

spanish-mathml-intent

SPANISH PARSE INTENTS

V1.1 2026-10-22^{*}

©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	51
3 The big four	8	13 Change history	52
3.1 El comando <code>\spnum</code>	8	14 Índice de la Documentación	53
3.1.1 Llaves para <code>\spnum</code>	8	15 Implementation	57
3.2 El comando <code>\spunit</code>	9	16 Índice de la Implementación	242
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`[9] en 2019 y, más tarde, `enumext`[10] 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`[22] ha realizado un trabajo extraordinario en este campo. Hoy es posible generar documentos etiquetados, validarlos con herramientas como `veraPDF`[23] o inspeccionar su estructura con `ngpdf`[24]. Sin embargo, siempre me quedó una duda: cuando un estudiante utiliza `NVDA`[13] junto con `MathCAT`[14], ¿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 v1.1 revisada por última vez el 2026-10-22.

[†]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 [17, 21, 19], las especificaciones de WTPDF [18] y la guía más reciente de la PDF Association, «Best Practice Guide: Math in PDF» [20], coinciden en que el camino hacia una accesibilidad matemática de calidad pasa por el uso de MathML. En ese contexto, $\text{\LuaTeX}$ 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` [2] y que se ejecute bajo $\text{\LuaTeX}$, 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 [16] 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` [22, 31], que está en inglés. Quien da sus primeros pasos en $\text{\LaTeX}$ puede apoyarse en `Learn LaTeX` [32], 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_ε [25].

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 $\text{\LaTeX}$ [29] release 2026-11-01.

Este paquete solo se puede utilizar con $\text{\LuaTeX}$ 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` [34], un encapsulado de `\luamml_attribute`: `een` provista por `luamml` [3]. ¡Gracias Marcel!

1.1 Carga del paquete

El paquete `spanish-mathml-intent` trabaja solo con $\text{\LuaTeX}$ 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` [33] y `documentmetadata-support` [34], 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`[12] 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 ocupan. El paquete `spanish-mathml-intent` provee el comando `\spshort` (§3.4) que cubre estos casos, de esta manera podemos cargar `babel`[11] 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`[4], 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`[5] y la fuente Erewhon-Math para el modo matemático cargada por el paquete `fourier-otf`[6]. Adicionalmente cargo la fuente Source Code Pro para el texto tipo «máquina de escribir» por medio del paquete `sourcecodepro`[7].

```
% \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`[8], 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`[26] 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 spintent},
  pdfinfo         =
  {
    Title   = {Test},
    Subject = {spintent},
  },
}

```

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 `\spcArc`, `\spAcirc` y `\spPcArc`, 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>	29
un polígono	<code>\$\spPoly{ABC}\$</code>	30
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>	46
una combinatoria	<code>\$\spcomb{5,2}\$</code>	47
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[13] y MathCAT[14] 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[23]:

```
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`[27, 28]. 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;número⟩}</code>		<code>\spnum[⟨key = val⟩]{⟨número,número;número e exponente⟩}</code>	
	<code>\spnum[⟨key = val⟩]{⟨número.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 ‘x’ 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, yards	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{\langle nuevo\ alias \rangle}{\langle expresión\ de\ unidades \rangle}`

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{\langle cifra \rangle}`
`\spmoney{\langle cifra\ símbolo\ moneda \rangle}`
`\spmoney{\langle cifra\ alias\ moneda \rangle}`

`\spmoney[\langle key = val \rangle]{\langle cifra \rangle}`
`\spmoney[\langle key = val \rangle]{\langle cifra\ símbolo\ moneda \rangle}`
`\spmoney[\langle key = val \rangle]{\langle cifra\ alias\ moneda \rangle}`

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* $[\langle key = val \rangle]$, `\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 = {\langle moneda | alias \rangle}`

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[33]` 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	« <i>doctora</i> »
dr.as	—	Dr. ^{as}	« <i>doctoras</i> »
prof.a	—	Prof. ^a	« <i>profesora</i> »
d.a	d. ^a	D. ^a	« <i>doña</i> »
m.a	—	M. ^a	« <i>maría</i> »
n.o	n. ^o	N. ^o	« <i>número</i> »
n.os	—	N. ^{os}	« <i>números</i> »
c.ia	—	C. ^{ia}	« <i>compañía</i> »
Abreviaturas comunes y expresiones latinas			
pág.	pag.	pág.	« <i>página</i> »
vol.	—	vol.	« <i>volumen</i> »
etc.	—	etc.	« <i>etcétera</i> »
et al.	et. al.	et al.	« <i>y otros</i> »

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»

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 `{<siglo>}`. Si se establece en `ac`, `\spsiglo[print-era=ac]{XXI}` imprimirá ‘xxi a. C.’ en la salida PDF y NVDA/MathCAT verbalizará «veintiuno antes de cristo»; si se establece en `dc`, `\spsiglo[print-era=dc]{XXI}` imprimirá ‘xxi d. C.’ en la salida PDF y NVDA/MathCAT verbalizará «veintiuno después de cristo»; si se establece en `none`, `\spsiglo[print-era=none]{XXI}` imprimirá ‘xxi’ en la salida PDF y NVDA/MathCAT verbalizará «veintiuno».

El comando `\sptime`

```
\sptime \sptime{<horas : minutos>} \sptime{<horas : minutos : segundos>}
```

El comando `\sptime` trabaja en *modo matemático* y recibe el *argumento obligatorio* `{<horas : minutos>}` o `{<horas : minutos : segundos>}`.

Por ejemplo: `\sptime{23:45}` imprimirá en «*modo texto*» la salida PDF ‘23:45’ y NVDA/MathCAT verbalizará «veintitrés cuarenta y cinco minutos».

El comando `\spdate`

```
\spdate \spdate{<DD/MM/YYYY>} \spdate{<YYYY/MM/DD>} \spdate{<YYYY-MM-DD>} \spdate{<DD-MM-YYYY>}
```

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

Por ejemplo: `\spdate{1981-01-02}` y `\spdate{02/01/1981}` imprimirán en «*modo texto*» la salida PDF ‘1981-01-02’ y ‘02/01/1981’ respectivamente y en ambos casos, NVDA/MathCAT verbalizará «dos de enero de mil novecientos ochenta y uno».

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

5 Utilidades para símbolos

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

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

- A diferencia de las otras utilidades implementadas por `spanish-mathml-intent`, que analizan y dan formato a la salida, estos no procesan los «*símbolos matemáticos*», solo afectan la *forma* semántica en que serán verbalizados por NVDA/MathCAT.

El comando `\spnewread`

```
\spnewread \spnewread{<símbolo>}{<verbalización NVDA>}
```

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

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

- El `{<símbolo>}` debe ser un único comando, incluso uno definido por el usuario. Por ejemplo: `\newcommand{\xcuad}{x^2}`, `\spnewread{\xcuad}{x-al-cuadrado}` registrará ‘`\xcuad`’ y cuando se utilice `\spread{\xcuad}` se imprimirá ‘ x^2 ’ en la salida PDF y NVDA/MathCAT verbalizará «x al cuadrado».
- Los espacios y los guiones bajos de `{<verbalización NVDA>}` se convierten en guiones ‘-’, por lo que {es congruente con} equivale a {es-congruente-con}. Los guiones repetidos se reducen a uno y se eliminan los del inicio y del final. Si el resultado queda vacío se devuelve un “error” y se detiene la compilación del documento.

El comando `\spread`

```
\spread \spread{<símbolo>} \spread[<key = val>]{<símbolo>}
\spread[<read-arg={<verbalización NVDA>}]{<comandos LATEX>}
```

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

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

Si el argumento es `{<comandos LATEX>}` y no se establece la llave `read-arg` se devuelve un “error” y se detiene la compilación del documento.

Llaves para `\spread`

`read-arg = {⟨verbalización NVDA⟩}`

por defecto: *no usado*

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

`silent = {⟨true | false⟩}`

por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT del argumento `{⟨símbolo⟩}` o `{⟨comandos  $\LaTeX$ ⟩}` pasado al comando sin modificar la impresión de este. Si se establece en *true*, NVDA/MathCAT NO verbalizará la salida del comando; si se establece en *false*, NVDA/MathCAT sí verbalizará la salida del comando.

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

El comando `\spusep`

`\spusep`

`\spusep{⟨\{ | [ ( | ) | ] | \}⟩}`

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

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

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

- El motivo de existencia de `\spusep` es que los comandos `\left`, `\right`, `\Uleft` y `\Uright` provistos por $\LaTeX$ crean un *grupo* implícito que dificulta el código para *tagged* PDF cuando se necesita interrumpir algo entre ellos, por ejemplo al construir expresiones complejas con `\spnZ`, `\spmQ` o `\spdiv`. El comando `\spusep` imprime el *delimitador* sin crear ningún grupo.
- Hay un segundo motivo para usar `\spusep`. Varios comandos aceptan un *argumento opcional* entre corchetes, `[⟨key = val⟩]`, o delimitado por paréntesis, `(⟨número, número, ...⟩)`, o ambos, escrito justo después del nombre del comando. Si a continuación de un comando necesita un ‘[’ o un ‘(’ explícito que no sea ese argumento, escríbalo con `\spusep`; de lo contrario, $\LaTeX$ lo interpretará como el comienzo del argumento opcional. Por ejemplo, `$\spmedia\spusep{[ ]}$` imprime el símbolo de `\spmedia` seguido de ‘[’, mientras que `\spmedia[` haría que $\LaTeX$ lea lo que sigue como llaves `[⟨key = val⟩]`.

Llaves para `\spusep`

`size = {⟨none | big | Big | bigg | Bigg⟩}`

por defecto: *none*

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

`silent = {⟨true | false⟩}`

por defecto: *false*

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

El comando `\spdsym`

`\spdsym`

`\spdsym[⟨key = val⟩]`

El comando `\spdsym` trabaja en *modo matemático* y recibe el *argumento opcional* `[⟨key = val⟩]` mediante el cual se establece la «*forma*» en la que se imprimirá el *símbolo de división* y cómo se verbalizará en NVDA/MathCAT. Si no se proporciona el *argumento opcional* `[⟨key = val⟩]`, `$\spdsym$` imprimirá ‘:’ en la salida PDF y NV-DA/MathCAT verbalizará «*dividido*».

Llaves para `\spdsym`

`prefix = {⟨prefijo⟩}`

por defecto: *no usado*

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

`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 \spmc d

\spmc d

\spmc d

\spmc d(<número, número, ...>)

\spmc d(<letra, letra, ...>)

\spmc d[<key = val>](<número, número, ...>)

El comando `\spmc d` 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 d$` imprimirá ‘mcd’ en la salida PDF y NVDA/MathCAT verbalizará «*máximo común divisor*».

Si se usa el argumento `(<número, número, ...>)`, `\$ \spmc d(2, 3, 6)$` imprimirá ‘mcd(2, 3, 6)’ en la salida PDF y NVDA/MathCAT verbalizará «*máximo común divisor entre dos, tres y seis*».

Si se usa el argumento `(<letra, letra, ...>)`, `\$ \spmc d(m, n)$` imprimirá ‘mcd(*m*, *n*)’ en la salida PDF y NVDA/MathCAT verbalizará «*máximo común divisor entre eme y ene*».

🚫 Los argumentos `(<número, número, ...>)` o `(<letra, letra, ...>)` requieren al menos dos elementos.

Llaves para \spmc d

`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**: `\spmc[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` $\{\langle \text{úmero entero} \rangle\}$ $\{\langle \text{key} = \text{val} \rangle\} \{\langle \text{úmero entero} \rangle\}$

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

Si no se proporciona el *argumento opcional* $[\langle \text{key} = \text{val} \rangle]$, $\$ \backslash \text{spnZ} \{23\} \$$ imprimirá ‘23’ en la salida PDF y NVDA/MathCAT verbalizará «veintitrés»; $\$ \backslash \text{spnZ} \{-23\} \$$ 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` = $\{\langle \text{true} \mid \text{false} \rangle\}$ por defecto: *false*

Establece si se *imprimen* paréntesis redondos ‘(...)’ alrededor del argumento $\{\langle \text{úmero entero} \rangle\}$ pasado a `\spnZ`. Si se establece en *true*, se imprimirán paréntesis redondos alrededor del $\{\langle \text{úmero entero} \rangle\}$; si se establece en *false*, no se imprimirán los paréntesis. Por ejemplo: $\$ \backslash \text{spnZ} [\text{wrap-parens}=\text{true}] \{-23\} \$$ imprimirá ‘(-23)’ en la salida PDF.

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

Establece si NVDA/MathCAT *verbalizará* los paréntesis redondos ‘(...)’ alrededor del *argumento* $\{\langle \text{úmero entero} \rangle\}$ 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: $\$ \backslash \text{spnZ} [\text{wrap-parens}=\text{true}, \text{read-parens}=\text{false}] \{-23\} \$$ verbalizará «menos veintitrés».

El comando `\spmQ`

`\spmQ` $\{\langle \text{entero} \rangle\} \{\langle \text{numerador} \rangle\} \{\langle \text{denominador} \rangle\}$ $\{\langle \text{key} = \text{val} \rangle\} \{\langle \text{entero} \rangle\} \{\langle \text{numerador} \rangle\} \{\langle \text{denominador} \rangle\}$

El comando `\spmQ` trabaja en *modo matemático* y recibe *tres argumentos obligatorios*: $\{\langle \text{úmero entero} \rangle\}$, $\{\langle \text{numerador} \rangle\}$ y $\{\langle \text{denominador} \rangle\}$, los procesa internamente y devuelve un «número mixto». Los tres deben ser enteros; el $\{\langle \text{numerador} \rangle\}$ y el $\{\langle \text{denominador} \rangle\}$ no pueden llevar signo y el $\{\langle \text{denominador} \rangle\}$ no puede ser cero. Cualquier otro valor devuelve un “error” y se detiene la compilación del documento.

Si no se proporciona el *argumento opcional* $[\langle \text{key} = \text{val} \rangle]$, $\$ \backslash \text{spmQ} \{2\} \{5\} \{7\} \$$ imprimirá ‘ $2\frac{5}{7}$ ’ en la salida PDF y NVDA/MathCAT verbalizará «dos y cinco séptimos».

- En el caso de las fracciones, y en especial con los «números mixtos», no se pueden manejar muchas cosas desde el lado de `spanish-mathml-intent`. Por ejemplo, la entrada $\$ \backslash \text{frac} \{5\} \{7\} \$$ 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` = $\{\langle \text{entero} \mid \text{enteros} \mid \text{y} \rangle\}$ 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*, $\$ \backslash \text{spmQ} [\text{join-word}=\text{entero}] \{1\} \{5\} \{7\} \$$ imprimirá ‘ $1\frac{5}{7}$ ’ en la salida PDF y NVDA/MathCAT verbalizará «un entero cinco séptimos»; si se establece en *enteros*, $\$ \backslash \text{spmQ} [\text{join-word}=\text{enteros}] \{2\} \{5\} \{7\} \$$ imprimirá ‘ $2\frac{5}{7}$ ’ en la salida PDF y NVDA/MathCAT verbalizará «dos enteros cinco séptimos»; si se establece en *y*, $\$ \backslash \text{spmQ} [\text{join-word}=\text{y}] \{2\} \{5\} \{7\} \$$ imprimirá ‘ $2\frac{5}{7}$ ’ en la salida PDF y NVDA/MathCAT verbalizará «dos y cinco séptimos».

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

Establece si el $\{\langle \text{numerador} \rangle\}$ y el $\{\langle \text{denominador} \rangle\}$ 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*, $\{\langle \text{numerador} \rangle\}$ y $\{\langle \text{denominador} \rangle\}$ se pasarán a `\spnZ` y, por ejemplo, $\$ \backslash \text{spmQ} \{2\} \{5\} \{7\} \$$ 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` = $\{\langle \text{true} \mid \text{false} \rangle\}$ 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` = $\{\langle \text{true} \mid \text{false} \rangle\}$ 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 = {⟨true | false⟩}`

por defecto: *true*

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

`silent = {⟨true | false⟩}`

por defecto: *false*

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

El comando `\sprazon`

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

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

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

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

Llaves para `\sprazon`

`silent = {⟨true | false⟩}`

por defecto: *false*

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

`use-dfrac = {⟨true | false⟩}`

por defecto: *false*

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

`inline = {⟨true | false⟩}`

por defecto: *false*

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

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

El comando `\spProp`

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

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

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

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

Llaves para `\spProp`

`silent = {⟨true | false⟩}`

por defecto: *false*

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

`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</code>	<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 ‘ (x_1, y_1) ’ y verbalizará «x subíndice uno coma y subíndice uno». La lectura del subíndice la decide NVDA/MathCAT, porque `\spcoord` no tiene la llave `read-cmd`.

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

Llaves para `\spcoord`

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

Establece si se *silencia* la verbalización en NVDA/MathCAT del `{\argumento}` pasado al comando sin modificar la impresión de este. Si se establece en `true`, NVDA/MathCAT NO verbalizará la salida del comando; si se establece en `false`, NVDA/MathCAT sí verbalizará la salida del comando.

El comando `\spPoint`

<code>\spPoint</code>	<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 ‘ A_1 ’ y verbalizará «punto A sub uno».

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

Llaves para `\spPoint`

`read-cmd = {\school | mathcat}` por defecto: `school`

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

`prefix = {\prefijo}` por defecto: `no usado`

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

`concept = {\true | false}` por defecto: `true`

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

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

por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT del concepto «punto» 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 `\spvec`

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

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

Si el *argumento* es de la forma $\{\langle \text{punto, punto} \rangle\}$ con prima(s) ‘*’* o subíndice(s) ‘*_*{...}’ opcionales se aplica `\overrightarrow` en la impresión; si el *argumento* es de la forma $\{\langle \text{letra} \rangle\}$ minúscula con subíndice ‘*_*{...}’ opcional se aplica `\vec` en la impresión.

Si no se proporciona el *argumento opcional* $[\langle \text{key} = \text{val} \rangle]$, `\$ \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 = {\langle school | mathcat \rangle}`

por defecto: *school*

Establece la *forma* en la que NVDA/MathCAT verbalizará el argumento $\{\langle \text{punto, punto} \rangle\}$. 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 = {\langle prefijo \rangle}`

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 = {\langle true | false \rangle}`

por defecto: *false*

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

`concept = {\langle true | false \rangle}`

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

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

por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT del concepto «vector» 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 `\sprecta`

<code>\sprecta</code>	<code>\sprecta{\langle punto, punto \rangle}</code>	<code>\sprecta[key = val](\langle punto, punto \rangle)</code>
	<code>\sprecta{\langle letra \rangle}</code>	<code>\sprecta[key = val](\langle letra \rangle)</code>

El comando `\sprecta` trabaja en *modo matemático* y recibe un *argumento obligatorio* de la forma $\{\langle \text{letra} \rangle\}$ o $\{\langle \text{punto, punto} \rangle\}$ con prima(s) ‘*’* o subíndice(s) ‘*_*{...}’ opcionales. Si el *argumento* es de la forma $\{\langle \text{punto, punto} \rangle\}$ se aplica `\overleftarrow` en la impresión; si el *argumento* es de la forma $\{\langle \text{letra} \rangle\}$, mayúscula o minúscula, se imprimirá solo la $\{\langle \text{letra} \rangle\}$ sin aplicar `\overleftarrow`.

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

Llaves para `\sprecta`

`read-cmd = {\langle school | mathcat \rangle}`

por defecto: *school*

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

`prefix = {⟨prefijo⟩}`

por defecto: *no usado*

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

`concept = {⟨true | false⟩}`

por defecto: *true*

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

`silent = {⟨true | false⟩}`

por defecto: *false*

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

El comando `\spray`

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

El comando `\spray` trabaja en *modo matemático* y recibe el *argumento obligatorio* {⟨punto, punto⟩} con prima(s) ' ' o subíndice(s) '_{...}' opcionales. Representa un *rayo* o *semirrecta* con origen en el primer punto y dirección hacia el segundo, aplicando `\overrightarrow` en la impresión.

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

Llaves para `\spray`

`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»: `\spray{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 `\spray[read-sty=rayo]{AB}`, NVDA/MathCAT verbalizará «rayo A B»; si se utiliza `\spray[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`

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

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

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

Llaves para `\spside`

`read-cmd = {⟨school | mathcat⟩}`

por defecto: *school*

Establece la *forma* en la que NVDA/MathCAT verbalizará el argumento {⟨punto, punto⟩}. Si se establece en *school*, NVDA/MathCAT verbaliza las letras del argumento sin el prefijo «mayúscula»: `\spside{AB}` se

verbaliza «lado A B»; si se establece en `mathcat`, NVDA/MathCAT aplica sus propias reglas, que pueden incluir el prefijo «mayúscula» dependiendo de la configuración del usuario.

`read-sty` = { `(lado | segmento | trazo)` } por defecto: *lado*

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

`prefix` = { `(prefijo)` } por defecto: *no usado*

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

`concept` = { `(true | false)` } por defecto: *true*

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

`read-arg` = { `(verbalización NVDA)` } por defecto: *no usado*

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

`silent` = { `(true | false)` } por defecto: *false*

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

El comando `\spmside`

`\spmside` `\spmside{ (punto, punto) }` `\spmside[ (key = val) ]{ (punto, punto) }`

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

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

Llaves para `\spmside`

`read-cmd` = { `(school | mathcat)` } por defecto: *school*

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

`read-sty` = { `(lado | segmento | trazo)` } por defecto: *lado*

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

`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`

`\spangle{(punto | letra griega | número)}`
`\spangle[key = val]{(punto | letra griega | número)}`

`\spangle{(punto, punto, punto)}`
`\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) ‘`_{\dots}`’ opcionales.

Si no se proporciona el *argumento opcional* `[key = val]`, `\spangle{A}` imprimirá ‘ $\angle A$ ’ en la salida PDF y NVDA/MathCAT verbalizará «ángulo en A»; `\spangle{\alpha}` imprimirá ‘ $\angle \alpha$ ’ en la salida PDF y NVDA/MathCAT verbalizará «ángulo alfa»; `\spangle{1}` imprimirá ‘ $\angle 1$ ’ en la salida PDF y NVDA/MathCAT verbalizará «ángulo uno» y `\spangle{AOB}` imprimirá ‘ $\angle AOB$ ’ en la salida PDF y NVDA/MathCAT verbalizará «ángulo A O B».

🔴 Si no se tiene instalada la fuente matemática NewCMMath-Regular.otf, parte del paquete `newcomputermodern[8]`, 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 ‘ $\angle A$ ’ en la salida PDF y NVDA/MathCAT verbalizará «ángulo recto en A», `\spangle[recto=true]{AOB}` imprimirá ‘ $\angle AOB$ ’ 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[key = val]{(punto | letra griega | número)}`
`\spmangle[key = val]{(punto, punto, punto)}`

`\spmangle{(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) ‘`_{\dots}`’ 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 {<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`

`\sparc {<punto, punto>}` `\sparc {<punto, punto, punto>}`
`\sparc [key = val] {<punto, punto>}` `\sparc [key = val] {<punto, punto, punto>}`

El comando `\sparc` trabaja en *modo matemático* y recibe un *argumento obligatorio* de la forma `{<punto, punto>}` o `{<punto, punto, punto>}` con prima(s) ‘’ o subíndice(s) ‘_ {...}’ opcionales aplicando `\widearc` en la impresión.

Si no se proporciona el *argumento opcional* [*key = val*], `\sparc{AB}` imprimirá ‘ $\widehat{AB}$ ’ 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` = $\{\langle school \mid mathcat \rangle\}$ por defecto: *school*

Establece la *forma* en la que NVDA/MathCAT verbalizará el argumento $\{\langle punto, punto \rangle\}$ o $\{\langle punto, punto, punto \rangle\}$. Si se establece en *school*, NVDA/MathCAT verbaliza las letras del argumento sin el prefijo «mayúscula»: $\$ \backslash \text{sparc}\{AB\} \$$ 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` = $\{\langle prefijo \rangle\}$ 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` = $\{\langle true \mid false \rangle\}$ 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 $\{\langle argumento \rangle\}$; si se establece en *false*, NVDA/MathCAT solo verbalizará el $\{\langle argumento \rangle\}$ acorde al valor establecido por la llave `read-cmd`.

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

Establece si se *silencia* la verbalización en NVDA/MathCAT del concepto «arco» y del $\{\langle argumento \rangle\}$ 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 `\spmarc`

<code>\spmarc</code>	<code>\spmarc\{\langle punto, punto \rangle\}</code>	<code>\spmarc\{\langle punto, punto, punto \rangle\}</code>
	<code>\spmarc[\langle key = val \rangle]\{\langle punto, punto \rangle\}</code>	<code>\spmarc[\langle key = val \rangle]\{\langle punto, punto, punto \rangle\}</code>

El comando `\spmarc` trabaja en *modo matemático* y recibe un *argumento obligatorio* de la forma $\{\langle punto, punto \rangle\}$ o $\{\langle punto, punto, punto \rangle\}$ con prima(s) $\prime$ o subíndice(s) $\subscript\{\dots\}$ opcionales colocando ‘m’ antes de aplicar `\widearc` en la impresión.

Si no se proporciona el *argumento opcional* $[\langle key = val \rangle]$, $\$ \backslash \text{spmarc}\{AB\} \$$ 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` = $\{\langle school \mid mathcat \rangle\}$ por defecto: *school*

Establece la *forma* en la que NVDA/MathCAT verbalizará el argumento $\{\langle punto, punto \rangle\}$ o $\{\langle punto, punto, punto \rangle\}$. Si se establece en *school*, NVDA/MathCAT verbaliza las letras del argumento sin el prefijo «mayúscula»: $\$ \backslash \text{spmarc}\{AB\} \$$ 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` = $\{\langle prefijo \rangle\}$ 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` = $\{\langle true \mid false \rangle\}$ 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 $\{\langle argumento \rangle\}$; si se establece en *false*, NVDA/MathCAT solo verbalizará el $\{\langle argumento \rangle\}$ acorde al valor establecido por la llave `read-cmd`.

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

Establece si se *silencia* la verbalización en NVDA/MathCAT del término «medida del», el concepto «arco» y el $\{\langle argumento \rangle\}$ 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 `\spPoly`

<code>\spPoly</code>	<code>\spPoly\{\langle punto, punto, punto, ... \rangle\}</code>	<code>\spPoly[\langle key = val \rangle]\{\langle punto, punto, punto, ... \rangle\}</code>
----------------------	--	---

El comando `\spPoly` trabaja en *modo matemático* y recibe el *argumento obligatorio* $\{\langle punto, punto, punto, ... \rangle\}$, una lista de al menos tres $\{\langle puntos \rangle\}$ con prima(s) $\prime$ o subíndice(s) $\subscript\{\dots\}$ 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*], $\$ \backslash \text{spPoly}\{ABC\}$ imprimirá ' $\triangle ABC$ ' en la salida PDF y NVDA/MathCAT verbalizará «triángulo A B C»; $\$ \backslash \text{spPoly}\{ABCD\}$ imprimirá ' $\square ABCD$ ' y verbalizará «cuadrado A B C D»; $\$ \backslash \text{spPoly}\{ABCDE\}$ imprimirá ' $ABCDE$ ' y verbalizará «pentágono A B C D E».

Llaves para $\backslash \text{spPoly}$

`read-cmd = { (school | mathcat) }` por defecto: *school*

Establece la *forma* en la que NVDA/MathCAT verbalizará el argumento $\{ (punto, punto, punto, \dots) \}$. Si se establece en *school*, NVDA/MathCAT verbaliza las letras del argumento sin el prefijo «mayúscula»: $\$ \backslash \text{spPoly}\{ABC\}$ 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* $\backslash \text{triangle}$ o $\backslash \text{mdlgwhtsquare}$ en la impresión sin modificar la verbalización de este. Si se establece en *true*, $\$ \backslash \text{spPoly}[\text{print-sym=true}]\{ABC\}$ imprimirá ' $\triangle ABC$ ' en la salida PDF y NVDA/MathCAT verbalizará «triángulo A B C»; $\$ \backslash \text{spPoly}[\text{print-sym=true}]\{ABCD\}$ 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 $\backslash \text{spAfig}$

$\backslash \text{spAfig}$	$\backslash \text{spAfig}$	$\backslash \text{spAfig}((punto, punto, punto, \dots))$
$\backslash \text{spAfig}[(key = val)]$	$\backslash \text{spAfig}[(key = val)]((punto, punto, punto, \dots))$	
$\backslash \text{spAfig}((1 2 3 \dots n))$	$\backslash \text{spAfig}((F P B L T))$	
$\backslash \text{spAfig}[(key = val)]((1 2 3 \dots n))$	$\backslash \text{spAfig}[(key = val)]((F P B L T))$	

El comando $\backslash \text{spAfig}$ trabaja en *modo matemático* y recibe el *argumento opcional delimitado por paréntesis* de la forma $((1 | 2 | 3 | \dots | n))$, $((punto, punto, punto, \dots))$ o $((F | P | B | L | T))$ aplicando $\backslash \text{mathcal}\{A\}$ 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* $((1 | 2 | 3 | \dots | n))$, $\$ \backslash \text{spAfig}(1)$ imprimirá ' $\mathcal{A}_1$ ' en la salida PDF y NVDA/MathCAT verbalizará «área uno»; $\$ \backslash \text{spAfig}(n)$ imprimirá ' $\mathcal{A}_n$ ' en la salida PDF y NVDA/MathCAT verbalizará «área ene».

Si se proporciona el *argumento* $((punto, punto, punto, \dots))$, $\$ \backslash \text{spAfig}(ABC)$ imprimirá ' $\mathcal{A}_{\triangle ABC}$ ' en la salida PDF y NVDA/MathCAT verbalizará «área del triángulo A B C»; $\$ \backslash \text{spAfig}(ABCD)$ imprimirá ' $\mathcal{A}_{\square ABCD}$ ' en la salida PDF y NVDA/MathCAT verbalizará «área del cuadrado A B C D»; $\$ \backslash \text{spAfig}(ABCDE)$ 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 unos».

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

Llaves para `\spAfig`

`read-cmd` = { `<school | mathcat>` }

por defecto: `school`

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

`prefix` = { `<prefijo>` }

por defecto: `no usado`

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

`concept` = { `<true | false>` }

por defecto: `true`

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

🔊 Utilizar `concept=false` sin el (`<argumento>`) es equivalente a usar `silent=true`.

`read-arg` = { `<verbalización NVDA>` }

por defecto: `no usado`

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

🔊 Esta llave no tiene efecto y emite un aviso con los argumentos (`<F | P | B | L | T>`) y (`<1 | 2 | 3 | ... | n>`).

`concept*` { `<valor prohibido>` }

por defecto: `no usado`

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

`read-sub` = { `<true | false>` }

por defecto: `false`

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

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

`print-sym` = { `<auto | triangle | square | none>` }

por defecto: `auto`

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

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

🔊 Esta llave no se puede utilizar con el argumento (`<F | P | B | L | T>`).

`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</code>	<code>\spPfig((punto, punto, punto, ...))</code>
<code>\spPfig[key = val]</code>	<code>\spPfig[key = val]</code>	<code>\spPfig((punto, punto, punto, ...))</code>
<code>\spPfig(1 2 3 ... n)</code>	<code>\spPfig(F P B L T)</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>	<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 ABC »; `\spPfig(ABCD)` imprimirá ' $\mathcal{P}_{\square ABCD}$ ' en la salida PDF y NVDA/MathCAT verbalizará «perímetro del cuadrado $ABCD$ »; `\spPfig(ABCDE)` imprimirá ' $\mathcal{P}_{ABCDE}$ ' en la salida PDF y NVDA/MathCAT verbalizará «perímetro del pentágono $ABCDE$ ». 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 ABC »; 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 $ABCD$ ».

- 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 = {\langle auto | triangle | square | none \rangle}`

por defecto: `auto`

Establece si se usará `\triangle` o `\mdlgwhtsquare` en la impresión para los argumentos (`\langle punto, punto, punto, ... \rangle`) y (`\langle 1 | 2 | 3 | ... | n \rangle`) pasado al comando sin modificar la verbalización de este. Si se utiliza (`\langle punto, punto, punto, ... \rangle`) 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 $\mathcal{P}_{\square ABC}$ en la salida PDF y en ambos casos NVDA/MathCAT verbalizará «perímetro del triángulo A B C».

Si se utiliza (`\langle 1 | 2 | 3 | ... | n \rangle`) 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 = {\langle true | false \rangle}`

por defecto: `false`

Establece si se silencia la verbalización en NVDA/MathCAT del término «perímetro del», el concepto «nombre de la figura» y el `\langle 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.

Los comandos `\sprad` y `\spdiam`

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

Los comandos `\sprad` y `\spdiam` trabajan en *modo matemático* y reciben el *argumento opcional* `[\langle key = val \rangle]` 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* `[\langle key = val \rangle]`, `\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 = {\langle true | false \rangle}`

por defecto: `true`

Establece si se verbaliza 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 = {\langle school | mathcat \rangle}`

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 = {\langle true | false \rangle}`

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 = {\langle true | false \rangle}`

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 ‘Ød’ en la salida PDF y NVDA/MathCAT verbalizará «diámetro».

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

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

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 = {⟨verbalización NVDA⟩}`

por defecto: no usado

Sobrescribe el *concepto* establecido por la llave `concept` que será verbalizado por NVDA/MathCAT al ejecutar el comando sin modificar la impresión de este. El valor de esta llave *siempre* debe pasarse entre llaves '{ }' y se debe usar en conjunto con la llave `concept`.

Por ejemplo: `\sprad[read-arg={medida exacta}]` imprimirá 'r' en la salida PDF y NVDA/MathCAT verbalizará «radio medida exacta»; `\sprad[concept=false, read-arg={medida exacta}]` imprimirá 'r' y NVDA/MathCAT verbalizará «medida exacta».

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

`silent = {⟨true | false⟩}`

por defecto: false

Establece si se *silencia* la verbalización en NVDA/MathCAT del *concepto* establecido por la llave `concept` y de las letras 'r', 'R', 'd' o 'D' al ejecutar el comando sin modificar la impresión de este. Si se establece en `true`, NVDA/MathCAT NO verbalizará la salida del comando; si se establece en `false`, NVDA/MathCAT sí verbalizará la salida del comando.

El comando `\spcirc`

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

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

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

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

Si se proporciona el *argumento* `(⟨centro⟩)`, `\spcirc(0)` imprimirá '⊙(O)' en la salida PDF y NVDA/MathCAT verbalizará «circunferencia de centro en O».

Si se proporciona el *argumento* `(⟨centro, radio⟩)`, `\spcirc(0,3)` imprimirá '⊙(O,3)' en la salida PDF y NVDA/MathCAT verbalizará «circunferencia de centro en O y radio tres».

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

Llaves para `\spcirc`

El comando `\spcirc` posee las *meta-keys* `spside` pasada al comando `\spside` (v. pág. 26) y `spmside` pasada al comando `\spmside` (v. pág. 27).

`read-cmd = {⟨school | mathcat⟩}`

por defecto: school

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

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

`prefix = {⟨prefijo⟩}`

por defecto: no usado

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

`print-sym = {⟨usual | odot⟩}`

por defecto: odot

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

`read-sty = {⟨circun | circulo⟩}`

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](0)` se verbalizará «circunferencia de centro en O» en NVDA/MathCAT; si se establece en `circulo`, `\spcirc[read-sty=circulo](0)` 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 $\odot_1(O, \overline{AB})$ 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 '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 = {\(school | mathcat)}`

por defecto: *school*

Establece la *forma* en la que NVDA/MathCAT verbalizará el argumento *(centro)*. Si se establece en `school`, NVDA/MathCAT verbaliza el argumento *(centro)* sin el prefijo «*mayúscula*»; si se establece en `mathcat`,

NVDA/MathCAT aplica sus propias reglas, que pueden incluir el prefijo «mayúscula» dependiendo de la configuración del usuario.

`prefix = {⟨prefijo⟩}`

por defecto: *no usado*

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

`read-sty = {⟨circun | círculo⟩}`

por defecto: *circun*

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

`concept = {⟨true | false⟩}`

por defecto: *true*

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

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

`read-arg = {⟨verbalización NVDA⟩}`

por defecto: *no usado*

Sobrescribe el *concepto* establecido por la llave `read-sty` que será verbalizado por NVDA/MathCAT al ejecutar el comando sin modificar la impresión de este. El valor de esta llave *siempre* debe pasarse entre llaves '{ }' y se debe usar en conjunto con la llave `concept`. Por ejemplo: `\spAcirc[read-arg={esfera}](0)` se verbalizará «área de la esfera con centro en O» en NVDA/MathCAT; `\spAcirc[concept=false,read-arg={esfera}](0)` se verbalizará «esfera con centro en O» en NVDA/MathCAT.

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

`print-sym = {⟨usual | odot⟩}`

por defecto: *odot*

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

`sub-index = {⟨subíndice⟩}`

por defecto: *no usado*

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

`read-sub = {⟨true | false⟩}`

por defecto: *false*

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

`silent = {⟨true | false⟩}`

por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT del término «área de ...», del concepto establecido por la llave `read-sty` y del argumento (`⟨centro⟩`) o (`⟨1 | 2 | 3 | ... | n⟩`) pasado al comando sin modificar la impresión de este. Si se establece en `true`, NVDA/MathCAT NO verbalizará la salida del comando; si se establece en `false`, NVDA/MathCAT sí verbalizará la salida del comando.

El comando `\spPcirc`

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

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

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

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* (`⟨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 '`_{\dots}`' 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 (`(1 | 2 | 3 | ... | n)`); si se establece en *false*, NVDA/MathCAT solo verbalizará el argumento (`(centro)`) acorde al valor establecido por la llave `read-cmd`.

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

`read-arg` = { `(verbalización NVDA)` }

por defecto: *no usado*

Sobrescribe el *concepto* establecido por la llave `read-sty` que será verbalizado por NVDA/MathCAT al ejecutar el comando sin modificar la impresión de este. El valor de esta llave *siempre* debe pasarse entre llaves '{ }' y se debe usar en conjunto con la llave `concept`.

Por ejemplo: `\spPcirc[read-sty=circulo,read-arg={disco}](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 (`(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 ' $\mathcal{P}_{O(O)}$ ' 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 ' $\mathcal{P}_{O_1(O)}$ ' en la salida PDF y NVDA/MathCAT verbalizará «*perímetro de la circunferencia uno con centro en O*».

`read-sub` = { `(true | false)` }

por defecto: *false*

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

`silent` = { `(true | false)` }

por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT del término «*perímetro de ...*», del concepto establecido por la llave `read-sty` y del argumento (`(centro)`) o (`(1 | 2 | 3 | ... | n)`) pasado al comando sin modificar la impresión de este. Si se establece en *true*, NVDA/MathCAT NO verbalizará la salida del comando; si se establece en *false*, NVDA/MathCAT sí verbalizará la salida del comando.

El comando `\spVol`

`\spVol` `\spVol` `\spVol[key = val]`

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

Si no se proporciona el *argumento opcional* [`key = val`], `\spVol` imprimirá '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 ‘*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 ‘*V*’ cuando `concept=false`. Si se establece en *school*, NVDA/MathCAT verbalizará únicamente la letra ‘*V*’; si se establece en *mathcat*, NVDA/MathCAT aplica sus propias reglas sobre ‘*V*’.

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

Establece un *subíndice* sobre ‘*V*’. El valor de esta llave *siempre* debe pasarse entre llaves ‘{ }’. Por ejemplo: `\spVol[sub-index={1}]` imprimirá ‘*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 ‘*V*’ en la salida PDF y NVDA/MathCAT verbalizará «*volumen medida exacta*»; `\spVol[concept=false, read-arg={medida exacta}]` imprimirá ‘*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 ‘*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 `\{ \langle intervalo \rangle \}` 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 `\{ \langle intervalo \rangle \}` siempre debe pasarse entre llaves ‘`{ }`’, incluso cuando comienza con un corchete: sin ellas, `\spinterval[1,5]` interpreta los corchetes como el argumento opcional `[\langle key = val \rangle]`.
- 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 = \{ \langle school | mathcat \rangle \}` 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 = \{ \langle prefijo \rangle \}` 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 = \{ \langle verbalización NVDA \rangle \}` 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 = \{ \langle true | false \rangle \}` 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 = \{ \langle auto | none | big | Big | bigg | Bigg \rangle \}` 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 = \{ \langle true | false \rangle \}` por defecto: *false*

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

`\spfun \{ \langle función \rangle \}` `\spfun[\langle key = val \rangle]{ \langle función \rangle }`

El comando `\spfun` trabaja en *modo matemático* y recibe el *argumento* obligatorio `\{ \langle función \rangle \}` de la forma `\{ \langle f(x) \rangle \}`, 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]`, `\spfun{f(x)}` imprimirá ‘*f(x)*’ en la salida PDF y NVDA/MathCAT verbalizará «*f de x*»; `\spfun{f_{1}(x)}` imprimirá ‘*f₁(x)*’ y verbalizará «*f sub uno de x*»; `\spfun{f'(x)}` imprimirá ‘*f'(x)*’ y verbalizará «*f prima de x*»; `\spfun{f''(x)}` imprimirá ‘*f''(x)*’ y verbalizará «*f doble prima de x*».

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

Un argumento puede ser a su vez una llamada, tantas veces como se quiera: $\$ \backslash \text{spfun}\{f(g(x))\}$ 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 $\{ \langle \text{función} \rangle \}$ 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` = $\{ \langle \text{school} \mid \text{mathcat} \rangle \}$ 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` = $\{ \langle \text{prefijo} \rangle \}$ 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` = $\{ \langle \text{true} \mid \text{false} \rangle \}$ 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` = $\{ \langle \text{true} \mid \text{false} \rangle \}$ por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT de la $\{ \langle \text{función} \rangle \}$ 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>	$\backslash \text{spdom}\{ \langle \text{función} \rangle \}$	$\backslash \text{spdom}[\langle \text{key} = \text{val} \rangle]\{ \langle \text{función} \rangle \}$
<code>\sprec</code>	$\backslash \text{sprec}\{ \langle \text{función} \rangle \}$	$\backslash \text{sprec}[\langle \text{key} = \text{val} \rangle]\{ \langle \text{función} \rangle \}$
<code>\spcod</code>	$\backslash \text{spcod}\{ \langle \text{función} \rangle \}$	$\backslash \text{spcod}[\langle \text{key} = \text{val} \rangle]\{ \langle \text{función} \rangle \}$

Los comandos `\spdom`, `\sprec` y `\spcod` trabajan en *modo matemático* y reciben el *argumento* obligatorio $\{ \langle \text{función} \rangle \}$: 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* $[\langle \text{key} = \text{val} \rangle]$, $\$ \backslash \text{spdom}\{f\}$ 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` = $\{ \langle \text{prefijo} \rangle \}$ 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 $\$ \backslash \text{spdom}[\text{prefix}=\langle \text{el} \rangle]\{f\}$, NVDA/MathCAT verbalizará «el dominio de f »; con $\$ \backslash \text{sprec}[\text{prefix}=\langle \text{el} \rangle]\{f\}$, «el recorrido de f »; con $\$ \backslash \text{spcod}[\text{prefix}=\langle \text{el} \rangle]\{f\}$, «el codominio de f ». La impresión no cambia.

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

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

El comando `\spfuncomp`

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

El comando `\spfuncomp` trabaja en *modo matemático* y recibe el *argumento obligatorio* $\langle\textit{composición}\rangle$: dos o más *funciones* separadas por `\circ`, como en $\langle f \circ g \rangle$ o $\langle f \circ g \circ h \rangle$. Cada función es un nombre como en `\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 `\spfun`: pueden ser letras, números, expresiones o llamadas.

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

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

Llaves para `\spfuncomp`

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

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 `\spabs`

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

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

Llaves para `\spabs`

`close = \langle\textit{auto} | \textit{true} | \textit{false}\rangle`

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: $\$ \backslash\text{spabs}[\textit{close}=\textit{true}]\{x\} \$$ verbalizará «*valor absoluto de x fin valor absoluto*». Con *false* nunca se verbaliza: $\$ \backslash\text{spabs}[\textit{close}=\textit{false}]\{x-3\} \$$ verbalizará «*valor absoluto de x menos 3*». La llave no tiene efecto con *silent* ni con *read-arg*.

`size = \langle\textit{none} | \textit{big} | \textit{Big} | \textit{bigg} | \textit{Bigg}\rangle`

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: $\$ \backslash\text{spabs}[\textit{size}=\textit{Big}]\{\dfrac{x}{2}\} \$$ imprime las barras más grandes que el texto y no cambia la verbalización.

`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: $\$ \backslash\text{spabs}[\textit{read-arg}=\{\textit{módulo de } x\}]\{x\} \$$ imprimirá ‘ $|x|$ ’ en la salida PDF y NVDA/MathCAT verbalizará «*módulo de x*».

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

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 `\sp0mega` 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`

<code>\spProm</code>	<code>\spProm{<argumento>}</code>	<code>\spProm[<key = val>]{<argumento>}</code>
----------------------	---	--

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`

<code>\spmedia</code>	<code>\spmedia</code>	<code>\spmedia[<key = val>]</code>
<code>\spmediana</code>	<code>\spmediana</code>	<code>\spmediana[<key = val>]</code>
<code>\spmoda</code>	<code>\spmoda</code>	<code>\spmoda[<key = val>]</code>
<code>\spdesv</code>	<code>\spdesv</code>	<code>\spdesv[<key = val>]</code>
<code>\spvarz</code>	<code>\spvarz</code>	<code>\spvarz[<key = val>]</code>

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 ‘s²’ 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 un *subíndice* sobre el símbolo del comando, que NVDA/MathCAT verbaliza después del concepto con la palabra «de», o «sub» en `\spdesv` y `\spvarz` según la llave `read-sub`. El valor de esta llave *siempre* debe pasarse entre llaves ‘{ }’. Por ejemplo: `\spmedia[sub-index={x}]` imprimirá ‘MA_x’ en la salida PDF y NVDA/MathCAT verbalizará «media aritmética de x »; `\spdesv[sub-index={x}]` imprimirá ‘s_x’ y

verbalizará «desviación estándar *s* sub *x*»; `$\spvarz[sub-index={x}]$` imprimirá ‘ s_x^2 ’ y verbalizará «varianza *s* sub *x* al cuadrado».

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

Establece si se *usa* la palabra «sub» en lugar de «de» al verbalizar el subíndice establecido por la llave `sub-index`, sin modificar la impresión del comando. Si se establece en `true`,

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

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

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

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

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

`silent = {true | false}` por defecto: false

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

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

<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 ‘Q₁’ en la salida PDF y NVDA/MathCAT verbalizará «cuartil 1»; `$\spdecil{3}$` imprimirá ‘D₃’ y verbalizará «decil 3»; `$\spPerc{25}$` imprimirá ‘P₂₅’ 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 ‘Q₁’ 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: `\spcase[read-arg={casos que cumplen}]{f}` imprimirá ‘ C_f ’ en la salida PDF y NVDA/MathCAT verbalizará «casos que cumplen».

`silent = {<{true | false}>}` por defecto: false

Establece si se *silencia* la verbalización en NVDA/MathCAT del símbolo al ejecutar el comando sin modificar la impresión de este. Si se establece en `true`, NVDA/MathCAT NO verbalizará la salida del comando; si se establece en `false`, sí verbalizará la salida del comando. Una letra inválida devuelve siempre un “error” y se detiene la compilación del documento, también con `silent=true`.

El comando `\spOmega`

