               }
8395             }
8396           }
8397         \rparen
8398       }
8399     }
8400   }
8401   {
8402     \MathMLintent { _ \tl_use:N \l__spintent_spProb_read_arg_tl :nofix }
8403     {
8404       \__spintent_inner_sep:
8405       P \lparen \__spintent_spProb_content: \rparen
8406     }
8407   }
8408 }
8409 }

```

(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:`.

8410 `\cs_new_protected:Nn \__spintent_spProb_exec:nn`

```

8411 {
8412   \keys_set:nn { spintent / spProb } {#1}
8413   \__spintent_luafun_spProb_parse_and_set:n { \exp_not:n {#2} }
8414   \str_if_eq:VnTF \__spintent_spProb_luaset_error_str { true }
8415     { \msg_error:nnn { spintent } { spProb-invalid-arg } {#2} }
8416     { \__spintent_spProb_mathml: }
8417 }

```

(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`.

```

8418 \NewDocumentCommand \spProb { O{} m }
8419 {
8420   \mode_if_math:TF
8421   {
8422     \group_begin:
8423       \attribute \__spintent_scope_attr_int = 1
8424       \__spintent_spProb_exec:nn {#1} {#2}
8425     \group_end:
8426   }
8427   { \msg_error:nnn { spintent } { need-math-mode } { \spProb } }
8428 }

```

(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 $\langle event \rangle^{\{c\}}$ or, with the key `bar`, as $\overline{\langle event \rangle}$, and reads it «*complemento de*» the event.

15.65.1 Definition of variables for `\spcomp`

`\__spintent_spcomp_read_arg_tl`
`\__spintent_spcomp_wrap_parens_str`

The variable `\__spintent_spcomp_read_arg_tl` stores the key `read-arg` and `\__spintent_spcomp_wrap_parens_str` the key `wrap-parens`.

```

8429 \tl_new:N \__spintent_spcomp_read_arg_tl
8430 \str_new:N \__spintent_spcomp_wrap_parens_str

```

(End of definition for `\__spintent_spcomp_read_arg_tl` and `\__spintent_spcomp_wrap_parens_str`.)

`\__spintent_spcomp_luaset_compound_str`
`\__spintent_spcomp_compound_bool`
`\__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 `\__spintent_spcomp_luaset_compound_str`. The boolean variable `\__spintent_spcomp_compound_bool` is true in that case. The boolean variable `\__spintent_spcomp_do_paren_bool` tells whether the event is printed in parentheses once the key and the event have been examined.

```

8431 \str_new:N \__spintent_spcomp_luaset_compound_str
8432 \bool_new:N \__spintent_spcomp_compound_bool
8433 \bool_new:N \__spintent_spcomp_do_paren_bool

```

(End of definition for `\__spintent_spcomp_luaset_compound_str`, `\__spintent_spcomp_compound_bool`, and `\__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*.

```

8434 \msg_new:nnnn { spintent } { spcomp-empty-arg }
8435 {
8436   Empty~argument~for~'\token_to_str:N \spcomp'\~\msg_line_context:.
8437 }
8438 {
8439   The~event~is~required.
8440 }

```

(End of definition for `spcomp-empty-arg`.)

15.65.3 Definition of keys for \spcomp

We define the *(keys)* `silent`, `bar`, `wrap-parens` and `read-arg`.

```

8441 \keys_define:nn { spintent / spcomp }
8442 {
8443   silent      .bool_set:N = \l__spintent_spcomp_silent_bool,
8444   silent      .initial:n = false,
8445   silent      .value_required:n = true,
8446   bar         .bool_set:N = \l__spintent_spcomp_bar_bool,
8447   bar         .initial:n = false,
8448   bar         .value_required:n = true,
8449   wrap-parens .choices:nn = { auto , true , false }
8450   {
8451     \int_case:nn { \l_keys_choice_int }
8452     {
8453       {1} { \str_set:Nn \l__spintent_spcomp_wrap_parens_str { auto } }
8454       {2} { \str_set:Nn \l__spintent_spcomp_wrap_parens_str { true } }
8455       {3} { \str_set:Nn \l__spintent_spcomp_wrap_parens_str { false } }
8456     }
8457   },
8458   wrap-parens .initial:n = auto,
8459   wrap-parens .value_required:n = true,
8460   wrap-parens / unknown .code:n =
8461     \msg_error:nneee { spintent } { unknown-choice }
8462     { wrap-parens } { auto,~true,~false } { \exp_not:n {#1} },
8463   read-arg     .code:n =
8464     { \__spintent_sanitizize_affix:Nn \l__spintent_spcomp_read_arg_tl {#1} },
8465   read-arg     .initial:n = ,
8466   read-arg     .value_required:n = true,
8467   unknown      .code:n =
8468     {
8469       \str_set:Nn \l__spintent_command_name_str { spcomp }
8470       \__spintent_unknown_keys_cmd:n {#1}
8471     },
8472 }

```

(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.

```

8473 \cs_new_protected:Nn \__spintent_spcomp_set_paren:n
8474 {
8475   \bool_set_false:N \l__spintent_spcomp_do_paren_bool
8476   \bool_set_false:N \l__spintent_spcomp_compound_bool
8477   \__spintent_luafun_spcomp_compound:n { \exp_not:n {#1} }
8478   \str_if_eq:VnT \l__spintent_spcomp_lua_set_compound_str { true }
8479   { \bool_set_true:N \l__spintent_spcomp_compound_bool }
8480   \bool_if:NF \l__spintent_spcomp_bar_bool
8481   {
8482     \str_case:Vn \l__spintent_spcomp_wrap_parens_str
8483     {
8484       { true } { \bool_set_true:N \l__spintent_spcomp_do_paren_bool }
8485       { auto }
8486       {
8487         \bool_if:NT \l__spintent_spcomp_compound_bool
8488         { \bool_set_true:N \l__spintent_spcomp_do_paren_bool }
8489       }
8490     }
8491   }
8492 }
8493 \cs_new_protected:Nn \__spintent_spcomp_event:n
8494 {
8495   \group_begin:
8496   \__spintent_spProb_nested_ops:

```

```

8497     \bool_if:NTF \l__spintent_spcomp_do_paren_bool
8498     { \lparen #1 \rparen }
8499     {#1}
8500   \group_end:
8501 }

```

(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`.

```

8502 \cs_new_protected:Nn \__spintent_spcomp_typeset:n
8503 {
8504   \bool_if:NTF \l__spintent_spcomp_bar_bool
8505   { \overline {#1} }
8506   { {#1} ^ { c } }
8507 }

```

(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`.

```

8508 \cs_new_protected:Nn \__spintent_spcomp_exec:nn
8509 {
8510   \keys_set:nn { spintent / spcomp } {#1}
8511   \__spintent_spcomp_set_paren:n {#2}
8512   \__spintent_spcomp_mathml:n {#2}
8513 }

```

(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.

```

8514 \cs_new_protected:Nn \__spintent_spcomp_mathml:n
8515 {
8516   \__spintent_invisible_sep:
8517   \bool_if:NTF \l__spintent_spcomp_silent_bool
8518   {
8519     \MathMLintent { _- }
8520     {
8521       \__spintent_inner_sep:
8522       \__spintent_spcomp_typeset:n { \__spintent_spcomp_event:n {#1} }
8523     }
8524   }
8525   {
8526     \tl_if_empty:NTF \l__spintent_spcomp_read_arg_tl
8527     {
8528       \MathMLintent
8529       {
8530         \bool_lazy_and:nnTF
8531         { \bool_if_p:N \l__spintent_spcomp_bar_bool }
8532         { \bool_if_p:N \l__spintent_spcomp_compound_bool }
8533         { _complemento-de-la-expresión:prefix($a) }
8534         { _complemento-de:prefix($a) }
8535       }
8536       {
8537         \__spintent_inner_sep:
8538         \__spintent_spcomp_typeset:n
8539         {
8540           \MathMLarg { a }
8541           { \__spintent_inner_sep: \__spintent_spcomp_event:n {#1} }
8542         }
8543       }
8544     }
8545     {
8546       \MathMLintent { _ \tl_use:N \l__spintent_spcomp_read_arg_tl :nofix }

```

```

8547         {
8548             \__spintent_inner_sep:
8549             \__spintent_spcomp_typeset:n { \__spintent_spcomp_event:n {#1} }
8550         }
8551     }
8552 }
8553 }

```

(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`.

```

8554 \NewDocumentCommand \spcomp { O{} m }
8555 {
8556     \mode_if_math:TF
8557     {
8558         \group_begin:
8559         \attribute \l__spintent_scope_attr_int = 1
8560         \tl_if_blank:nTF {#2}
8561         { \msg_error:nn { spintent } { spcomp-empty-arg } }
8562         { \__spintent_spcomp_exec:nn {#1} {#2} }
8563         \group_end:
8564     }
8565     { \msg_error:nnn { spintent } { need-math-mode } { \spcomp } }
8566 }

```

(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` 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.

```

8567 \str_new:N \l__spintent_spcomb_luaset_single_str
8568 \str_new:N \l__spintent_spcomb_luaset_raw_n_str
8569 \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` 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.

```

8570 \str_new:N \l__spintent_spcomb_luaset_error_str
8571 \str_new:N \l__spintent_spcomb_luaset_n_str
8572 \str_new:N \l__spintent_spcomb_luaset_k_str
8573 \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` 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*.

```

\l__spintent_spcomb_kind_str
\l__spintent_spcomb_print_sty_str
8574 \tl_new:N \l__spintent_spcomb_read_arg_tl
8575 \str_new:N \l__spintent_spcomb_kind_str
8576 \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`.)

Variables computed from the keys. The token list variable `\l__spinttent_spcomb_letter_tl` holds the letters of the symbol: C, V, A or P, followed by R and/or C; and `\l__spinttent_spcomb_concept_tl` the name of the concept that is read, before the final -de. The string variable `\l__spinttent_spcomb_mode_str` is one, pair or list according to the arguments the symbol takes, and `\l__spinttent_spcomb_problem_str` describes an incompatible combination of keys, it is empty when there is none. The boolean variable `\l__spinttent_spcomb_use_binom_bool` is true when the symbol is printed with `\binom`. The sequence `\l__spinttent_spcomb_list_seq` holds the multiplicities to print, and the token list `\l__spinttent_spcomb_last_tl` the last one.

```

8577 \tl_new:N \l__spinttent_spcomb_letter_tl
8578 \tl_new:N \l__spinttent_spcomb_concept_tl
8579 \str_new:N \l__spinttent_spcomb_mode_str
8580 \str_new:N \l__spinttent_spcomb_problem_str
8581 \bool_new:N \l__spinttent_spcomb_use_binom_bool
8582 \seq_new:N \l__spinttent_spcomb_list_seq
8583 \tl_new:N \l__spinttent_spcomb_last_tl

```

(End of definition for `\l__spinttent_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`.

```

8584 \msg_new:nnnn { spinttent } { spcomb-invalid-keys }
8585 {
8586   Incompatible~keys~for~'\token_to_str:N \spcomb':~'#1'~\msg_line_context:.
8587 }
8588 {
8589   'circular'~needs~'kind=permutation'~and~'repeat=false';~'print-sty=binom'~needs~
8590   'kind=combination'~and~'repeat=false'.
8591 }
8592 \msg_new:nnnn { spinttent } { spcomb-missing-arg }
8593 {
8594   Missing~'k'~(after~the~comma)~for~'\token_to_str:N \spcomb'~\msg_line_context:.
8595 }
8596 {
8597   This~kind~of~symbol~needs~'k'~or~the~list~of~repetitions~after~the~
8598   comma:~'\token_to_str:N \spcomb {n,k}'.
8599 }
8600 \msg_new:nnnn { spinttent } { spcomb-extra-arg }
8601 {
8602   Unexpected~comma~and~'k'~for~'\token_to_str:N \spcomb'~\msg_line_context:.
8603 }
8604 {
8605   Circular~permutations~take~only~'n':~'\token_to_str:N \spcomb {n}'.
8606 }
8607 \msg_new:nnnn { spinttent } { spcomb-invalid-arg }
8608 {
8609   Invalid~arguments~'#1'~and~'#2'~for~'\token_to_str:N \spcomb'~\msg_line_context:.
8610 }
8611 {
8612   Use~'n',~'n,k'~or~'n,a,b,...'~(at~least~two~repetitions,~none~empty);~if~
8613   integers,~'k'~<~n'~and~'a+b+...<~n'.
8614 }