```
\spOmega <\spOmega> <\spOmega[<{key = val}>]>
```

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

Si no se proporciona el *argumento opcional*, $\backslash\mathrm{spOmega}$ imprimirá ‘ Ω ’ en la salida PDF y NVDA/MathCAT verbalizará «*espacio muestral*».

Llaves para $\backslash\mathrm{spOmega}$

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

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

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

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

$\mathrm{silent} = \{\langle\mathrm{true} \mid \mathrm{false}\rangle\}$ 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 $\backslash\mathrm{spPval}$

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

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

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

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

Llaves para $\backslash\mathrm{spPval}$

$\mathrm{silent} = \{\langle\mathrm{true} \mid \mathrm{false}\rangle\}$ 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.

$\mathrm{use-dfrac} = \{\langle\mathrm{true} \mid \mathrm{false}\rangle\}$ por defecto: *false*

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

El comando $\backslash\mathrm{spProb}$

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

El comando $\backslash\mathrm{spProb}$ trabaja en *modo matemático* y recibe el *argumento obligatorio* $\{\langle\mathrm{evento}\rangle\}$: un evento, o dos eventos unidos por *exactamente uno* de los operadores $\backslash\mathrm{cap}$, $\backslash\mathrm{cup}$, $\backslash\mathrm{mid}$ 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 $\backslash\mathrm{spProb}\{A \ \backslash\mathrm{cap} \ (B \ \backslash\mathrm{cup} \ C)\}$ es válido.

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

Llaves para `\spProb`

`read-arg = {⟨verbalización NVDA⟩}`

por defecto: *no usado*

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

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

`silent = {⟨true | false⟩}`

por defecto: *false*

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

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

El comando `\spcomp`

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

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

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

Llaves para `\spcomp`

`bar = {⟨true | false⟩}`

por defecto: *false*

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

`wrap-parens = {⟨auto | true | false⟩}`

por defecto: *auto*

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

`read-arg = {⟨verbalización NVDA⟩}`

por defecto: *no usado*

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

Por ejemplo: `$\spcomp[read-arg={no ocurre A}]{A}$` imprimirá ‘ A^c ’ en la salida PDF y NVDA/MathCAT verbalizará «*no ocurre A*».

`silent = {⟨true | false⟩}`

por defecto: *false*

Establece si se *silencia* la verbalización en NVDA/MathCAT del complemento al ejecutar el comando sin modificar la impresión de este. Si se establece en *true*, NVDA/MathCAT NO verbalizará la salida del comando; si se establece en *false*, sí verbalizará la salida del comando.

El comando `\spcomb`

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

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

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

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

Llaves para $\backslash \text{spcomb}$

$\text{kind} = \{ \langle combination \mid variation \mid arrangement \mid permutation \rangle \}$ 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 $\$ \backslash \text{spcomb}[\text{kind=permutation}]\{n\}$, se imprime P_n y se verbaliza «permutación de n elementos», y con él, con $\$ \backslash \text{spcomb}[\text{kind=permutation}]\{n,k\}$, se imprime P_k^n y se verbaliza «permutación de n elementos tomados de a k».

$\text{repeat} = \{ \langle true \mid false \rangle \}$ 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: $\$ \backslash \text{spcomb}[\text{kind=variation,repeat=true}]\{5,2\}$ 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: $\$ \backslash \text{spcomb}[\text{kind=permutation,repeat=true}]\{10,3,3,4\}$ imprimirá $PR_{10}^{3,3,4}$ y verbalizará «permutación con repetición de 10 elementos con repeticiones 3 3 y 4».

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

Establece si se trata de una *permutación circular*. Si se establece en *true*, $\$ \backslash \text{spcomb}[\text{kind=permutation,circular=true}]\{n\}$ 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 $\{ \langle n \rangle \}$.

$\text{print-sty} = \{ \langle auto \mid binom \mid letter \rangle \}$ 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*, $\$ \backslash \text{spcomb}[\text{print-sty=letter}]\{5,2\}$ 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.

$\text{read-arg} = \{ \langle verbalización NVDA \rangle \}$ 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 $\{ \quad \}$.

Por ejemplo: $\$ \backslash \text{spcomb}[\text{read-arg}=\{ \langle cinco sobre dos \rangle \}]\{5,2\}$ imprimirá $\binom{5}{2}$ en la salida PDF y NVDA/MathCAT verbalizará «cinco sobre dos».

$\text{silent} = \{ \langle true \mid false \rangle \}$ 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.

- 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 $\backslash \text{spfact}$

$\backslash \text{spfact} \quad \backslash \text{spfact}\{ \langle n \rangle \} \quad \backslash \text{spfact}[\langle key = val \rangle]\{ \langle n \rangle \}$

El comando $\backslash \text{spfact}$ 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: $\$ \backslash \text{spfact}\{ (n-1) \}$ imprimirá $(n-1)!$.

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

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

Llaves para $\backslash \text{spfact}$

$\text{read-arg} = \{ \langle verbalización NVDA \rangle \}$ 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 $\{ \quad \}$.

Por ejemplo: `$\spfact[read-arg={cinco factorial}]{5}$` imprimirá ‘5!’ en la salida PDF y NV-DA/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-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-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-tool

El script `spmml-tool` se instala junto con el paquete. Prepara los archivos de `MathML` que genera `luamml[3]` 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 `spmml-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 `spmml-i-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=(<i>carpeta</i>)	Carpeta donde se escriben los archivos, que se crea si no existe.
-e, --engine=(<i>motor</i>)	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 `spmml-i-tool` es distinto de cero. Pueden indicarse varios archivos a la vez y la extensión puede omitirse. Algunos ejemplos:

```
spmml-i-tool -c ejemplo.pdf
spmml-i-tool -c -s ejemplo
spmml-i-tool ejemplo-luamml-mathml.html
spmml-i-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 Lua_lTeX. 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 Lua_lTeX. The attached files are intended as examples and, in a way, to save time for Matthew Bertucci (@mbertucci), who runs the tests on the _lTeX side, when he sees this package and adds it to The _lTeX 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 `spmml-i-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`, `\sp0omega`, `\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 [30], 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` [23], a construir baterías de tests de regresión, a comprender los modos `mathml-SE` y `mathml-AF`, a escuchar con `NVDA/MathCAT` [13, 14] 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, pero esto no termina aquí: debo seguirle el paso al proyecto de etiquetado de $\LaTeX$ [22] y a `luamml` [3], y pensar en las utilidades que hoy quedaron fuera. 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 [1]. 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.

12 Referencias

- [1] 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.
- [2] ROBERTSON, WILL. “unicode-math - Unicode mathematics support for $X_{\LaTeX}$ and $\text{Lua}\TeX$ ”. Available from CTAN, <https://www.ctan.org/pkg/unicode-math>, 2023.
- [3] KRUEGER, MARCEL. “LuaMML - Automatically generate MathML from $\text{Lua}\TeX$ math mode material”. Available from CTAN, <https://ctan.org/pkg/luamml>, 2026.
- [4] The $\LaTeX$ Project. “fontspec - Advanced font selection for $X_{\LaTeX}$ and $\text{Lua}\TeX$ ”. Available from CTAN, <https://www.ctan.org/pkg/fontspec>, 2025.
- [5] VOß, HERBERT. “libertinus-otf - Support for Libertinus OpenType”. Available from CTAN, <https://www.ctan.org/pkg/libertinus-otf>, 2025.
- [6] FLIPO, DANIEL. “erewhon-math - Utopia based OpenType Math font”. Available from CTAN, <https://ctan.org/pkg/erewhon-math>, 2026.
- [7] HOFSTRA, SILKE. “sourcecodepro - Use SourceCodePro with $\TeX$ (-alike) systems”. Available from CTAN, <https://ctan.org/pkg/sourcecodepro>, 2025.
- [8] TSOLOMITIS, ANTONIS. “newcomputermodern - The New Computer Modern family of fonts”. Available from CTAN, <https://ctan.org/pkg/newcomputermodern>, 2026.
- [9] GONZÁLEZ, PABLO. “scontents - Stores $\LaTeX$ contents in memory or files”. Available from CTAN, <https://www.ctan.org/pkg/scontents>, 2026.
- [10] GONZÁLEZ, PABLO. “enumext - Enumerate exercise sheets”. Available from CTAN, <https://www.ctan.org/pkg/enumext>, 2026.
- [11] BRAAMS, JOHANNES AND BEZOS, JAVIER. “babel - Multilingual support for $\LaTeX$, $\text{Lua}\TeX$, $X_{\LaTeX}$, and Plain $\TeX$ ”. Available from CTAN, <https://www.ctan.org/pkg/babel>, 2026.
- [12] BEZOS, JAVIER. “babel-spanish - Babel support for Spanish”. Available from CTAN, <https://www.ctan.org/pkg/babel-spanish>, 2026.
- [13] NV ACCESS. “NVDA - NonVisual Desktop Access screen reader”. Available from NV ACCESS, <https://www.nvaccess.org/>, 2026.

- [14] SOIFFER, NEIL. “MathCAT - Math to speech and braille translation”. Available from GITHUB, <https://github.com/daisy/MathCAT>, 2026.
- [15] SOIFFER, NEIL. “MathCAT Demo - Online demonstration of MathCAT”. Available from GITHUB, <https://daisy.github.io/MathCATDemo/>, 2026.
- [16] ADOBE INC. “Adobe Acrobat Reader - Reliable PDF viewer”. Available from ADOBE, <https://get.adobe.com/reader/>, 2026.
- [17] ISO. “ISO 32000-2:2020 - Portable Document Format (PDF 2.0)”. Available from PDF ASSOCIATION, <https://pdfa.org/resource/iso-32000-2/>, 2020.
- [18] PDF ASSOCIATION. “WTPDF - Well-Tagged PDF specification”. Available from PDFA, <https://pdfa.org/wtpdf/>, 2024.
- [19] PDF ASSOCIATION. “PDF 2.0 Errata Collection 3”. Available from PDFA, <https://pdfa.org/PDF-2-0-errata-collection-3-now-available/>, 2026.
- [20] LATEX PROJECT LWG. “Best Practice Guide: Math in PDF”. Available from PDFA, <https://pdfa.org/resource/best-practice-guide-math-in-PDF/>, 2026.
- [21] JOHNSON, DUFF. “PDF 2.0: New Features, Real-World Impact”. Available from PDFA, <https://pdfa.org/PDF-2-0-new-features-real-world-impact/>, 2026.
- [22] The L^AT_EX Project. “LaTeX Tagged PDF Project - Documentation and resources”. Available from LATEX PROJECT, <https://latex3.github.io/tagging-project/documentation/>, 2026.
- [23] VERAPDF CONSORTIUM. “veraPDF - Industry supported PDF/A and PDF/UA validator”. Available from VERAPDF, <https://verapdf.org/>, 2026.
- [24] DUAL LAB. “ngPDF - Next Generation PDF demonstrator and validator”. Available from NGPDF, <https://ngpdf.com/>, 2026.
- [25] BERRY, KARL. “L^AT_EX 2_ε: An Unofficial Reference Manual”. Available from CTAN, <https://ctan.org/pkg/latex2e-help-texinfo>, 2026.
- [26] The L^AT_EX Project. “Extensive support for hypertext in L^AT_EX”. Available from CTAN, <https://www.ctan.org/pkg/hyperref>, 2026.
- [27] The L^AT_EX Project. “The expl3 package”. Available from CTAN, <https://www.ctan.org/pkg/l3kernel>, 2026.
- [28] The L^AT_EX Project. “The L^AT_EX 3 Interfaces”. Available from CTAN, <https://www.ctan.org/pkg/l3kernel>, 2026.
- [29] The L^AT_EX Project. “The L^AT_EX 2_ε sources”. Available from CTAN, <https://ctan.org/tex-archive/macros/latex/base>, 2026.
- [30] The L^AT_EX Project. “L^AT_EX News, Issue 42, November 2025”. Available from CTAN, <https://ctan.org/tex-archive/macros/latex/base>, 2025.
- [31] The L^AT_EX Project. “L^AT_EX for authors current version”. Available from CTAN, <https://ctan.org/pkg/latex-base>, 2026.
- [32] LEARNLATEX.ORG. “Learn L^AT_EX - Your first steps in L^AT_EX, in short lessons”. Available from LEARNLATEX.ORG, <https://www.learnlatex.org/es/>, 2026.
- [33] FISCHER, ULRIKE. “tagpdf – L^AT_EX kernel code for PDF tagging”. Available from CTAN, <https://www.ctan.org/pkg/tagpdf>, 2026.
- [34] The L^AT_EX Project. “latex-lab – L^AT_EX laboratory”. Available from CTAN, <https://www.ctan.org/pkg/latex-lab>, 2026.

13 Change history

- v1.1 (ctan), 2026-10-22**
 - The command-line tool `spitool` is renamed `spmml-intent-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	36
\spPerc	43
\spPfig	32
\spPoint	23
\spPoly	30
\spProb	46
\spProm	42
\spProp	22
\spPval	45
\spVol	38
\spabs	41
\spangle	27
\spang	28
\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	17
\spdom	40
\spdsym	16
\spfact	47
\spfuncomp	41
\spfun	39
\spinterval	38
\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	24
\sprec	40
\spriq	43
\spshort	12
\spside	26
\spsiglo	14
\sptime	15
\spunitalias	10
\spunit	8
\spusep	16
\spvarz	42
\spvec	24
K	
Llaves de \spabs provistas por spanish-mathml-intent :	
close	41
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	27
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	44
silent	45
Llaves de \spcirc provistas por spanish-mathml-intent :	
concept	35
diameter	35

prefix	34	silent	48
print-sym	34	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	34	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	39
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	29
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	33	Llaves de \spmcm provistas por spanish-mathml-intent:	
read-arg	34	concept	20
read-cmd	33	prefix	20
read-sub	34	read-sty	20
silent	34	result	20
sub-index	33	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	40	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	37
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*	32
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	26	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	23
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	18	silent	22
result	18	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	33
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

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

read-arg	38
read-cmd	38
read-sub	38
silent	38
sub-index	38
Llaves de \usepackage provistas por spanish-mathml-intent:	
mathcal	2

M

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

P

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

T

Herramienta provista por spanish-mathml-intent:	
spmml-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 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>
```

15.3 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} {2026-10-22} {1.1} {Spanish parse intents}
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, Lua_TE_X 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 \IfDocumentMetadataF
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 }

```

15.4 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`.)

15.5 Variables for the key `\mathcal`, `\rightanglesqr` and `\attribute`

```

\l__spintent_luaset_cal_fam_ok_str
\l__spintent_luaset_sym_fam_ok_str
\l__spintent_luaset_scope_attr_str

```

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

```

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`.)

15.6 Loading the Lua module and functions

```
spintent.lua
```

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

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

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

`\__spintent_luafun_set_debug_ghost:n` Passes `AFghost` to Lua as a plain string. If it is `true`, the two ghost characters are marked with `sptmp="outer"` or `sptmp="inner"` in the file `-luamml-mathml.html`, so that the Lua script `spmml-intent-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 { \__spintent_luafun_set_debug_ghost:n { true } }
79 { \__spintent_luafun_set_debug_ghost:n { false } }

```

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

`\l__spintent_scope_attr_int` A LuaTeX attribute number allocated from Lua with `luatexbase.new_attribute` and exposed as an integer. Every command that uses the function `\__spintent_luafun_mrow_intent_arg:nn` sets `\attribute` `\l__spintent_scope_attr_int = 1` inside its own group, which marks the nodes typeset within it as eligible for the queue.

```

80 \int_new:N \l__spintent_scope_attr_int
81 \int_set:NV \l__spintent_scope_attr_int \l__spintent_luafun_set_scope_attr_str

```

(End of definition for `\l__spintent_scope_attr_int`.)