```

(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`.

```

8615 \keys_define:nn { spinttent / spcomb }
8616 {
8617   silent      .bool_set:N = \l__spinttent_spcomb_silent_bool,
8618   silent      .initial:n = false,
8619   read-arg    .value_required:n = true,
8620   repeat      .bool_set:N = \l__spinttent_spcomb_repeat_bool,

```

```

8621     repeat      .initial:n = false,
8622     repeat      .value_required:n = true,
8623     circular    .bool_set:N = \l__spinttent_spcomb_circular_bool,
8624     circular    .initial:n = false,
8625     circular    .value_required:n = true,
8626     kind        .choices:nn =
8627       { combination , variation , arrangement , permutation }
8628     {
8629       \int_case:nn { \l_keys_choice_int }
8630       {
8631         {1} { \str_set:Nn \l__spinttent_spcomb_kind_str { combination } }
8632         {2} { \str_set:Nn \l__spinttent_spcomb_kind_str { variation } }
8633         {3} { \str_set:Nn \l__spinttent_spcomb_kind_str { arrangement } }
8634         {4} { \str_set:Nn \l__spinttent_spcomb_kind_str { permutation } }
8635       }
8636     },
8637     kind        .initial:n = combination,
8638     kind        .value_required:n = true,
8639     kind / unknown .code:n =
8640       \msg_error:nneee { spinttent } { unknown-choice }
8641       { kind } { { combination,~variation,~arrangement,~permutation }
8642         { \exp_not:n {#1} } },
8643     print-sty .choices:nn = { auto , binom , letter }
8644     {
8645       \int_case:nn { \l_keys_choice_int }
8646       {
8647         {1} { \str_set:Nn \l__spinttent_spcomb_print_sty_str { auto } }
8648         {2} { \str_set:Nn \l__spinttent_spcomb_print_sty_str { binom } }
8649         {3} { \str_set:Nn \l__spinttent_spcomb_print_sty_str { letter } }
8650       }
8651     },
8652     print-sty .initial:n = auto,
8653     print-sty .value_required:n = true,
8654     print-sty / unknown .code:n =
8655       \msg_error:nneee { spinttent } { unknown-choice }
8656       { print-sty } { { auto,~binom,~letter } { \exp_not:n {#1} } },
8657     read-arg .code:n =
8658       { \__spinttent_sanitizetl:Nn \l__spinttent_spcomb_read_arg_tl {#1} },
8659     read-arg .initial:n = ,
8660     read-arg .value_required:n = true,
8661     unknown .code:n =
8662       {
8663         \str_set:Nn \l__spinttent_command_name_str { spcomb }
8664         \__spinttent_unknown_keys_cmd:n {#1}
8665       },
8666   }

```

(End of definition for *silent* and others.)

15.66.4 Internal functions for \spcomb

`\__spinttent_spcomb_setup:` The function `\__spinttent_spcomb_setup:` computes the letters and the concept from the keys, decides whether `\binom` is used, and leaves in `\l__spinttent_spcomb_problem_str` the description of an incompatible combination of keys, if any.

```

8667 \cs_new_protected:Nn \__spinttent_spcomb_setup:
8668 {
8669   \str_clear:N \l__spinttent_spcomb_problem_str
8670   \str_case:Vn \l__spinttent_spcomb_kind_str
8671   {
8672     { combination }
8673     {
8674       \tl_set:Nn \l__spinttent_spcomb_letter_tl { C }
8675       \tl_set:Nn \l__spinttent_spcomb_concept_tl { combinación }
8676     }
8677     { variation }
8678     {
8679       \tl_set:Nn \l__spinttent_spcomb_letter_tl { V }
8680       \tl_set:Nn \l__spinttent_spcomb_concept_tl { variación }
8681     }
8682     { arrangement }
8683     {

```

```

8684         \tl_set:Nn \l__spintent_spcomb_letter_tl { A }
8685         \tl_set:Nn \l__spintent_spcomb_concept_tl { arreglo }
8686     }
8687     { permutation }
8688     {
8689         \tl_set:Nn \l__spintent_spcomb_letter_tl { P }
8690         \tl_set:Nn \l__spintent_spcomb_concept_tl { permutación }
8691     }
8692 }
8693 \bool_if:NT \l__spintent_spcomb_circular_bool
8694 {
8695     \bool_lazy_and:nnTF
8696     { \str_if_eq_p:Nn \l__spintent_spcomb_kind_str { permutation } }
8697     { ! \bool_if_p:N \l__spintent_spcomb_repeat_bool }
8698     {
8699         \tl_put_right:Nn \l__spintent_spcomb_letter_tl { C }
8700         \tl_put_right:Nn \l__spintent_spcomb_concept_tl { -circular }
8701     }
8702     {
8703         \str_set:Nn \l__spintent_spcomb_problem_str
8704         { 'circular'~requires~'kind=permutation'~and~'repeat=false' }
8705     }
8706 }
8707 \bool_if:NT \l__spintent_spcomb_repeat_bool
8708 {
8709     \tl_put_right:Nn \l__spintent_spcomb_letter_tl { R }
8710     \tl_put_right:Nn \l__spintent_spcomb_concept_tl { -con-repetición }
8711 }
8712 \bool_set_false:N \l__spintent_spcomb_use_binom_bool
8713 \bool_lazy_and:nnTF
8714 { \str_if_eq_p:Nn \l__spintent_spcomb_kind_str { combination } }
8715 { ! \bool_if_p:N \l__spintent_spcomb_repeat_bool }
8716 {
8717     \str_if_eq:NnF \l__spintent_spcomb_print_sty_str { letter }
8718     { \bool_set_true:N \l__spintent_spcomb_use_binom_bool }
8719 }
8720 {
8721     \str_if_eq:NnT \l__spintent_spcomb_print_sty_str { binom }
8722     {
8723         \str_if_empty:NT \l__spintent_spcomb_problem_str
8724         {
8725             \str_set:Nn \l__spintent_spcomb_problem_str
8726             { 'print=binom'~requires~'kind=combination'~and~'repeat=false' }
8727         }
8728     }
8729 }
8730 }

```

(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.

```

8731 \cs_new_protected:Nn \__spintent_spcomb_arg_ann:nn
8732 {
8733     \MathMLarg {#1} { \__spintent_inner_sep: #2 }
8734 }
8735 \cs_new_protected:Nn \__spintent_spcomb_arg_plain:nn
8736 {
8737     #2
8738 }
8739 \cs_new_eq:NN \__spintent_spcomb_arg:nn \__spintent_spcomb_arg_plain:nn
8740 \cs_new_protected:Nn \__spintent_spcomb_sep_ann:
8741 {
8742     \MathMLintent { :silent } { , }
8743 }
8744 \cs_new_protected:Nn \__spintent_spcomb_sep_plain:
8745 {
8746     ,

```

```

8747 }
8748 \cs_new_eq:NN \__spintent_spcomb_sep: \__spintent_spcomb_sep_plain:
8749 \cs_new_protected:Nn \__spintent_spcomb_last_ann:n
8750 {
8751   \MathMLintent { _y-:prefix($\l) }
8752   { \__spintent_inner_sep: \MathMLarg { \l } { \__spintent_inner_sep: #1 } }
8753 }
8754 \cs_new_protected:Nn \__spintent_spcomb_last_plain:n
8755 {
8756   #1
8757 }
8758 \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)`.

```

8759 \cs_new_protected:Nn \__spintent_spcomb_list:
8760 {
8761   \seq_clear:N \__spintent_spcomb_list_seq
8762   \tl_map_inline:Nn \__spintent_spcomb_luaset_list_tl
8763   { \seq_put_right:Nn \__spintent_spcomb_list_seq {##1} }
8764   \seq_pop_right:NN \__spintent_spcomb_list_seq \__spintent_spcomb_last_tl
8765   \seq_use:Nn \__spintent_spcomb_list_seq { \__spintent_spcomb_sep: }
8766   \__spintent_spcomb_sep:
8767   \exp_args:NV \__spintent_spcomb_last:n \__spintent_spcomb_last_tl
8768 }

```

(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.

```

8769 \cs_new_protected:Nn \__spintent_spcomb_symbol:
8770 {
8771   \bool_if:NTF \__spintent_spcomb_use_binom_bool
8772   {
8773     \int_compare:nNnTF { \mathstyle } < { 2 } { \Bigl } { \bigl } \lparen
8774     \genfrac { } { } { } { \opt } { }
8775     {
8776       \__spintent_spcomb_arg:nn { n }
8777       { \str_use:N \__spintent_spcomb_luaset_n_str }
8778     }
8779     {
8780       \__spintent_spcomb_arg:nn { k }
8781       { \str_use:N \__spintent_spcomb_luaset_k_str }
8782     }
8783     \int_compare:nNnTF { \mathstyle } < { 2 } { \Bigr } { \bigr } \rparen
8784   }
8785   {
8786     { \tl_use:N \__spintent_spcomb_letter_tl }
8787     \str_case:Vn \__spintent_spcomb_mode_str
8788     {
8789       { pair }
8790       {
8791         \c_math_subscript_token
8792         {
8793           \__spintent_spcomb_arg:nn { k }
8794           { \str_use:N \__spintent_spcomb_luaset_k_str }
8795         }
8796         ^
8797         {
8798           \__spintent_spcomb_arg:nn { n }
8799           { \str_use:N \__spintent_spcomb_luaset_n_str }
8800         }
8801       }
8802       { one }
8803       {
8804         \c_math_subscript_token
8805         {

```

```

8806         \\\spinttent_spcomb_arg:nn { n }
8807         { \str_use:N \l_\spinttent_spcomb_luaset_n_str }
8808     }
8809 }
8810 { list }
8811 {
8812     \c_math_subscript_token
8813     {
8814         \\\spinttent_spcomb_arg:nn { n }
8815         { \str_use:N \l_\spinttent_spcomb_luaset_n_str }
8816     }
8817     ^
8818     {
8819         \\\spinttent_spcomb_arg:nn { m } { \\\spinttent_spcomb_list: }
8820     }
8821 }
8822 }
8823 }
8824 }

```

(End of definition for \\\spinttent_spcomb_symbol:.)

`\\spinttent_spcomb_mathml:` The function `\\spinttent_spcomb_mathml:` prints the symbol inside one native `\MathMLintent`, preceded by the outer ghost `\\spinttent_invisible_sep:` and by the inner ghost `\\spinttent_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 `\\spinttent_spcomb_list:`.

```

8825 \cs_new_protected:Nn \\\spinttent_spcomb_mathml:
8826 {
8827     \\\spinttent_invisible_sep:
8828     \bool_if:NTF \l_\spinttent_spcomb_silent_bool
8829     {
8830         \cs_set_eq:NN \\\spinttent_spcomb_arg:nn \\\spinttent_spcomb_arg_plain:nn
8831         \cs_set_eq:NN \\\spinttent_spcomb_sep: \\\spinttent_spcomb_sep_plain:
8832         \cs_set_eq:NN \\\spinttent_spcomb_last:n \\\spinttent_spcomb_last_plain:n
8833         \MathMLintent { - }
8834         { \\\spinttent_inner_sep: \\\spinttent_spcomb_symbol: }
8835     }
8836     {
8837         \tl_if_empty:NTF \l_\spinttent_spcomb_read_arg_tl
8838         {
8839             \cs_set_eq:NN \\\spinttent_spcomb_arg:nn \\\spinttent_spcomb_arg_ann:nn
8840             \cs_set_eq:NN \\\spinttent_spcomb_sep: \\\spinttent_spcomb_sep_ann:
8841             \cs_set_eq:NN \\\spinttent_spcomb_last:n \\\spinttent_spcomb_last_ann:n
8842             \str_case:Vn \l_\spinttent_spcomb_mode_str
8843             {
8844                 { one }
8845                 {
8846                     \MathMLintent
8847                     { _ \tl_use:N \l_\spinttent_spcomb_concept_tl -de:prefix($c) }
8848                     {
8849                         \\\spinttent_inner_sep:
8850                         \MathMLarg { c }
8851                         {
8852                             \MathMLintent { _elementos:postfix($n) }
8853                             { \\\spinttent_inner_sep: \\\spinttent_spcomb_symbol: }
8854                         }
8855                     }
8856                 }
8857                 { pair }
8858                 {
8859                     \bool_if:NTF \l_\spinttent_spcomb_use_binom_bool
8860                     {
8861                         \MathMLintent { _sobre:infix($n,$k) }
8862                         { \\\spinttent_inner_sep: \\\spinttent_spcomb_symbol: }

```

```

8863         }
8864     {
8865         \MathMLintent
8866         { _ \tl_use:N \l__spinttent_spcomb_concept_tl -de:prefix($c) }
8867         {
8868             \__spinttent_inner_sep:
8869             \MathMLarg { c }
8870             {
8871                 \MathMLintent { _elementos-tomados-de-a:infix($n,$k) }
8872                 { \__spinttent_inner_sep: \__spinttent_spcomb_symbol: }
8873             }
8874         }
8875     }
8876 }
8877 { list }
8878 {
8879     \MathMLintent
8880     { _ \tl_use:N \l__spinttent_spcomb_concept_tl -de:prefix($c) }
8881     {
8882         \__spinttent_inner_sep:
8883         \MathMLarg { c }
8884         {
8885             \MathMLintent { _elementos-con-repeticiones:infix($n,$m) }
8886             { \__spinttent_inner_sep: \__spinttent_spcomb_symbol: }
8887         }
8888     }
8889 }
8890 }
8891 }
8892 {
8893     \cs_set_eq:NN \__spinttent_spcomb_arg:nn \__spinttent_spcomb_arg_plain:nn
8894     \cs_set_eq:NN \__spinttent_spcomb_sep: \__spinttent_spcomb_sep_plain:
8895     \cs_set_eq:NN \__spinttent_spcomb_last:n \__spinttent_spcomb_last_plain:n
8896     \MathMLintent { _ \tl_use:N \l__spinttent_spcomb_read_arg_tl :nofix }
8897     { \__spinttent_inner_sep: \__spinttent_spcomb_symbol: }
8898 }
8899 }
8900 }

```

(End of definition for `\__spinttent_spcomb_mathml:`)

`\__spinttent_spcomb_exec:nn`

The function `\__spinttent_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__spinttent_spcomb_luaset_single_str`, and calls `\__spinttent_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 `\__spinttent_spcomb_mathml:`.

```

8901 \cs_new_protected:Nn \__spinttent_spcomb_exec:nn
8902 {
8903     \keys_set:nn { spinttent / spcomb } {#1}
8904     \__spinttent_spcomb_setup:
8905     \str_if_empty:NTF \l__spinttent_spcomb_problem_str
8906     {
8907         \__spinttent_luafun_spcomb_split:n { \exp_not:n {#2} }
8908         \bool_if:NTF \l__spinttent_spcomb_circular_bool
8909         {
8910             \str_if_eq:VnTF \l__spinttent_spcomb_luaset_single_str { true }
8911             { \str_set:Nn \l__spinttent_spcomb_mode_str { one } }
8912             { \str_set:Nn \l__spinttent_spcomb_mode_str { extra } }
8913         }
8914         {
8915             \bool_lazy_and:nnTF
8916             { \str_if_eq_p:Vn \l__spinttent_spcomb_kind_str { permutation } }
8917             { \bool_if_p:N \l__spinttent_spcomb_repeat_bool }
8918             {
8919                 \str_if_eq:VnTF \l__spinttent_spcomb_luaset_single_str { true }
8920                 { \str_set:Nn \l__spinttent_spcomb_mode_str { missing } }
8921                 { \str_set:Nn \l__spinttent_spcomb_mode_str { list } }
8922             }
8923         }
8924         \str_if_eq:VnTF \l__spinttent_spcomb_luaset_single_str { true }
8925         {

```

```

8926         \str_if_eq:VnTF \l__spinttent_spcomb_kind_str { permutation }
8927         { \str_set:Nn \l__spinttent_spcomb_mode_str { one } }
8928         { \str_set:Nn \l__spinttent_spcomb_mode_str { missing } }
8929     }
8930     { \str_set:Nn \l__spinttent_spcomb_mode_str { pair } }
8931 }
8932 }
8933 \str_case:VnF \l__spinttent_spcomb_mode_str
8934 {
8935     { extra }
8936     { \msg_error:nn { spinttent } { spcomb-extra-arg } }
8937     { missing }
8938     { \msg_error:nn { spinttent } { spcomb-missing-arg } }
8939 }
8940 {
8941     \__spinttent_luafun_spcomb_prepare:nnnn
8942     { \l__spinttent_spcomb_mode_str }
8943     { \l__spinttent_spcomb_luaset_raw_n_str }
8944     { \l__spinttent_spcomb_luaset_raw_k_str }
8945     {
8946         \bool_if:NTF \l__spinttent_spcomb_repeat_bool
8947         { false } { true }
8948     }
8949     \str_if_eq:VnTF \l__spinttent_spcomb_luaset_error_str { true }
8950     {
8951         \msg_error:nnee { spinttent } { spcomb-invalid-arg }
8952         { \l__spinttent_spcomb_luaset_raw_n_str }
8953         { \l__spinttent_spcomb_luaset_raw_k_str }
8954     }
8955     { \__spinttent_spcomb_mathml: }
8956 }
8957 }
8958 {
8959     \msg_error:nne { spinttent } { spcomb-invalid-keys }
8960     { \l__spinttent_spcomb_problem_str }
8961 }
8962 }

```

(End of definition for `\__spinttent_spcomb_exec:nn`.)

`\__spinttent_spcomb_begin:nn`

The function `\__spinttent_spcomb_begin:nn` opens a local group, activates the scope attribute and delegates execution to `\__spinttent_spcomb_exec:nn`.

```

8963 \cs_new_protected:Nn \__spinttent_spcomb_begin:nn
8964 {
8965     \group_begin:
8966     \attribute \l__spinttent_scope_attr_int = 1
8967     \__spinttent_spcomb_exec:nn {#1} {#2}
8968     \group_end:
8969 }

```

(End of definition for `\__spinttent_spcomb_begin:nn`.)

`\spcomb`

The public command `\spcomb` enforces *math mode* and delegates execution to `\__spinttent_spcomb_begin:nn`, which opens a local group. The mandatory argument ‘`#2`’ is *n* or *n*, *k*, and Lua splits it.

```

8970 \NewDocumentCommand \spcomb { O{ } m }
8971 {
8972     \mode_if_math:TF
8973     { \__spinttent_spcomb_begin:nn {#1} {#2} }
8974     { \msg_error:nnn { spinttent } { need-math-mode } { \spcomb } }
8975 }

```

(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

`\__spinttent_spfact_read_arg_tl` The token list variable `\__spinttent_spfact_read_arg_tl` stores the key `read-arg`.

```
8976 \tl_new:N \__spinttent_spfact_read_arg_tl
```

(End of definition for `\__spinttent_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*.

```
8977 \msg_new:nnnn { spinttent } { spfact-empty-arg }
8978 {
8979   Empty~argument~for~'\token_to_str:N \spfact'\msg_line_context:.
8980 }
8981 {
8982   The~number~n~is~required.
8983 }
```

(End of definition for `spfact-empty-arg`.)

15.67.3 Definition of keys for \spfact

`silent` We define the *(keys)* `silent` and `read-arg`.

```
read-arg
unknown
8984 \keys_define:nn { spinttent / spfact }
8985 {
8986   silent .bool_set:N = \__spinttent_spfact_silent_bool,
8987   silent .initial:n = false,
8988   silent .value_required:n = true,
8989   read-arg .code:n =
8990     { \__spinttent_sanitizize_affix:Nn \__spinttent_spfact_read_arg_tl {#1} },
8991   read-arg .initial:n = ,
8992   read-arg .value_required:n = true,
8993   unknown .code:n =
8994     {
8995       \str_set:Nn \__spinttent_command_name_str { spfact }
8996       \__spinttent_unknown_keys_cmd:n {#1}
8997     },
8998 }
```

(End of definition for `silent`, `read-arg`, and `unknown`.)

15.67.4 Internal functions for \spfact

`\__spinttent_spfact_exec:nn` The function `\__spinttent_spfact_exec:nn` sets the *(keys)* in ‘#1’ and calls `\__spinttent_spfact_mathml:n` with the argument in ‘#2’.

```
8999 \cs_new_protected:Nn \__spinttent_spfact_exec:nn
9000 {
9001   \keys_set:nn { spinttent / spfact } {#1}
9002   \__spinttent_spfact_mathml:n {#2}
9003 }
```

(End of definition for `\__spinttent_spfact_exec:nn`.)

`\__spinttent_spfact_mathml:n` The function `\__spinttent_spfact_mathml:n` prints the factorial of ‘#1’ inside one native `\MathMLintent`, preceded by the outer ghost `\__spinttent_invisible_sep:` and followed by the inner ghost `\__spinttent_inner_sep:`. With `silent=true` the intent is ‘_’; with `read-arg` given it is `_⟨read-arg⟩:nofix`; otherwise it is `_factorial:postfix($n)`.

```
9004 \cs_new_protected:Nn \__spinttent_spfact_mathml:n
9005 {
9006   \__spinttent_invisible_sep:
9007   \bool_if:NTF \__spinttent_spfact_silent_bool
9008     { \MathMLintent { _ } { \__spinttent_inner_sep: #1 \mathexclam } }
9009     {
9010       \tl_if_empty:NTF \__spinttent_spfact_read_arg_tl
9011         {
9012           \MathMLintent { _factorial:postfix($n) }
9013         }
9014         {
9015           \__spinttent_inner_sep:
9016           \MathMLarg { n } { \__spinttent_inner_sep: #1 }
9017           \mathexclam
9018         }
9019     }
```

```

9017         }
9018     }
9019     {
9020         \MathMLintent { _ \tl_use:N \l__spinttent_spfact_read_arg_tl :nofix }
9021         { \__spinttent_inner_sep: #1 \mathexclam }
9022     }
9023 }
9024 }

```

(End of definition for `\__spinttent_spfact_mathml:n`.)

`\spfact` The public command `\spfact` enforces *math mode*, opens a local group and delegates execution to `\__spinttent_spfact_exec:nn`.

```

9025 \NewDocumentCommand \spfact { 0{} m }
9026 {
9027     \mode_if_math:TF
9028     {
9029         \group_begin:
9030         \attribute \l__spinttent_scope_attr_int = 1
9031         \tl_if_blank:nTF {#2}
9032         { \msg_error:nn { spinttent } { spfact-empty-arg } }
9033         { \__spinttent_spfact_exec:nn {#1} {#2} }
9034     \group_end:
9035     }
9036     { \msg_error:nnn { spinttent } { need-math-mode } { \spfact } }
9037 }

```

(End of definition for `\spfact`. This function is documented on page 48.)

15.68 Finish package

Finish package implementation.