`\__spintent_luafun_spunit_parse_product:nV` Variants generated for `\__spintent_luafun_spunit_parse_product:nn`, used by `\spunit`;
`\__spintent_luafun_spmoney_lookup_data:V` `\__spintent_luafun_spmoney_lookup_data:n`, used by `\spmoney`; `\__spintent_luafun_spmQ_scale_int:n` used by `\spmQ`; and `\__spintent_luafun_spfun_parse_and_set:n`, used by `\spfun`.

```

82 \cs_generate_variant:Nn \__spintent_luafun_spunit_parse_product:nn { nV }
83 \cs_generate_variant:Nn \__spintent_luafun_spmoney_lookup_data:n { V }
84 \cs_generate_variant:Nn \__spintent_luafun_spmQ_scale_int:nn { Vn }
85 \cs_generate_variant:Nn \__spintent_luafun_spfun_parse_and_set:n { v }

```

(End of definition for `\__spintent_luafun_spunit_parse_product:nV` and others.)

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

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

`mathcal-fallback` 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   \__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`.)

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

```

100 \cs_new_protected:Nn \__spintent_mathcal:n
101 {
102   \bool_if:NTF \l__spintent_mathcal_bool
103   {
104     \str_if_eq:VnTF \l__spintent_luafun_classic_mathcal:n { true }
105     { \__spintent_luafun_classic_mathcal:n {#1} }
106     {
107       \msg_warning:nn { spintent } { mathcal-fallback }
108       \mathcal {#1}

```

```

109     }
110   }
111   { \mathcal {#1} }
112 }

```

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

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

```

113 \cs_new_protected:Nn \__spintenc_right_angle_sqr:
114 {
115   \str_if_eq:VnTF \__spintenc_luaset_sym_fam_ok_str { true }
116   { \__spintenc_luafun_right_angle_sqr: }
117   {
118     \msg_warning:nn { spintenc } { right-angle-sqr-fallback }
119     \rightangle
120   }
121 }

```

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

15.8 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 `spintenc.lua`.

`\__spintenc_luafun_mrow_intent_arg:nn` The internal function `\__spintenc_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[3]`, 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 `\__spintenc_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.

`\__spintenc_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 \__spintenc_invisible_sep:
123 {
124   \__spintenc_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 `spmmlt-tool`.

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

`\__spintenc_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 \__spintenc_inner_sep:
127 {
128   \__spintenc_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 `sम्मli-tool`.

(End of definition for `\__spinttent_inner_sep:`)

`\c__spinttent_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`.

```
130 \hook_gput_code:nnn { begindocument } { spinttent }
131 {
132   \Umathchardef \c__spinttent_invisible_sep_int = "0 "0 "2063
133 }
```

- 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__spinttent_invisible_sep_int`)

15.9 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.

`\__spinttent_collapse_hyphens:N` A recursive function to collapse multiple hyphens into a single one.

```
134 \cs_new_protected:Nn \__spinttent_collapse_hyphens:N
135 {
136   \tl_if_in:NnT #1 { -- }
137   {
138     \tl_replace_all:Nnn #1 { -- } { - }
139     \__spinttent_collapse_hyphens:N #1
140   }
141 }
```

(End of definition for `\__spinttent_collapse_hyphens:N`)

`\__spinttent_sanitize_affix:Nn` The function `\__spinttent_sanitize_affix:Nn` checks if the `{⟨argument⟩}` passed in `#2` is *blank*. If so, it clears the target token list variable passed in `#1`. Otherwise, it converts the `{⟨argument⟩}` into a string with every character in category code 12, so it no longer matters how an underscore ‘`_`’ was typed, trims the bounds, converts spaces ‘`␣`’ and underscores ‘`_`’ to hyphens ‘`-`’, collapses multiple hyphens, and strips leading/trailing hyphens. Because underscores are converted *before* collapsing, any run mixing hyphens and underscores is reduced to a single hyphen, and the stored value never starts or ends with ‘`-`’ or ‘`_`’: the command that uses the value is the one that places the leading ‘`_`’ of the concept name. For example, if you pass a target variable and raw text:

```
\__spinttent_sanitize_affix:Nn \__spinttent_spsomesym_prefix_tl{ foo bar baz }
```

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

```
142 \cs_new_protected:Nn \__spinttent_sanitize_affix:Nn
143 {
144   \tl_if_blank:nTF {#2}
145   { \tl_clear:N #1 }
146   {
147     \tl_set:Ne #1 { \tl_to_str:e {#2} }
148     \tl_trim_spaces:N #1
149     \str_if_eq:VnTF #1 { _- }
150     { \tl_set:Nn #1 { - } }
151     {
152       \tl_replace_all:Nnn #1 { ~ } { - }
```

```

153         \tl_replace_all:Nee #1 { \tl_to_str:n { _ } } { - }
154         \__spinttent_collapse_hyphens:N #1
155         \tl_put_left:Nn #1 { \q_mark }
156         \tl_put_right:Nn #1 { \q_stop }
157         \tl_replace_all:Nnn #1 { \q_mark - } { \q_mark }
158         \tl_replace_all:Nnn #1 { - \q_stop } { \q_stop }
159         \tl_remove_all:Nn #1 { \q_mark }
160         \tl_remove_all:Nn #1 { \q_stop }
161     }
162 }
163 }
164 \cs_generate_variant:Nn \__spinttent_sanitizize_affix:Nn { cn }

```

(End of definition for __spinttent_sanitizize_affix:Nn.)

15.10 The command \setspinttent

The `\setspinttent` command is used to configure *(keys)* “globally” for the utilities provided by `spanish-mathml-intent`.

15.10.1 Definition of variables and error messages for \setspinttent

__spinttent_setspinttent_input_arg_cmd_tl
unknown-command-setup

The variable `\__spinttent_setspinttent_input_arg_cmd_tl` will *store* the command or command name passed to the “first” `{(argument)}` ‘#1’ of the command `\setspinttent`.

```

165 \tl_new:N \__spinttent_setspinttent_input_arg_cmd_tl

```

The “error” `unknown-command-setup` is raised when the `{(argument)}` ‘#1’ passed to `\setspinttent` is a “non-existent” command.

```

166 \msg_new:nnnn { spinttent } { unknown-command-setup }
167 {
168     Cannot~configure~unknown~command~'\c_backslash_str #1'~\msg_line_context:.
169 }
170 {
171     The~command~'\c_backslash_str #1'~is~not~defined.~Check~the~spelling~of~the~command~
172     name.
173 }

```

(End of definition for __spinttent_setspinttent_input_arg_cmd_tl and unknown-command-setup.)

15.10.2 Internal functions for \setspinttent

__spinttent_setspinttent_set_keys:nn
__spinttent_setspinttent_set_keys:en

The function `\__spinttent_setspinttent_set_keys:nn` will check if the command passed as `{(argument)}` exists in ‘#1’ and set the *(keys)* passed as `{(argument)}` in ‘#2’ to the correct path; otherwise, it will return an error.

```

174 \cs_new_protected:Nn \__spinttent_setspinttent_set_keys:nn
175 {
176     \cs_if_exist:CTF {#1}
177     {
178         \keys_set:nn { spinttent / #1 } {#2}
179     }
180     {
181         \msg_error:nnn { spinttent } { unknown-command-setup } {#1}
182     }
183 }
184 \cs_generate_variant:Nn \__spinttent_setspinttent_set_keys:nn { en }

```

(End of definition for __spinttent_setspinttent_set_keys:nn.)

__spinttent_setspinttent_parse_args:nn
__spinttent_setspinttent_parse_args:Vn

The function `\__spinttent_setspinttent_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 `\__spinttent_setspinttent_set_keys:en`. Otherwise, i.e., if ‘\’ was not passed, it will execute `\__spinttent_setspinttent_set_keys:nn`.

```

185 \cs_new_protected:Nn \__spinttent_setspinttent_parse_args:nn
186 {
187     \tl_if_single:nTF {#1}
188     {
189         \token_if_cs:NTF #1
190         {
191             \__spinttent_setspinttent_set_keys:en { \cs_to_str:N #1 } {#2}
192         }

```

```

193         {
194             \__spintent_setspinttent_set_keys:nn {#1} {#2}
195         }
196     }
197     {
198         \__spintent_setspinttent_set_keys:nn {#1} {#2}
199     }
200 }
201 \cs_generate_variant:Nn \__spintent_setspinttent_parse_args:nn { Vn }

```

(End of definition for __spintent_setspinttent_parse_args:nn.)

`\setspinttent` Now we define the user command `\setspinttent`.

```

202 \NewDocumentCommand \setspinttent { m m }
203 {
204     \tl_if_blank:nTF {#1}
205     {
206         \msg_error:nnn { spinttent } { unknown-command-setup } { <empty> }
207     }
208     {
209         \tl_set:Nn \l__spinttent_setspinttent_input_arg_cmd_tl {#1}
210         \tl_trim_spaces:N \l__spinttent_setspinttent_input_arg_cmd_tl
211         \__spinttent_setspinttent_parse_args:Vn \l__spinttent_setspinttent_input_arg_cmd_tl
212             {#2}
213     }
214 }

```

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

15.11 Handling unknown keys

`\l__spinttent_command_name_str`
`unknown-choice`
`cmd-key-unknown`
`cmd-key-value-unknown`

We define the variable `\l__spinttent_command_name_str` which will store the “name” of each command within the implementation along with the messages `unknown-choice` used by `<keys>` that use `.choice:n` in their implementation and the messages `cmd-key-unknown`, `cmd-key-value-unknown` used by the `<key>` `unknown` in commands that use `[<key=val>]`.

```

215 \str_new:N \l__spinttent_command_name_str
216 \msg_new:nnnn { spinttent } { unknown-choice }
217 {
218     Invalid~value~'#3'~for~key~'#1';~use~one~of:~#2~\msg_line_context:.
219 }
220 {
221     The~key~'#1'~only~accepts~one~of:~#2.
222 }
223 \msg_new:nnnn { spinttent } { cmd-key-unknown }
224 {
225     The~key~'#1'~is~unknown~by~command~
226     '\c_backslash_str \l__spinttent_command_name_str'~and~is~being~ignored~
227     \msg_line_context:.
228 }
229 {
230     Check~the~spelling~of~the~key~name.
231 }
232 \msg_new:nnnn { spinttent } { cmd-key-value-unknown }
233 {
234     The~key~'#1=#2'~is~unknown~by~command~
235     '\c_backslash_str \l__spinttent_command_name_str'~and~is~being~ignored~
236     \msg_line_context:.
237 }
238 {
239     Check~the~spelling~of~the~key~name.
240 }

```

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

`\__spinttent_unknown_keys_cmd:nn`
`\__spinttent_unknown_keys_cmd:Vn`
`\__spinttent_unknown_keys_cmd:n`

The internal function `\__spinttent_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 `\__spinttent_unknown_keys_cmd:Vn` takes the name of the key from a variable. The function `\__spinttent_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 `\__spintent_command_name_str`.

```

241 \cs_new_protected:Nn \__spintent_unknown_keys_cmd:nn
242 {
243   \tl_if_blank:nTF {#2}
244     { \msg_error:nnn { spintent } { cmd-key-unknown } {#1} }
245     { \msg_error:nnnn { spintent } { cmd-key-value-unknown } {#1} {#2} }
246 }
247 \cs_generate_variant:Nn \__spintent_unknown_keys_cmd:nn { Vn }
248 \cs_new_protected:Nn \__spintent_unknown_keys_cmd:n
249 {
250   \__spintent_unknown_keys_cmd:Vn \l_keys_key_str {#1}
251 }

```

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

15.12 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”.

```

252 \tl_new:N \l__spintent_luaset_sign_tl
253 \tl_new:N \l__spintent_luaset_part_int_tl
254 \tl_new:N \l__spintent_luaset_dec_sep_tl
255 \tl_new:N \l__spintent_luaset_part_dec_tl
256 \tl_new:N \l__spintent_luaset_part_period_tl
257 \str_new:N \l__spintent_luaset_has_integer_str

```

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

`\l__spintent_luaset_only_part_int_str`
`\l__spintent_luaset_only_part_dec_str`
`\l__spintent_luaset_only_part_period_str`
`\l__spintent_luaset_format_part_int_str`
`\l__spintent_luaset_format_part_dec_str`

The literal entries of the numbers that we will pass to `\MathMLintent` with the *intent* property :number are stored in `\l__spintent_luaset_only_part_int_str` for the “integer part”, in `\l__spintent_luaset_only_part_dec_str` for the “finite decimal part”, and in `\l__spintent_luaset_only_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”.

```

258 \str_new:N \l__spintent_luaset_only_part_int_str
259 \str_new:N \l__spintent_luaset_only_part_dec_str
260 \str_new:N \l__spintent_luaset_only_part_period_str
261 \str_new:N \l__spintent_luaset_format_part_int_str
262 \str_new:N \l__spintent_luaset_format_part_dec_str

```

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

`\l__spintent_luaset_millions_str`
`\l__spintent_luaset_decimal_period_str`
`\l__spintent_luaset_not_decimal_period_str`
`\l__spintent_luaset_not_decimal_not_period_str`
`\l__spintent_luaset_dec_is_all_zeros_str`
`\l__spintent_luaset_has_exponent_str`
`\l__spintent_luaset_exponent_tl`

The flag variable `\l__spintent_luaset_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`.

```

263 \str_new:N \l__spintent_luaset_millions_str

```

```

264 \str_new:N \__spintent_luaset_decimal_period_str
265 \str_new:N \__spintent_luaset_not_decimal_period_str
266 \str_new:N \__spintent_luaset_not_decimal_not_period_str
267 \str_new:N \__spintent_luaset_dec_is_all_zeros_str
268 \str_new:N \__spintent_luaset_has_exponent_str
269 \tl_new:N \__spintent_luaset_exponent_tl

```

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

```

\__spintent_luaset_has_units_str
\__spintent_luaset_status_str
\__spintent_luaset_arg_above_tl
\__spintent_luaset_arg_below_tl
\__spintent_luaset_denom_has_number_str

```

Since the function `\__spintent_luafun_clean_split_and_set:n` is common to `\spnum` (§15.14) and `\spunit` (§15.15), 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 `\__spintent_luaset_has_units_str` accordingly. Following this, it verifies if this text contains more than one “/”. It then sets the variable `\__spintent_luaset_status_str` to `valid` or `multislash`. If the structure is `valid`, it splits the unit into `\__spintent_luaset_arg_above_tl` and `\__spintent_luaset_arg_below_tl`. Finally, it analyzes `\__spintent_luaset_arg_below_tl`; if the denominator starts with a number, it sets `\__spintent_luaset_denom_has_number_str` to `true`, otherwise to `false`.

```

270 \str_new:N \__spintent_luaset_has_units_str
271 \str_new:N \__spintent_luaset_status_str
272 \tl_new:N \__spintent_luaset_arg_above_tl
273 \tl_new:N \__spintent_luaset_arg_below_tl
274 \str_new:N \__spintent_luaset_denom_has_number_str

```

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

15.13 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.

```

275 \msg_new:nnnn { spintent } { need-math-mode }
276 {
277   Math~mode~required~for~'\tl_trim_spaces:n {#1}'~\msg_line_context:.
278 }
279 {
280   Use~the~command~inside~'\$....\$'~or~in~a~math~environment.
281 }
282 \msg_new:nnnn { spintent } { need-text-mode }
283 {
284   Text~mode~required~for~'\tl_trim_spaces:n {#1}'~\msg_line_context:.
285 }
286 {
287   Use~the~command~outside~of~'\$....\$'~and~math~environments.
288 }

```

15.14 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.

15.14.1 Definition of variables for `\spnum`

```

\__spintent_spnum_read_terse_bool
\__spintent_spnum_read_period_sep_str
\__spintent_spnum_notation_sym_tl
\__spintent_spnum_intent_read_sign_str
\__spintent_spnum_intent_read_dec_sep_str

```

The variable `\__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 `\__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 $\{\langle argument \rangle\}$ 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 `\__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 `\__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_spnun_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`.

```

289 \bool_new:N \l__spintent_spnun_read_terse_bool
290 \str_new:N \l__spintent_spnun_read_period_sep_str
291 \tl_new:N \l__spintent_spnun_notation_sym_tl
292 \str_new:N \l__spintent_spnun_intent_read_sign_str
293 \str_new:N \l__spintent_spnun_intent_read_dec_sep_str

```

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

15.14.2 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 $\langle argument \rangle$ passed in ‘#1’, advising them to use the `\spunit` command instead. The “error” `spnum-no-digits` is raised when the provided $\langle argument \rangle$ contains no numerical digits. Since other commands, such as `\spdiv` and `\spmul`, delegate to `\spnum`, these messages do not mention the command name.

```

294 \msg_new:nnnn { spintent } { spnum-has-units }
295 {
296   Units~are~not~allowed~in~a~numerical~value~('#1')~\msg_line_context:.
297 }
298 {
299   Use~'\token_to_str:N \spunit'~to~print~numbers~with~units.
300 }
301 \msg_new:nnnn { spintent } { spnum-no-digits }
302 {
303   Invalid~numerical~value~'#1'~\msg_line_context:.
304 }
305 {
306   A~numerical~value~must~contain~at~least~one~digit.
307 }

```

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

15.14.3 Definition of keys for `\spnum`

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

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

```

308 \keys_define:nn { spintent / spnum }
309 {
310   cifras-sep      .bool_set:N = \l__spintent_spnun_cifras_sep_bool,
311   cifras-sep      .initial:n = true,
312   cifras-sep      .value_required:n = true,
313   read-sign       .choices:nn =
314     { terse , clear }
315     {
316       \int_case:nn { \l_keys_choice_int }
317       {
318         {1} { \bool_set_true:N \l__spintent_spnun_read_terse_bool }
319         {2} { \bool_set_false:N \l__spintent_spnun_read_terse_bool }
320       }
321     },
322   read-sign       .initial:n = terse,
323   read-sign       .value_required:n = true,
324   read-sign / unknown .code:n =
325     \msg_error:nnee { spintent } { unknown-choice }
326     { read-sign } { terse , ~clear } { \exp_not:n {#1} },
327   read-period-sep .choices:nn =
328     { coma , punto }
329     {
330       \int_case:nn { \l_keys_choice_int }
331       {
332         {1} { \str_set:Nn \l__spintent_spnun_read_period_sep_str { coma } }
333         {2} { \str_set:Nn \l__spintent_spnun_read_period_sep_str { punto } }
334       }
335     },
336   read-period-sep .initial:n = coma,
337   read-period-sep .value_required:n = true,
338   read-period-sep / unknown .code:n =
339     \msg_error:nnee { spintent } { unknown-choice }
340     { read-period-sep } { coma , ~punto } { \exp_not:n {#1} },
341   notation-sym    .choices:nn =

```

```

342     { times , cdot }
343     {
344         \int_case:nn { \l_keys_choice_int }
345         {
346             {1} { \tl_set:Nn \l__spinttent_spnun_notation_sym_tl { \times } }
347             {2} { \tl_set:Nn \l__spinttent_spnun_notation_sym_tl { \cdot } }
348         }
349     },
350     notation-sym .initial:n = cdot,
351     notation-sym .value_required:n = true,
352     notation-sym / unknown .code:n =
353         \msg_error:nneee { spinttent } { unknown-choice }
354         { notation-sym } { times , ~ cdot } { \exp_not:n {#1} },
355     unknown .code:n =
356     {
357         \str_set:Nn \l__spinttent_command_name_str { spnum }
358         \__spinttent_unknown_keys_cmd:n {#1}
359     },
360 }

```

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

15.14.4 Internal functions for \spnum

```

\__spinttent_spnun_render_part:n
\__spinttent_spnun_render_int:
\__spinttent_spnun_render_dec:

```

The function `\__spinttent_spnun_render_part:n` processes the $\{ \langle argument \rangle \}$ passed in ‘#1’ used by the functions `\__spinttent_spnun_render_int:` and `\__spinttent_spnun_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.

```

361 \cs_new_protected:Nn \__spinttent_spnun_render_part:n
362 {
363     \MathMLintent { \str_use:c { \l__spinttent_luaaset_only_part_#1_str } :number }
364     {
365         \bool_if:NTF \l__spinttent_spnun_cifras_sep_bool
366         { \str_use:c { \l__spinttent_luaaset_format_part_#1_str } }
367         { \tl_use:c { \l__spinttent_luaaset_part_#1_tl } }
368     }
369 }
370 \cs_new_protected:Nn \__spinttent_spnun_render_int:
371 {
372     \__spinttent_spnun_render_part:n { int }
373 }
374 \cs_new_protected:Nn \__spinttent_spnun_render_dec:
375 {
376     \__spinttent_spnun_render_part:n { dec }
377 }

```

(End of definition for `\__spinttent_spnun_render_part:n`, `\__spinttent_spnun_render_int:`, and `\__spinttent_spnun_render_dec:`.)

```

\__spinttent_spnun_render_only_period:
\__spinttent_spnun_print_period_bar:
\__spinttent_spnun_render_period_sep:

```

The function `\__spinttent_spnun_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 `\__spinttent_spnun_print_period_bar:`.

```

378 \cs_new_protected:Nn \__spinttent_spnun_render_only_period:
379 {
380     \MathMLintent { \l__spinttent_luaaset_only_part_period_str :number }
381     {
382         \l__spinttent_luaaset_part_period_tl
383     }
384 }

```

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

```

385 \cs_new_protected:Nn \__spinttent_spnun_print_period_bar:
386 {
387     \MathMLintent { _periódico:postfix($dp) }
388     {
389         {
390             \overline
391             {

```

```

392         \MathMLarg { dp } { \__spinttent_snum_render_only_period: }
393     }
394 }
395 }
396 }

```

The function `\__spinttent_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.

```

397 \cs_new_protected:Nn \__spinttent_snum_render_period_sep:
398 {
399     \str_if_eq:VnTF \__spinttent_snum_read_period_sep_str { coma }
400     {
401         \MathMLintent { _coma:prefix($pp) }
402         {
403             \mathord{,}
404             \MathMLarg { pp } { \__spinttent_snum_print_period_bar: }
405         }
406     }
407     {
408         \MathMLintent { _punto:prefix($pp) }
409         {
410             \mathord{.}
411             \MathMLarg { pp } { \__spinttent_snum_print_period_bar: }
412         }
413     }
414 }

```

(End of definition for `\__spinttent_snum_render_only_period:`, `\__spinttent_snum_print_period_bar:`, and `\__spinttent_snum_render_period_sep:`.)

`\__spinttent_snum_render_decimal_sep:`

The function `\__spinttent_snum_render_decimal_sep:` sets the variable `\__spinttent_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 `\__spinttent_luaset_dec_sep_tl`. We print the decimal separator stored in `\__spinttent_luaset_dec_sep_tl` using `\mathord` to remove spaces around it and then use the command `\MathMLarg` where we insert `\__spinttent_invisible_sep:` for compatibility with `MathML` and finally call the function `\__spinttent_snum_render_dec:` to print the finite decimal part.

```

415 \cs_new_protected:Nn \__spinttent_snum_render_decimal_sep:
416 {
417     \str_if_eq:VnTF \__spinttent_luaset_dec_sep_tl { , }
418     { \str_set:Nn \__spinttent_snum_intent_read_dec_sep_str { _coma } }
419     { \str_set:Nn \__spinttent_snum_intent_read_dec_sep_str { _punto } }
420     \MathMLintent { \__spinttent_snum_intent_read_dec_sep_str :prefix($dp) }
421     {
422         \mathord { \__spinttent_luaset_dec_sep_tl }
423         \MathMLarg { dp }
424         {
425             % keep "dp" in mathml-AF
426             \MathMLintent { :silent } { \c__spinttent_invisible_sep_int }
427             \__spinttent_snum_render_dec:
428         }
429     }
430     % period part
431     \str_if_eq:VnTF \__spinttent_luaset_decimal_period_str { true }
432     {
433         \str_if_eq:VnTF \__spinttent_luaset_dec_is_all_zeros_str { true }
434         {
435             \__spinttent_inner_sep:
436             \__spinttent_snum_print_period_bar:
437         }
438         {
439             \MathMLintent { _con:prefix($pnum) }
440             {
441                 \__spinttent_inner_sep:
442                 \MathMLarg { pnum }
443                 {
444                     \__spinttent_snum_print_period_bar:
445                 }
446             }
447         }
448     }
449 }

```

(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.

```

450 \cs_new_protected:Nn \__spintent_snum_print_part_dec:
451 {
452   \tl_if_empty:NF \l__spintent_luaset_part_dec_tl
453   {
454     \__spintent_snum_render_decimal_sep:
455   }
456 }
457 \cs_new_protected:Nn \__spintent_snum_print_part_period:
458 {
459   \tl_if_empty:NF \l__spintent_luaset_part_period_tl
460   {
461     \str_if_eq:VnTF \l__spintent_luaset_not_decimal_period_str { true }
462     { \__spintent_snum_render_period_sep: }
463     {
464       \tl_if_empty:NT \l__spintent_luaset_part_dec_tl
465       {
466         \__spintent_snum_render_period_sep:
467       }
468     }
469   }
470 }

```

(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:`.

```

471 \cs_new_protected:Nn \__spintent_snum_print_integer:
472 {
473   \str_if_eq:VnTF \l__spintent_luaset_not_decimal_not_period_str { true }
474   { \__spintent_snum_render_int: }
475   {
476     \MathMLintent { \l__spintent_luaset_only_part_int_str :number:prefix($next) }
477     {
478       \bool_if:NTF \l__spintent_snum_cifras_sep_bool
479       { \l__spintent_luaset_format_part_int_str }
480       { \l__spintent_luaset_part_int_tl }
481       \MathMLarg { next }
482       {
483         \tl_if_empty:NTF \l__spintent_luaset_part_dec_tl
484         { \__spintent_snum_print_part_period: }
485         { \__spintent_snum_print_part_dec: }
486       }
487     }
488   }
489 }
490 \cs_new_protected:Nn \__spintent_snum_print_num_start:
491 {
492   \tl_if_empty:NTF \l__spintent_luaset_sign_tl
493   {
494     \__spintent_invisible_sep: % need
495     \__spintent_snum_print_integer:

```

```

496     }
497   {
498     \bool_if:NTF \l__spintent_spnum_read_terse_bool
499     {
500       \str_if_eq:NnTF \l__spintent_luaset_sign_tl { + }
501       { \str_set:Nn \l__spintent_spnum_intent_read_sign_str { _más:prefix($n)} }
502       {
503         \str_set:Nn \l__spintent_spnum_intent_read_sign_str { _menos:prefix($n)}
504       }
505     }
506     {
507       \str_if_eq:NnTF \l__spintent_luaset_sign_tl { + }
508       {
509         \str_set:Nn \l__spintent_spnum_intent_read_sign_str
510         { _positivo:postfix($n) }
511       }
512       {
513         \str_set:Nn \l__spintent_spnum_intent_read_sign_str
514         { _negativo:postfix($n) }
515       }
516     }
517     \__spintent_invisible_sep: % need
518     \MathMLintent { \l__spintent_spnum_intent_read_sign_str }
519     {
520       \l__spintent_luaset_sign_tl
521       \MathMLarg { n } { \__spintent_spnum_print_integer: }
522     }
523   }
524 }
525 \cs_new_protected:Nn \__spintent_spnum_mathml:
526 {
527   \__spintent_spnum_print_num_start:
528   \str_if_eq:NnTF \l__spintent_luaset_has_exponent_str { true }
529   {
530     \MathMLintent { _por } { \l__spintent_spnum_notation_sym_tl }
531     10^{ \l__spintent_luaset_exponent_tl }
532   }
533 }

```

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

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

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

```

534 \NewDocumentCommand \spnum { O{} m }
535 {
536   \mode_if_math:TF
537   {
538     \group_begin:
539     \keys_set:nn { spintent / spnum } {#1}
540     \__spintent_luafun_clean_split_and_set:n { \exp_not:n {#2} }
541     \str_if_eq:NnTF \l__spintent_luaset_has_units_str { true }
542     { \msg_error:nnn { spintent } { spnum-has-units } {#2} }
543     {
544       \tl_if_empty:NTF \l__spintent_luaset_part_int_tl
545       { \msg_error:nnn { spintent } { spnum-no-digits } {#2} }
546       { \__spintent_spnum_mathml: }
547     }
548     \group_end:
549   }
550   { \msg_error:nnn { spintent } { need-math-mode } { \spnum } }
551 }

```

(End of definition for `\spnum`. This function is documented on page 8.)

15.15 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 $\langle \textit{argument} \rangle$ to Lua module `spintent.lua` via `\__spintent_luafun_clean_split_and_set:n` to separate and clean the $\langle \textit{argument} \rangle$.

This ensures that all “units”, including SI fractions, products, and sexagesimal angles, are processed consistently by the `spintent.lua`.

15.15.1 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 $\langle i \rangle$ of a product $\langle p \rangle$, Lua also leaves the canonical symbol, the speech text and the exponent. The canonical symbol is stored in `\__spintent_spunit_luaset_<p>_<i>_canonical_str`, the speech text in `\__spintent_spunit_luaset_<p>_<i>_read_str` and the exponent, which is empty when there is none, in `\__spintent_spunit_luaset_<p>_<i>_exp_tl`.

```
552 \str_new:N \__spintent_spunit_luaset_above_n_str
553 \str_new:N \__spintent_spunit_luaset_above_found_str
554 \str_new:N \__spintent_spunit_luaset_above_sexa_str
555 \str_new:N \__spintent_spunit_luaset_below_n_str
556 \str_new:N \__spintent_spunit_luaset_below_found_str
```

(End of definition for `\__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.

```
557 \str_new:N \__spintent_spunit_luaset_canonical_str
558 \str_new:N \__spintent_spunit_luaset_read_str
559 \str_new:N \__spintent_spunit_luaset_status_str
```

(End of definition for `\__spintent_spunit_luaset_canonical_str`, `\__spintent_spunit_luaset_read_str`, and `\__spintent_spunit_luaset_status_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.

```
560 \tl_new:N \__spintent_spunit_exponent_tl
561 \str_new:N \__spintent_spunit_read_slash_str
```

(End of definition for `\__spintent_spunit_exponent_tl` and `\__spintent_spunit_read_slash_str`.)