```

9038 \file_input_stop:
9039 \endpackage

```

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	<u>458</u> <u>1</u>
B	
bar	<u>844</u> <u>1</u>
C	
card	<u>816</u> <u>9</u>
cifras-sep	<u>312</u> , <u>589</u> , <u>875</u> , <u>1963</u> , <u>2038</u> , <u>2696</u>
circular	<u>8615</u>
Document class:	
l3doc	<u>58</u>
close	<u>4192</u>
cmd-key-unknown	<u>219</u>
cmd-key-value-unknown	<u>219</u>
Commands provided by spanish-mathml-intent :	
\setspintent	<u>64</u>
\spAcirc	<u>213</u>
\spAfig	<u>196</u>
\spFrec	<u>228</u>
\spOmega	<u>231</u>
\spPcirc	<u>213</u>
\spPerc	<u>226</u>
\spPfig	<u>196</u>
\spPoint	<u>158</u>
\spPoly	<u>187</u>
\spProb	<u>236</u>
\spProm	<u>219</u>
\spProp	<u>116</u>
\spPval	<u>232</u>
\spVol	<u>217</u>
\spabs	<u>153</u>
\spangle	<u>180</u>
\spang	<u>117</u>
\sparc	<u>183</u>
\spcase	<u>230</u>
\spcirc	<u>206</u>
\spcod	<u>150</u>
\spcomb	<u>246</u>
\spcomp	<u>239</u>
\spcoord	<u>130</u>
\spcuartil	<u>226</u>
\spdate	<u>94</u>
\spdecil	<u>226</u>
\spdesv	<u>222</u>
\spdiam	<u>200</u>
\spdiv	<u>103</u>
\spdom	<u>150</u>
\spdsym	<u>100</u>
\spfact	<u>248</u>
\spfuncomp	<u>147</u>
\spfun	<u>144</u>
\spinterval	<u>138</u>
\spmQ	<u>125</u>
\spmangle	<u>180</u>
\spmarc	<u>183</u>
\spmmax	<u>223</u>
\spmcd	<u>109</u>
\spmcm	<u>109</u>
\spmediana	<u>222</u>
\spmedia	<u>222</u>
\spmin	<u>223</u>
\spmoda	<u>222</u>
\spmoney	<u>85</u>
\spmside	<u>174</u>
\spmsym	<u>102</u>
\spmul	<u>105</u>
\spnD	<u>114</u>
\spnM	<u>114</u>
\spnZ	<u>119</u>
\spnewread	<u>98</u>
\spnewunit	<u>78</u>
\spnum	<u>72</u>
\sprad	<u>200</u>
\sprango	<u>223</u>
\sprayo	<u>170</u>
\sprazon	<u>115</u>
\spread	<u>98</u>
\sprecta	<u>167</u>
\sprec	<u>150</u>
\spriq	<u>223</u>
\spshort	<u>88</u>
\spside	<u>174</u>
\spsiglo	<u>91</u>
\sptime	<u>92</u>
\spunitalias	<u>78</u>
\spunit	<u>77</u>
\spusep	<u>96</u>
\spvarz	<u>222</u>
\spvec	<u>162</u>
concept	<u>2118</u> , <u>2356</u> , <u>3242</u> , <u>3634</u> , <u>4471</u> , <u>4581</u> , <u>4830</u> , <u>5090</u> , <u>5234</u> , <u>5451</u> , <u>5748</u> , <u>5904</u> , <u>6097</u> , <u>6511</u> , <u>6773</u> , <u>7083</u> , <u>7400</u>
concept*	<u>6097</u>
currency	<u>875</u>
D	
diameter	<u>6773</u>
I	
inline	<u>2541</u>
int commands:	
\int_set:Nn	<u>70</u>
J	
join-word	<u>2774</u>
K	
Keys for \spabs provided by spanish-mathml-intent :	
close	<u>150–152</u>
read-arg	<u>150</u> , <u>151</u>
silent	<u>150</u> , <u>151</u>
size	<u>150</u> , <u>151</u>
Keys for \spAcirc provided by spanish-mathml-intent :	
concept	<u>208</u>
prefix	<u>207</u> , <u>208</u>
print-sym	<u>207–209</u>
read-arg	<u>207</u> , <u>208</u>
read-cmd	<u>207</u> , <u>208</u>
read-sty	<u>207</u> , <u>208</u>
read-sub	<u>208</u>

silent	208	read-arg	196
sub-index	207, 208	read-cmd	196
Keys for \spAfig provided by spanish-mathml-intent :		read-sub	196
concept	189	silent	196
prefix	187, 189	sub-index	196
print-sym	187–189	upper-case	196
read-arg	187, 189, 190	Keys for \spdiv provided by spanish-mathml-intent :	
read-cmd	187, 189	cifras-sep	102
read-sub	189	notation-sym	102
silent	189	read-parens	102
Keys for \spangle provided by spanish-mathml-intent :		read-period-sep	102
concept	176	read-sign	102
prefix	175, 176	read-sym	102
print-sym	176	set-sym	102
read-cmd	175, 176	wrap-parens	102
recto	175, 176	Keys for \spdom provided by spanish-mathml-intent :	
silent	176	prefix	148
Keys for \sparc provided by spanish-mathml-intent :		silent	148
concept	181	Keys for \spdsym provided by spanish-mathml-intent :	
prefix	180, 181	prefix	99
read-cmd	180, 181	read-sym	99
silent	181	set-sym	99
Keys for \spcase provided by spanish-mathml-intent :		suffix	99
read-arg	229, 230	Keys for \spfact provided by spanish-mathml-intent :	
silent	229, 230	read-arg	246, 247
Keys for \spcirc provided by spanish-mathml-intent :		silent	246, 247
concept	202	Keys for \spFrec provided by spanish-mathml-intent :	
diameter	202	read-arg	227, 228
prefix	201, 202	read-sub	227
print-sym	201–203	silent	227
radio-sty	201, 202	sub-index	227
read-arg	201, 202	Keys for \spfuncomp provided by spanish-mathml-intent :	
read-cmd	201, 202	silent	145
read-sty	201, 202	Keys for \spfun provided by spanish-mathml-intent :	
read-sub	202	concept	138, 139
silent	202	prefix	138, 139
sub-index	201, 202	read-cmd	138, 139
Keys for \spcomb provided by spanish-mathml-intent :		silent	138, 139
circular	239, 240	Keys for \spinterval provided by spanish-mathml-intent :	
kind	239, 240	concept	132, 135
print-sty	239, 240	prefix	131, 132, 135
read-arg	239, 240	read-arg	131, 132, 135
repeat	239, 240	read-cmd	131, 132, 136
silent	240	silent	132, 135, 137
Keys for \spcomp provided by spanish-mathml-intent :		size	131–133
bar	236–238	Keys for \spmangle provided by spanish-mathml-intent :	
read-arg	236, 237	concept	176
silent	237	prefix	175, 176
wrap-parens	236, 237	print-sym	175, 176
Keys for \spcoord provided by spanish-mathml-intent :		read-cmd	175, 176
silent	127	recto	175, 176
Keys for \spcuartil provided by spanish-mathml-intent :		silent	176
read-arg	224	Keys for \spmarc provided by spanish-mathml-intent :	
silent	224	concept	181
Keys for \spdecil provided by spanish-mathml-intent :		prefix	180, 181
read-arg	224	read-cmd	180, 181
silent	224	silent	181
Keys for \spdesv provided by spanish-mathml-intent :		Keys for \spmax provided by spanish-mathml-intent :	
read-arg	219	read-arg	220
read-sub	219	silent	220
sigma	220	Keys for \spmcd provided by spanish-mathml-intent :	
silent	219	concept	106
sub-index	219	prefix	106
Keys for \spdiam provided by spanish-mathml-intent :		read-sty	106
concept	196	result	106

silent	106	result-num	111
upper-case	106	result	111
Keys for \spmc provided by spanish-mathml-intent:		silent	111
concept	106	Keys for \spnM provided by spanish-mathml-intent:	
prefix	106	concept	111
read-sty	106	prefix	111
result	106	result-num	111
silent	106	result	111
upper-case	106	silent	111
Keys for \spmediana provided by spanish-mathml-intent:		Keys for \spnum provided by spanish-mathml-intent:	
read-arg	219	cifras-sep	66, 68
read-sub	219	notation-sym	67, 68
silent	219	read-period-sep	67–69
sub-index	219	read-sign	67, 68
Keys for \spmedia provided by spanish-mathml-intent:		Keys for \spnZ provided by spanish-mathml-intent:	
read-arg	219	cifras-sep	118
read-sub	219	read-parens	118
silent	219	read-sign	118
sub-index	219	wrap-parens	118
Keys for \spmin provided by spanish-mathml-intent:		Keys for \spOmega provided by spanish-mathml-intent:	
read-arg	220	card	230, 231
silent	220	read-arg	230
Keys for \spmoda provided by spanish-mathml-intent:		silent	230
read-arg	219	sub-index	230
read-sub	219	Keys for \spPcirc provided by spanish-mathml-intent:	
silent	219	concept	208
sub-index	219	prefix	207, 208
Keys for \spmoney provided by spanish-mathml-intent:		print-sym	207–209
cifras-sep	80	read-arg	207, 208
currency	80	read-cmd	207, 208
money-math	80	read-sty	207, 208
sub-units	79, 80	read-sub	208
Keys for \spmQ provided by spanish-mathml-intent:		silent	208
join-word	119, 120, 122	sub-index	207, 208
read-parens	120, 121	Keys for \spPerc provided by spanish-mathml-intent:	
silent	120, 124, 125	read-arg	224
use-dfrac	120–123	silent	224
use-spN	120, 123	Keys for \spPfig provided by spanish-mathml-intent:	
wrap-parens	119–121	concept	189
Keys for \spmside provided by spanish-mathml-intent:		prefix	187, 189
concept	171	print-sym	187–189
prefix	170, 171	read-arg	187, 189, 190
print-sym	170, 171, 196	read-cmd	187, 189
read-arg	170, 171	read-sub	189
read-cmd	170, 171, 173	silent	189
read-sty	170, 171	Keys for \spPoint provided by spanish-mathml-intent:	
silent	171, 173	concept	156
Keys for \spmsym provided by spanish-mathml-intent:		prefix	156
prefix	100, 101	read-cmd	156
print-sym	101	silent	156
read-sym	100, 101	Keys for \spPoly provided by spanish-mathml-intent:	
set-sym	100, 101	concept	184
suffix	100, 101	prefix	184
Keys for \spmul provided by spanish-mathml-intent:		print-sym	184
cifras-sep	104	read-cmd	184
notation-sym	104	silent	184
read-parens	104	Keys for \spProb provided by spanish-mathml-intent:	
read-period-sep	104	read-arg	233
read-sign	104	silent	233
read-sym	104	Keys for \spProm provided by spanish-mathml-intent:	
set-sym	104	read-arg	218
wrap-parens	104	silent	218
Keys for \spnD provided by spanish-mathml-intent:		Keys for \spPval provided by spanish-mathml-intent:	
concept	111	silent	231
prefix	111	use-dfrac	231

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	220
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	8615

M

mathcal-fallback	86
mc-too-few-args	2342
md-invalid-arg	2342
Lua module provided by spanish-mathml-intent :	
spintent.lua	58–61, 65–67, 72, 77, 78, 80, 85, 88, 90, 91, 93, 105, 106, 110, 116, 119, 125, 138, 144, 156
money-math	875

N

notation-sym	312, 589, 1963, 2038
--------------	----------------------

P

Packages :

expl3	58, 65, 67
lua-unicode-math	59
luamml	61, 119, 122, 183, 186, 206
tagpdf	85, 86, 89
unicode-math	59, 95
prefix	1802, 1877, 2118, 2356, 3242, 3634, 4081, 4471, 4581, 4830, 5090, 5234, 5451, 5748, 5904, 6097, 6773, 7083
print-cdot	589
print-era	1307
print-sty	8615
print-sym	1877, 5234, 5451, 5904, 6097, 6511, 6773, 7083

R

radio-sty	6773
read-arg	1165, 1718, 3242, 4192, 5234, 6097, 6511, 6773, 7083, 7400, 7601, 7666, 7704, 7890, 8026, 8119, 8169, 8280, 8441, 8615, 8984
read-cdot	589
read-cmd	3242, 3634, 4471, 4581, 4830, 5090, 5234, 5451, 5748, 5904, 6097, 6511, 6773, 7083, 7400
read-parens	1963, 2038, 2696, 2774
read-period-sep	312, 589, 1963, 2038
read-sign	312, 1963, 2038, 2696
read-slash	589
read-sty	2118, 5090, 5234, 6773, 7083
read-sub	6097, 6511, 6773, 7083, 7400, 7666, 8026
read-sym	1802, 1877, 1963, 2038
recto	5451
repeat	8615
result	2118, 2356
result-num	2356
right-angle-sqr-fallback	86

S

seq commands:

<code>\seq_set_from_clist:Nn</code>	70
<code>set-sym</code>	1802 , 1877 , 1963 , 2038
<code>\setspintent</code>	6 , 63 , 206
<code>sigma</code>	7694
<code>silent</code>	1576 , 1718 , 2118 , 2356 , 2541 , 2774 , 3036 , 3242 , 3634 , 3895 , 4081 , 4192 , 4471 , 4581 , 4830 , 5090 , 5234 , 5451 , 5748 , 5904 , 6097 , 6511 , 6773 , 7083 , 7400 , 7601 , 7666 , 7704 , 7890 , 8026 , 8119 , 8169 , 8211 , 8280 , 8441 , 8615 , 8984
<code>size</code>	1576 , 3242 , 4192
<code>small-caps</code>	1165
<code>\spabs</code>	41 , 150 , 4326
<code>spabs-empty-arg</code>	4185
<code>\spAcirc</code>	35 , 207 , 7367
<code>spAcirc-invalid-arg</code>	7074
<code>\spAfig</code>	30 , 187 , 6476
<code>spAfig-invalid-arg</code>	6057
<code>spAfig-invalid-points</code>	6057
<code>spAfig-read-arg-key</code>	6057
<code>spAfig-sym-cardinality-conflict</code>	6057
<code>spAfig-sym-letter-conflict</code>	6057
<code>\spang</code>	29 , 116 , 2679
<code>spang-invalid-arg</code>	2634
<code>\spangle</code>	27 , 175 , 5706
<code>spangle-invalid-arg</code>	5440
<code>\sparc</code>	29 , 180 , 5869
<code>sparc-invalid-points</code>	5738
<code>\spcase</code>	44 , 229 , 8153
<code>spcase-invalid-letter</code>	8112
<code>\spcirc</code>	34 , 200 , 7046
<code>spcirc-invalid-arg</code>	6766
<code>\spcod</code>	40 , 148 , 4148
<code>\spcomb</code>	47 , 239 , 8970
<code>spcomb-extra-arg</code>	8584
<code>spcomb-invalid-arg</code>	8584
<code>spcomb-invalid-keys</code>	8584
<code>spcomb-missing-arg</code>	8584
<code>\spcomp</code>	46 , 236 , 8554
<code>spcomp-empty-arg</code>	8434
<code>\spcoord</code>	23 , 126 , 3187
<code>spcoord-invalid-arg</code>	3029
<code>\spcuartil</code>	43 , 7971
<code>\spdate</code>	15 , 93 , 1548
<code>spdate-invalid-arg</code>	1523
<code>\spdecil</code>	43 , 7971
<code>\spdesv</code>	42 , 7817
<code>\spdiam</code>	33 , 196 , 6727
<code>\spdiv</code>	18 , 102 , 2021
<code>spdiv-empty-arg</code>	1956
<code>\spdom</code>	40 , 148 , 4148
<code>spdom-invalid-arg</code>	4074
<code>\spdsym</code>	17 , 99 , 1862
<code>\spfact</code>	47 , 246 , 9025
<code>spfact-empty-arg</code>	8977
<code>\spFrec</code>	44 , 226 , 8096
<code>spFrec-invalid-sub-index</code>	8011
<code>spFrec-invalid-symbol</code>	8011
<code>\spfun</code>	40 , 138 , 3868
<code>spfun-invalid-arg</code>	3626

<code>\spfuncomp</code>	41 , 144 , 4057
<code>spfuncomp-invalid-arg</code>	3888
spintent internal commands:	
<code>__spintent_collapse_hyphens:N</code>	138
<code>\l__spintent_command_name_str</code>	219
<code>\l__spintent_geo_final_print_tl</code>	6046
<code>\l__spintent_geo_luaset_angulo_case_-str</code>	3010
<code>\l__spintent_geo_luaset_circ_radio_raw_-str</code>	6749
<code>\l__spintent_geo_luaset_circ_radio_type_-str</code>	6749
<code>\l__spintent_geo_luaset_concept_str</code> ...	3008
<code>\l__spintent_geo_luaset_error_str</code> ...	3000
<code>\l__spintent_geo_luaset_intent_str</code> ...	3000
<code>\l__spintent_geo_luaset_is_name_str</code> ...	3003
<code>\l__spintent_geo_luaset_letra_conector_-str</code>	3012
<code>\l__spintent_geo_luaset_letra_str</code> ...	3003
<code>\l__spintent_geo_luaset_letra_word_str</code>	3012
<code>\l__spintent_geo_luaset_nombre_es_num_-str</code>	3010
<code>\l__spintent_geo_luaset_points_clist</code> ..	3003
<code>\l__spintent_geo_luaset_poly_sym_str</code> ..	3008
<code>\l__spintent_geo_luaset_print_tl</code>	3000
<code>\l__spintent_geo_luaset_sub_spoken_str</code> ..	3012
<code>\l__spintent_geo_luaset_sub_str</code>	3003
<code>\l__spintent_geo_luaset_tipo_str</code>	3003
<code>\l__spintent_geo_print_sym_str</code>	6048
<code>\l__spintent_geo_temp_tl</code>	6055
<code>__spintent_inner_sep:</code>	126
<code>__spintent_invisible_sep:</code>	122
<code>\c__spintent_invisible_sep_int</code>	134
<code>__spintent_luafun_clean_split_and_-set:n</code>	256
<code>__spintent_luafun_mrow_intent_arg:nn</code> ..	122
<code>__spintent_luafun_set_debug_ghost:n</code> ...	77
<code>__spintent_luafun_sfun_parse_and_set:n</code> ..	82
<code>__spintent_luafun_spmoney_lookup_data:n</code> ..	82
<code>__spintent_luafun_spmQ_scale_int:nn</code> ...	82
<code>__spintent_luafun_spunit_parse_-product:nn</code>	82
<code>\l__spintent_luaset_arg_above_tl</code>	274
<code>\l__spintent_luaset_arg_below_tl</code>	274
<code>\l__spintent_luaset_cal_fam_ok_str</code>	73
<code>\l__spintent_luaset_dec_is_all_zeros_-str</code>	267
<code>\l__spintent_luaset_dec_sep_tl</code>	256
<code>\l__spintent_luaset_decimal_period_str</code> ..	267
<code>\l__spintent_luaset_denom_has_number_-str</code>	274
<code>\l__spintent_luaset_exponent_tl</code>	267
<code>\l__spintent_luaset_format_part_dec_str</code> ..	262
<code>\l__spintent_luaset_format_part_int_str</code> ..	262
<code>\l__spintent_luaset_has_exponent_str</code> ..	267
<code>\l__spintent_luaset_has_integer_str</code> ...	256
<code>\l__spintent_luaset_has_units_str</code>	274
<code>\l__spintent_luaset_millions_str</code>	267
<code>\l__spintent_luaset_not_decimal_not_-period_str</code>	267
<code>\l__spintent_luaset_not_decimal_period_-str</code>	267
<code>\l__spintent_luaset_only_part_dec_str</code> ..	262

<code>\l__spintent_luaset_only_part_int_str</code> .	262	<code>\l__spintent_spAfig_prefix_tl</code>	6039
<code>\l__spintent_luaset_only_part_period-str</code>	262	<code>\l__spintent_spAfig_print_sym_str</code>	6039
<code>\l__spintent_luaset_part_dec_tl</code>	256	<code>\l__spintent_spAfig_read_arg_tl</code>	6039
<code>\l__spintent_luaset_part_int_tl</code>	256	<code>\l__spintent_spang_exec:n</code>	2642
<code>\l__spintent_luaset_part_period_tl</code> . . .	256	<code>\l__spintent_spang_luaset_error_str</code> . . .	2630
<code>\l__spintent_luaset_scope_attr_str</code>	73	<code>\l__spintent_spang_luaset_grado_str</code> . . .	2630
<code>\l__spintent_luaset_sign_tl</code>	256	<code>\l__spintent_spang_luaset_minuto_str</code> . .	2630
<code>\l__spintent_luaset_status_str</code>	274	<code>\l__spintent_spang_luaset_segundo_str</code> .	2630
<code>\l__spintent_luaset_sym_fam_ok_str</code>	73	<code>\l__spintent_spang_mathml:</code>	2652
<code>\l__spintent_mathcal:n</code>	100	<code>\l__spintent_spangle_exec:nnn</code>	5510
<code>\l__spintent_mathmlarg_point:nnn</code>	4343	<code>\l__spintent_spangle_intent_arg_tl</code> . . .	5438
<code>\l__spintent_mathmlarg_point_kv_tl</code> . . .	4339	<code>\l__spintent_spangle_intent_tl</code>	5433
<code>\l__spintent_mathmlarg_point_mode_str</code> .	4341	<code>\l__spintent_spangle_kv_tl</code>	5438
<code>\l__spintent_mathmlarg_point_ref_tl</code> . . .	4339	<code>\l__spintent_spangle_mathml:n</code>	5547
<code>\l__spintent_mathmlarg_point_silent-bool</code>	4341	<code>\l__spintent_spangle_mode_str</code>	5428
<code>\l__spintent_mc_args_seq</code>	2108	<code>\l__spintent_spangle_prefix_tl</code>	5428
<code>\l__spintent_mc_idx_int</code>	2108	<code>\l__spintent_spangle_print_sym_now-bool</code>	5433
<code>\l__spintent_mc_luaset_all_numeric_str</code>	2106	<code>\l__spintent_spangle_pt_idx_int</code>	5435
<code>\l__spintent_mc_luaset_types_clist</code> . . .	2106	<code>\l__spintent_spangle_pt_seq</code>	5435
<code>\l__spintent_mc_mathml_args:n</code>	2179	<code>\l__spintent_spangle_refs_tl</code>	5435
<code>\l__spintent_mc_process_args:nn</code>	2159	<code>\l__spintent_spangle_sym:n</code>	5491
<code>\l__spintent_mc_raw_args_seq</code>	2108	<code>\l__spintent_sparc_exec:nnn</code>	5792
<code>\l__spintent_mc_types_seq</code>	2108	<code>\l__spintent_sparc_intent_arg_tl</code>	5737
<code>\l__spintent_md_out_seq</code>	2340	<code>\l__spintent_sparc_intent_tl</code>	5733
<code>\l__spintent_point_sep:</code>	130	<code>\l__spintent_sparc_mathml:n</code>	5810
<code>\l__spintent_print_spunit_hierarchy:</code> . . .	679	<code>\l__spintent_sparc_mode_str</code>	5728
<code>\l__spintent_right_angle_sqr:</code>	113	<code>\l__spintent_sparc_prefix_tl</code>	5728
<code>\l__spintent_sanitize_affix:Nn</code>	146	<code>\l__spintent_sparc_print_arc:n</code>	5786
<code>\l__spintent_saved_luamml_flag_int</code> . . .	2751	<code>\l__spintent_sparc_pt_idx_int</code>	5734
<code>\l__spintent_scope_attr_int</code>	80	<code>\l__spintent_sparc_pt_seq</code>	5734
<code>\l__spintent_setspintent_input_arg_cmd-tl</code>	169	<code>\l__spintent_sparc_refs_tl</code>	5734
<code>\l__spintent_setspintent_parse_args:nn</code> .	189	<code>\l__spintent_sparc_wide_arc:n</code>	5782
<code>\l__spintent_setspintent_set_keys:nn</code> . . .	178	<code>\l__spintent_spcase_concept_tl</code>	8110
<code>\l__spintent_sp_rad_diam_exec:nn</code>	6722	<code>\l__spintent_spcase_exec:n</code>	8134
<code>\l__spintent_sp_rad_diam_letra_tl</code>	6502	<code>\l__spintent_spcase_letter_str</code>	8110
<code>\l__spintent_sp_rad_diam_mathml:n</code>	6558	<code>\l__spintent_spcase_read_arg_tl</code>	8108