15.15.2 Definition of error messages for `\spunit`

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 $\langle \textit{argument} \rangle$, 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.

```
562 \msg_new:nnnn { spintent } { spunit-invalid-unit }
563 {
564   Invalid~unit~expression~'~#1'~\msg_line_context:.
565 }
566 {
567   The~expression~contains~an~unknown~or~malformed~unit.~Check~the~spelling~or~verify~
568   that~the~unit~is~defined.
569 }
570 \msg_new:nnnn { spintent } { spunit-no-units }
571 {
572   No~unit~found~in~'~#1'~\msg_line_context:.
573 }
574 {
575   At~least~one~unit~is~required;~use~'\token_to_str:N \spnum'~for~pure~
576   numbers.
577 }
578 \msg_new:nnnn { spintent } { spunit-inline-numeric-denominator }
579 {
580   Invalid~numeric~coefficient~in~the~inline~denominator~'~#1'~\msg_line_context:.
581 }
```

```

582 {
583     Numbers~are~not~allowed~after~the~slash~'/'~in~the~inline~mode~of~units.
584 }

```

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

15.15.3 Definition of keys for `\spunit`

We define the *keys* `print-cdot`, `read-cdot`, `read-slash`, `cifras-sep`, `read-period-sep` and `notation-sym`.

```

585 \keys_define:nn { spintent / spunit }
586 {
587     print-cdot      .bool_set:N = \l__spintent_spunit_print_cdot_bool,
588     print-cdot      .initial:n = false,
589     print-cdot      .value_required:n = true,
590     read-cdot       .bool_set:N = \l__spintent_spunit_read_cdot_bool,
591     read-cdot       .initial:n = false,
592     read-cdot       .value_required:n = true,
593     read-slash      .choices:nn =
594         { por , partido-por }
595         {
596             \int_case:nn { \l_keys_choice_int }
597             {
598                 {1} { \str_set:Nn \l__spintent_spunit_read_slash_str { _por } }
599                 {2} { \str_set:Nn \l__spintent_spunit_read_slash_str { _partido-por } }
600             }
601         },
602     read-slash      .initial:n = por,
603     read-slash      .value_required:n = true,
604     read-slash / unknown .code:n =
605         \msg_error:nnee { spintent } { unknown-choice }
606         { read-slash } { por, ~partido-por } { \exp_not:n {#1} },
607     cifras-sep      .meta:nn = { spintent / spnum } { cifras-sep = {#1} },
608     cifras-sep      .value_required:n = true,
609     read-period-sep .meta:nn = { spintent / spnum } { read-period-sep = {#1} },
610     read-period-sep .value_required:n = true,
611     notation-sym    .meta:nn = { spintent / spnum } { notation-sym = {#1} },
612     notation-sym    .value_required:n = true,
613     unknown         .code:n =
614         {
615             \str_set:Nn \l__spintent_command_name_str { spunit }
616             \__spintent_unknown_keys_cmd:n {#1}
617         },
618 }

```

(End of definition for `print-cdot` and others.)

15.15.4 Internal functions for `\spunit`

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

```

619 \cs_new_protected:Nn \__spintent_spunit_format_canonical:
620 {
621     \str_case:VnF \l__spintent_spunit_luaset_canonical_str
622     {
623         { \ell } { \ell }
624     }
625     { \mathrm { \l__spintent_spunit_luaset_canonical_str } }
626 }

```

(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.

```

627 \cs_new_protected:Nn \__spintent_spunit_load_term:nn
628 {
629     \str_set_eq:Nc \l__spintent_spunit_luaset_canonical_str
630     { \__spintent_spunit_luaset_ #1 _ #2 _canonical_str }
631     \str_set_eq:Nc \l__spintent_spunit_luaset_read_str
632     { \__spintent_spunit_luaset_ #1 _ #2 _read_str }

```

```

633     \tl_set_eq:Nc \l__spintent_spunit_exponent_tl
634     { \l__spintent_spunit_luaset_ #1 _ #2 _exp_tl }
635 }

```

(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.

```

636 \cs_new_protected:Nn \__spintent_spunit_print_term:nn
637 {
638     \int_compare:nNnT {#2} > { 1 }
639     {
640         \bool_if:NTF \l__spintent_spunit_print_cdot_bool
641         {
642             \bool_if:NTF \l__spintent_spunit_read_cdot_bool
643             { \MathMLintent { _por } { \cdot } }
644             { \MathMLintent { :silent } { \cdot } }
645         }
646         {
647             \,
648             \bool_if:NT \l__spintent_spunit_read_cdot_bool
649             { \MathMLintent { _por } { \mathopen { \c__spintent_invisible_sep_int } } }
650         }
651     }

```

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

```

652     \__spintent_spunit_load_term:nn {#1} {#2}
653     \tl_if_empty:NTF \l__spintent_spunit_exponent_tl
654     {
655         \MathMLintent { \l__spintent_spunit_luaset_read_str }
656         {
657             \__spintent_spunit_format_canonical:
658         }
659     }
660     {
661         {
662             \MathMLintent { \l__spintent_spunit_luaset_read_str }
663             {
664                 \__spintent_spunit_format_canonical:
665             }
666         }
667         ^ { \l__spintent_spunit_exponent_tl }
668     }
669 }

```

(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.

```

670 \cs_new_protected:Nn \__spintent_spunit_print_product:n
671 {
672     \int_step_inline:nn { \str_use:c { \l__spintent_spunit_luaset_ #1 _n_str } }
673     { \__spintent_spunit_print_term:nn {#1} {##1} }
674 }

```

(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*”.

```

675 \cs_new_protected:Nn \__spintent_print_spunit_hierarchy:
676 {
677     \tl_if_empty:NF \l__spintent_luaset_arg_above_tl
678     {
679         \__spintent_spunit_print_product:n { above }
680         \tl_if_empty:NF \l__spintent_luaset_arg_below_tl
681         {

```

```

682         \MathMLintent { \l__spinttent_spunit_read_slash_str } { / }
683         \__spinttent_spunit_print_product:n { below }
684     }
685 }
686 }

```

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

`\__spinttent_spunit_exec:n`

The function `\__spinttent_spunit_exec:n` performs the main validation and layout for “units”. It first checks if the `{\langle argument \rangle}` 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 `\__spinttent_spunit_mathml:.`

```

687 \cs_new_protected:Nn \__spinttent_spunit_exec:n
688 {
689     \str_if_eq:VnTF \l__spinttent_luaset_has_units_str { false }
690     { \msg_error:nnn { spinttent } { spunit-no-units } { #1 } }
691     {
692         \str_if_eq:VnTF \l__spinttent_luaset_status_str { valid }
693         {
694             \str_if_eq:VnTF
695             \l__spinttent_luaset_denom_has_number_str { true }
696             {
697                 \msg_error:nnn { spinttent }
698                 { spunit-inline-numeric-denominator } { #1 }
699             }
700             {
701                 \__spinttent_luafun_spunit_parse_product:nV { above }
702                 \l__spinttent_luaset_arg_above_tl
703                 \tl_if_empty:NF \l__spinttent_luaset_arg_below_tl
704                 {
705                     \__spinttent_luafun_spunit_parse_product:nV { below }
706                     \l__spinttent_luaset_arg_below_tl
707                 }
708                 \bool_lazy_and:nnTF
709                 {
710                     \str_if_eq_p:ee { \l__spinttent_spunit_luaset_above_found_str }
711                     { true }
712                 }
713                 {
714                     \bool_lazy_or_p:nn
715                     { \tl_if_empty_p:N \l__spinttent_luaset_arg_below_tl }
716                     {
717                         \str_if_eq_p:ee { \l__spinttent_spunit_luaset_below_found_str }
718                         { true }
719                     }
720                 }
721                 {
722                     \__spinttent_spunit_mathml:
723                 }
724                 { \msg_error:nnn { spinttent } { spunit-invalid-unit } { #1 } }
725             }
726         }
727         { \msg_error:nnn { spinttent } { spunit-invalid-unit } { #1 } }
728     }
729 }

```

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

`\__spinttent_spunit_mathml:`

The function `\__spinttent_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.

```

730 \cs_new_protected:Nn \__spinttent_spunit_mathml:
731 {
732     \str_if_eq:VnTF \l__spinttent_spunit_luaset_above_sexa_str { true }

```

```

733 {
734   \__spintent_spunit_load_term:nn { above } { 1 }
735   \tl_if_empty:NF \__spintent_luaset_part_int_tl
736   {
737     \__spintent_sprnum_mathml:
738   }
739   \bool_lazy_and:nnT
740   { \str_if_eq_p:ee { \__spintent_luaset_part_int_tl } { 1 } }
741   { \tl_if_empty_p:N \__spintent_luaset_part_dec_tl }
742   {
743     \str_set:Nx \__spintent_spunit_luaset_read_str
744     {
745       \str_case:Vn \__spintent_spunit_luaset_canonical_str
746       {
747         { ° } { _grado:unit }
748         { ' } { _minuto:unit }
749         { " } { _segundo:unit }
750       }
751     }
752   }
753   \MathMLintent { \__spintent_spunit_luaset_read_str }
754   {
755     \mathord
756     {
757       \str_case:Vn \__spintent_spunit_luaset_canonical_str
758       {
759         { " } { \prime \! \prime }
760         { ' } { \prime }
761         { ° } { ° }
762       }
763     }
764   }
765 }
766 {
767   \tl_if_empty:NF \__spintent_luaset_part_int_tl
768   {
769     \__spintent_sprnum_mathml: \,
770   }
771   \__spintent_print_spunit_hierarchy:
772 }
773 }

```

(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.

```

774 \NewDocumentCommand \spunit { O{} m }
775 {
776   \mode_if_math:TF
777   {
778     \group_begin:
779     \keys_set:nn { spintent / spunit } {#1}
780     \__spintent_luafun_clean_split_and_set:n { \exp_not:n {#2} }
781     \__spintent_spunit_exec:n {#2}
782     \group_end:
783   }
784   { \msg_error:nnn { spintent } { need-math-mode } { \spunit } }
785 }

```

(End of definition for `\spunit`. This function is documented on page 9.)

15.16 The commands `\spnewunit` and `\spunitalias`

The user commands `\spnewunit` and `\spunitalias` allow users to add new custom “units” or “aliases” to the package’s dictionary. Similar to `\spunit`, these commands delegate the processing and validation to the `spintent.lua` via `\__spintent_luafun_define_custom_unit:nn` and `\__spintent_luafun_define_spunit_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 T_EX using the `\l__spintent_spunit_luaset_status_str` variable. If a collision is detected or the $\langle \textit{argument} \rangle$ is invalid, the package throws the corresponding “error” message.

15.16.1 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.

```

786 \msg_new:nnnn { spintent } { spnewunit-duplicate }
787 {
788   The~unit~symbol~or~alias~'#1'~is~already~defined~\msg_line_context:.
789 }
790 {
791   Existing~units~and~aliases~cannot~be~redefined~with~
792   '\token_to_str:N \spnewunit'.
793 }
794 \msg_new:nnnn { spintent } { spunitalias-duplicate }
795 {
796   The~alias~'#1'~is~already~defined~or~reserved~\msg_line_context:.
797 }
798 {
799   Existing~units~and~aliases~cannot~be~redefined.
800 }
801 \msg_new:nnnn { spintent } { spunitalias-invalid-expr }
802 {
803   Invalid~expression~'#2'~for~alias~'#1'~\msg_line_context:.
804 }
805 {
806   The~expression~must~contain~only~defined~units,~exponents~and~one~'/' .
807 }

```

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

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

```

808 \NewDocumentCommand \spnewunit { m m }
809 {
810   \mode_if_math:TF
811   {
812     \msg_error:nnn { spintent } { need-text-mode } { \spnewunit }
813   }
814   {
815     \__spintent_luafun_define_custom_unit:nn {#1} {#2}
816     \str_if_eq:VnT \l__spintent_spunit_luaset_status_str { duplicate }
817     {
818       \msg_error:nnn { spintent } { spnewunit-duplicate } {#1}
819     }
820   }
821 }

```

(End of definition for `\spnewunit`. This function is documented on page 10.)

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

```

822 \NewDocumentCommand \spunitalias { m m }
823 {
824   \mode_if_math:TF
825   {
826     \msg_error:nnn { spintent } { need-text-mode } { \spunitalias }
827   }
828   {
829     \__spintent_luafun_define_spunit_alias:nn {#1} {#2}
830     \str_if_eq:VnT \l__spintent_spunit_luaset_status_str { duplicate }

```

```

831         {
832             \msg_error:nnn { spintent } { spunitalias-duplicate } {#1}
833         }
834         \str_if_eq:VnT \l__spintent_spunit_luaset_status_str { invalid-expr }
835         {
836             \msg_error:nnnn { spintent } { spunitalias-invalid-expr } {#1} {#2}
837         }
838     }
839 }

```

(End of definition for `\spunitalias`. This function is documented on page 11.)

15.17 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 delegates `{\argument}` parsing to the Lua module `spintent.lua` via `\__spintent_luafun_clean_split_and_set:n`. The module then retrieves the correct “currency symbol”, pluralization rules, and layout structures from the database.

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

```

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

```

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

```

840 \str_new:N \l__spintent_spmoney_luaset_symbol_str
841 \str_new:N \l__spintent_spmoney_luaset_position_str
842 \str_new:N \l__spintent_spmoney_luaset_sing_str
843 \str_new:N \l__spintent_spmoney_luaset_plur_str
844 \str_new:N \l__spintent_spmoney_luaset_conde_str
845 \str_new:N \l__spintent_spmoney_luaset_subsing_str
846 \str_new:N \l__spintent_spmoney_luaset_subplur_str
847 \str_new:N \l__spintent_spmoney_luaset_print_iso_str
848 \str_new:N \l__spintent_spmoney_luaset_currency_arg_str
849 \str_new:N \l__spintent_spmoney_luaset_status_str

```

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

```

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

```

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

```

850 \bool_new:N \l__spintent_spmoney_sub_units_bool
851 \str_new:N \l__spintent_spmoney_intent_money_str
852 \str_new:N \l__spintent_spmoney_intent_sign_str
853 \str_new:N \l__spintent_spmoney_intent_sub_unit_str
854 \str_new:N \l__spintent_spmoney_res_main_voice_str
855 \int_new:N \l__spintent_spmoney_subnum_int
856 \str_new:N \l__spintent_spmoney_extracted_curr_str
857 \tl_new:N \l__spintent_spmoney_extracted_num_tl

```

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

15.17.1 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.

```

858 \msg_new:nnnn { spintent } { spmoney-no-subunits }
859 {
860     The~option~'sub-units=true'~is~ignored~for~the~currency~'~#1'~\msg_line_context:.
861 }
862 {
863     Currencies~like~'JPY'~or~'CLP'~have~no~subunits~in~the~Lua~module.
864 }
865 \msg_new:nnnn { spintent } { spmoney-invalid-currency }
866 {

```

```

867     Invalid~currency~'#1'~for~'\token_to_str:N \spmoney'~\msg_line_context:.
868   }
869   {
870     The~symbol,~alias~or~ISO~code~does~not~exist~in~the~Lua~module.
871   }

```

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

15.17.2 Definition of keys for `\spmoney`

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

```

currency
sub-units
money-math
cifras-sep
unknown
872 \keys_define:nn { spintent / spmoney }
873   {
874     currency .code:n = { \__spintent_spmoney_set_currency:n {#1} },
875     currency .value_required:n = true,
876     sub-units .bool_set:N = \__spintent_spmoney_sub_units_bool,
877     sub-units .initial:n = { false },
878     sub-units .value_required:n = true,
879     money-math .bool_set:N = \__spintent_spmoney_money_math_bool,
880     money-math .initial:n = { true },
881     money-math .value_required:n = true,
882     cifras-sep .meta:nn = { spintent / spnum } { cifras-sep = {#1} },
883     cifras-sep .value_required:n = true,
884     unknown .code:n =
885       {
886         \str_set:Nn \__spintent_command_name_str { spmoney }
887         \__spintent_unknown_keys_cmd:n {#1}
888       },
889   }

```

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

15.17.3 Internal functions for `\spmoney`

The function `\__spintent_spmoney_set_currency:n` validates the currency identifier provided in ‘#1’ against the database. It delegates the lookup to the function `\__spintent_spmoney_normalize_key:n`, which calls the Lua module `spintent.lua`. If the module returns a notfound status, an “error” is triggered. Otherwise, it stores the valid currency identifier in `\__spintent_spmoney_luaset_currency_arg_str` for subsequent processing.

```

890 \cs_new_protected:Nn \__spintent_spmoney_set_currency:n
891   {
892     \__spintent_spmoney_normalize_key:n {#1}
893     \str_if_eq:VnTF \__spintent_spmoney_luaset_status_str { notfound }
894       {
895         \msg_error:nnn { spintent } { spmoney-invalid-currency } {#1}
896       }
897     { \str_set:Nn \__spintent_spmoney_luaset_currency_arg_str {#1} }
898   }
899 \cs_new_protected:Nn \__spintent_spmoney_normalize_key:n
900   {
901     \__spintent_luafun_spmoney_normalize_key:n { \exp_not:n {#1} }
902   }
903 \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.

```

904 \keys_set:nn { spintent / spmoney } { currency = pesos }

```

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

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`.

```

905 \cs_new_protected:Nn \__spintent_spmoney_validate_money:
906   {
907     \str_if_eq:VnT \__spintent_luaset_has_units_str { true }
908     {
909       \__spintent_spmoney_normalize_key:V \__spintent_luaset_arg_above_tl

```

```

910     \str_if_eq:VnTF \l__spinttent_spmoney_luaset_status_str { notfound }
911     {
912         \msg_error:nne { spinttent } { spmoney-invalid-currency }
913         { \l__spinttent_luaset_arg_above_tl }
914     }
915     {
916         \str_set:NV
917         \l__spinttent_spmoney_luaset_currency_arg_str
918         \l__spinttent_luaset_arg_above_tl
919     }
920 }
921 \str_set:Ne \l__spinttent_spmoney_luaset_status_str { found }
922 \__spinttent_luafun_spmoney_lookup_data:V \l__spinttent_spmoney_luaset_currency_arg_str
923 }

```

(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.

```

924 \cs_new_protected:Nn \__spinttent_spmoney_print_symbol:
925 {
926     \str_if_eq:VnTF \l__spinttent_spmoney_luaset_print_iso_str { true }
927     { \mathrm { \l__spinttent_spmoney_luaset_currency_arg_str } }
928     {
929         \str_if_eq:VnTF \l__spinttent_spmoney_luaset_symbol_str { $ }
930         {
931             \bool_if:NTF \l__spinttent_spmoney_money_math_bool
932             { \$ } { \text { \$ } }
933         }
934         {
935             \bool_if:NTF \l__spinttent_spmoney_money_math_bool
936             { \l__spinttent_spmoney_luaset_symbol_str }
937             { \text { \l__spinttent_spmoney_luaset_symbol_str } }
938         }
939     }
940 }

```

(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*”.

```

941 \cs_new_protected:Nn \__spinttent_spmoney_render_layout:
942 {
943     \str_if_eq:VnTF \l__spinttent_spmoney_luaset_position_str { pre }
944     {
945         \__spinttent_spmoney_print_symbol:
946         \MathMLarg { n } { \__spinttent_spmoney_render_number_node: }
947     }
948     {
949         \MathMLarg { n } { \__spinttent_spmoney_render_number_node: }
950         \__spinttent_spmoney_print_symbol:
951     }
952 }

```

(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 full numeric value, encompassing both the integer and decimal components. Conversely, the function `\__spinttent_spmoney_render_only_integer_node:` outputs exclusively the integer portion, omitting any decimal data.

```

953 \cs_new_protected:Nn \__spinttent_spmoney_render_number_node:
954 {
955     \__spinttent_spmoney_render_int:
956     \__spinttent_spmoney_print_part_dec:
957 }
958 \cs_new_protected:Nn \__spinttent_spmoney_render_only_integer_node:

```

```

959 {
960   \__spintent_spmoney_render_int:
961 }

```

(End of definition for __spintent_spmoney_render_number_node: and __spintent_spmoney_render_only_integer_node:.)

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

```

962 \cs_new_protected:Nn \__spintent_spmoney_mathml:
963 {
964   \tl_if_empty:NTF \l__spintent_luaset_sign_tl
965   {
966     \bool_if:NTF \l__spintent_spmoney_sub_units_bool
967     { \__spintent_spmoney_set_and_print_sub_units: }
968     { \__spintent_spmoney_intent_int: }
969   }
970   {
971     \str_if_eq:VnTF \l__spintent_luaset_sign_tl { + }
972     {
973       \str_set:Nn \l__spintent_spmoney_intent_sign_str { _más:prefix ($n) }
974     }
975     {
976       \str_set:Nn \l__spintent_spmoney_intent_sign_str { _menos:prefix ($n) }
977     }
978     \__spintent_invisible_sep:
979     \MathMLintent { \l__spintent_spmoney_intent_sign_str }
980     {
981       \l__spintent_luaset_sign_tl
982       \MathMLarg { n }
983       {
984         \bool_if:NTF \l__spintent_spmoney_sub_units_bool
985         { \__spintent_spmoney_set_and_print_sub_units: }
986         { \__spintent_spmoney_intent_int: }
987       }
988     }
989   }
990 }

```

(End of definition for __spintent_spmoney_mathml:.)

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

```

991 \cs_new_protected:Nn \__spintent_spmoney_intent_int:
992 {
993   \tl_if_empty:NTF \l__spintent_luaset_dec_sep_tl
994   {
995     \str_if_eq:VnTF \l__spintent_luaset_only_part_int_str { 1 }
996     {
997       \str_set:NV \l__spintent_spmoney_intent_money_str
998       \l__spintent_spmoney_luaset_sing_str
999     }
1000     {
1001       \str_set:NV \l__spintent_spmoney_intent_money_str
1002       \l__spintent_spmoney_luaset_plur_str
1003     }
1004     \str_if_eq:VnT \l__spintent_luaset_millions_str { true }
1005     {
1006       \str_set:NV \l__spintent_spmoney_intent_money_str
1007       \l__spintent_spmoney_luaset_conde_str
1008     }
1009   }
1010   {
1011     \str_set:NV \l__spintent_spmoney_intent_money_str
1012     \l__spintent_spmoney_luaset_plur_str
1013     \str_if_eq:VnT \l__spintent_luaset_millions_str { true }
1014     {

```

```

1015         \str_if_eq:VnTF \l__spintent_luaset_dec_is_all_zeros_str { true }
1016     {
1017         \str_set:NV \l__spintent_spmoney_intent_money_str
1018             \l__spintent_spmoney_luaset_conde_str
1019     }
1020     {
1021         \str_set:NV \l__spintent_spmoney_intent_money_str
1022             \l__spintent_spmoney_luaset_plur_str
1023     }
1024 }
1025 }
1026 \__spintent_invisible_sep:
1027 \MathMLintent { _ \l__spintent_spmoney_intent_money_str :postfix($n) }
1028 {
1029     \__spintent_spmoney_render_layout:
1030 }
1031 }

```

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

```

1032 \cs_new_protected:Nn \__spintent_spmoney_check_sub_units:
1033 {
1034     \str_if_empty:NT \l__spintent_spmoney_luaset_subsing_str
1035     {
1036         \msg_warning:nne { spintent } { spmoney-no-subunits }
1037         { \l__spintent_spmoney_luaset_currency_arg_str }
1038         \bool_set_false:N \l__spintent_spmoney_sub_units_bool
1039     }
1040     \tl_if_empty:NT \l__spintent_luaset_dec_sep_tl
1041     { \bool_set_false:N \l__spintent_spmoney_sub_units_bool }
1042 }
1043 \cs_new_protected:Nn \__spintent_spmoney_set_and_print_sub_units:
1044 {
1045     \int_compare:nNnTF { \str_count:N \l__spintent_luaset_only_part_dec_str } = { 1 }
1046     {
1047         \int_set:Nn \l__spintent_spmoney_subnum_int
1048             { \l__spintent_luaset_only_part_dec_str * 10 }
1049         \str_set:Ne \l__spintent_luaset_only_part_dec_str
1050             { \l__spintent_luaset_only_part_dec_str 0 }
1051     }
1052     {
1053         \int_set:Nn \l__spintent_spmoney_subnum_int
1054             { \l__spintent_luaset_only_part_dec_str }
1055     }
1056     \int_compare:nNnTF { \l__spintent_spmoney_subnum_int } = { 1 }
1057     {
1058         \str_set:Ne \l__spintent_spmoney_intent_sub_unit_str
1059             { _ \l__spintent_spmoney_luaset_subsing_str :postfix($sub) }
1060     }
1061     {
1062         \str_set:Ne \l__spintent_spmoney_intent_sub_unit_str
1063             { _ \l__spintent_spmoney_luaset_subplur_str :postfix($sub) }
1064     }
1065     \str_if_eq:VnTF \l__spintent_luaset_only_part_int_str { 1 }
1066     {
1067         \str_set:NV \l__spintent_spmoney_res_main_voice_str
1068             \l__spintent_spmoney_luaset_sing_str
1069     }
1070     {
1071         \str_set:NV \l__spintent_spmoney_res_main_voice_str
1072             \l__spintent_spmoney_luaset_plur_str
1073     }

```

```

1074     \str_if_eq:VnT \l__spinttent_luaset_millions_str { true }
1075     {
1076         \str_set:NV \l__spinttent_spmoney_res_main_voice_str
1077         \l__spinttent_spmoney_luaset_conde_str
1078     }
1079     \__spinttent_spmoney_build_tree:
1080 }

```

(End of definition for `\__spinttent_spmoney_check_sub_units:` and `\__spinttent_spmoney_set_and_print_sub_units:`)

```

\__spinttent_spmoney_build_tree:
\__spinttent_spmoney_print_sep_con:
\__spinttent_spmoney_print_dec_sub_unit:

```

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

`\__spinttent_spmoney_print_sep_con:` also applies the specific speech intent ‘_con’ to the decimal separator, while `\__spinttent_spmoney_print_dec_sub_unit:` binds the computed “subunit” intent to the decimal digits.

```

1081 \cs_new_protected:Nn \__spinttent_spmoney_build_tree:
1082 {
1083     \str_if_eq:VnTF \l__spinttent_spmoney_luaset_position_str { pre }
1084     {
1085         \MathMLintent { _ \l__spinttent_spmoney_res_main_voice_str :postfix($n) }
1086         {
1087             \__spinttent_spmoney_print_symbol:
1088             \MathMLarg { n } { \__spinttent_spmoney_render_only_integer_node: }
1089         }
1090         \__spinttent_spmoney_print_sep_con:
1091         \__spinttent_spmoney_print_dec_sub_unit:
1092     }
1093     {
1094         \MathMLintent { _ \l__spinttent_spmoney_res_main_voice_str :postfix($n) }
1095         {
1096             \__spinttent_inner_sep:
1097             \MathMLarg { n } { \__spinttent_spmoney_render_only_integer_node: }
1098         }
1099         \__spinttent_spmoney_print_sep_con:
1100         \__spinttent_spmoney_print_dec_sub_unit:
1101         \MathMLintent { :silent } { \__spinttent_spmoney_print_symbol: }
1102     }
1103 }
1104 \cs_new_protected:Nn \__spinttent_spmoney_print_sep_con:
1105 {
1106     \MathMLintent { _con }
1107     {
1108         \mathord { \l__spinttent_luaset_dec_sep_tl }
1109     }
1110 }
1111 \cs_new_protected:Nn \__spinttent_spmoney_print_dec_sub_unit:
1112 {
1113     \MathMLintent { \l__spinttent_spmoney_intent_sub_unit_str }
1114     {
1115         \__spinttent_inner_sep:
1116         \MathMLarg { sub }
1117         {
1118             \__spinttent_spmoney_render_dec:
1119         }
1120     }
1121 }

```

(End of definition for `\__spinttent_spmoney_build_tree:`, `\__spinttent_spmoney_print_sep_con:`, and `\__spinttent_spmoney_print_dec_sub_unit:`)

```
\__spinttent_spmoney_parse_and_route:n
```

The internal function `\__spinttent_spmoney_parse_and_route:n` processes the $\{\langle argument \rangle\}$ passed in ‘#1’ using Lua module to extract the numeric value and any “currency” identifier. If a “currency” is found in `\l__spinttent_spmoney_extracted_curr_str`, it sets the corresponding variables. It then calls `\__spinttent_spmoney_validate_money:` to verify the “currency”. If “subunit” rendering is enabled in `\l__spinttent_spmoney_sub_units_bool`, it calls `\__spinttent_spmoney_check_sub_units:`. Finally, it calls `\__spinttent_spmoney_mathml:` to format the output.

```

1122 \cs_new_protected:Nn \__spinttent_spmoney_parse_and_route:n
1123 {

```

```

1124     \__spintent_luafun_spmoney_prepare_input:n { \exp_not:n {#1} }
1125     \__spintent_luafun_clean_split_and_set:n
1126     { \exp_not:V \__spintent_spmoney_extracted_num_tl }
1127     \str_if_empty:NF \__spintent_spmoney_extracted_curr_str
1128     {
1129         \str_set:Ne \__spintent_luaset_has_units_str { true }
1130         \str_set:NV \__spintent_luaset_arg_above_tl
1131         \__spintent_spmoney_extracted_curr_str
1132     }
1133     \__spintent_spmoney_validate_money:
1134     \bool_if:NT \__spintent_spmoney_sub_units_bool
1135     { \__spintent_spmoney_check_sub_units: }
1136     \__spintent_spmoney_mathml:
1137 }

```

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

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

```

1138 \NewDocumentCommand \spmoney { 0{} m }
1139 {
1140     \mode_if_math:TF
1141     {
1142         \group_begin:
1143         \keys_set:nn { spintent / spmoney } {#1}
1144         \__spintent_spmoney_parse_and_route:n {#2}
1145         \group_end:
1146     }
1147     { \msg_error:nnn { spintent } { need-math-mode } { \spmoney } }
1148 }

```

(End of definition for `\spmoney`. This function is documented on page 11.)

15.18 The command `\spshort`

The user command `\spshort` is the *fourth* and *final* of “*the big four*”. It manages the semantic injection and typographical formatting of *text mode* abbreviations, ordinals, and acronyms. Unlike *math mode* commands, which rely on **MathML** attributes, this command uses the PDF *tagging tools* provided by the `tagpdf`[33] package. Specifically, it uses the expanded key **E** to ensure that NVDA interpret the full form of the abbreviation, ordinal, or acronym in *text mode*.

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

```

1149 \str_new:N \__spintent_spshort_luaset_status_str
1150 \str_new:N \__spintent_spshort_luaset_layout_str
1151 \str_new:N \__spintent_spshort_luaset_expand_str
1152 \str_new:N \__spintent_spshort_luaset_output_str
1153 \str_new:N \__spintent_spshort_luaset_base_str
1154 \str_new:N \__spintent_spshort_luaset_suffix_str

```

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

15.18.1 Definition of error messages for `\spshort`

`spshort-unknown-abbreviation`

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

```

1155 \msg_new:nnnn { spintent } { spshort-unknown-abbreviation }
1156 {
1157     The~abbreviation~or~ordinal~'#1'~is~invalid~or~not~recognized~\msg_line_context:.
1158 }
1159 {
1160     Check~the~spelling~or~the~grammar~rules~for~ordinals.
1161 }

```

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

15.18.2 Definition of keys for `\spshort`

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

`read-arg`
`unknown`

```

1162 \keys_define:nn { spintenc / spshort }
1163 {
1164     small-caps .bool_set:N = \l__spintenc_spshort_smallcaps_bool,
1165     small-caps .initial:n = false,
1166     small-caps .value_required:n = true,
1167     read-arg .str_set:N = \l__spintenc_spshort_read_arg_str,
1168     read-arg .initial:n = {},
1169     read-arg .value_required:n = true,
1170     unknown .code:n =
1171     {
1172         \str_set:Nn \l__spintenc_command_name_str { spshort }
1173         \__spintenc_unknown_keys_cmd:n {#1}
1174     },
1175 }
```

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

15.18.3 Internal functions for `\spshort`

`\__spintenc_spshort_tag_span:nn`
`\__spintenc_spshort_tag_span:en`

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

```

1176 \cs_new_protected:Nn \__spintenc_spshort_tag_span:nn
1177 {
1178     \tag_mc_end_push:
1179     \tag_struct_begin:n { tag=Span, E={#1} }
1180     \tag_mc_begin:n { tag=Span }
1181     \tag_suspend:n { \spshort }
1182     #2
1183     \tag_resume:n { \spshort }
1184     \tag_mc_end:
1185     \tag_struct_end:
1186     \tag_mc_begin_pop:n {}
1187 }
1188 \cs_generate_variant:Nn \__spintenc_spshort_tag_span:nn { en }
```

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

`\__spintenc_spshort_print:nn`

The internal function `\__spintenc_spshort_print:nn` acts as the primary formatter for text abbreviations. It clears the string variable `\l__spintenc_spshort_read_arg_str` and evaluates the optional `(keys)` passed in ‘#1’. It then checks if a manual reading override was provided via those keys. If that string variable remains empty, it calls `\__spintenc_spshort_exec:n` to process the `{(argument)}` passed in ‘#2’. Otherwise, if an override is detected, it delegates the execution to `\__spintenc_spshort_render_bypass:n`.

```

1189 \cs_new_protected:Nn \__spintenc_spshort_print:nn
1190 {
1191     \str_clear:N \l__spintenc_spshort_read_arg_str
1192     \keys_set:nn { spintenc / spshort } {#1}
1193     \str_if_empty:NTF \l__spintenc_spshort_read_arg_str
1194     {
1195         \__spintenc_spshort_exec:n {#2}
1196     }
1197     {
1198         \__spintenc_spshort_render_bypass:n {#2}
1199     }
1200 }
```

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

`\__spintenc_spshort_exec:n`

The internal function `\__spintenc_spshort_exec:n` interfaces with the Lua module `spintenc.lua` function `\__spintenc_luafun_spshort_process:n` to resolve the abbreviation or ordinal `{(argument)}` passed in ‘#1’.

It subsequently evaluates the execution state stored within the string variable `\l__spintenc_spshort_luaset_status` and dynamically routes execution to the corresponding rendering function `success`, `fallback` or `error`. Any unrecognized status defaults to the “error” case.

```

1201 \cs_new_protected:Nn \__spintenc_spshort_exec:n
```

```

1202 {
1203   \__spintent_luafun_spshort_process:n {#1}
1204   \str_case:VnF \l__spintent_spshort_luaset_status_str
1205   {
1206     { success } { \__spintent_spshort_render_success: }
1207     { fallback } { \__spintent_spshort_render_fallback: }
1208     { error } { \__spintent_spshort_render_error:n {#1} }
1209   }
1210   { \__spintent_spshort_render_error:n {#1} }
1211 }

```

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

Conversely, the function `\__spintent_spshort_render_bypass:n` maps the user-defined “speech” to the $\langle argument \rangle$, bypassing the automated lookup Lua module. Finally, the function `\__spintent_spshort_render_error:n` triggers a structural exception while yielding the unformatted $\langle argument \rangle$ to prevent fatal compilation.

```

1212 \cs_new_protected:Nn \__spintent_spshort_render_success:
1213 {
1214   \__spintent_spshort_tag_span:en
1215   { \l__spintent_spshort_luaset_expand_str }
1216   { \__spintent_spshort_execute_layout: }
1217 }
1218 \cs_new_protected:Nn \__spintent_spshort_render_fallback:
1219 {
1220   \__spintent_spshort_tag_span:en
1221   { \l__spintent_spshort_luaset_expand_str }
1222   {
1223     \bool_if:NTF \l__spintent_spshort_smallcaps_bool
1224     { \textsc{ \l__spintent_spshort_luaset_output_str } }
1225     { \l__spintent_spshort_luaset_output_str }
1226   }
1227 }
1228 \cs_new_protected:Nn \__spintent_spshort_render_bypass:n
1229 {
1230   \__spintent_spshort_tag_span:en { \l__spintent_spshort_read_arg_str } {#1}
1231 }
1232 \cs_new_protected:Nn \__spintent_spshort_render_error:n
1233 {
1234   \msg_error:nnn { spintent } { spshort-unknown-abbreviation } {#1}
1235 }

```

(End of definition for `\__spintent_spshort_render_success:` and others.)

```
\__spintent_spshort_execute_layout:
```

The internal function `\__spintent_spshort_execute_layout:` executes the typographical layout. It evaluates the descriptor “stored” within the string variable `\l__spintent_spshort_luaset_layout_str`, which holds a value such as `linear_regular`, `superscript`, or `small_caps_pure`, and maps it to the corresponding $\TeX$ formatting. It also manages complex nested configurations, dynamically applying combinations such as superscripts integrated with conditional `SMALL CAPS` based on the active boolean variables.

```

1236 \cs_new_protected:Nn \__spintent_spshort_execute_layout:
1237 {
1238   \str_case:VnF \l__spintent_spshort_luaset_layout_str
1239   {
1240     { linear_regular }
1241     {
1242       \bool_if:NTF \l__spintent_spshort_smallcaps_bool
1243       { \textsc{ \l__spintent_spshort_luaset_output_str } }
1244       { \l__spintent_spshort_luaset_output_str }
1245     }
1246     { linear_caps }
1247     { \l__spintent_spshort_luaset_output_str }
1248     { linear_mixed }
1249     { \l__spintent_spshort_luaset_output_str }
1250     { superscript }
1251     {
1252       \l__spintent_spshort_luaset_base_str

```

```

1253     \bool_if:NTF \l__spintent_spshort_smallcaps_bool
1254     {
1255         \textsuperscript{
1256             \textsc{
1257                 \str_lowercase:V \l__spintent_spshort_luaset_suffix_str
1258             }
1259         }
1260     }
1261     {
1262         \textsuperscript{ \l__spintent_spshort_luaset_suffix_str }
1263     }
1264 }
1265 { small_caps_pure }
1266 {
1267     \bool_if:NTF \l__spintent_spshort_smallcaps_bool
1268     {
1269         \textsc{
1270             \str_lowercase:V \l__spintent_spshort_luaset_output_str
1271         }
1272     }
1273     { \l__spintent_spshort_luaset_output_str }
1274 }
1275 { acronym_long }
1276 { \l__spintent_spshort_luaset_output_str }
1277 }
1278 { \l__spintent_spshort_luaset_output_str }
1279 }

```

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

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

```

1280 \NewDocumentCommand \spshort { 0{} m }
1281 {
1282     \mode_if_math:TF
1283     {
1284         \msg_error:nnn { spintent } { need-text-mode } { \spshort }
1285     }
1286     {
1287         \mode_leave_vertical:
1288         \group_begin:
1289             \l__spintent_spshort_print:nn {#1} {#2}
1290         \group_end:
1291     }
1292 }

```

(End of definition for `\spshort`. This function is documented on page 13.)

15.19 The command `\spsiglo`

The public document command `\spsiglo` provides a semantic interface for typesetting century designations such as siglo XXI. It accepts both Arabic and Roman numerals as $\langle \textit{argument} \rangle$, interfacing with the Lua module `spintent.lua` to validate, parse, and normalize chronological values.

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

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

```

1293 \str_new:N \l__spintent_spsiglo_luaset_error_str
1294 \str_new:N \l__spintent_spsiglo_luaset_arabic_str
1295 \str_new:N \l__spintent_spsiglo_luaset_roman_min_str
1296 \str_new:N \l__spintent_spsiglo_print_era_str

```

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

15.19.1 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.

```

1297 \msg_new:nnnn { spintent } { spsiglo-invalid-arg }
1298 {
1299   Invalid~argument~'#2'~for~'\tl_trim_spaces:n {#1}'~\msg_line_context:.
1300 }
1301 {
1302   The~argument~must~be~an~Arabic~or~Roman~numeral~between~'1'~and~'4000'.
1303 }

```

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

15.19.2 Definition of keys for \spsiglo

`print-era` We define the $\langle key \rangle$ `print-era`.

`unknown`

```

1304 \keys_define:nn { spintent / spsiglo }
1305 {
1306   print-era .choices:nn =
1307     { none , ac , dc }
1308   {
1309     \int_case:nn { \l_keys_choice_int }
1310     {
1311       {1} { \str_set:Nn \l__spintent_spsiglo_print_era_str { none } }
1312       {2} { \str_set:Nn \l__spintent_spsiglo_print_era_str { ac } }
1313       {3} { \str_set:Nn \l__spintent_spsiglo_print_era_str { dc } }
1314     }
1315   },
1316   print-era .initial:n = none,
1317   print-era .value_required:n = true,
1318   print-era / unknown .code:n =
1319     \msg_error:nneee { spintent } { unknown-choice }
1320     { print-era } { none, ~ac, ~dc } { \exp_not:n {#1} },
1321   unknown .code:n =
1322     {
1323       \str_set:Nn \l__spintent_command_name_str { spsiglo }
1324       \__spintent_unknown_keys_cmd:n {#1}
1325     },
1326 }

```

(End of definition for `print-era` and `unknown`.)

15.19.3 Internal functions for \spsiglo

`\__spintent_spsiglo_render_simple:nn`

`\__spintent_spsiglo_render_simple:VV`

`\__spintent_spsiglo_render_with_era:nnn`

`\__spintent_spsiglo_render_with_era:VVV`

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

```

1327 \cs_new_protected:Nn \__spintent_spsiglo_render_simple:nn
1328 {
1329   \MathMLintent { #1:number } { \text { \textsc {#2} } }
1330 }
1331 \cs_generate_variant:Nn \__spintent_spsiglo_render_simple:nn { VV }
1332 \cs_new_protected:Nn \__spintent_spsiglo_render_with_era:nnn
1333 {
1334   \str_if_eq:nnTF {#1} { ac }
1335   {
1336     \MathMLintent { _antes-de-cristo:postfix($z) }
1337     {
1338       \__spintent_inner_sep:
1339       \MathMLarg { z }
1340       {
1341         \MathMLintent { #2:number }
1342         {
1343           \text { \textsc {#3}~a.~C. }
1344         }
1345       }
1346     }
1347 }

```

```

1348     {
1349         \MathMLintent { _después-de-cristo:postfix($z) }
1350     {
1351         \__spintent_inner_sep:
1352         \MathMLarg { z }
1353         {
1354             \MathMLintent { #2:number }
1355             {
1356                 \text { \textsc {#3}~d.~C. }
1357             }
1358         }
1359     }
1360 }
1361 }
1362 \cs_generate_variant:Nn \__spintent_spsiglo_render_with_era:nnn { VVV }

```

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

__spintent_spsiglo_tag_span:nn
__spintent_spsiglo_tag_span:en

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

```

1363 \cs_new_protected:Nn \__spintent_spsiglo_tag_span:nn
1364 {
1365     \tag_mc_end_push:
1366     \tag_struct_begin:n { tag=Span, alt={#1} }
1367     \tag_mc_begin:n { tag=Span }
1368     \tag_suspend:n { \spsiglo }
1369     #2
1370     \tag_resume:n { \spsiglo }
1371     \tag_mc_end:
1372     \tag_struct_end:
1373     \tag_mc_begin_pop:n {}
1374 }
1375 \cs_generate_variant:Nn \__spintent_spsiglo_tag_span:nn { en }

```

(End of definition for __spintent_spsiglo_tag_span:nn.)

__spintent_spsiglo_exec:n
__spintent_spsiglo_mathml:
__spintent_spsiglo_text:

The function `\__spintent_spsiglo_exec:n` formats century designations based on the active mode. It first checks the “error” status in the variable `\__spintent_spsiglo_luaset_error_str` returned by the Lua module `spintent.lua` and issues an “error” if validation fails.

If the `{⟨argument⟩}` is valid, in *math mode* it calls `\__spintent_spsiglo_mathml:`, which applies the `MathML` structures depending on whether an era modifier is present. In *text mode* it leaves vertical mode first and calls `\__spintent_spsiglo_text:`, which utilizes the `tag` and `alt` keys to expand the value of the `print-era` option into full text such as “antes de cristo” for NVDA/MathCAT.

```

1376 \cs_new_protected:Nn \__spintent_spsiglo_exec:n
1377 {
1378     \mode_if_math:F { \mode_leave_vertical: }
1379     \str_if_eq:VnTF \__spintent_spsiglo_luaset_error_str { true }
1380     {
1381         \msg_error:nnnn { spintent } { spsiglo-invalid-arg }
1382         { \spsiglo } { #1 }
1383     }
1384     {
1385         \mode_if_math:TF
1386         { \__spintent_spsiglo_mathml: }
1387         { \__spintent_spsiglo_text: }
1388     }
1389 }
1390 \cs_new_protected:Nn \__spintent_spsiglo_mathml:
1391 {
1392     \str_if_eq:VnTF \__spintent_spsiglo_print_era_str { none }
1393     {
1394         \__spintent_spsiglo_render_simple:VV
1395         \__spintent_spsiglo_luaset_arabic_str
1396         \__spintent_spsiglo_luaset_roman_min_str
1397     }
1398     {
1399         \__spintent_spsiglo_render_with_era:VVV

```

```

1400         \l__spinttent_spsiglo_print_era_str
1401         \l__spinttent_spsiglo_luaset_arabic_str
1402         \l__spinttent_spsiglo_luaset_roman_min_str
1403     }
1404 }
1405 \cs_new_protected:Nn \__spinttent_spsiglo_text:
1406 {
1407     \str_case:Vn \l__spinttent_spsiglo_print_era_str
1408     {
1409         { none }
1410         {
1411             \__spinttent_spsiglo_tag_span:en
1412             { \l__spinttent_spsiglo_luaset_arabic_str }
1413             { \textsc { \l__spinttent_spsiglo_luaset_roman_min_str } }
1414         }
1415         { ac }
1416         {
1417             \__spinttent_spsiglo_tag_span:en
1418             { \l__spinttent_spsiglo_luaset_arabic_str ~ antes ~ de ~ cristo }
1419             { \textsc { \l__spinttent_spsiglo_luaset_roman_min_str } ~ a.~C. }
1420         }
1421         { dc }
1422         {
1423             \__spinttent_spsiglo_tag_span:en
1424             { \l__spinttent_spsiglo_luaset_arabic_str ~ después ~ de ~ cristo }
1425             { \textsc { \l__spinttent_spsiglo_luaset_roman_min_str } ~ d.~C. }
1426         }
1427     }
1428 }

```

(End of definition for `\__spinttent_spsiglo_exec:n`, `\__spinttent_spsiglo_mathml:`, and `\__spinttent_spsiglo_text:`.)

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

```

1429 \NewDocumentCommand \spsiglo { 0{} m }
1430 {
1431     \group_begin:
1432     \keys_set:nn { spintent / spsiglo } {#1}
1433     \__spinttent_luafun_spsiglo_parse:n {#2}
1434     \__spinttent_spsiglo_exec:n {#2}
1435     \group_end:
1436 }

```

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

15.20 The command `\sptime`

The command `\sptime` is the main public interface for typesetting and tagging time expressions. It parses time string formats, such as HH:MM or HH:MM:SS, using the Lua function `\__spinttent_luafun_sptime_parse:n`. This separates the hours, minutes, seconds, and fractions to construct a **MathML** *intent* tree for NVDA/MathCAT.

These internal string variables store the values returned from the Lua module `spintent.lua`. They hold the extracted time units, error flags, and boolean indicators, such as the presence of seconds or decimal fractions, used during formatting.

```

1437 \str_new:N \l__spinttent_sptime_luaset_seconds_str
1438 \str_new:N \l__spinttent_sptime_luaset_has_seconds_str
1439 \str_new:N \l__spinttent_sptime_luaset_error_str
1440 \str_new:N \l__spinttent_sptime_luaset_is_fraction_str
1441 \str_new:N \l__spinttent_sptime_luaset_base_str

```

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

15.20.1 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.

```

1442 \msg_new:nnnn { spintent } { sptime-invalid-arg }

```

```

1443 {
1444   Invalid~argument~'#2'~for~'\tl_trim_spaces:n {#1}'~\msg_line_context:.
1445 }
1446 {
1447   Allowed~formats:~'HH:MM',~'HH:MM:SS'~or~'HH:MM:SS.ss'.~
1448   Hours~must~be~'0-23'~and~minutes/seconds~'0-59'.
1449 }

```

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

15.20.2 Internal functions for `\sptime`

`\__spintent_sptime_build_seconds:nn`

The internal function `\__spintent_sptime_build_seconds:nn` builds the **MathML** structure for the seconds component of a time expression. It wraps the numeric value ‘#2’ in a speech intent, applying a connector prefix, passed in ‘#1’, and the “segundos” postfix so it is read correctly by NVDA/MathCAT.

```

1450 \cs_new_protected:Nn \__spintent_sptime_build_seconds:nn
1451 {
1452   \MathMLintent { #1:prefix($z) }
1453   {
1454     \text { : }
1455     \MathMLarg { z }
1456     {
1457       \MathMLintent { _segundos:postfix($m) }
1458       {
1459         \__spintent_inner_sep:
1460         \MathMLarg { m }
1461         {
1462           \text { #2 }
1463         }
1464       }
1465     }
1466   }
1467 }

```

(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.

```

1468 \cs_new_protected:Nn \__spintent_sptime_mathml:
1469 {
1470   \str_if_eq:VnTF \__spintent_sptime_luaset_has_seconds_str { true }
1471   {
1472     \MathMLintent { :time }
1473     {
1474       \text { \__spintent_sptime_luaset_base_str }
1475     }
1476     \str_if_eq:VnTF \__spintent_sptime_luaset_is_fraction_str { true }
1477     {
1478       \__spintent_sptime_build_seconds:nn { con }
1479       { \__spintent_sptime_luaset_seconds_str }
1480     }
1481     {
1482       \__spintent_sptime_build_seconds:nn { _minutos-con }
1483       { \__spintent_sptime_luaset_seconds_str }
1484     }
1485   }
1486   {
1487     \MathMLintent { :time }
1488     {
1489       \text { \__spintent_sptime_luaset_base_str }
1490     }
1491     \str_if_eq:VnTF \__spintent_sptime_luaset_is_fraction_str { false }
1492     {
1493       \MathMLintent { _minutos }
1494       {
1495         \c__spintent_invisible_sep_int

```

```

1496         }
1497     }
1498 }
1499 }

```

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

`\sptime` The command `\sptime` is the public interface for formatting time expressions. It first checks if it is being used in math mode, issuing an error if not. Inside a local group, it parses the input using `\__spintent_luafun_sptime_parse:n` and checks the error status in `\l__spintent_sptime_luaset_error_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.

```

1500 \NewDocumentCommand \sptime { m }
1501 {
1502     \mode_if_math:TF
1503     {
1504         \group_begin:
1505         \__spintent_luafun_sptime_parse:n {#1}
1506         \str_if_eq:VnTF \l__spintent_sptime_luaset_error_str { true }
1507         {
1508             \msg_error:nnnn { spintent } { sptime-invalid-arg }
1509             { \sptime } {#1}
1510         }
1511         {
1512             \__spintent_sptime_mathml:
1513         }
1514     } \group_end:
1515 }
1516 { \msg_error:nnn { spintent } { need-math-mode } { \sptime } }
1517 }

```

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

15.21 The command `\spdate`

The command `\spdate` is the main public command for typesetting and tagging date expressions. It parses date string formats, such as DD/MM/YYYY or YYYY-MM-DD, using the Lua function `\__spintent_luafun_spdate_parse:n`. This verifies the input and constructs a **MathML** *intent* tree for NVDA/MathCAT.

```

\l__spintent_spdate_luaset_output_str
\l__spintent_spdate_luaset_error_str

```

These internal string variables store the values returned from the Lua module `spintent.lua`. They hold the formatted date string and the error status used during formatting.

```

1518 \str_new:N \l__spintent_spdate_luaset_output_str
1519 \str_new:N \l__spintent_spdate_luaset_error_str

```

(End of definition for `\l__spintent_spdate_luaset_output_str` and `\l__spintent_spdate_luaset_error_str`.)

15.21.1 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`.

```

1520 \msg_new:nnnn { spintent } { spdate-invalid-arg }
1521 {
1522     Invalid~argument~'#2'~for~'\tl_trim_spaces:n {#1}'~\msg_line_context:.
1523 }
1524 {
1525     Use~'DD/MM/YYYY',~'DD-MM-YYYY'~or~'YYYY-MM-DD',~with~a~valid~day~
1526     and~month.
1527 }

```

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

15.21.2 Internal functions for `\spdate`

```
\__spintent_spdate_exec:n
```

The internal function `\__spintent_spdate_exec:n` parses the input using `\__spintent_luafun_spdate_parse:n` and checks the error status in `\l__spintent_spdate_luaset_error_str`. If an invalid format is detected, it issues the `spdate-invalid-arg` error. Otherwise, it calls `\__spintent_spdate_mathml:`.

```

1528 \cs_new_protected:Nn \__spintent_spdate_exec:n
1529 {
1530     \__spintent_luafun_spdate_parse:n {#1}

```

```

1531 \str_if_eq:VnTF \l__spinttent_spdate_luaset_error_str { true }
1532 {
1533   \msg_error:nnnn { spinttent } { spdate-invalid-arg }
1534   { \spdate } {#1}
1535 }
1536 { \__spinttent_spdate_mathml: }
1537 }

```

(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`.

```

1538 \cs_new_protected:Nn \__spinttent_spdate_mathml:
1539 {
1540   \MathMLintent { :date }
1541   {
1542     \text { \l__spinttent_spdate_luaset_output_str }
1543   }
1544 }

```

(End of definition for `\__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 `\__spinttent_spdate_exec:n` to process the input.

```

1545 \NewDocumentCommand \spdate { m }
1546 {
1547   \mode_if_math:TF
1548   {
1549     \group_begin:
1550     \__spinttent_spdate_exec:n {#1}
1551     \group_end:
1552   }
1553   { \msg_error:nnn { spinttent } { need-math-mode } { \spdate } }
1554 }

```

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

15.22 Semantic Operators

This section provides semantic operators. Unlike their formatting counterparts, these commands do not process mandatory arguments. Instead, they only inject a mathematical glyph wrapped in its corresponding regional intent. This grants teachers absolute control when building complex algebraic expressions, mixed numbers, or fraction divisions.

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.

`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.

```

1555 \msg_new:nnnn { spinttent } { spusep-empty-arg }
1556 {
1557   Empty~argument~for~'\token_to_str:N \spusep'\~\msg_line_context:.
1558 }
1559 {
1560   A~non-empty~argument~is~required.
1561 }
1562 \msg_new:nnnn { spinttent } { spusep-invalid-arg }
1563 {
1564   Invalid~argument~'#1'~for~'\token_to_str:N \spusep'\~\msg_line_context:.
1565 }
1566 {
1567   Only~the~delimiters~listed~in~the~documentation~of~'\token_to_str:N \spusep'~are~
1568   accepted.
1569 }

```

(End of definition for `spusep-empty-arg` and `spusep-invalid-arg`.)

`\l__spintent_spusep_arg_tl`
`\l__spintent_spusep_left_tl`
`\l__spintent_spusep_right_tl`

The variable `\l__spintent_spusep_arg_tl` stores the $\langle argument \rangle$, the variable `\l__spintent_spusep_left_tl` and `\l__spintent_spusep_right_tl` stores the parentheses.

```
1570 \tl_new:N \l__spintent_spusep_arg_tl
1571 \tl_new:N \l__spintent_spusep_left_tl
1572 \tl_new:N \l__spintent_spusep_right_tl
```

(End of definition for `\l__spintent_spusep_arg_tl`, `\l__spintent_spusep_left_tl`, and `\l__spintent_spusep_right_tl`.)

`size`
`silent`
`unknown`

We define the $\langle keys \rangle$ `size` and `silent`.

```
1573 \keys_define:nn { spintent / spusep }
1574 {
1575   size .choices:nn =
1576     { none , big , Big, bigg, Bigg }
1577     {
1578       \int_case:nn { \l_keys_choice_int }
1579       {
1580         { 1 }
1581         {
1582           \tl_clear:N \l__spintent_spusep_left_tl
1583           \tl_clear:N \l__spintent_spusep_right_tl
1584         }
1585         { 2 }
1586         {
1587           \tl_set:Nn \l__spintent_spusep_left_tl { \bigl }
1588           \tl_set:Nn \l__spintent_spusep_right_tl { \bigr }
1589         }
1590         { 3 }
1591         {
1592           \tl_set:Nn \l__spintent_spusep_left_tl { \Bigl }
1593           \tl_set:Nn \l__spintent_spusep_right_tl { \Bigr }
1594         }
1595         { 4 }
1596         {
1597           \tl_set:Nn \l__spintent_spusep_left_tl { \biggl }
1598           \tl_set:Nn \l__spintent_spusep_right_tl { \biggr }
1599         }
1600         { 5 }
1601         {
1602           \tl_set:Nn \l__spintent_spusep_left_tl { \Biggl }
1603           \tl_set:Nn \l__spintent_spusep_right_tl { \Biggr }
1604         }
1605       }
1606     },
1607   size .initial:n = none,
1608   size .value_required:n = true,
1609   size / unknown .code:n =
1610     \msg_error:nneee { spintent } { unknown-choice }
1611     { size } { none,~big,~Big,~bigg,~Bigg } { \exp_not:n {#1} },
1612   silent .bool_set:N = \l__spintent_spusep_silent_bool,
1613   silent .initial:n = false,
1614   silent .value_required:n = true,
1615   unknown .code:n =
1616     {
1617       \str_set:Nn \l__spintent_command_name_str { spusep }
1618       \__spintent_unknown_keys_cmd:n {#1}
1619     },
1620 }
```

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

`\__spintent_spusep_mathml:nn`

The function `\__spintent_spusep_mathml:nn` will place the argument passed in ‘#2’ to the right of the variables `\l__spintent_spusep_left_tl` and `\l__spintent_spusep_right_tl` and then check the state of the variable `\l__spintent_spusep_silent_bool` handled by the key `silent` to print.

```
1621 \cs_new_protected:Nn \__spintent_spusep_mathml:nn
1622 {
1623   \tl_put_right:cn { \l__spintent_spusep_ #1 _tl } {#2}
1624   \bool_if:NTF \l__spintent_spusep_silent_bool
1625   {
1626     \MathMLintent { :silent } { \tl_use:c { \l__spintent_spusep_ #1 _tl } }
```

```

1627     }
1628     {
1629         \tl_use:c { l__spintent_spusep_ #1 _tl }
1630     }
1631 }

```

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

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

```

1632 \cs_new_protected:Nn \__spintent_spusep_exec:n
1633 {
1634     \tl_set:Nc \l__spintent_spusep_arg_tl { \tl_trim_spaces:n {#1} }
1635     \tl_if_empty:NT \l__spintent_spusep_arg_tl
1636     {
1637         \msg_error:nn { spintent } { spusep-empty-arg }
1638     }
1639     \str_case:VnF \l__spintent_spusep_arg_tl
1640     {
1641         { ( }
1642         {
1643             \__spintent_spusep_mathml:nn { left } { \lparen }
1644         }
1645         { ) }
1646         {
1647             \__spintent_spusep_mathml:nn { right } { \rparen }
1648         }
1649         { [ ]
1650         {
1651             \__spintent_spusep_mathml:nn { left } { \lbrack }
1652         }
1653         { ] }
1654         {
1655             \__spintent_spusep_mathml:nn { right } { \rbrack }
1656         }
1657         { \{ }
1658         {
1659             \__spintent_spusep_mathml:nn { left } { \lbrace }
1660         }
1661         { \} }
1662         {
1663             \__spintent_spusep_mathml:nn { right } { \rbrace }
1664         }
1665     }
1666     {
1667         \tl_if_empty:NF \l__spintent_spusep_arg_tl
1668         { \msg_error:nnn { spintent } { spusep-invalid-arg } {#1} }
1669     }
1670 }

```

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

`\spusep` The command `\spusep` checks that it is running in math mode, opens a local group, sets the keys passed in ‘#1’ and processes the $\langle argument \rangle$ passed in ‘#2’ by means of the function `\__spintent_spusep_exec:n`.

```

1671 \NewDocumentCommand \spusep { O{} m }
1672 {
1673     \mode_if_math:TF
1674     {
1675         \group_begin:
1676         \keys_set:nn { spintent / spusep } {#1}
1677         \__spintent_spusep_exec:n {#2}
1678         \group_end:
1679     }
1680     { \msg_error:nnn { spintent } { need-math-mode } { \spusep } }
1681 }

```

(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.

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`.

```

1682 \msg_new:nnnn { spintent } { spnewread-empty-args }
1683 {
1684   Empty~argument~for~'\token_to_str:N \spnewread'\msg_line_context:.
1685 }
1686 {
1687   Two~non-empty~arguments~are~required.
1688 }
1689 \msg_new:nnnn { spintent } { spnewread-duplicate }
1690 {
1691   The~symbol~'#1'~is~already~defined~\msg_line_context:.
1692 }
1693 {
1694   Existing~symbols~cannot~be~redefined~with~
1695   '\token_to_str:N \spnewread'.
1696 }
1697 \msg_new:nnnn { spintent } { spread-empty-sym }
1698 {
1699   Empty~argument~for~'\token_to_str:N \spread'\msg_line_context:.
1700 }
1701 {
1702   A~non-empty~symbol~is~required.
1703 }
1704 \msg_new:nnnn { spintent } { spread-unknown-sym }
1705 {
1706   The~symbol~'#1'~is~not~registered~and~no~'read-arg'~was~given~\msg_line_context:.
1707 }
1708 {
1709   Register~the~symbol~first~with~'\token_to_str:N \spnewread'~or~pass~'read-arg'.
1710 }

```

(End of definition for `spnewread-empty-args` and others.)

\l_spintent_spread_luaset_status_str
\l_spintent_spread_luaset_reading_str
\l_spintent_spread_read_arg_tl
\l_spintent_spread_read_tmp_tl

The variables `\l_spintent_spread_luaset_status_str` and `\l_spintent_spread_luaset_reading_str` are set by `\__spintent_luafun_spread_new:nn/\__spintent_luafun_spread_lookup:n`. The variable `\l_spintent_spread_read_arg_tl` stores `read-arg`.

```

1711 \str_new:N \l_spintent_spread_luaset_status_str
1712 \str_new:N \l_spintent_spread_luaset_reading_str
1713 \tl_new:N \l_spintent_spread_read_arg_tl
1714 \tl_new:N \l_spintent_spread_read_tmp_tl

```

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

read-arg
silent
unknown

We define the `(keys)` `read-arg` and `silent`.

```

1715 \keys_define:nn { spintent / spread }
1716 {
1717   read-arg .code:n =
1718   {
1719     \__spintent_sanitizé_affix:Nn \l_spintent_spread_read_arg_tl {#1}
1720   },
1721   read-arg .initial:n = ,
1722   read-arg .value_required:n = true,
1723   silent .bool_set:N = \l_spintent_spread_silent_bool,
1724   silent .initial:n = false,
1725   silent .value_required:n = true,
1726   unknown .code:n =
1727   {
1728     \str_set:Nn \l_spintent_command_name_str { spread }
1729     \__spintent_unknown_keys_cmd:n {#1}
1730   },
1731 }

```

(End of definition for `read-arg`, `silent`, and `unknown`.)

`\__spintent_spread_exec:nn`

The function `\__spintent_spread_exec:nn` receives the optional $\langle keys \rangle$ in ‘#1’ and the $\langle symbol \rangle$ 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.