<code>\l__spintent_sp_rad_diam_read_arg_full-tl</code>	6502	<code>\l__spintent_spcase_sub_index_str</code>	8108
<code>\l__spintent_sp_rad_diam_read_arg-print:n</code>	6684	<code>\l__spintent_spcirc_articulo_str</code>	6758
<code>\l__spintent_sp_rad_diam_word_tl</code>	6502	<code>\l__spintent_spcirc_build_concept:</code>	6919
<code>\l__spintent_spabs_close_bool</code>	4183	<code>\l__spintent_spcirc_connector_arg_tl</code> . .	6760
<code>\l__spintent_spabs_close_mode_str</code>	4178	<code>\l__spintent_spcirc_connector_tl</code>	6760
<code>\l__spintent_spabs_delim_l_tl</code>	4181	<code>\l__spintent_spcirc_exec:nn</code>	6974
<code>\l__spintent_spabs_delim_r_tl</code>	4181	<code>\l__spintent_spcirc_has_arg_bool</code>	6760
<code>\l__spintent_spabs_exec:nn</code>	4282	<code>\l__spintent_spcirc_intent_arg_tl</code>	6760
<code>\l__spintent_spabs_luaset_close_str</code> . . .	4183	<code>\l__spintent_spcirc_intent_tl</code>	6760
<code>\l__spintent_spabs_mathml:n</code>	4288	<code>\l__spintent_spcirc_mathml:</code>	6991
<code>\l__spintent_spabs_read_arg_tl</code>	4178	<code>\l__spintent_spcirc_mode_str</code>	6751
<code>\l__spintent_spabs_set_close:n</code>	4268	<code>\l__spintent_spcirc_prefix_tl</code>	6751
<code>\l__spintent_spabs_set_delims:</code>	4237	<code>\l__spintent_spcirc_print_sym_str</code>	6751
<code>\l__spintent_spabs_size_str</code>	4178	<code>\l__spintent_spcirc_radio_arg_tl</code>	6760
<code>\l__spintent_spAcirc_mode_str</code>	7059	<code>\l__spintent_spcirc_radio_print:</code>	6902
<code>\l__spintent_spAcirc_prefix_tl</code>	7059	<code>\l__spintent_spcirc_radio_sty_str</code>	6751
<code>\l__spintent_spAcirc_print_sym_str</code> . . .	7059	<code>\l__spintent_spcirc_read_arg_tl</code>	6751
<code>\l__spintent_spAcirc_read_arg_tl</code>	7059	<code>\l__spintent_spcirc_read_sty_str</code>	6751
<code>\l__spintent_spAcirc_read_sty_str</code>	7059	<code>\l__spintent_spcirc_sub_index_str</code>	6751
<code>\l__spintent_spAcirc_sub_index_str</code> . . .	7059	<code>\l__spintent_spcirc_sym:</code>	6871
<code>\l__spintent_spAfig_mode_str</code>	6039	<code>\l__spintent_spcirc_word_and_article:</code> . .	6881
		<code>\l__spintent_spcirc_word_tl</code>	6758
		<code>\l__spintent_spcircfig_build_concept:nn</code>	7230
		<code>\l__spintent_spcircfig_build_short:n</code> . .	7189

<code>__spintent_spcircfig_del_word:n</code>	7176	<code>\l__spintent_spcord_luaset_error_str</code> .	3015
<code>\l__spintent_spcircfig_del_word_tl</code> . . .	7070	<code>\l__spintent_spcord_luaset_sep_str</code> . . .	3015
<code>__spintent_spcircfig_exec:nnn</code>	7277	<code>\l__spintent_spcord_luaset_x_composite_</code> <code>str</code>	3017
<code>__spintent_spcircfig_finish_intent:nn</code>	7214	<code>\l__spintent_spcord_luaset_x_has_frac_</code> <code>str</code>	3017
<code>\l__spintent_spcircfig_intent_arg_tl</code> . .	7070	<code>\l__spintent_spcord_luaset_x_sign_str</code>	3017
<code>\l__spintent_spcircfig_intent_tl</code>	7070	<code>\l__spintent_spcord_luaset_x_type_str</code>	3017
<code>\l__spintent_spcircfig_luaset_comma_</code> <code>str</code>	7068	<code>\l__spintent_spcord_luaset_x_value_</code> <code>str</code>	3017
<code>__spintent_spcircfig_mathml:nn</code>	7324	<code>\l__spintent_spcord_luaset_y_composite_</code> <code>str</code>	3017
<code>\l__spintent_spcircfig_op_tl</code>	7070	<code>\l__spintent_spcord_luaset_y_has_frac_</code> <code>str</code>	3017
<code>\l__spintent_spcircfig_short_bool</code>	7069	<code>\l__spintent_spcord_luaset_y_sign_str</code>	3017
<code>__spintent_spcircfig_sym:n</code>	7152	<code>\l__spintent_spcord_luaset_y_type_str</code>	3017
<code>__spintent_spcircfig_word_and_</code> <code>article:n</code>	7158	<code>\l__spintent_spcord_luaset_y_value_</code> <code>str</code>	3017
<code>__spintent_spcomb_arg:nn</code>	8731	<code>__spintent_spcord_mathml:N</code>	3152
<code>__spintent_spcomb_arg_ann:nn</code>	8731	<code>__spintent_spcord_print:n</code>	3109
<code>__spintent_spcomb_arg_plain:nn</code>	8731	<code>\l__spintent_spcord_raw_arg_tl</code>	3025
<code>__spintent_spcomb_begin:nn</code>	8963	<code>__spintent_spcord_set_wrap_parens_</code> <code>size:</code>	3047
<code>\l__spintent_spcomb_concept_tl</code>	8577	<code>__spintent_spcord_value:n</code>	3100
<code>__spintent_spcomb_exec:nn</code>	8901	<code>\l__spintent_spcord_wrap_parens_l_tl</code> .	3025
<code>\l__spintent_spcomb_kind_str</code>	8574	<code>\l__spintent_spcord_wrap_parens_r_tl</code> .	3025
<code>__spintent_spcomb_last:n</code>	8731	<code>\l__spintent_spcuartil_read_arg_tl</code> . . .	7877
<code>__spintent_spcomb_last_ann:n</code>	8731	<code>__spintent_spdate_exec:n</code>	1531
<code>__spintent_spcomb_last_plain:n</code>	8731	<code>\l__spintent_spdate_luaset_error_str</code> . .	1521
<code>\l__spintent_spcomb_last_tl</code>	8577	<code>\l__spintent_spdate_luaset_output_str</code> .	1521
<code>\l__spintent_spcomb_letter_tl</code>	8577	<code>__spintent_spdate_mathml:</code>	1541
<code>__spintent_spcomb_list:</code>	8759	<code>\l__spintent_spdecil_read_arg_tl</code>	7877
<code>\l__spintent_spcomb_list_seq</code>	8577	<code>\l__spintent_spdesv_read_arg_tl</code>	7656
<code>\l__spintent_spcomb_luaset_error_str</code> . .	8570	<code>\l__spintent_spdesv_sub_index_str</code>	7656
<code>\l__spintent_spcomb_luaset_k_str</code>	8570	<code>\l__spintent_spdiam_mode_str</code>	6505
<code>\l__spintent_spcomb_luaset_list_tl</code> . . .	8570	<code>\l__spintent_spdiam_read_arg_tl</code>	6505
<code>\l__spintent_spcomb_luaset_n_str</code>	8570	<code>\l__spintent_spdiam_sub_index_str</code>	6505
<code>\l__spintent_spcomb_luaset_raw_k_str</code> . .	8567	<code>__spintent_spdiv_exec:nnn</code>	2008
<code>\l__spintent_spcomb_luaset_raw_n_str</code> . .	8567	<code>__spintent_spdiv_mathml:nn</code>	1992
<code>\l__spintent_spcomb_luaset_single_str</code> .	8567	<code>__spintent_spdiv_render_op:nn</code>	1983
<code>__spintent_spcomb_mathml:</code>	8825	<code>\l__spintent_spdom_cmd_str</code>	4068
<code>\l__spintent_spcomb_mode_str</code>	8577	<code>__spintent_spdom_exec:nnnnn</code>	4134
<code>\l__spintent_spcomb_print_sty_str</code>	8574	<code>\l__spintent_spdom_intent_arg_tl</code>	4071
<code>\l__spintent_spcomb_problem_str</code>	8577	<code>\l__spintent_spdom_luaset_error_str</code> . . .	4073
<code>\l__spintent_spcomb_read_arg_tl</code>	8574	<code>__spintent_spdom_mathml:</code>	4105
<code>__spintent_spcomb_sep:</code>	8731	<code>__spintent_spdom_name:</code>	4096
<code>__spintent_spcomb_sep_ann:</code>	8731	<code>\l__spintent_spdom_op_tl</code>	4068
<code>__spintent_spcomb_sep_plain:</code>	8731	<code>\l__spintent_spdom_prefix_tl</code>	4071
<code>__spintent_spcomb_setup:</code>	8667	<code>\l__spintent_spdom_word_tl</code>	4068
<code>__spintent_spcomb_symbol:</code>	8769	<code>__spintent_spdsym_mathml:</code>	1849
<code>\l__spintent_spcomb_use_binom_bool</code> . . .	8577	<code>\l__spintent_spdsym_prefix_tl</code>	1798
<code>\l__spintent_spcomp_compound_bool</code>	8431	<code>\l__spintent_spdsym_read_sym_tl</code>	1798
<code>\l__spintent_spcomp_do_paren_bool</code>	8431	<code>\l__spintent_spdsym_suffix_tl</code>	1798
<code>__spintent_spcomp_event:n</code>	8473	<code>\l__spintent_spdsym_symbol_tl</code>	1798
<code>__spintent_spcomp_exec:nn</code>	8508	<code>__spintent_spfact_exec:nn</code>	8999
<code>\l__spintent_spcomp_luaset_compound_</code> <code>str</code>	8431	<code>__spintent_spfact_mathml:n</code>	9004
<code>__spintent_spcomp_mathml:n</code>	8514	<code>\l__spintent_spfact_read_arg_tl</code>	8976
<code>\l__spintent_spcomp_read_arg_tl</code>	8429	<code>__spintent_spfig_build_concept_</code> <code>vertices:n</code>	6343
<code>__spintent_spcomp_set_paren:n</code>	8473	<code>__spintent_spfig_build_refs:</code>	6328
<code>__spintent_spcomp_typeset:n</code>	8502	<code>__spintent_spfig_build_simple:n</code>	6193
<code>\l__spintent_spcomp_wrap_parens_str</code> . . .	8429	<code>\l__spintent_spfig_desc_tl</code>	6049
<code>__spintent_spcoord_exec:nn</code>	3169		
<code>__spintent_spcoord_inner_sep:n</code>	3132		
<code>__spintent_spcoord_intent_infix:</code>	3138		
<code>\l__spintent_spcoord_intent_str</code>	3025		

<code>__spintent_spfexec:nnn</code>	6167	<code>\l__spintent_spfuncomp_tipo_str</code>	3883
<code>\l__spintent_spfexec_intent_arg_tl</code>	6055	<code>\l__spintent_spinterval_delim_l_tl</code>	3226
<code>\l__spintent_spfexec_intent_tl</code>	6049	<code>\l__spintent_spinterval_delim_r_tl</code>	3226
<code>__spintent_spfexec_mathml:n</code>	6434	<code>__spintent_spinterval_exec:nn</code>	3584
<code>\l__spintent_spfexec_op_tl</code>	6049	<code>__spintent_spinterval_inner_sep:n</code>	3480
<code>__spintent_spfexec_points_loop:n</code>	6422	<code>\l__spintent_spinterval_intent_arg_tl</code>	3222
<code>\l__spintent_spfexec_pt_idx_int</code>	6052	<code>\l__spintent_spinterval_intent_group-</code> <code>tl</code>	3222
<code>\l__spintent_spfexec_pt_seq</code>	6052	<code>\l__spintent_spinterval_luaset_a_braced-</code> <code>str</code>	3205
<code>\l__spintent_spfexec_refs_tl</code>	6052	<code>\l__spintent_spinterval_luaset_a-</code> <code>composite_str</code>	3205
<code>__spintent_spfexec_sym:</code>	6159	<code>\l__spintent_spinterval_luaset_a-</code> <code>inf_str</code>	3205