```

1732 \cs_new_protected:Nn \__spintent_spread_exec:nn
1733 {
1734   \keys_set:nn { spintent / spread } {#1}
1735   \tl_if_blank:nTF {#2}
1736   { \msg_error:nn { spintent } { spread-empty-sym } }
1737   {
1738     \bool_if:NTF \__spintent_spread_silent_bool
1739     { \__spintent_spread_mathml:nn {#2} { - } }
1740     {
1741       \tl_if_empty:NTF \__spintent_spread_read_arg_tl
1742       {
1743         \__spintent_luafun_spread_lookup:n { \exp_not:n {#2} }
1744         \str_if_eq:VnTF \__spintent_spread_luaset_status_str { found }
1745         {
1746           \__spintent_spread_mathml:nV {#2}
1747           \__spintent_spread_luaset_reading_str
1748         }
1749         {
1750           \msg_error:nne { spintent } { spread-unknown-sym }
1751           { \exp_not:n {#2} }
1752         }
1753       }
1754       { \__spintent_spread_mathml:nV {#2} \__spintent_spread_read_arg_tl }
1755     }
1756   }
1757 }

```

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

`\__spintent_spread_mathml:nn`

`\__spintent_spread_mathml:nV`

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 ‘_’.

```

1758 \cs_new_protected:Nn \__spintent_spread_mathml:nn
1759 {
1760   \__spintent_invisible_sep:
1761   \MathMLintent { _#2 } { \__spintent_inner_sep: #1 }
1762 }
1763 \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`.

```

1764 \NewDocumentCommand \spread { O{} m }
1765 {
1766   \mode_if_math:TF
1767   {
1768     \group_begin:
1769     \__spintent_spread_exec:nn {#1} {#2}
1770     \group_end:
1771   }
1772   { \msg_error:nnn { spintent } { need-math-mode } { \spread } }
1773 }

```

(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.

```

1774 \NewDocumentCommand \spnewread { m m }
1775 {
1776   \mode_if_math:TF
1777   {
1778     \msg_error:nnn { spintent } { need-text-mode } { \spnewread }
1779   }
1780   {
1781     \__spintent_sanitizet_affix:Nn \__spintent_spread_read_tmp_tl {#2}
1782     \bool_lazy_or:nnTF
1783     { \tl_if_blank_p:n {#1} } { \tl_if_empty_p:N \__spintent_spread_read_tmp_tl }
1784     { \msg_error:nn { spintent } { spnewread-empty-args } }
1785     {
1786       \__spintent_luafun_spread_new:nn { \exp_not:n {#1} }
1787       { \exp_not:V \__spintent_spread_read_tmp_tl }
1788       \str_if_eq:VnT \__spintent_spread_luaet_status_str { duplicate }
1789       {
1790         \msg_error:nnn { spintent } { spnewread-duplicate } {#1}
1791       }
1792     }
1793   }
1794 }

```

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

15.25 The command `\spdsym`

```

\__spintent_spdsym_read_sym_tl
\__spintent_spdsym_symbol_tl
\__spintent_spdsym_prefix_tl
\__spintent_spdsym_suffix_tl

```

Variables to store the regional reading string, the visual operator, and the optional semantic affixes specifically for `\spdsym`.

```

1795 \tl_new:N \__spintent_spdsym_read_sym_tl
1796 \tl_new:N \__spintent_spdsym_symbol_tl
1797 \tl_new:N \__spintent_spdsym_prefix_tl
1798 \tl_new:N \__spintent_spdsym_suffix_tl

```

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

```

prefix
suffix
set-sym
read-sym
unknown

```

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

```

1799 \keys_define:nn { spintent / spdsym }
1800 {
1801   prefix .code:n =
1802   { \__spintent_sanitizet_affix:Nn \__spintent_spdsym_prefix_tl {#1} },
1803   prefix .initial:n = ,
1804   prefix .value_required:n = true,
1805   suffix .code:n =
1806   { \__spintent_sanitizet_affix:Nn \__spintent_spdsym_suffix_tl {#1} },
1807   suffix .initial:n = ,
1808   suffix .value_required:n = true,
1809   set-sym .choices:nn =
1810   { old-style , two-dots , slash }
1811   {
1812     \int_case:nn { \l_keys_choice_int }
1813     {
1814       {1} { \tl_set:Nn \__spintent_spdsym_symbol_tl { \div } }
1815       {2} { \tl_set:Nn \__spintent_spdsym_symbol_tl { : } }
1816       {3} { \tl_set:Nn \__spintent_spdsym_symbol_tl { / } }
1817     }
1818   },
1819   set-sym .initial:n = two-dots,
1820   set-sym .value_required:n = true,
1821   set-sym / unknown .code:n =
1822   \msg_error:nneee { spintent } { unknown-choice }
1823   { set-sym } { old-style, ~two-dots, ~slash } { \exp_not:n {#1} },
1824   read-sym .choices:nn =
1825   { dividido , dividido-por , dividido-entre }
1826   {
1827     \int_case:nn { \l_keys_choice_int }
1828     {
1829       {1} { \tl_set:Nn \__spintent_spdsym_read_sym_tl { dividido } }
1830       {2} { \tl_set:Nn \__spintent_spdsym_read_sym_tl { dividido-por } }

```

```

1831         {3} { \tl_set:Nn \l__spintent_spdsym_read_sym_tl { dividido-entre } }
1832     }
1833 },
1834 read-sym .initial:n = dividido,
1835 read-sym .value_required:n = true,
1836 read-sym / unknown .code:n =
1837     \msg_error:nneee { spintent } { unknown-choice }
1838     { read-sym } { dividido, ~dividido-por, ~dividido-entre }
1839     { \exp_not:n {#1} },
1840 unknown .code:n =
1841     {
1842         \str_set:Nn \l__spintent_command_name_str { spdsym }
1843         \l__spintent_unknown_keys_cmd:n {#1}
1844     },
1845 }

```

(End of definition for *prefix* and others.)

`\l__spintent_spdsym_mathml:` The internal function `\l__spintent_spdsym_mathml:` outputs the configured intent structure, attaching prefixes and suffixes if present.

```

1846 \cs_new_protected:Nn \l__spintent_spdsym_mathml:
1847 {
1848     \MathMLintent
1849     {
1850         -
1851         \tl_if_empty:NF \l__spintent_spdsym_prefix_tl
1852         { \l__spintent_spdsym_prefix_tl - }
1853         \l__spintent_spdsym_read_sym_tl
1854         \tl_if_empty:NF \l__spintent_spdsym_suffix_tl
1855         { - \l__spintent_spdsym_suffix_tl }
1856     }
1857     { \l__spintent_spdsym_symbol_tl }
1858 }

```

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

`\spdsym` Defines the `\spdsym` operator. It takes a single optional argument for local setup and delegates the output to `\l__spintent_spdsym_mathml:`.

```

1859 \NewDocumentCommand \spdsym { 0{} }
1860 {
1861     \mode_if_math:TF
1862     {
1863         \group_begin:
1864         \keys_set:nn { spintent / spdsym } {#1}
1865         \l__spintent_spdsym_mathml:
1866         \group_end:
1867     }
1868     { \msg_error:nnn { spintent } { need-math-mode } { \spdsym } }
1869 }

```

(End of definition for `\spdsym`. This function is documented on page 17.)

15.26 The command `\spmsym`

`\l__spintent_spmsym_read_sym_tl` Variables to store the regional reading string, the visual operator, and the optional semantic affixes specifically for `\spmsym`.

```

\l__spintent_spmsym_symbol_tl
\l__spintent_spmsym_prefix_tl
\l__spintent_spmsym_suffix_tl
1870 \tl_new:N \l__spintent_spmsym_read_sym_tl
1871 \tl_new:N \l__spintent_spmsym_symbol_tl
1872 \tl_new:N \l__spintent_spmsym_prefix_tl
1873 \tl_new:N \l__spintent_spmsym_suffix_tl

```

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

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

```

suffix
set-sym
read-sym
print-sym
unknown
1874 \keys_define:nn { spintent / spmsym }
1875 {
1876     prefix .code:n =
1877     { \l__spintent_sanitizize_affix:Nn \l__spintent_spmsym_prefix_tl {#1} },
1878     prefix .initial:n = ,

```

```

1879   prefix      .value_required:n = true,
1880   suffix      .code:n =
1881     { \__spinttent_sanitizet_affix:Nn \__spinttent_spmsym_suffix_tl {#1} },
1882   suffix      .initial:n = ,
1883   suffix      .value_required:n = true,
1884   set-sym     .choices:nn =
1885     { cross , dot , asterisk }
1886     {
1887       \int_case:nn { \_keys_choice_int }
1888       {
1889         {1} { \tl_set:Nn \__spinttent_spmsym_symbol_tl { \times } }
1890         {2} { \tl_set:Nn \__spinttent_spmsym_symbol_tl { \cdot } }
1891         {3} { \tl_set:Nn \__spinttent_spmsym_symbol_tl { \ast } }
1892       }
1893     },
1894   set-sym     .initial:n = dot,
1895   set-sym     .value_required:n = true,
1896   set-sym / unknown .code:n =
1897     \msg_error:nneee { spinttent } { unknown-choice }
1898     { set-sym } { cross, ~dot, ~asterisk } { \exp_not:n {#1} },
1899   read-sym    .choices:nn =
1900     { por , multiplicado-por , veces }
1901     {
1902       \int_case:nn { \_keys_choice_int }
1903       {
1904         {1} { \tl_set:Nn \__spinttent_spmsym_read_sym_tl { por } }
1905         {2} { \tl_set:Nn \__spinttent_spmsym_read_sym_tl { multiplicado-por } }
1906         {3} { \tl_set:Nn \__spinttent_spmsym_read_sym_tl { veces } }
1907       }
1908     },
1909   read-sym    .initial:n = por,
1910   read-sym    .value_required:n = true,
1911   read-sym / unknown .code:n =
1912     \msg_error:nneee { spinttent } { unknown-choice }
1913     { read-sym } { por, ~multiplicado-por, ~veces } { \exp_not:n {#1} },
1914   print-sym   .bool_set:N = \__spinttent_spmsym_print_sym_bool,
1915   print-sym   .initial:n = true,
1916   print-sym   .value_required:n = true,
1917   unknown     .code:n =
1918     {
1919       \str_set:Nn \__spinttent_command_name_str { spmsym }
1920       \__spinttent_unknown_keys_cmd:n {#1}
1921     },
1922 }

```

(End of definition for *prefix* and others.)

`\__spinttent_spmsym_mathml:` The internal function `\__spinttent_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.

```

1923 \cs_new_protected:Nn \__spinttent_spmsym_mathml:
1924 {
1925   \MathMLintent
1926   {
1927     -
1928     \tl_if_empty:NF \__spinttent_spmsym_prefix_tl
1929       { \__spinttent_spmsym_prefix_tl - }
1930     \__spinttent_spmsym_read_sym_tl
1931     \tl_if_empty:NF \__spinttent_spmsym_suffix_tl
1932       { - \__spinttent_spmsym_suffix_tl }
1933   }
1934   {
1935     \bool_if:NTF \__spinttent_spmsym_print_sym_bool
1936       { \__spinttent_spmsym_symbol_tl }
1937       {
1938         \mathopen { \c__spinttent_invisible_sep_int }
1939       }
1940   }
1941 }

```

(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:`.

```

1942 \NewDocumentCommand \spmsym { 0{} }
1943 {
1944   \mode_if_math:TF
1945   {
1946     \group_begin:
1947       \keys_set:nn { spinttent / spmsym } {#1}
1948       \__spinttent_spmsym_mathml:
1949     \group_end:
1950   }
1951   { \msg_error:nnn { spinttent } { need-math-mode } { \spmsym } }
1952 }

```

(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*.

```

1953 \msg_new:nnnn { spinttent } { spdiv-empty-arg }
1954 {
1955   Empty~argument~for~'\token_to_str:N \spdiv'\~\msg_line_context:.
1956 }
1957 {
1958   A~dividend~and~a~divisor~are~required.
1959 }

```

(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`.

```

1960 \keys_define:nn { spinttent / spdiv }
1961 {
1962   cifras-sep      .meta:nn = { spinttent / spnum } { cifras-sep      = {#1} },
1963   read-sign       .meta:nn = { spinttent / spnum } { read-sign       = {#1} },
1964   read-period-sep .meta:nn = { spinttent / spnum } { read-period-sep = {#1} },
1965   notation-sym    .meta:nn = { spinttent / spnum } { notation-sym    = {#1} },
1966   set-sym         .meta:nn = { spinttent / spdsym } { set-sym         = {#1} },
1967   read-sym        .meta:nn = { spinttent / spdsym } { read-sym        = {#1} },
1968   wrap-parens     .bool_set:N = \__spinttent_spdiv_wrap_parens_bool,
1969   wrap-parens     .initial:n = false,
1970   wrap-parens     .value_required:n = true,
1971   read-parens     .bool_set:N = \__spinttent_spdiv_read_parens_bool,
1972   read-parens     .initial:n = true,
1973   read-parens     .value_required:n = true,
1974   unknown         .code:n =
1975   {
1976     \str_set:Nn \__spinttent_command_name_str { spdiv }
1977     \__spinttent_unknown_keys_cmd:n {#1}
1978   },
1979 }

```

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

15.27.3 Internal functions for `\spdiv`

`\__spinttent_spdiv_render_op:nn` The function `\__spinttent_spdiv_render_op:nn` parses both `{\arguments}` through the `\spnum` command and embeds the semantic operator from `\spdsym`.

```

1980 \cs_new_protected:Nn \__spinttent_spdiv_render_op:nn
1981 {
1982   \spnum {#1}
1983   \MathMLintent { _ \__spinttent_spdsym_read_sym_tl }

```

```

1984     {
1985         \l__spintent_spdsym_symbol_tl
1986     }
1987     \spnum {#2}
1988 }

```

(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`.

```

1989 \cs_new_protected:Nn \__spintent_spdiv_mathml:nn
1990 {
1991     \bool_if:NTF \l__spintent_spdiv_wrap_parens_bool
1992     {
1993         \bool_if:NTF \l__spintent_spdiv_read_parens_bool
1994         {
1995             \lparen \__spintent_spdiv_render_op:nn {#1} {#2} \rparen
1996         }
1997         {
1998             \MathMLintent { :silent } { \lparen }
1999             \__spintent_spdiv_render_op:nn {#1} {#2}
2000             \MathMLintent { :silent } { \rparen }
2001         }
2002     }
2003     { \__spintent_spdiv_render_op:nn {#1} {#2} }
2004 }

```

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

`\__spintent_spdiv_exec:nnn`

The function `\__spintent_spdiv_exec:nnn` sets *⟨keys⟩*, validates *⟨arguments⟩*, and calls `\__spintent_spdiv_mathml:nn`.

```

2005 \cs_new_protected:Nn \__spintent_spdiv_exec:nnn
2006 {
2007     \keys_set:nn { spintent / spdiv } {#1}
2008     \tl_if_blank:NTF {#2}
2009     { \msg_error:nn { spintent } { spdiv-empty-arg } }
2010     {
2011         \tl_if_blank:NTF {#3}
2012         { \msg_error:nn { spintent } { spdiv-empty-arg } }
2013         {
2014             \__spintent_spdiv_mathml:nn {#2} {#3}
2015         }
2016     }
2017 }

```

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

`\spdiv`

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

```

2018 \NewDocumentCommand \spdiv { O{} m m }
2019 {
2020     \mode_if_math:TF
2021     {
2022         \group_begin:
2023         \__spintent_spdiv_exec:nnn {#1} {#2} {#3}
2024         \group_end:
2025     }
2026     { \msg_error:nnn { spintent } { need-math-mode } { \spdiv } }
2027 }

```

(End of definition for `\spdiv`. This function is documented on page 18.)

15.28 The command `\spmul`

The user-level command `\spmul` formats multiplication representations. It takes two mandatory arguments: the *⟨first factor⟩* and the *⟨second factor⟩*. Both *⟨arguments⟩* are internally processed by the `\spnum` command to ensure proper digit separation. This command checks that neither the *⟨first factor⟩* nor the *⟨second factor⟩* is *blank*.

15.28.1 Definition of error messages for \spmul

spmul-empty-arg

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

```
2028 \msg_new:nnnn { spintent } { spmul-empty-arg }
2029 {
2030   Empty~argument~for~'\token_to_str:N \spmul'~\msg_line_context:.
2031 }
2032 {
2033   A~first~factor~and~a~second~factor~are~required.
2034 }
```

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

15.28.2 Definition of keys for \spmul

cifras-sep

read-sign

read-period-sep

notation-sym

set-sym

read-sym

wrap-parens

read-parens

unknown

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

```
2035 \keys_define:nn { spintent / spmul }
2036 {
2037   cifras-sep      .meta:nn = { spintent / spnum } { cifras-sep      = {#1} },
2038   read-sign       .meta:nn = { spintent / spnum } { read-sign       = {#1} },
2039   read-period-sep .meta:nn = { spintent / spnum } { read-period-sep = {#1} },
2040   notation-sym    .meta:nn = { spintent / spnum } { notation-sym    = {#1} },
2041   set-sym         .meta:nn = { spintent / spmsym } { set-sym         = {#1} },
2042   read-sym        .meta:nn = { spintent / spmsym } { read-sym        = {#1} },
2043   wrap-parens     .bool_set:N = \l__spintent_spmul_wrap_parens_bool,
2044   wrap-parens     .initial:n = false,
2045   wrap-parens     .value_required:n = true,
2046   read-parens     .bool_set:N = \l__spintent_spmul_read_parens_bool,
2047   read-parens     .initial:n = true,
2048   read-parens     .value_required:n = true,
2049   unknown         .code:n =
2050   {
2051     \str_set:Nn \l__spintent_command_name_str { spmul }
2052     \__spintent_unknown_keys_cmd:n {#1}
2053   },
2054 }
```

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

15.28.3 Internal functions for \spmul

__spintent_spmul_render_op:nn

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

```
2055 \cs_new_protected:Nn \__spintent_spmul_render_op:nn
2056 {
2057   \spnum {#1}
2058   \MathMLintent { _ \l__spintent_spmsym_read_sym_tl }
2059   {
2060     \l__spintent_spmsym_symbol_tl
2061   }
2062   \spnum {#2}
2063 }
```

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

__spintent_spmul_mathml:nn

The function `\__spintent_spmul_mathml:nn` builds the `MathML` structure of the multiplication. If `wrap-parens` is true, it adds the structural parentheses around `\__spintent_spmul_render_op:nn`, read or silenced according to `read-parens`.

```
2064 \cs_new_protected:Nn \__spintent_spmul_mathml:nn
2065 {
2066   \bool_if:NTF \l__spintent_spmul_wrap_parens_bool
2067   {
2068     \bool_if:NTF \l__spintent_spmul_read_parens_bool
2069     {
2070       \lparen \__spintent_spmul_render_op:nn {#1} {#2} \rparen
2071     }
2072     {
2073       \MathMLintent { :silent } { \lparen }
```

```

2074         \__spinttent_spmul_render_op:nn {#1} {#2}
2075         \MathMLintent { :silent } { \rparen }
2076     }
2077 }
2078 { \__spinttent_spmul_render_op:nn {#1} {#2} }
2079 }

```

(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`.

```

2080 \cs_new_protected:Nn \__spinttent_spmul_exec:nnn
2081 {
2082     \keys_set:nn { spinttent / spmul } {#1}
2083     \tl_if_blank:nTF {#2}
2084     { \msg_error:nn { spinttent } { spmul-empty-arg } }
2085     {
2086         \tl_if_blank:nTF {#3}
2087         { \msg_error:nn { spinttent } { spmul-empty-arg } }
2088         {
2089             \__spinttent_spmul_mathml:nn {#2} {#3}
2090         }
2091     }
2092 }

```

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

`\spmul`

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

```

2093 \NewDocumentCommand \spmul { O{} m m }
2094 {
2095     \mode_if_math:TF
2096     {
2097         \group_begin:
2098         \__spinttent_spmul_exec:nnn {#1} {#2} {#3}
2099         \group_end:
2100     }
2101     { \msg_error:nnn { spinttent } { need-math-mode } { \spmul } }
2102 }

```

(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 delegate argument validation and calculation to the Lua module `spinttent.lua`, constructing a valid intent.

15.29.1 Definition of variables for `\spgcd` and `\spmcn`

Internal variables generated dynamically via a comma-separated list to avoid code duplication for both symmetric commands.

```

2103 \clist_new:N \l__spinttent_mc_luaset_types_clist
2104 \str_new:N \l__spinttent_mc_luaset_all_numeric_str
2105 \seq_new:N \l__spinttent_mc_args_seq
2106 \seq_new:N \l__spinttent_mc_raw_args_seq
2107 \seq_new:N \l__spinttent_mc_types_seq
2108 \int_new:N \l__spinttent_mc_idx_int
2109 \clist_map_inline:nn { gcd , mcm }
2110 {
2111     \tl_new:c { \l__spinttent_sp #1 _luaset_ #1 _value_tl }
2112     \bool_new:c { \l__spinttent_sp #1 _read_terse_bool }
2113     \tl_new:c { \l__spinttent_sp #1 _prefix_tl }
2114 }

```

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

15.29.2 Definition of keys for `\spgcd` and `\spmcn`

`result`

`prefix`

`read-sty`

`upper-case`

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

```

2115 \clist_map_inline:nn { gcd , mcm }

```

```

2116 {
2117   \keys_define:nn { spintent / sp #1 }
2118   {
2119     result      .bool_set:c = { l__spintent_sp #1 _print_result_bool },
2120     result      .initial:n = false,
2121     result      .value_required:n = true,
2122     prefix      .code:n =
2123       { \__spintent_sanitize_affix:cn { l__spintent_sp #1 _prefix_tl } {##1} },
2124     prefix      .initial:n = ,
2125     prefix      .value_required:n = true,
2126     read-sty    .choices:nn =
2127       { terse , clear }
2128       {
2129         \int_case:nn { \l_keys_choice_int }
2130         {
2131           {1} { \bool_set_true:c { l__spintent_sp #1 _read_terse_bool } }
2132           {2} { \bool_set_false:c { l__spintent_sp #1 _read_terse_bool } }
2133         }
2134       },
2135     read-sty    .initial:n = clear,
2136     read-sty    .value_required:n = true,
2137     read-sty / unknown .code:n =
2138       \msg_error:nneee { spintent } { unknown-choice }
2139       { read-sty } { terse , ~clear } { \exp_not:n {##1} },
2140     upper-case .bool_set:c = { l__spintent_sp #1 _print_upper_bool },
2141     upper-case .initial:n = false,
2142     upper-case .value_required:n = true,
2143     concept    .bool_set:c = { l__spintent_sp #1 _concept_bool },
2144     concept    .initial:n = true,
2145     concept    .value_required:n = true,
2146     silent     .bool_set:c = { l__spintent_sp #1 _silent_bool },
2147     silent     .initial:n = false,
2148     silent     .value_required:n = true,
2149     unknown    .code:n =
2150       {
2151         \str_set:Nn \l__spintent_command_name_str { sp #1 }
2152         \__spintent_unknown_keys_cmd:n {##1}
2153       },
2154   }
2155 }

```

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

15.29.3 Internal functions for \spmc and \spmc

__spintent_mc_process_args:nn

The function `\__spintent_mc_process_args:nn` parses the operation ‘#1’ and its raw argument list ‘#2’. Calls `\__spintent_luafun_mc_classify_args:n`, splits ‘#2’ into `\l__spintent_mc_raw_args_seq` and `\l__spintent_mc_luaset_types_clist` into `\l__spintent_mc_types_seq`, then walks the former by position, fetching the matching type from the latter through `\seq_item:Nn`, to build `\l__spintent_mc_args_seq`: each argument wrapped through `\spnum` when its type is `number`, verbatim otherwise. Then it calls `\__spintent_mc_mathml_args:n`.

```

2156 \cs_new_protected:Nn \__spintent_mc_process_args:nn
2157 {
2158   \__spintent_luafun_mc_classify_args:n {#2}
2159   \seq_set_from_clist:Nn \l__spintent_mc_raw_args_seq {#2}
2160   \seq_set_from_clist:NN \l__spintent_mc_types_seq \l__spintent_mc_luaset_types_clist
2161   \seq_clear:N \l__spintent_mc_args_seq
2162   \int_zero:N \l__spintent_mc_idx_int
2163   \seq_map_inline:Nn \l__spintent_mc_raw_args_seq
2164   {
2165     \int_incr:N \l__spintent_mc_idx_int
2166     \str_if_eq:eeTF
2167     { \seq_item:Nn \l__spintent_mc_types_seq { \l__spintent_mc_idx_int } }
2168     { number }
2169     {
2170       \seq_put_right:Nn \l__spintent_mc_args_seq { \spnum {##1} }
2171     }
2172     { \seq_put_right:Nn \l__spintent_mc_args_seq {##1} }
2173   }
2174   \__spintent_mc_mathml_args:n {#1}
2175 }

```

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

`\__spintenc_mc_mathml_args:n`

The function `\__spintenc_mc_mathml_args:n` receives the short identifier in ‘#1’ and injects the intents around the already-formatted `\l__spintenc_mc_args_seq`.

```

2176 \cs_new_protected:Nn \__spintenc_mc_mathml_args:n
2177 {
2178   \bool_if:cTF { \l__spintenc_sp #1 _silent_bool }
2179   {
2180     \seq_set_map:Nnn \l__spintenc_mc_args_seq \l__spintenc_mc_args_seq
2181     {
2182       \MathMLintent { _- } { \__spintenc_inner_sep: ##1 }
2183     }
2184     \MathMLintent { :silent } { \c__spintenc_invisible_sep_int } % prevent "por"
2185     \MathMLintent { :silent } { \lparen }
2186     % prevent "intervalo"
2187     \MathMLintent { :silent } { \c__spintenc_invisible_sep_int }
2188     \seq_use:Nnnn \l__spintenc_mc_args_seq
2189     { \MathMLintent { :silent } { , } }
2190     { \MathMLintent { :silent } { , } }
2191     { \MathMLintent { :silent } { , } }
2192   }
2193   {
2194     \MathMLintent { :silent } { \c__spintenc_invisible_sep_int } % prevent doble "de"
2195     \bool_if:cTF { \l__spintenc_sp #1 _concept_bool }
2196     {
2197       \MathMLintent { _entre } { \lparen }
2198     }
2199     {
2200       \MathMLintent { :silent } { \lparen }
2201     }
2202     % prevent "intervalo"
2203     \MathMLintent { :silent } { \c__spintenc_invisible_sep_int }
2204     \seq_use:Nnnn \l__spintenc_mc_args_seq
2205     {
2206       \MathMLintent { :silent } { , }
2207       \MathMLintent { _y- } { \c__spintenc_invisible_sep_int } % "y" medio
2208     }
2209     {
2210       \MathMLintent { :silent } { , }
2211     }
2212     {
2213       \MathMLintent { :silent } { , }
2214       \MathMLintent { _y- } { \c__spintenc_invisible_sep_int } % "y" final
2215     }
2216   }
2217   \MathMLintent { :silent } { \rparen }
2218 }

```

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

`\__spintenc_spmc_exec:nnnn`

The internal function `\__spintenc_spmc_exec:nnnn` is the shared function for both `\spmc` and `\spmc`. 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 `\__spintenc_spmc_mathml:nn` to print the operator name.

With ‘#4’ given, classifies it through `\__spintenc_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 `\__spintenc_spmc_result_mathml:n`. `\__spintenc_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__spintenc_invisible_sep_int` is applied.

```

2219 \cs_new_protected:Nn \__spintenc_spmc_exec:nnnn
2220 {
2221   \keys_set:nn { spintenc / sp #1 } {#3}
2222   \__spintenc_spmc_mathml:nn {#1} {#2}

```

With ‘#4’ given, it is classified with `\__spintenc_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 `\__spinttent_mc_process_args:n` prints the argument list.

```

2223 \tl_if_novalue:nF {#4}
2224 {
2225   \seq_set_from_clist:Nn \l__spinttent_mc_raw_args_seq {#4}
2226   \int_compare:nNnTF { \seq_count:N \l__spinttent_mc_raw_args_seq } < { 2 }
2227   {
2228     \msg_error:nne { spinttent } { mc-too-few-args } { \exp_not:c { sp #1 } }
2229   }
2230   {
2231     \__spinttent_luafun_mc_classify_args:n {#4}
2232     \str_if_eq:VnT \l__spinttent_mc_luaaset_all_numeric_str { true }
2233     {
2234       \use:c { __spinttent_luafun_calculate_ #1 :n } {#4}
2235     }
2236     \__spinttent_mc_process_args:nn {#1} {#4}
2237     \__spinttent_spmc_result_mathml:n {#1}
2238   }
2239 }
2240 }

```

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

`\__spinttent_spmc_mathml:nn`

The function `\__spinttent_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.

```

2241 \cs_new_protected:Nn \__spinttent_spmc_mathml:nn
2242 {
2243   \bool_if:cTF { l__spinttent_sp #1 _silent_bool }
2244   {
2245     \MathMLintent { :silent }
2246     {
2247       \mathrm
2248       {
2249         \bool_if:cTF { l__spinttent_sp #1 _print_upper_bool }
2250         { \text_uppercase:n {#1} } {#1}
2251       }
2252     }
2253   }
2254   {
2255     \bool_if:cTF { l__spinttent_sp #1 _concept_bool }
2256     {
2257       \MathMLintent
2258       {
2259         -
2260         \tl_if_empty:cF { l__spinttent_sp #1 _prefix_tl }
2261         {
2262           \tl_use:c { l__spinttent_sp #1 _prefix_tl } -
2263         }
2264         \bool_if:cTF { l__spinttent_sp #1 _read_terse_bool }
2265         {#1} {#2}
2266       }
2267       {
2268         \mathrm
2269         {
2270           \bool_if:cTF { l__spinttent_sp #1 _print_upper_bool }
2271           { \text_uppercase:n {#1} } {#1}
2272         }
2273       }
2274     }
2275     {
2276       \MathMLintent { :silent }
2277       {
2278         \mathrm
2279         {
2280           \bool_if:cTF { l__spinttent_sp #1 _print_upper_bool }
2281           { \text_uppercase:n {#1} } {#1}
2282         }
2283       }
2284     }
2285   }

```

```

2284     }
2285   }
2286 }

```

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

`\__spintenc_spmc_result_mathml:n`

The function `\__spintenc_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.

```

2287 \cs_new_protected:Nn \__spintenc_spmc_result_mathml:n
2288 {
2289   \str_if_eq:VnT \__spintenc_mc_luaset_all_numeric_str { true }
2290   {
2291     \bool_if:cT { \__spintenc_sp #1 _print_result_bool }
2292     {
2293       \bool_if:cTF { \__spintenc_sp #1 _silent_bool }
2294       {
2295         \MathMLintent { :silent } { = }
2296         \MathMLintent { _- }
2297         {
2298           \__spintenc_inner_sep:
2299           \tl_use:c { \__spintenc_sp #1 _luaset_ #1 _value_tl }
2300         }
2301       }
2302       {
2303         \MathMLintent { _es-igual-a } { = }
2304         \tl_use:c { \__spintenc_sp #1 _luaset_ #1 _value_tl }
2305       }
2306     }
2307   }
2308 }

```

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

`\spmc`
`\spmc`

The commands `\spmc` and `\spmc` 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 `\__spintenc_spmc_exec:nnnn`, passing along the appropriate reading strings and arguments.

```

2309 \NewDocumentCommand \spmc { 0{} d() }
2310 {
2311   \mode_if_math:TF
2312   {
2313     \group_begin:
2314     \__spintenc_spmc_exec:nnnn { mcd } { máximo-común-divisor } {#1} {#2}
2315     \group_end:
2316   }
2317   { \msg_error:nnn { spintenc } { need-math-mode } { \spmc } }
2318 }
2319 \NewDocumentCommand \spmc { 0{} d() }
2320 {
2321   \mode_if_math:TF
2322   {
2323     \group_begin:
2324     \__spintenc_spmc_exec:nnnn { mcm } { mínimo-común-múltiplo } {#1} {#2}
2325     \group_end:
2326   }
2327   { \msg_error:nnn { spintenc } { need-math-mode } { \spmc } }
2328 }

```

(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 delegate argument validation and calculation to the Lua module `spintent.lua`, constructing a valid intent.

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

`\l__spintenc_md_out_seq`
`\l__spintenc_spnmd_luaset_is_number_str`
`\l__spintenc_spnM_spnD_luaset_error_str`
`\l__spintenc_spnM_luaset_result_tl`

Internal variables generated dynamically for multiples and divisors.

```

2329 \seq_new:N \l__spintenc_md_out_seq

```

©2026 by Pablo González L

```

2330 \str_new:N \__spinttent_spnM_spnD_luaset_error_str
2331 \str_new:N \__spinttent_spnmd_luaset_is_number_str
2332 \clist_map_inline:nn { nM , nD }
2333 {
2334     \tl_new:c { \__spinttent_sp #1 _luaset_result_tl }
2335     \str_new:c { \__spinttent_sp #1 _luaset_is_truncated_str }
2336     \tl_new:c { \__spinttent_sp #1 _prefix_tl }
2337 }

```

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

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.

```

2338 \msg_new:nnnn { spinttent } { md-invalid-arg }
2339 {
2340     Invalid~argument~for~'\tl_trim_spaces:n {#1}'~\msg_line_context:.
2341 }
2342 {
2343     The~argument~must~be~a~single~positive~integer.
2344 }

```

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

```

2345 \msg_new:nnnn { spinttent } { mc-too-few-args }
2346 {
2347     Invalid~arguments~for~'\tl_trim_spaces:n {#1}'~\msg_line_context:.
2348 }
2349 {
2350     At~least~two~arguments~are~required.
2351 }

```

(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`.

```

2352 \clist_map_inline:nn { nM , nD }
2353 {
2354     \keys_define:nn { spinttent / sp #1 }
2355     {
2356         result      .bool_set:c = { \__spinttent_sp #1 _print_result_bool },
2357         result      .initial:n = false,
2358         result      .value_required:n = true,
2359         silent      .bool_set:c = { \__spinttent_sp #1 _silent_bool },
2360         silent      .initial:n = false,
2361         silent      .value_required:n = true,
2362         result-num .tl_set:c = { \__spinttent_sp #1 _print_result_num_tl },
2363         result-num .initial:n = ,
2364         result-num .value_required:n = true,
2365         prefix      .code:n =
2366             { \__spinttent_sanitize_affix:cn { \__spinttent_sp #1 _prefix_tl } {##1} },
2367         prefix      .initial:n = ,
2368         prefix      .value_required:n = true,
2369         concept      .bool_set:c = { \__spinttent_sp #1 _concept_bool },
2370         concept      .initial:n = true,
2371         concept      .value_required:n = true,
2372         unknown      .code:n =
2373             {
2374                 \str_set:Nn \__spinttent_command_name_str { sp #1 }
2375                 \__spinttent_unknown_keys_cmd:n {##1}
2376             },
2377     }
2378 }

```

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

15.30.4 Internal functions for `\spnM` and `\spnD`

The internal function `\__spinttent_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*”, and appends ‘-de’.

```

2379 \cs_new:Nn \__spinttent_spnM_spnD_intent:nn

```

```

2380 {
2381   -
2382   \tl_if_empty:cF { l__spintent_sp #1 _prefix_tl }
2383   { \tl_use:c { l__spintent_sp #1 _prefix_tl } - }
2384   \str_case:nnF {#2} { { multiplos } { múltiplos } }
2385   {#2}
2386   -de
2387 }

```

(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 injected 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.

```

2388 \cs_new_protected:Nn \__spintent_spnM_spnD_silent:nn
2389 {
2390   \MathMLintent { :silent } { \operatorname {#1} }
2391   \MathMLintent { :silent } { \lparen }
2392   \MathMLintent { :silent } { \c__spintent_invisible_sep_int } % prevent doble "de"
2393   \MathMLintent { _- } { \__spintent_inner_sep: #2 }
2394   \MathMLintent { :silent } { \rparen }
2395 }

```

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

`\__spintent_spnM_spnD_result:nn`

The internal function `\__spintent_spnM_spnD_result:nn` calculates and formats the computed set. It takes the command identifier ‘#1’ and the classified numeric argument ‘#2’. It invokes the Lua calculation, reporting `md-invalid-arg` if it fails. On success, it maps the resulting clist into a sequence and formats the printed output with custom intents for the separators and the conjunctions.

```

2396 \cs_new_protected:Nn \__spintent_spnM_spnD_result:nn
2397 {
2398   \use:c { __spintent_luafun_calcular_ #1 :nn } {#2}
2399   {
2400     \tl_use:c { l__spintent_sp #1 _print_result_num_tl }
2401   }
2402   \str_if_eq:VnTF \__spintent_spnM_spnD_luaset_error_str { true }
2403   {
2404     \msg_error:nne { spintent } { md-invalid-arg } { \exp_not:c { sp #1 } }
2405   }
2406   {
2407     \MathMLintent { :silent } { \c__spintent_invisible_sep_int } % prevent "por"
2408     \MathMLintent { _son } { = \{ }
2409     \seq_set_from_clist:Ne
2410       \__spintent_md_out_seq { \tl_use:c { l__spintent_sp #1 _luaset_result_tl } }
2411     \seq_use:Nnnn \__spintent_md_out_seq
2412     {
2413       \MathMLintent { :silent } { , }
2414       \str_if_eq:vnT { l__spintent_sp #1 _luaset_is_truncated_str } { false }
2415       {
2416         \MathMLintent { _y- } { \c__spintent_invisible_sep_int } % "y" (solo dos)
2417       }
2418     }
2419     { \MathMLintent { :silent } { , } }
2420     {
2421       \MathMLintent { :silent } { , }
2422       \str_if_eq:vnT { l__spintent_sp #1 _luaset_is_truncated_str } { false }
2423       {
2424         \MathMLintent { _y- } { \c__spintent_invisible_sep_int } % "y" final
2425       }
2426     }
2427     \str_if_eq:vnT { l__spintent_sp #1 _luaset_is_truncated_str } { true }
2428     {
2429       \MathMLintent { :silent } { , } ~ \dots
2430     }
2431     \MathMLintent { :silent } { \} }
2432   }
2433 }

```

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

`\__spinttent_spnM_spnD_exec:nnnnn`

The internal function `\__spinttent_spnM_spnD_exec:nnnnn` acts as the primary entry point for the multiples and divisors commands. It takes the command identifier `'#1'`, its printed mathematical symbol `'#2'`, its base reading string `'#3'`, the local key-value options `'#4'`, and the optional argument `'#5'`. It first sets the provided keys within the current scope. With `'#5'` given, `\__spinttent_luafun_spnmd_classify:n` classifies it, and a zero is rejected with `md-invalid-arg`. Then it calls `\__spinttent_spnM_spnD_mathml:nnnn`.

```

2434 \cs_new_protected:Nn \__spinttent_spnM_spnD_exec:nnnnn
2435 {
2436   \keys_set:nn { spinttent / sp #1 } {#4}
2437   \tl_if_novalue:nF {#5}
2438   {
2439     \__spinttent_luafun_spnmd_classify:n { \exp_not:n {#5} }
2440     \str_if_eq:VnT \__spinttent_spnM_spnD_luaset_error_str { true }
2441     { \msg_error:nne { spinttent } { md-invalid-arg } { \exp_not:c { sp #1 } } }
2442   }
2443   \__spinttent_spnM_spnD_mathml:nnnn {#1} {#2} {#3} {#5}
2444 }

```

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

`\__spinttent_spnM_spnD_mathml:nnnn`

The function `\__spinttent_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 `\__spinttent_spnmd_luaset_is_number_str` decides whether it is wrapped with `\spnum`, and then `result` decides whether `\__spinttent_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 `\__spinttent_spnM_spnD_silent:nn`; otherwise `concept=true` carries the literal intent from `\__spinttent_spnM_spnD_intent:nn`, `false` silences only the operator symbol, leaving the argument to read normally.

```

2445 \cs_new_protected:Nn \__spinttent_spnM_spnD_mathml:nnnn
2446 {
2447   \tl_if_novalue:nTF {#4}
2448   {
2449     \bool_if:cTF { \__spinttent_sp #1 _silent_bool }
2450     { \__spinttent_spnM_spnD_silent:nn {#2} { n } }
2451     {
2452       \bool_if:cTF { \__spinttent_sp #1 _concept_bool }
2453       {
2454         \MathMLintent
2455         { \__spinttent_spnM_spnD_intent:nn {#1} {#3} }
2456         { \operatorname {#2} }
2457       }
2458       { \MathMLintent { :silent } { \operatorname {#2} } }
2459       \MathMLintent { :silent } { \lparen }
2460       % prevent doble "de"
2461       \MathMLintent { :silent } { \c__spinttent_invisible_sep_int }
2462       n
2463       \MathMLintent { :silent } { \rparen }
2464     }
2465   }
2466   {
2467     \bool_if:cTF { \__spinttent_sp #1 _silent_bool }
2468     {
2469       \str_if_eq:VnTF \__spinttent_spnmd_luaset_is_number_str { true }
2470       { \__spinttent_spnM_spnD_silent:nn {#2} { \spnum {#4} } }
2471       { \__spinttent_spnM_spnD_silent:nn {#2} {#4} }
2472     }
2473     {
2474       \bool_if:cTF { \__spinttent_sp #1 _concept_bool }
2475       {
2476         \MathMLintent
2477         { \__spinttent_spnM_spnD_intent:nn {#1} {#3} }
2478         { \operatorname {#2} }
2479       }
2480       { \MathMLintent { :silent } { \operatorname {#2} } }

```

```

2481 \MathMLintent { :silent } { \lparen }
2482 % prevent doble "de"
2483 \MathMLintent { :silent } { \c__spinttent_invisible_sep_int }
2484 \str_if_eq:VnTF \l__spinttent_spnmd_luaset_is_number_str { true }
2485 { \spnum {#4} }
2486 {#4}
2487 \MathMLintent { :silent } { \rparen }
2488 }
2489 \str_if_eq:VnT \l__spinttent_spnmd_luaset_is_number_str { true }
2490 {
2491   \bool_if:cT { l__spinttent_sp #1 _print_result_bool }
2492   {
2493     \bool_if:cTF { l__spinttent_sp #1 _silent_bool }
2494     {
2495       % prevent "por"
2496       \MathMLintent { :silent } { \c__spinttent_invisible_sep_int }
2497       \MathMLintent { _- }
2498       {
2499         \__spinttent_inner_sep:
2500         \__spinttent_spnM_spnD_result:nn {#1} {#4}
2501       }
2502     }
2503     {
2504       \__spinttent_spnM_spnD_result:nn {#1} {#4}
2505     }
2506   }
2507 }
2508 }
2509 }

```

(End of definition for `\__spinttent_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 `\__spinttent_spnM_spnD_exec:nnnnn`, passing along the optional *⟨keys⟩* ‘#1’ and the numeric *⟨argument⟩* ‘#2’.

```

2510 \NewDocumentCommand \spnM { 0{} d() }
2511 {
2512   \mode_if_math:TF
2513   {
2514     \group_begin:
2515     \__spinttent_spnM_spnD_exec:nnnnn { nM } { M } { multiplos } {#1} {#2}
2516     \group_end:
2517   }
2518   { \msg_error:nnn { spinttent } { need-math-mode } { \spnM } }
2519 }
2520 \NewDocumentCommand \spnD { 0{} d() }
2521 {
2522   \mode_if_math:TF
2523   {
2524     \group_begin:
2525     \__spinttent_spnM_spnD_exec:nnnnn { nD } { D } { divisores } {#1} {#2}
2526     \group_end:
2527   }
2528   { \msg_error:nnn { spinttent } { need-math-mode } { \spnD } }
2529 }

```

(End of definition for `\spnM` and `\spnD`. These functions are documented on page 20.)

15.31 The command `\sprazon`

The user command `\sprazon` typesets a ratio $a:b$ as a fraction, using `\dfrac` or `\frac` according to *use-dfrac*, read natively, without parsing in Lua.

15.31.1 Definition of variables for `\sprazon`

`\l__spinttent_sprazon_cmd_str` The string variable `\l__spinttent_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`.

```

2530 \str_new:N \l__spinttent_sprazon_cmd_str
2531 \str_set:Nn \l__spinttent_sprazon_cmd_str { sprazon }

```

(End of definition for `\l__spinttent_sprazon_cmd_str`.)

15.31.2 Definition of keys for \sprazon

`silent` We define the $\langle keys \rangle$ `silent`, `use-dfrac` and `inline`.

`use-dfrac`

`inline`

`unknown`

```

2532 \keys_define:nn { spintent / sprazon }
2533 {
2534     silent      .bool_set:N = \l__spintent_sprazon_silent_bool,
2535     silent      .initial:n = false,
2536     silent      .value_required:n = true,
2537     use-dfrac    .bool_set:N = \l__spintent_sprazon_use_dfrac_bool,
2538     use-dfrac    .initial:n = false,
2539     use-dfrac    .value_required:n = true,
2540     inline      .bool_set:N = \l__spintent_sprazon_inline_bool,
2541     inline      .initial:n = false,
2542     inline      .value_required:n = true,
2543     unknown     .code:n =
2544     {
2545         \str_set_eq:NN \l__spintent_command_name_str \l__spintent_sprazon_cmd_str
2546         \__spintent_unknown_keys_cmd:n {#1}
2547     },
2548 }
```

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

`\__spintent_sprazon_exec:nnn` The function `\__spintent_sprazon_exec:nnn` receives the optional $\langle keys \rangle$ 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`.

```

2549 \cs_new_protected:Nn \__spintent_sprazon_exec:nnn
2550 {
2551     \keys_set:nn { spintent / sprazon } {#1}
2552     \__spintent_sprazon_mathml:nn {#2} {#3}
2553 }
```

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

`\__spintent_sprazon_mathml:nn` The function `\__spintent_sprazon_mathml:nn` receives the numerator in ‘#1’ and the denominator in ‘#2’. With `silent`, the whole expression is silenced as one unit: `\frac` or `\dfrac`, according to `use-dfrac`, or `#1 : #2` with `inline`.

Otherwise, with `inline=false`, the default, the numerator and the denominator are tagged with a native `\MathMLarg`, as num and denom, inside `\frac` or `\dfrac`, and the outer `\MathMLintent` reads `_es-a:infix($num,$denom)`. With `inline=true`, the two references are tagged side by side and the `:` between them carries the `_es-a:infix($num,$denom)` intent.

```

2554 \cs_new_protected:Nn \__spintent_sprazon_mathml:nn
2555 {
2556     \__spintent_invisible_sep:
2557     \bool_if:NTF \l__spintent_sprazon_silent_bool
2558     {
2559         \MathMLintent { _- }
2560         {
2561             \__spintent_inner_sep:
2562             \bool_if:NTF \l__spintent_sprazon_inline_bool
2563             { #1 : #2 }
2564             {
2565                 \bool_if:NTF \l__spintent_sprazon_use_dfrac_bool
2566                 { \dfrac {#1} {#2} }
2567                 { \frac {#1} {#2} }
2568             }
2569         }
2570     }
2571     {
2572         \bool_if:NTF \l__spintent_sprazon_inline_bool
2573         {
2574             \MathMLintent { _es-a:infix($num,$denom) }
2575             {
2576                 \MathMLarg { num } {#1}
2577                 :
2578                 \MathMLarg { denom } {#2}
2579             }
2580         }
2581         {
```

```

2582         \MathMLintent { _es-a:infix($num,$denom) }
2583     {
2584         \__spintent_inner_sep:
2585         \bool_if:NTF \l__spintent_sprazon_use_dfrac_bool
2586         {
2587             \dfrac { \MathMLarg { num } {#1} } { \MathMLarg { denom } {#2} }
2588         }
2589         { \frac { \MathMLarg { num } {#1} } { \MathMLarg { denom } {#2} } }
2590     }
2591 }
2592 }
2593 }

```

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

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

```

2594 \NewDocumentCommand \sprazon { 0{} m m }
2595 {
2596     \mode_if_math:TF
2597     {
2598         \group_begin:
2599         \__spintent_sprazon_exec:nnn {#1} {#2} {#3}
2600         \group_end:
2601     }
2602     { \msg_error:nnn { spintent } { need-math-mode } { \sprazon } }
2603 }

```

(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.

\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.

```

2604 \NewDocumentCommand \spProp { 0{} m m m m }
2605 {
2606     \mode_if_math:TF
2607     {
2608         \group_begin:
2609         \str_set:Nn \l__spintent_sprazon_cmd_str { spProp }
2610         \__spintent_sprazon_exec:nnn {#1} {#2} {#3}
2611         \bool_if:NTF \l__spintent_sprazon_silent_bool
2612         {
2613             \MathMLintent { :silent } { = }
2614         }
2615         { \MathMLintent { _como } { = } }
2616         \__spintent_sprazon_exec:nnn {#1} {#4} {#5}
2617         \group_end:
2618     }
2619     { \msg_error:nnn { spintent } { need-math-mode } { \spProp } }
2620 }

```

(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.

These internal string variables store the values returned from the Lua module `spintent.lua`. They hold the extracted numerical values for degrees, minutes, and seconds, as well as the error status used during formatting.

```

2621 \str_new:N \l__spintent_spang_luaset_grado_str
2622 \str_new:N \l__spintent_spang_luaset_minuto_str
2623 \str_new:N \l__spintent_spang_luaset_segundo_str
2624 \str_new:N \l__spintent_spang_luaset_error_str

```

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

15.33.1 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.

```

2625 \msg_new:nnnn { spintent } { spang-invalid-arg }
2626 {
2627   Invalid~argument~'#2'~for~'\tl_trim_spaces:n {#1}'~\msg_line_context:.
2628 }
2629 {
2630   The~argument~must~have~the~form~'degrees',~'degrees:minutes'~or~
2631   'degrees:minutes:seconds'.
2632 }

```

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

15.33.2 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:**.

```

2633 \cs_new_protected:Nn \__spintent_spang_exec:n
2634 {
2635   \str_set:Nn \l__spintent_spang_luaset_error_str { false }
2636   \__spintent_luafun_spang_parse:n {#1}
2637   \str_if_eq:VnTF \l__spintent_spang_luaset_error_str { true }
2638   {
2639     \msg_error:nnnn { spintent } { spang-invalid-arg } { \spang } {#1}
2640   }
2641   { \__spintent_spang_mathml: }
2642 }

```

(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 $\langle argument \rangle$ from expansion.

```

2643 \cs_new_protected:Nn \__spintent_spang_mathml:
2644 {
2645   \str_if_empty:NF \l__spintent_spang_luaset_grado_str
2646   {
2647     \exp_args:Ne \spunit { \l__spintent_spang_luaset_grado_str ° }
2648   }
2649   \str_if_empty:NF \l__spintent_spang_luaset_minuto_str
2650   {
2651     \MathMLintent { _minutos:postfix($m) }
2652     {
2653       \MathMLarg { m }
2654       {
2655         \exp_args:Ne \spunit { \l__spintent_spang_luaset_minuto_str ' }
2656       }
2657     }
2658   }
2659   \str_if_empty:NF \l__spintent_spang_luaset_segundo_str
2660   {
2661     \MathMLintent { _segundos:postfix($s) }
2662     {
2663       \MathMLarg { s }
2664       {
2665         \exp_args:Ne \spunit { \l__spintent_spang_luaset_segundo_str " }
2666       }
2667     }
2668   }
2669 }

```

(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 $\langle argument \rangle$.

```

2670 \NewDocumentCommand \spang { m }
2671 {
2672   \mode_if_math:TF
2673   {
2674     \group_begin:
2675     __spintent_spang_exec:n {#1}
2676     \group_end:
2677   }
2678   { \msg_error:nnn { spintent } { need-math-mode } { \spang } }
2679 }

```

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

```

2680 \msg_new:nnnn { spintent } { spnZ-not-integer }
2681 {
2682   Invalid~argument~'~#1'~for~'\token_to_str:N \spnZ'~\msg_line_context:.
2683 }
2684 {
2685   Only~integers~are~accepted;~decimal~separators~and~units~are~not~allowed.
2686 }

```

(End of definition for `spnZ-not-integer`.)

15.34.2 Definition of keys for `\spnZ`

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

```

2687 \keys_define:nn { spintent / spnZ }
2688 {
2689   cifras-sep .meta:nn = { spintent / spnum } { cifras-sep = {#1} },
2690   read-sign .meta:nn = { spintent / spnum } { read-sign = {#1} },
2691   wrap-parens .bool_set:N = \__spintent_spnz_wrap_parens_bool,
2692   wrap-parens .initial:n = false,
2693   wrap-parens .value_required:n = true,
2694   read-parens .bool_set:N = \__spintent_spnz_read_parens_bool,
2695   read-parens .initial:n = true,
2696   read-parens .value_required:n = true,
2697   unknown .code:n =
2698   {
2699     \str_set:Nn \__spintent_command_name_str { spnZ }
2700     \__spintent_unknown_keys_cmd:n {#1}
2701   },
2702 }

```

(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:`.

```

2703 \cs_new_protected:Nn __spintent_spnz_mathml:
2704 {
2705   \bool_if:NTF \__spintent_spnz_wrap_parens_bool

```

```

2706     {
2707         \bool_if:NTF \l__spinttent_spnz_read_parens_bool
2708         {
2709             \lparen \l__spinttent_spnnum_print_num_start: \rparen
2710         }
2711         {
2712             \MathMLintent { :silent } { \lparen }
2713             \l__spinttent_spnnum_print_num_start:
2714             \MathMLintent { :silent } { \rparen }
2715         }
2716     }
2717     { \l__spinttent_spnnum_print_num_start: }
2718 }

```

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

`\l__spinttent_spnz_exec:nn`

The function `\l__spinttent_spnz_exec:nn` first processes the *<keys>* passed in ‘#1’ and then calls `\l__spinttent_luafun_clean_split_and_set:n` on ‘#2’. It verifies that the input is a valid integer by checking if the variable `\l__spinttent_luaset_has_integer_str` is `true`. If not `true`, an “error” is raised.

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

```

2719 \cs_new_protected:Nn \l__spinttent_spnz_exec:nn
2720 {
2721     \keys_set:nn { spinttent / spnZ } {#1}
2722     \l__spinttent_luafun_clean_split_and_set:n { \exp_not:n {#2} }
2723     \str_if_eq:VnTF \l__spinttent_luaset_has_integer_str { true }
2724     { \l__spinttent_spnz_mathml: }
2725     { \msg_error:nnn { spinttent } { spnZ-not-integer } {#2} }
2726 }

```

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

`\spnZ`

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

```

2727 \NewDocumentCommand \spnZ { O{} m }
2728 {
2729     \mode_if_math:TF
2730     {
2731         \group_begin:
2732         \l__spinttent_spnz_exec:nn {#1} {#2}
2733         \group_end:
2734     }
2735     { \msg_error:nnn { spinttent } { need-math-mode } { \spnZ } }
2736 }

```

(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 delegates the scaling of the integer part to the Lua module `spintent.lua` via `\l__spinttent_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`

The variable `\l__spinttent_spmQ_join_word_str` stores the value set by the key `join-word` which will be verbalized by MathCAT as a grammatical connector between the integer and fractional parts.

The variable `\l__spinttent_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 `\l__spinttent_spmQ_print_scale_int:n`.

The variables `\l__spinttent_spmQ_wrap_parens_l_tl` and `\l__spinttent_spmQ_wrap_parens_r_tl` store the sized parentheses set by the function `\l__spinttent_spmQ_set_wrap_parens_size:` and used by the key `wrap-parens`.

The boolean variable `\l__spinttent_spmQ_error_bool` is set to `true` by `\l__spinttent_spmQ_check_part:nn` when any of the three arguments is rejected, so that nothing is typeset.

The integer variable `\l__spinttent_saved_luamml_flag_int` temporarily stores the current value of `\l__luamml_flag_int` before disabling it during the injection of the scaled integer node, and restores it

afterwards. This prevents `\luamml` from generating a spurious `<math>` element around the `sub_box` processed in isolation, before `\frac` has expanded.

```

2737 \str_new:N \l__spinttent_spmQ_join_word_str
2738 \str_new:N \l__spinttent_spmQ_intent_read_sign_str
2739 \tl_new:N \l__spinttent_spmQ_wrap_parens_l_tl
2740 \tl_new:N \l__spinttent_spmQ_wrap_parens_r_tl
2741 \bool_new:N \l__spinttent_spmQ_error_bool
2742 \int_new:N \l__spinttent_saved_luamml_flag_int

```

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

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.

```

2743 \msg_new:nnnn { spinttent } { spmQ-zero-denominator }
2744 {
2745   Invalid~denominator~for~'\token_to_str:N \spmQ'\~\msg_line_context:.
2746 }
2747 {
2748   The~denominator~of~a~fraction~cannot~be~zero.
2749 }
2750 \msg_new:nnnn { spinttent } { spmQ-not-integer }
2751 {
2752   Invalid~argument~'#1'~for~'\token_to_str:N \spmQ'\~\msg_line_context:.
2753 }
2754 {
2755   Only~integers~are~accepted.
2756 }
2757 \msg_new:nnnn { spinttent } { spmQ-signed-part }
2758 {
2759   Invalid~argument~'#1'~for~'\token_to_str:N \spmQ'\~\msg_line_context:.
2760 }
2761 {
2762   The~numerator~and~the~denominator~must~be~integers~without~sign;~only~the~integer~
2763   part~may~have~one.
2764 }

```

(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`.

```

2765 \keys_define:nn { spinttent / spmQ }
2766 {
2767   join-word .choices:nn =
2768     { entero , enteros , y }
2769     {
2770       \int_case:nn { \l_keys_choice_int }
2771       {
2772         {1} { \str_set:Nn \l__spinttent_spmQ_join_word_str { entero } }
2773         {2} { \str_set:Nn \l__spinttent_spmQ_join_word_str { enteros } }
2774         {3} { \str_set:Nn \l__spinttent_spmQ_join_word_str { y } }
2775       }
2776     },
2777   join-word .initial:n = y,
2778   join-word .value_required:n = true,
2779   join-word / unknown .code:n =
2780     \msg_error:nneee { spinttent } { unknown-choice }
2781     { join-word } { entero, ~enteros, ~y } { \exp_not:n {#1} },
2782   use-spnZ .bool_set:N = \l__spinttent_spmQ_use_spnZ_bool,
2783   use-spnZ .initial:n = true,
2784   use-spnZ .value_required:n = true,
2785   use-dfrac .bool_set:N = \l__spinttent_spmQ_use_dfrac_bool,
2786   use-dfrac .initial:n = false,
2787   use-dfrac .value_required:n = true,
2788   wrap-parens .bool_set:N = \l__spinttent_spmQ_wrap_parens_bool,
2789   wrap-parens .initial:n = false,
2790   wrap-parens .value_required:n = true,

```

```

2791     read-parens .bool_set:N = \l__spinttent_spmQ_read_parens_bool,
2792     read-parens .initial:n = true,
2793     read-parens .value_required:n = true,
2794     silent .bool_set:N = \l__spinttent_spmQ_silent_bool,
2795     silent .initial:n = false,
2796     silent .value_required:n = true,
2797     unknown .code:n =
2798     {
2799         \str_set:Nn \l__spinttent_command_name_str { spmQ }
2800         \__spinttent_unknown_keys_cmd:n {#1}
2801     },
2802 }

```

(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 unsized `\lparen` and `\rparen`.

If the key `use-dfrac` is active, the size is always forced to `\Bigl/\Bigr` regardless of the current mathematical style, since `\dfrac` always produces a display-height fraction.

```

2803 \cs_new_protected:Nn \__spinttent_spmQ_set_wrap_parens_size:
2804 {
2805     \int_case:nnF { \mathstyle }
2806     {
2807         { 0 }
2808         {
2809             \tl_set:Nn \l__spinttent_spmQ_wrap_parens_l_tl { \Bigl \lparen }
2810             \tl_set:Nn \l__spinttent_spmQ_wrap_parens_r_tl { \Bigr \rparen }
2811         }
2812         { 1 }
2813         {
2814             \tl_set:Nn \l__spinttent_spmQ_wrap_parens_l_tl { \Bigl \lparen }
2815             \tl_set:Nn \l__spinttent_spmQ_wrap_parens_r_tl { \Bigr \rparen }
2816         }
2817         { 2 }
2818         {
2819             \tl_set:Nn \l__spinttent_spmQ_wrap_parens_l_tl { \bigl \lparen }
2820             \tl_set:Nn \l__spinttent_spmQ_wrap_parens_r_tl { \bigr \rparen }
2821         }
2822         { 3 }
2823         {
2824             \tl_set:Nn \l__spinttent_spmQ_wrap_parens_l_tl { \bigl \lparen }
2825             \tl_set:Nn \l__spinttent_spmQ_wrap_parens_r_tl { \bigr \rparen }
2826         }
2827     }
2828     {
2829         \tl_set:Nn \l__spinttent_spmQ_wrap_parens_l_tl { \lparen }
2830         \tl_set:Nn \l__spinttent_spmQ_wrap_parens_r_tl { \rparen }
2831     }

```

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.

```

2832     \bool_if:NT \l__spinttent_spmQ_use_dfrac_bool
2833     {
2834         \tl_set:Nn \l__spinttent_spmQ_wrap_parens_l_tl { \Bigl \lparen }
2835         \tl_set:Nn \l__spinttent_spmQ_wrap_parens_r_tl { \Bigr \rparen }
2836     }
2837 }

```

(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 `\__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.

```

2838 \cs_new_protected:Nn \__spinttent_spmQ_start_wrap_parens:
2839 {
2840     \bool_if:NT \__spinttent_spmQ_wrap_parens_bool
2841     {
2842         \__spinttent_spmQ_set_wrap_parens_size:
2843         \bool_if:NTF \__spinttent_spmQ_read_parens_bool
2844         { \__spinttent_spmQ_wrap_parens_l_tl }
2845         { \MathMLintent { :silent } { \__spinttent_spmQ_wrap_parens_l_tl } }
2846     }
2847 }
2848 \cs_new_protected:Nn \__spinttent_spmQ_stop_wrap_parens:
2849 {
2850     \bool_if:NT \__spinttent_spmQ_wrap_parens_bool
2851     {
2852         \__spinttent_spmQ_set_wrap_parens_size:
2853         \bool_if:NTF \__spinttent_spmQ_read_parens_bool
2854         { \__spinttent_spmQ_wrap_parens_r_tl }
2855         { \MathMLintent { :silent } { \__spinttent_spmQ_wrap_parens_r_tl } }
2856     }
2857 }

```

(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:` injects 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.

```

2858 \cs_new_protected:Nn \__spinttent_spmQ_set_join_word:
2859 {
2860     \str_case:Nn \__spinttent_spmQ_join_word_str
2861     {
2862         { entero } { \MathMLintent { _entero } { \c__spinttent_invisible_sep_int } }
2863         { enteros } { \MathMLintent { _enteros } { \c__spinttent_invisible_sep_int } }
2864         { y } { \MathMLintent { _y- } { \c__spinttent_invisible_sep_int } }
2865     }
2866 }

```

(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:Nn`, it temporarily disables `luamml` by saving the current value of `\__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.

```

2867 \cs_new_protected:Nn \__spinttent_spmQ_print_scale_int:
2868 {
2869     \int_set_eq:Nc \__spinttent_saved_luamml_flag_int { \__luamml_flag_int }
2870     \int_set:Nn \__luamml_flag_int { 0 }
2871     \bool_if:NTF \__spinttent_spmQ_use_dfrac_bool
2872     { \__spinttent_luafun_spmQ_scale_int:Nn \__spinttent_luaset_part_int_tl { dfrac } }
2873     { \__spinttent_luafun_spmQ_scale_int:Nn \__spinttent_luaset_part_int_tl { frac } }
2874     \int_set_eq:cN { \__luamml_flag_int } \__spinttent_saved_luamml_flag_int
2875 }

```

(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 populate the shared Lua variables, then checks the status of `\__spinttent_luaset_has_integer_str`.