<code>\l__spintent_spfexec_symbol_tl</code>	6046	<code>\l__spintent_spinterval_luaset_a_integer-</code> <code>str</code>	3205
<code>\l__spintent_spFrec_concept_tl</code>	8009	<code>\l__spintent_spinterval_luaset_a_short-</code> <code>str</code>	3205
<code>__spintent_spFrec_exec:n</code>	8048	<code>\l__spintent_spinterval_luaset_a_sign-</code> <code>str</code>	3205
<code>\l__spintent_spFrec_read_arg_tl</code>	8007	<code>\l__spintent_spinterval_luaset_a_tall-</code> <code>str</code>	3205
<code>\l__spintent_spFrec_sub_index_str</code>	8007	<code>\l__spintent_spinterval_luaset_a_type-</code> <code>str</code>	3205
<code>\l__spintent_spFrec_symbol_str</code>	8009	<code>\l__spintent_spinterval_luaset_a_value-</code> <code>str</code>	3205
<code>__spintent_spfun_aname:n</code>	3665	<code>\l__spintent_spinterval_luaset_b_braced-</code> <code>str</code>	3205
<code>\l__spintent_spfun_aname_tl</code>	3620	<code>\l__spintent_spinterval_luaset_b-</code> <code>composite_str</code>	3205
<code>__spintent_spfun_arg:n</code>	3735	<code>\l__spintent_spinterval_luaset_b_inf-</code> <code>str</code>	3205
<code>__spintent_spfun_args:</code>	3758	<code>\l__spintent_spinterval_luaset_b_integer-</code> <code>str</code>	3205
<code>__spintent_spfun_args_plain:</code>	3767	<code>\l__spintent_spinterval_luaset_b_short-</code> <code>str</code>	3205
<code>__spintent_spfun_body:</code>	3810	<code>\l__spintent_spinterval_luaset_b_sign-</code> <code>str</code>	3205
<code>\l__spintent_spfun_call_tl</code>	3620	<code>\l__spintent_spinterval_luaset_b_tall-</code> <code>str</code>	3205
<code>\l__spintent_spfun_concept_bool</code>	3615	<code>\l__spintent_spinterval_luaset_b_type-</code> <code>str</code>	3205
<code>\l__spintent_spfun_depth_int</code>	3617	<code>\l__spintent_spinterval_luaset_b_value-</code> <code>str</code>	3205
<code>__spintent_spfun_exec:nn</code>	3854	<code>\l__spintent_spinterval_luaset_error-</code> <code>code_str</code>	3198
<code>__spintent_spfun_fname:</code>	3665	<code>\l__spintent_spinterval_luaset_error-</code> <code>str</code>	3198
<code>\l__spintent_spfun_fname_tl</code>	3620	<code>\l__spintent_spinterval_luaset_left_sym-</code> <code>str</code>	3200
<code>\l__spintent_spfun_intent_arg_tl</code>	3620	<code>\l__spintent_spinterval_luaset_right_sym-</code> <code>str</code>	3200
<code>\l__spintent_spfun_kv_tl</code>	3620	<code>\l__spintent_spinterval_luaset_sep_str</code>	3200
<code>\l__spintent_spfun_luaset_error_str</code>	3612	<code>\l__spintent_spinterval_luaset_tall-</code> <code>str</code>	3200
<code>\l__spintent_spfun_luaset_inv_str</code>	3612	<code>\l__spintent_spinterval_luaset_type-</code> <code>str</code>	3200
<code>\l__spintent_spfun_luaset_nargs_str</code>	3612	<code>__spintent_spinterval_mathml:</code>	3558
<code>__spintent_spfun_mathml:</code>	3805	<code>\l__spintent_spinterval_mode_str</code>	3217
<code>\l__spintent_spfun_mode_str</code>	3615	<code>__spintent_spinterval_plain_number:n</code>	3486
<code>__spintent_spfun_name_mathcat:</code>	3723	<code>\l__spintent_spinterval_prefix_tl</code>	3217
<code>__spintent_spfun_name_plain:</code>	3684	<code>__spintent_spinterval_print:n</code>	3520
<code>__spintent_spfun_name_school:</code>	3696	<code>\l__spintent_spinterval_read_arg_tl</code>	3217
<code>\l__spintent_spfun_nargs_int</code>	3617	<code>\l__spintent_spinterval_reason_str</code>	3221
<code>__spintent_spfun_nested:n</code>	3844		
<code>\l__spintent_spfun_prefix_tl</code>	3615		
<code>\l__spintent_spfun_refs_tl</code>	3620		
<code>__spintent_spfun_set_intent:</code>	3775		
<code>\l__spintent_spfun_silent_bool</code>	3615		
<code>\l__spintent_spfun_tmp_str</code>	3617		
<code>__spintent_spfuncomp_exec:nn</code>	4043		
<code>__spintent_spfuncomp_fun:n</code>	3906		
<code>__spintent_spfuncomp_group:</code>	3942		
<code>\l__spintent_spfuncomp_intent_tl</code>	3883		
<code>\l__spintent_spfuncomp_kv_tl</code>	3883		
<code>\l__spintent_spfuncomp_luaset_apply-</code> <code>str</code>	3879		
<code>\l__spintent_spfuncomp_luaset_error-</code> <code>str</code>	3879		
<code>\l__spintent_spfuncomp_luaset_n_str</code>	3879		
<code>\l__spintent_spfuncomp_luaset_paren-</code> <code>str</code>	3879		
<code>__spintent_spfuncomp_mathml:</code>	3980		
<code>\l__spintent_spfuncomp_n_int</code>	3883		
<code>__spintent_spfuncomp_plain:</code>	3966		
<code>\l__spintent_spfuncomp_refs_tl</code>	3883		

258 / 261

<code>\l__spinttent_spnM_spnD_luaset_error_-</code>	<code>__spinttent_spProb_event:N</code>	8295
<code>str</code>	<code>__spinttent_spProb_exec:nn</code>	8410
2332	<code>\l__spinttent_spProb_luaset_a_str</code>	8265
<code>__spinttent_spnM_spnD_mathml:nnnn</code>	<code>\l__spinttent_spProb_luaset_b_str</code>	8265
2452	<code>\l__spinttent_spProb_luaset_error_str</code>	8265
<code>__spinttent_spnM_spnD_result:nn</code>	<code>\l__spinttent_spProb_luaset_op_str</code>	8265
2403	<code>__spinttent_spProb_mathml:</code>	8337
<code>__spinttent_spnM_spnD_silent:nn</code>	<code>__spinttent_spProb_mid:</code>	8295
2395	<code>\l__spinttent_spProb_nested_bool</code>	8270
<code>\l__spinttent_spnmd_luaset_is_number_-</code>	<code>__spinttent_spProb_nested_ops:</code>	8295
<code>str</code>	<code>\l__spinttent_spProb_read_arg_tl</code>	8269
2332	<code>__spinttent_spProm_exec:nn</code>	7616
<code>\l__spinttent_spnnum_intent_read_dec_sep_-</code>	<code>\l__spinttent_spProm_intent_tl</code>	7599
<code>str</code>	<code>__spinttent_spProm_mathml:n</code>	7621
293	<code>\l__spinttent_spProm_read_arg_tl</code>	7599
<code>\l__spinttent_spnnum_intent_read_sign_str</code>	<code>__spinttent_spPval_exec:nnn</code>	8225
293	<code>__spinttent_spPval_mathml:nn</code>	8230
<code>__spinttent_spnnum_mathml:</code>	<code>\l__spinttent_sprad_mode_str</code>	6505
475	<code>\l__spinttent_sprad_read_arg_tl</code>	6505
<code>\l__spinttent_spnnum_notation_sym_tl</code>	<code>\l__spinttent_sprad_sub_index_str</code>	6505
293	<code>\l__spinttent_sprango_read_arg_tl</code>	7656
<code>__spinttent_spnnum_print_integer:</code>	<code>\l__spinttent_sprango_sub_index_str</code>	7656
475	<code>__spinttent_sprayo_exec:nn</code>	5134
<code>__spinttent_spnnum_print_num_start:</code>	<code>\l__spinttent_sprayo_intent_arg_tl</code>	5082
475	<code>__spinttent_sprayo_mathml:</code>	5148
<code>__spinttent_spnnum_print_part_dec:</code>	<code>\l__spinttent_sprayo_mode_str</code>	5076
454	<code>\l__spinttent_sprayo_prefix_tl</code>	5076
<code>__spinttent_spnnum_print_part_period:</code>	<code>\l__spinttent_sprayo_pt_idx_int</code>	5079
454	<code>\l__spinttent_sprayo_pt_seq</code>	5079
<code>__spinttent_spnnum_print_period_bar:</code>	<code>\l__spinttent_sprayo_read_sty_str</code>	5076
382	<code>\l__spinttent_sprayo_refs_tl</code>	5079
<code>\l__spinttent_spnnum_read_period_sep_str</code>	<code>__spinttent_sprazon_cmd_str</code>	2539
293	<code>__spinttent_sprazon_exec:nnn</code>	2558
<code>\l__spinttent_spnnum_read_terse_bool</code>	<code>__spinttent_sprazon_mathml:nn</code>	2563
293	<code>__spinttent_spread_exec:nn</code>	1735
<code>__spinttent_spnnum_render_dec:</code>	<code>\l__spinttent_spread_luaset_reading_str</code>	1685
365	<code>\l__spinttent_spread_luaset_status_str</code>	1685
<code>__spinttent_spnnum_render_decimal_sep:</code>	<code>__spinttent_spread_mathml:nn</code>	1761
419	<code>\l__spinttent_spread_read_arg_tl</code>	1685
<code>__spinttent_spnnum_render_int:</code>	<code>\l__spinttent_spread_read_tmp_tl</code>	1685
365	<code>__spinttent_sprecta_exec:nn</code>	4861
<code>__spinttent_spnnum_render_only_period:</code>	<code>\l__spinttent_sprecta_intent_arg_tl</code>	4821
382	<code>\l__spinttent_sprecta_kv_tl</code>	4821
<code>__spinttent_spnnum_render_part:n</code>	<code>__spinttent_sprecta_mathml:n</code>	4871
365	<code>\l__spinttent_sprecta_mode_str</code>	4816
<code>__spinttent_spnnum_render_period_sep:</code>	<code>\l__spinttent_sprecta_prefix_tl</code>	4816
382	<code>\l__spinttent_sprecta_pt_idx_int</code>	4818
<code>__spinttent_spnz_exec:nn</code>	<code>\l__spinttent_sprecta_pt_seq</code>	4818
2728	<code>\l__spinttent_sprecta_refs_tl</code>	4818
<code>__spinttent_spnz_mathml:</code>	<code>\l__spinttent_spriq_read_arg_tl</code>	7656
2712	<code>\l__spinttent_spriq_sub_index_str</code>	7656
<code>\l__spinttent_sp0mega_read_arg_tl</code>	<code>__spinttent_spsshort_exec:n</code>	1204
8165	<code>__spinttent_spsshort_execute_layout:</code>	1239
<code>\l__spinttent_sp0mega_read_sub_bool</code>	<code>\l__spinttent_spsshort_luaset_base_str</code>	1152
8165	<code>\l__spinttent_spsshort_luaset_expand_str</code>	1152
<code>\l__spinttent_sp0mega_sub_index_str</code>	<code>\l__spinttent_spsshort_luaset_layout_str</code>	1152
8165	<code>\l__spinttent_spsshort_luaset_output_str</code>	1152
<code>\l__spinttent_spPcirc_mode_str</code>	<code>\l__spinttent_spsshort_luaset_status_str</code>	1152
7059	<code>\l__spinttent_spsshort_luaset_suffix_str</code>	1152
<code>\l__spinttent_spPcirc_prefix_tl</code>	<code>__spinttent_spsshort_print:nn</code>	1192
7059	<code>__spinttent_spsshort_render_bypass:n</code>	1215
<code>\l__spinttent_spPcirc_print_sym_str</code>		
7059		
<code>\l__spinttent_spPcirc_read_arg_tl</code>		
7059		
<code>\l__spinttent_spPcirc_read_sty_str</code>		
7059		
<code>\l__spinttent_spPcirc_sub_index_str</code>		
7059		
<code>\l__spinttent_spPerc_read_arg_tl</code>		
7877		
<code>\l__spinttent_spPfig_mode_str</code>		
6039		
<code>\l__spinttent_spPfig_prefix_tl</code>		
6039		
<code>\l__spinttent_spPfig_print_sym_str</code>		
6039		
<code>\l__spinttent_spPfig_read_arg_tl</code>		
6039		
<code>__spinttent_spPoint_exec:nnn</code>		
4502		
<code>\l__spinttent_spPoint_intent_arg_tl</code>		
4462		
<code>__spinttent_spPoint_mathml:n</code>		
4547		
<code>\l__spinttent_spPoint_mode_str</code>		
4460		
<code>\l__spinttent_spPoint_prefix_tl</code>		
4460		
<code>__spinttent_spPoly_build_concept:</code>		
5961		
<code>__spinttent_spPoly_build_refs:</code>		
5946		
<code>__spinttent_spPoly_exec:nn</code>		
5997		
<code>\l__spinttent_spPoly_intent_arg_tl</code>		
5893		
<code>__spinttent_spPoly_mathml:</code>		
6008		
<code>\l__spinttent_spPoly_mode_str</code>		
5891		
<code>__spinttent_spPoly_points_loop:</code>		
5985		
<code>\l__spinttent_spPoly_prefix_tl</code>		
5891		
<code>\l__spinttent_spPoly_pt_idx_int</code>		
5893		
<code>\l__spinttent_spPoly_pt_seq</code>		
5893		
<code>\l__spinttent_spPoly_refs_tl</code>		
5893		
<code>__spinttent_spPoly_sym:</code>		
5938		
<code>__spinttent_spProb_cap:</code>		
8295		
<code>__spinttent_spProb_content:</code>		
8327		
<code>__spinttent_spProb_cup:</code>		
8295		

<code>__spintent_spsshort_render_error:n</code> ...	1215	<code>\l__spintent_spvarz_read_arg_tl</code>	7656
<code>__spintent_spsshort_render_fallback:</code> ..	1215	<code>\l__spintent_spvarz_sub_index_str</code>	7656
<code>__spintent_spsshort_render_success:</code> ...	1215	<code>__spintent_spvec_exec:nnn</code>	4615
<code>__spintent_spsshort_tag_span:nn</code>	1179	<code>\l__spintent_spvec_intent_arg_tl</code>	4572
<code>__spintent_spside_exec:nnn</code>	5291	<code>\l__spintent_spvec_kv_tl</code>	4572
<code>\l__spintent_spside_intent_arg_tl</code>	5223	<code>__spintent_spvec_mathml:n</code>	4648
<code>\l__spintent_spside_intent_tl</code>	5219	<code>\l__spintent_spvec_mode_str</code>	4567
<code>__spintent_spside_mathml:n</code>	5326	<code>\l__spintent_spvec_prefix_tl</code>	4567
<code>\l__spintent_spside_mode_str</code>	5212	<code>\l__spintent_spvec_pt_idx_int</code>	4569
<code>__spintent_spside_points_loop:n</code>	5394	<code>\l__spintent_spvec_pt_seq</code>	4569
<code>\l__spintent_spside_prefix_tl</code>	5212	<code>\l__spintent_spvec_refs_tl</code>	4569
<code>\l__spintent_spside_pt_idx_int</code>	5220	<code>__spintent_spVol_attach_subscript:</code> ...	7525
<code>\l__spintent_spside_pt_seq</code>	5220	<code>__spintent_spVol_mathml:</code>	7438
<code>\l__spintent_spside_read_arg_tl</code>	5212	<code>\l__spintent_spVol_mode_str</code>	7393
<code>\l__spintent_spside_read_sty_str</code>	5212	<code>\l__spintent_spVol_native_bool</code>	7396
<code>\l__spintent_spside_refs_tl</code>	5220	<code>\l__spintent_spVol_read_arg_full_tl</code> ...	7399
<code>__spintent_spsiglo_exec:n</code>	1379	<code>__spintent_spVol_read_arg_print:</code>	7557
<code>\l__spintent_spsiglo_luaset_arabic_str</code>	1296	<code>\l__spintent_spVol_read_arg_tl</code>	7393
<code>\l__spintent_spsiglo_luaset_error_str</code> .	1296	<code>__spintent_spVol_silent:</code>	7445
<code>\l__spintent_spsiglo_luaset_roman_min-</code> <code>str</code>	1296	<code>\l__spintent_spVol_sub_index_str</code>	7393
<code>__spintent_spsiglo_mathml:</code>	1379	<code>\l__spintent_spVol_suffix_tl</code>	7396
<code>\l__spintent_spsiglo_print_era_str</code> ...	1296	<code>__spintent_spVol_symbol_print:</code>	7487
<code>__spintent_spsiglo_render_simple:nn</code> ..	1330	<code>__spintent_spVol_verbal:</code>	7466
<code>__spintent_spsiglo_render_with-</code> <code>era:nnn</code>	1330	<code>\l__spintent_spVol_word_tl</code>	7396
<code>__spintent_spsiglo_tag_span:nn</code>	1366	<code>__spintent_stat_exec:nnn</code>	7780
<code>__spintent_spsiglo_text:</code>	1379	<code>__spintent_stat_intent:nn</code>	7736
<code>__spintent_sptime_build_seconds:nn</code> ...	1453	<code>__spintent_stat_intent:nnn</code>	7736
<code>\l__spintent_sptime_luaset_base_str</code> ...	1440	<code>__spintent_stat_intent:nnnn</code>	7736
<code>\l__spintent_sptime_luaset_error_str</code> ..	1440	<code>\l__spintent_stat_intent_tl</code>	7665
<code>\l__spintent_sptime_luaset_has_seconds-</code> <code>str</code>	1440	<code>__spintent_stat_letter:nnnn</code>	7780
<code>\l__spintent_sptime_luaset_is_fraction-</code> <code>str</code>	1440	<code>__spintent_stat_mathml:nnnn</code>	7771
<code>\l__spintent_sptime_luaset_seconds_str</code>	1440	<code>__spintent_stat_pos_check:nnn</code>	7908
<code>__spintent_sptime_mathml:</code>	1471	<code>\l__spintent_stat_pos_error_bool</code>	7881
<code>__spintent_spunit_exec:n</code>	691	<code>__spintent_stat_pos_exec:nnnnn</code>	7962
<code>\l__spintent_spunit_exponent_tl</code>	564	<code>__spintent_stat_pos_mathml:nnnn</code>	7932
<code>__spintent_spunit_format_canonical:</code> ..	623	<code>__spintent_stat_pos_typeset:nn</code>	7927
<code>__spintent_spunit_load_term:nn</code>	631	<code>\l__spintent_stat_pos_value_int</code>	7881
<code>\l__spintent_spunit_luaset_above_found-</code> <code>str</code>	556	<code>__spintent_stat_simple:nnn</code>	7780
<code>\l__spintent_spunit_luaset_above_n_str</code>	556	<code>__spintent_stat_typeset:nnn</code>	7722
<code>\l__spintent_spunit_luaset_above_sexa-</code> <code>str</code>	556	<code>__spintent_unknown_keys_cmd:n</code>	245
<code>\l__spintent_spunit_luaset_below_found-</code> <code>str</code>	556	<code>__spintent_unknown_keys_cmd:nn</code>	245
<code>\l__spintent_spunit_luaset_below_n_str</code>	556	<code>spintent.lua</code>	76
<code>\l__spintent_spunit_luaset_canonical-</code> <code>str</code>	561	<code>\spinterval</code>	39, 130, 3601
<code>\l__spintent_spunit_luaset_read_str</code> ...	561	<code>spinterval-invalid-arg</code>	3233
<code>\l__spintent_spunit_luaset_status_str</code> .	561	<code>\spmangle</code>	28, 175, 5706
<code>__spintent_spunit_mathml:</code>	734	<code>spmangle-invalid-arg</code>	5440
<code>__spintent_spunit_print_product:n</code> ...	674	<code>\spmarc</code>	29, 180, 5869
<code>__spintent_spunit_print_term:nn</code>	640	<code>spmarc-invalid-points</code>	5738
<code>\l__spintent_spunit_read_slash_str</code> ...	564	<code>\spmax</code>	43, 7851
<code>\l__spintent_spusep_arg_tl</code>	1558	<code>\spmcd</code>	19, 105, 2312
<code>__spintent_spusep_exec:n</code>	1635	<code>\spmcm</code>	20, 105, 2312
<code>\l__spintent_spusep_left_tl</code>	1558	<code>\spmedia</code>	42, 7802
<code>__spintent_spusep_mathml:nn</code>	1624	<code>\spmediana</code>	42, 7802
<code>\l__spintent_spusep_right_tl</code>	1558	<code>\spmmin</code>	43, 7851
		<code>\spmmoda</code>	42, 7802
		<code>\spmoney</code>	10, 78, 1141
		<code>spmoney-invalid-currency</code>	861
		<code>spmoney-no-subunits</code>	861
		<code>\spmQ</code>	21, 119, 2990
		<code>spmQ-not-integer</code>	2752

spmQ-signed-part	2752
spmQ-zero-denominator	2752
\spmside	26, 170, 5406
spmside	6773
spmside-invalid-points	5224
\spmsym	17, 100, 1945
\spmul	18, 104, 2096
spmul-empty-arg	2031
\spnD	18, 110, 2519
\spnewread	15, 1777
spnewread-duplicate	1689
spnewread-empty-args	1689
\spnewunit	10, 77, 812
spnewunit-duplicate	790
\spnM	19, 110, 2519
\spnum	7, 67, 538
spnum-has-units	298
spnum-no-digits	298
\spnZ	21, 118, 2736
spnZ-not-integer	2689
\spOmega	45, 230, 8191
\spPcirc	37, 207, 7367
spPcirc-invalid-arg	7074
\spPerc	43, 7971
\spPfig	32, 187, 6476
spPfig-invalid-arg	6057
spPfig-invalid-points	6057
spPfig-read-arg-key	6057
spPfig-sym-cardinality-conflict	6057
spPfig-sym-letter-conflict	6057
\spPoint	23, 156, 4556
spPoint-invalid-point	4463
\spPoly	30, 184, 6028
spPoly-invalid-points	5897
\spProb	46, 232, 8418
spProb-invalid-arg	8271
\spProm	42, 217, 7645
\spProp	22, 116, 2613
\spPval	45, 231, 8255
\sprad	33, 196, 6727
\sprango	43, 7851
\sprayo	25, 167, 5201
sprayo-invalid-points	5083
\sprazon	22, 114, 2603
\spread	16, 96, 1767
spread-empty-sym	1689
spread-unknown-sym	1689
\sprec	40, 148, 4148
\sprecta	25, 162, 5065
sprecta-invalid-points	4823
\spriq	43, 7851
\spshort	12, 85, 1283
spshort-unknown-abbreviation	1158
\spside	26, 170, 5406
spside	6773
spside-invalid-points	5224
\spsiglo	15, 88, 1432
spsiglo-invalid-arg	1300
\sptime	15, 91, 1503
sptime-invalid-arg	1445
\spunit	8, 72, 778
spunit-inline-numeric-denominator	566
spunit-invalid-unit	566
spunit-no-units	566
\spunitalias	10, 77, 826
spunitalias-duplicate	790
spunitalias-invalid-expr	790
\spusep	16, 94, 1674
spusep-empty-arg	1561
spusep-invalid-arg	1561
\spvarz	42, 7817
\spvec	24, 158, 4805
spvec-invalid-arg	4574
\spVol	38, 213, 7587
stat-pos-invalid-index	7883
str commands:	
\str_lowercase:n	70
sub-index	6511, 6773, 7083, 7400, 7666, 8026, 8169
sub-units	875
suffix	1802, 1877
T	
Tool provided by spanish-mathml-intent :	
spmml-tool	60, 62
U	
unknown	312, 589, 875, 1165, 1307, 1576, 1718, 1802, 1877, 1963, 2038, 2118, 2356, 2541, 2696, 2774, 3036, 3242, 3634, 3895, 4081, 4192, 4471, 4581, 4830, 5090, 5234, 5451, 5748, 5904, 6097, 6511, 6773, 7083, 7400, 7601, 7666, 7704, 7890, 8026, 8119, 8169, 8211, 8280, 8441, 8615, 8984
unknown-choice	219
unknown-command-setup	169
upper-case	2118, 6511
use-dfrac	2541, 2774, 8211
use-spnZ	2774
W	
wrap-parens	1963, 2038, 2696, 2774, 8441