If the status is `true`, it inspects `\l__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 `\l__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.

```

2876 \cs_new_protected:Nn \__spinttent_spmQ_print_scale_int:n
2877 {
2878   \__spinttent_luafun_clean_split_and_set:n { \exp_not:n {#1} }
2879   \str_if_eq:VnTF \l__spinttent_luaset_has_integer_str { true }
2880   {
2881     \tl_if_empty:NTF \l__spinttent_luaset_sign_tl
2882     {
2883       \__spinttent_spmQ_print_scale_int:
2884     }
2885     {
2886       \str_if_eq:VnTF \l__spinttent_luaset_sign_tl { + }
2887       {
2888         \str_set:Nn
2889           \l__spinttent_spmQ_intent_read_sign_str { _más:prefix($e) }
2890       }
2891       {
2892         \str_set:Nn
2893           \l__spinttent_spmQ_intent_read_sign_str { _menos:prefix($e) }
2894       }
2895       \MathMLintent { \l__spinttent_spmQ_intent_read_sign_str }
2896       {
2897         % \__spinttent_invisible_sep:
2898         \l__spinttent_luaset_sign_tl \c__spinttent_invisible_sep_int
2899         \MathMLarg { e } { \l__spinttent_spmQ_print_scale_int: }
2900       }
2901     }
2902   }
2903   { \msg_error:nnn { spinttent } { spmQ-not-integer } {#1} }
2904 }

```

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

`\__spinttent_spmQ_print_spmZ_dfrac:nn`

The function `\__spinttent_spmQ_print_spmZ_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 `\__spinttent_spmQ_check_part:nn`. The function branches on the two boolean variables `\l__spinttent_spmQ_use_spmZ_bool` and `\l__spinttent_spmQ_use_dfrac_bool`, controlled respectively by the keys `use-spmZ` and `use-dfrac`. When `use-spmZ` 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 dynamically via `\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.

```

2905 \cs_new_protected:Nn \__spinttent_spmQ_print_spmZ_dfrac:nn
2906 {
2907   \bool_if:NTF \l__spinttent_spmQ_use_spmZ_bool
2908   {
2909     \group_begin:
2910     \cs_set_protected:Nn \__spinttent_invisible_sep:
2911     {
2912       \__spinttent_luafun_invisible_sep_mark:
2913     }
2914     \use:c { \bool_if:NT \l__spinttent_spmQ_use_dfrac_bool { d } frac }
2915     { \spnZ {#1} } { \spnZ {#2} }
2916   \group_end:
2917   }
2918   {
2919     \use:c { \bool_if:NT \l__spinttent_spmQ_use_dfrac_bool { d } frac }
2920     {#1} {#2}
2921   }
2922 }

```

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

`\__spinttent_spmQ_print:nnn`

The function `\__spinttent_spmQ_print:nnn` is the main rendering orchestrator for `\spmQ`. It takes three arguments: the integer part `#1`, the numerator `#2`, and the denominator `#3`. It opens the optional parentheses via `\__spinttent_spmQ_start_wrap_parens:`, then calls `\__spinttent_spmQ_print_scale_int:n` to scale and render the integer part. Next, it injects the grammatical connector between the integer and the fraction via `\__spinttent_spmQ_set_join_word:`. Then it calls `\__spinttent_spmQ_print_spnZ_dfrac:nn` to typeset the fraction using the keys `use-dfrac` and `use-spnZ`. Finally, it closes the optional parentheses via `\__spinttent_spmQ_stop_wrap_parens:`.

```
2923 \cs_new_protected:Nn \__spinttent_spmQ_print:nnn
2924 {
2925   \__spinttent_spmQ_start_wrap_parens:
2926   \__spinttent_spmQ_print_scale_int:n {#1}
2927   \__spinttent_spmQ_set_join_word:
2928   \__spinttent_spmQ_print_spnZ_dfrac:nn {#2} {#3}
2929   \__spinttent_spmQ_stop_wrap_parens:
2930 }
```

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

`\__spinttent_spmQ_check_part:nn`

The function `\__spinttent_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 `\__spinttent_luafun_clean_split_and_set:n` on it. If `\l__spinttent_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 `\l__spinttent_spmQ_error_bool` to `true`; the arguments are rejected whatever the value of the key `silent`.

```
2931 \cs_new_protected:Nn \__spinttent_spmQ_check_part:nn
2932 {
2933   \__spinttent_luafun_clean_split_and_set:n { \exp_not:n {#2} }
2934   \str_if_eq:VnTF \l__spinttent_luaaset_has_integer_str { true }
2935   {
2936     \bool_lazy_or:nnTF
2937     { \tl_if_empty_p:N \l__spinttent_luaaset_sign_tl }
2938     { \str_if_eq_p:nn {#1} { int } }
2939     {
2940       \str_if_eq:nnT {#1} { den }
2941       {
2942         \int_compare:nNnT { \l__spinttent_luaaset_part_int_tl } = { 0 }
2943         {
2944           \msg_error:nn { spinttent } { spmQ-zero-denominator }
2945           \bool_set_true:N \l__spinttent_spmQ_error_bool
2946         }
2947       }
2948     }
2949     {
2950       \msg_error:nnn { spinttent } { spmQ-signed-part } {#2}
2951       \bool_set_true:N \l__spinttent_spmQ_error_bool
2952     }
2953   }
2954   {
2955     \msg_error:nnn { spinttent } { spmQ-not-integer } {#2}
2956     \bool_set_true:N \l__spinttent_spmQ_error_bool
2957   }
2958 }
```

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

`\__spinttent_spmQ_exec:nnnn`

The internal function `\__spinttent_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 `\__spinttent_spmQ_check_part:nn`. If any of them is rejected, nothing is typeset.

Otherwise, it calls `\__spinttent_spmQ_mathml:nnn`.

```
2959 \cs_new_protected:Nn \__spinttent_spmQ_exec:nnnn
2960 {
2961   \keys_set:nn { spinttent / spmQ } {#1}
2962   \bool_set_false:N \l__spinttent_spmQ_error_bool
```

```

2963     \__spinttent_spmQ_check_part:nn { int } {#2}
2964     \__spinttent_spmQ_check_part:nn { num } {#3}
2965     \__spinttent_spmQ_check_part:nn { den } {#4}
2966     \bool_if:NF \__spinttent_spmQ_error_bool
2967     { \__spinttent_spmQ_mathml:nnn {#2} {#3} {#4} }
2968 }

```

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

`\__spinttent_spmQ_mathml:nnn`

The internal function `\__spinttent_spmQ_mathml:nnn` takes the integer ‘`#1`’, the numerator ‘`#2`’, and the denominator ‘`#3`’, and calls `\__spinttent_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.

```

2969 \cs_new_protected:Nn \__spinttent_spmQ_mathml:nnn
2970 {
2971     \bool_if:NTF \__spinttent_spmQ_silent_bool
2972     {
2973         \__spinttent_invisible_sep:
2974         \MathMLintent { _- }
2975         {
2976             \__spinttent_inner_sep:
2977             \__spinttent_spmQ_print:nnn {#1} {#2} {#3}
2978         }
2979     }
2980     { \__spinttent_spmQ_print:nnn {#1} {#2} {#3} }
2981 }

```

(End of definition for `\__spinttent_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 `\__spinttent_spmQ_exec:nnnn` to process the optional *(keys)* and the *{(arguments)}*.

```

2982 \NewDocumentCommand \spmQ { O{} m m m }
2983 {
2984     \mode_if_math:TF
2985     {
2986         \group_begin:
2987         \__spinttent_spmQ_exec:nnnn {#1} {#2} {#3} {#4}
2988         \group_end:
2989     }
2990     { \msg_error:nnn { spinttent } { need-math-mode } { \spmQ } }
2991 }

```

(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 `spinttent.lua` and share the variables `\l__spinttent_geo_luaset_error_str`, `\l__spinttent_geo_luaset_intent_str`, `\l__spinttent_geo_luaset_print_tl`, `\l__spinttent_geo_luaset_letra_str`, `\l__spinttent_geo_luaset_points_clist` and `\l__spinttent_geo_luaset_tipo_str`, with which the intent is built. A single point is always annotated with `\__spinttent_mathmlarg_point:nnn`. Every command activates `\l__spinttent_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 shared variables

Internal variables shared across all commands of the family to avoid repeated calls into Lua.

```

\l__spinttent_geo_luaset_error_str
\l__spinttent_geo_luaset_intent_str
\l__spinttent_geo_luaset_print_tl
\l__spinttent_geo_luaset_points_clist
\l__spinttent_geo_luaset_is_name_str
\l__spinttent_geo_luaset_poly_sym_str
\l__spinttent_geo_luaset_angulo_case_str
\l__spinttent_geo_luaset_concept_str
\l__spinttent_geo_luaset_letra_word_str
\l__spinttent_geo_luaset_sub_spoken_str
\l__spinttent_geo_luaset_nombre_es_num_str
\l__spinttent_geo_luaset_letra_str
\l__spinttent_geo_luaset_sub_spoken_str

```

```

2992 \str_new:N \l__spinttent_geo_luaset_error_str
2993 \str_new:N \l__spinttent_geo_luaset_intent_str
2994 \tl_new:N \l__spinttent_geo_luaset_print_tl
2995 \clist_new:N \l__spinttent_geo_luaset_points_clist
2996 \str_new:N \l__spinttent_geo_luaset_is_name_str
2997 \str_new:N \l__spinttent_geo_luaset_poly_sym_str
2998 \str_new:N \l__spinttent_geo_luaset_angulo_case_str
2999 \str_new:N \l__spinttent_geo_luaset_concept_str
3000 \str_new:N \l__spinttent_geo_luaset_letra_word_str
3001 \str_new:N \l__spinttent_geo_luaset_sub_spoken_str

```

```

3002 \str_new:N \l__spintent_geo_luaset_nombre_es_num_str
3003 \str_new:N \l__spintent_geo_luaset_letra_str
3004 \str_new:N \l__spintent_geo_luaset_sub_str
3005 \str_new:N \l__spintent_geo_luaset_tipo_str

```

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

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`

Variables set by the function `\__spintent_luafun_spcoord_parse_and_set:n`. The string variable `\l__spintent_spcoord_luaset_sep_str` is , or ;, whichever the argument actually used. The variables for the abscissa x and the ordinate y have the same meaning: `_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.

```

3006 \str_new:N \l__spintent_spcoord_luaset_error_str
3007 \str_new:N \l__spintent_spcoord_luaset_sep_str
3008 \clist_map_inline:nn { x , y }
3009 {
3010   \str_new:c { \l__spintent_spcoord_luaset_ #1 _sign_str      }
3011   \str_new:c { \l__spintent_spcoord_luaset_ #1 _type_str      }
3012   \str_new:c { \l__spintent_spcoord_luaset_ #1 _value_str     }
3013   \str_new:c { \l__spintent_spcoord_luaset_ #1 _has_frac_str  }
3014   \str_new:c { \l__spintent_spcoord_luaset_ #1 _composite_str }
3015 }

```

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

The variable `\l__spintent_spcoord_raw_arg_tl` holds the $\langle argument \rangle$ exactly as given. The variable `\l__spintent_spcoord_intent_str` holds the `:infix` word for the separator `_coma:infix` or `_punto-y-coma:infix`. The variables `\l__spintent_spcoord_wrap_parens_l_tl`/`\l__spintent_spcoord_wrap_parens_r_tl` hold the opening/closing delimiter, sized by the function `\__spintent_spcoord_set_wrap_parens_size:.`

```

3016 \tl_new:N \l__spintent_spcoord_raw_arg_tl
3017 \str_new:N \l__spintent_spcoord_intent_str
3018 \tl_new:N \l__spintent_spcoord_wrap_parens_l_tl
3019 \tl_new:N \l__spintent_spcoord_wrap_parens_r_tl

```

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

15.37.2 Definition of error messages for `\spcoord`

`spcoord-invalid-arg`

The “error” `spcoord-invalid-arg` is raised when the $\langle argument \rangle$ does not have the form *abscissa,ordinate* or *abscissa;ordinate*.

```

3020 \msg_new:nnnn { spintent } { spcoord-invalid-arg }
3021 {
3022   Invalid~argument~'#1'~for~'\token_to_str:N \spcoord'~\msg_line_context:.
3023 }
3024 {
3025   The~argument~must~have~the~form~'abscissa,~ordinate'~or~'abscissa;~ordinate'.
3026 }

```

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

15.37.3 Definition of keys for `\spcoord`

`silent`
`unknown`

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

```

3027 \keys_define:nn { spintent / spcoord }
3028 {
3029   silent .bool_set:N = \l__spintent_spcoord_silent_bool,
3030   silent .initial:n = false,
3031   silent .value_required:n = true,
3032   unknown .code:n =
3033   {
3034     \str_set:Nn \l__spintent_command_name_str { spcoord }

```

```

3035     \__spinttent_unknown_keys_cmd:n {#1}
3036   },
3037 }

```

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

15.37.4 Internal functions for \spcoord

__spinttent_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.

```

3038 \cs_new_protected:Nn \__spinttent_spcoord_set_wrap_parens_size:
3039 {
3040   \tl_set:Nn \__spinttent_spcoord_wrap_parens_l_tl { \lparen }
3041   \tl_set:Nn \__spinttent_spcoord_wrap_parens_r_tl { \rparen }
3042   \bool_if:nT
3043   {
3044     \bool_lazy_or_p:nn
3045     {
3046       \bool_not_p:n
3047       { \str_if_eq_p:Vn \__spinttent_spcoord_luaset_x_has_frac_str { none } }
3048     }
3049     {
3050       \bool_not_p:n
3051       { \str_if_eq_p:Vn \__spinttent_spcoord_luaset_y_has_frac_str { none } }
3052     }
3053   }
3054   {
3055     \int_case:nnF { \mathstyle }
3056     {
3057       { 0 }
3058       {
3059         \tl_set:Nn \__spinttent_spcoord_wrap_parens_l_tl { \Bigl \lparen }
3060         \tl_set:Nn \__spinttent_spcoord_wrap_parens_r_tl { \Bigr \rparen }
3061       }
3062       { 1 }
3063       {
3064         \tl_set:Nn \__spinttent_spcoord_wrap_parens_l_tl { \Bigl \lparen }
3065         \tl_set:Nn \__spinttent_spcoord_wrap_parens_r_tl { \Bigr \rparen }
3066       }
3067       { 2 }
3068       {
3069         \tl_set:Nn \__spinttent_spcoord_wrap_parens_l_tl { \bigl \lparen }
3070         \tl_set:Nn \__spinttent_spcoord_wrap_parens_r_tl { \bigr \rparen }
3071       }
3072       { 3 }
3073       {
3074         \tl_set:Nn \__spinttent_spcoord_wrap_parens_l_tl { \bigl \lparen }
3075         \tl_set:Nn \__spinttent_spcoord_wrap_parens_r_tl { \bigr \rparen }
3076       }
3077     }
3078   }
3079   \bool_if:nT
3080   {
3081     \bool_lazy_or_p:nn
3082     { \str_if_eq_p:Vn \__spinttent_spcoord_luaset_x_has_frac_str { dfraction } }
3083     { \str_if_eq_p:Vn \__spinttent_spcoord_luaset_y_has_frac_str { dfraction } }
3084   }
3085   {
3086     \tl_set:Nn \__spinttent_spcoord_wrap_parens_l_tl { \Bigl \lparen }
3087     \tl_set:Nn \__spinttent_spcoord_wrap_parens_r_tl { \Bigr \rparen }
3088   }
3089 }
3090 }

```

(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’.

```

3091 \cs_new_protected:Nn \__spintent_spcoord_value:n
3092 {
3093   \str_if_eq:eeTF
3094   { \str_use:c { \__spintent_spcoord_luaset_ #1 _type_str } } { number }
3095   {
3096     \exp_args:Ne \spnum { \str_use:c { \__spintent_spcoord_luaset_ #1 _value_str } }
3097   }
3098   { \str_use:c { \__spintent_spcoord_luaset_ #1 _value_str } }
3099 }

```

(End of definition for $__spintent_spcoord_value:n$.)

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

```

3100 \cs_new_protected:Nn \__spintent_spcoord_print:n
3101 {
3102   \str_case:enF { \str_use:c { \__spintent_spcoord_luaset_ #1 _sign_str } }
3103   {
3104     { + }
3105     {
3106       \MathMLintent { _más:prefix(\$vp) }
3107       {
3108         \str_use:c { \__spintent_spcoord_luaset_ #1 _sign_str }
3109         \MathMLarg{ vp } { \__spintent_spcoord_value:n {#1} }
3110       }
3111     }
3112     { - }
3113     {
3114       \MathMLintent { _menos:prefix(\$vn) }
3115       {
3116         \str_use:c { \__spintent_spcoord_luaset_ #1 _sign_str }
3117         \MathMLarg{ vn } { \__spintent_spcoord_value:n {#1} }
3118       }
3119     }
3120   }
3121   { \__spintent_spcoord_value:n {#1} }
3122 }

```

(End of definition for $__spintent_spcoord_print:n$.)

$__spintent_spcoord_inner_sep:n$ Receives x or y in ‘#1’. With the variable $_composite_str$ of that endpoint equal to `true`, places $__spintent_inner_sep:$ first, so that the group $\MathMLarg$ wraps holds two children and keeps its `arg`: `luamml` drops it from a sole node that carries a subscript, a superscript or a radical.

```

3123 \cs_new_protected:Nn \__spintent_spcoord_inner_sep:n
3124 {
3125   \str_if_eq:eeT
3126   { \str_use:c { \__spintent_spcoord_luaset_ #1 _composite_str } } { true }
3127   { \__spintent_inner_sep: }
3128 }

```

(End of definition for $__spintent_spcoord_inner_sep:n$.)

$__spintent_spcoord_intent_infix:$ Sets $__spintent_spcoord_intent_str$ to $_coma:infix$ or $_punto-y-coma:infix$, according to $__spintent_spcoord_luaset_sep_str$, and wraps both sides, each referenced with $_x$ or $_y$ in ‘#1’. The names $\$x$ and $\$y$ stand for the output of $__spintent_spcoord_print:n$.

```

3129 \cs_new_protected:Nn \__spintent_spcoord_intent_infix:
3130 {
3131   \str_if_eq:VnTF \__spintent_spcoord_luaset_sep_str { ; }
3132   { \str_set:Nn \__spintent_spcoord_intent_str { _punto-y-coma:infix } }
3133   { \str_set:Nn \__spintent_spcoord_intent_str { _coma:infix } }
3134   \MathMLintent { \__spintent_spcoord_intent_str (\$x, \$y) }
3135   {
3136     \MathMLarg{x}
3137     { \__spintent_spcoord_inner_sep:n { x } \__spintent_spcoord_print:n { x } }
3138     \MathMLintent { :silent } { \__spintent_spcoord_luaset_sep_str }
3139     \MathMLarg{y}
3140     { \__spintent_spcoord_inner_sep:n { y } \__spintent_spcoord_print:n { y } }

```

```

3141     }
3142 }

```

(End of definition for `\__spintent_spcoord_intent_infix:`)

`\__spintent_spcoord_mathml:N`

The argument is the boolean variable that holds the value of `silent` of the command that calls it. With `silent=true`, the parentheses and the argument are wrapped in a single ‘`_-`’ intent, and `\l__spintent_spcoord_raw_arg_tl` is typeset as given. Otherwise, the parentheses are silenced individually and `\__spintent_spcoord_intent_infix:` provides the reading between them.

```

3143 \cs_new_protected:Nn \__spintent_spcoord_mathml:N
3144 {
3145   \bool_if:NTF #1
3146   {
3147     \MathMLintent { _- }
3148     {
3149       \l__spintent_spcoord_wrap_parens_l_tl
3150       \l__spintent_spcoord_raw_arg_tl
3151       \l__spintent_spcoord_wrap_parens_r_tl
3152     }
3153   }
3154   {
3155     \MathMLintent { :silent } { \l__spintent_spcoord_wrap_parens_l_tl }
3156     \__spintent_spcoord_intent_infix:
3157     \MathMLintent { :silent } { \l__spintent_spcoord_wrap_parens_r_tl }
3158   }
3159 }

```

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

`\__spintent_spcoord_exec:nn`

Sets the *(keys)* in ‘`#1`’ and `\l__spintent_spcoord_raw_arg_tl` to ‘`#2`’. Calls `\__spintent_luafun_spcoord_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 `\__spintent_spcoord_mathml:N` preceded by an invisible separator placed *before* the opening parenthesis. The whole pair is wrapped in `\mathord`, which prevents MathCAT from reading `(1,2)` as an interval when an equals sign or `\in` follows it.

```

3160 \cs_new_protected:Nn \__spintent_spcoord_exec:nn
3161 {
3162   \keys_set:nn { spintent / spcoord } { #1 }
3163   \tl_set:Nn \l__spintent_spcoord_raw_arg_tl { #2 }
3164   \__spintent_luafun_spcoord_parse_and_set:n { \exp_not:n { #2 } }
3165   \str_if_eq:VnT \l__spintent_spcoord_luaset_error_str { true }
3166   { \msg_error:nne { spintent } { spcoord-invalid-arg } { \exp_not:n { #2 } } }
3167   \__spintent_spcoord_set_wrap_parens_size:
3168   \str_if_eq:VnF \l__spintent_spcoord_luaset_error_str { true }
3169   {
3170     \mathord
3171     {
3172       % prevent "intervalo"
3173       \MathMLintent { :silent } { \c__spintent_invisible_sep_int }
3174       \__spintent_spcoord_mathml:N \l__spintent_spcoord_silent_bool
3175     }
3176   }
3177 }

```

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

`\spcoord`

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

```

3178 \NewDocumentCommand \spcoord { 0 } { m }
3179 {
3180   \mode_if_math:TF
3181   {
3182     \group_begin:
3183     \attribute \l__spintent_scope_attr_int = 1
3184     \__spintent_spcoord_exec:nn { #1 } { #2 }
3185     \group_end:
3186   }
3187   { \msg_error:nnn { spintent } { need-math-mode } { \spcoord } }
3188 }

```

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

15.38 The command `\spinterval`

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

Variables set by the function `\__spintent_luafun_spinterval_parse_and_set:n`.

```
\__spintent_spinterval_luaset_error_str
\__spintent_spinterval_luaset_error_code_str
\__spintent_spinterval_luaset_left_sym_str
\__spintent_spinterval_luaset_right_sym_str
\__spintent_spinterval_luaset_type_str
\__spintent_spinterval_luaset_sep_str
\__spintent_spinterval_luaset_tall_str

3189 \str_new:N \__spintent_spinterval_luaset_error_str
3190 \str_new:N \__spintent_spinterval_luaset_error_code_str
3191 \str_new:N \__spintent_spinterval_luaset_left_sym_str
3192 \str_new:N \__spintent_spinterval_luaset_right_sym_str
3193 \str_new:N \__spintent_spinterval_luaset_type_str
3194 \str_new:N \__spintent_spinterval_luaset_sep_str
3195 \str_new:N \__spintent_spinterval_luaset_tall_str
```

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

The same variables for each endpoint, a and b. The integer and short ones are `true` when the value is an integer and when it is an integer of at most three digits, respectively; braced holds the value with each comma protected by braces.

```
3196 \clist_map_inline:nn { a , b }
3197 {
3198   \str_new:c { \__spintent_spinterval_luaset_ #1 _sign_str      }
3199   \str_new:c { \__spintent_spinterval_luaset_ #1 _type_str      }
3200   \str_new:c { \__spintent_spinterval_luaset_ #1 _value_str     }
3201   \str_new:c { \__spintent_spinterval_luaset_ #1 _inf_str       }
3202   \str_new:c { \__spintent_spinterval_luaset_ #1 _tall_str      }
3203   \str_new:c { \__spintent_spinterval_luaset_ #1 _composite_str }
3204   \str_new:c { \__spintent_spinterval_luaset_ #1 _integer_str   }
3205   \str_new:c { \__spintent_spinterval_luaset_ #1 _short_str     }
3206   \str_new:c { \__spintent_spinterval_luaset_ #1 _braced_str    }
3207 }
```

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

The variables `\__spintent_spinterval_mode_str`, `\__spintent_spinterval_prefix_tl`, `\__spintent_spinterval_read_arg_tl` and `\__spintent_spinterval_size_str` store the values set by the keys `read-cmd`, `prefix`, `read-arg` and `size`, and `\__spintent_spinterval_reason_str` the text of the error reason. The variables `\__spintent_spinterval_word_w_tl` and `\__spintent_spinterval_word_g_tl` store the word that names the interval and the word that joins its endpoints, and `\__spintent_spinterval_intent_arg_tl` and `\__spintent_spinterval_intent_group_tl` the intents of the whole interval and of the group of endpoints. The variables `\__spintent_spinterval_sym_l_tl`, `\__spintent_spinterval_sym_r_tl`, `\__spintent_spinterval_size_l_tl`, `\__spintent_spinterval_size_r_tl`, `\__spintent_spinterval_delim_l_tl` and `\__spintent_spinterval_delim_r_tl` store the delimiter symbols, their size commands and the delimiters ready to be typeset; `\__spintent_spinterval_tmp_tl` is a scratch variable.

```
3208 \tl_new:N \__spintent_spinterval_prefix_tl
3209 \tl_new:N \__spintent_spinterval_read_arg_tl
3210 \str_new:N \__spintent_spinterval_mode_str
3211 \str_new:N \__spintent_spinterval_size_str
3212 \str_new:N \__spintent_spinterval_reason_str
3213 \tl_new:N \__spintent_spinterval_word_w_tl
3214 \tl_new:N \__spintent_spinterval_word_g_tl
3215 \tl_new:N \__spintent_spinterval_intent_arg_tl
3216 \tl_new:N \__spintent_spinterval_intent_group_tl
3217 \tl_new:N \__spintent_spinterval_sym_l_tl
3218 \tl_new:N \__spintent_spinterval_sym_r_tl
3219 \tl_new:N \__spintent_spinterval_size_l_tl
3220 \tl_new:N \__spintent_spinterval_size_r_tl
3221 \tl_new:N \__spintent_spinterval_delim_l_tl
3222 \tl_new:N \__spintent_spinterval_delim_r_tl
3223 \tl_new:N \__spintent_spinterval_tmp_tl
```

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

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 $\langle argument \rangle$; it reports the argument and the reason set by `\__spinttent_spinterval_set_reason:`.

```

3224 \msg_new:nnnn { spinttent } { spinterval-invalid-arg }
3225 {
3226   Invalid~argument~'#1'~for~'\token_to_str:N \spinterval'~(#{2})~
3227   \msg_line_context:.
3228 }
3229 {
3230   Use~'(a;~b)'~or~'(a,~b)',~with~'(',~'[~'~or~']'~on~the~left~and~')',~'~'~or~'[~'~
3231   on~the~right.
3232 }

```

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

15.38.3 Definition of keys for \spinterval

read-cmd

prefix

read-arg

size

silent

concept

unknown

We define the $\langle keys \rangle$ `read-cmd`, `prefix`, `read-arg`, `size`, `silent` and `concept`.

```

3233 \keys_define:nn { spinttent / spinterval }
3234 {
3235   read-cmd .choices:nn = { school , mathcat }
3236   {
3237     \int_case:nn { \l_keys_choice_int }
3238     {
3239       {1} { \str_set:Nn \__spinttent_spinterval_mode_str { school } }
3240       {2} { \str_set:Nn \__spinttent_spinterval_mode_str { mathcat } }
3241     }
3242   },
3243   read-cmd .initial:n = school,
3244   read-cmd .value_required:n = true,
3245   read-cmd / unknown .code:n =
3246     \msg_error:nneee { spinttent } { unknown-choice }
3247     { read-cmd } { school,~mathcat } { \exp_not:n {#1} },
3248   prefix .code:n =
3249     { \__spinttent_sanitizize_affix:Nn \__spinttent_spinterval_prefix_tl {#1} },
3250   prefix .initial:n = ,
3251   prefix .value_required:n = true,
3252   read-arg .code:n =
3253     { \__spinttent_sanitizize_affix:Nn \__spinttent_spinterval_read_arg_tl {#1} },
3254   read-arg .initial:n = ,
3255   read-arg .value_required:n = true,
3256   size .choices:nn = { auto , none , big , Big , bigg , Bigg }
3257   {
3258     \int_case:nn { \l_keys_choice_int }
3259     {
3260       {1} { \str_set:Nn \__spinttent_spinterval_size_str { auto } }
3261       {2} { \str_set:Nn \__spinttent_spinterval_size_str { none } }
3262       {3} { \str_set:Nn \__spinttent_spinterval_size_str { big } }
3263       {4} { \str_set:Nn \__spinttent_spinterval_size_str { Big } }
3264       {5} { \str_set:Nn \__spinttent_spinterval_size_str { bigg } }
3265       {6} { \str_set:Nn \__spinttent_spinterval_size_str { Bigg } }
3266     }
3267   },
3268   size .initial:n = auto,
3269   size .value_required:n = true,
3270   size / unknown .code:n =
3271     \msg_error:nneee { spinttent } { unknown-choice }
3272     { size } { auto,~none,~big,~Big,~bigg,~Bigg } { \exp_not:n {#1} },
3273   silent .bool_set:N = \__spinttent_spinterval_silent_bool,
3274   silent .initial:n = false,
3275   silent .value_required:n = true,
3276   concept .bool_set:N = \__spinttent_spinterval_concept_bool,
3277   concept .initial:n = true,
3278   concept .value_required:n = true,
3279   unknown .code:n =
3280     {
3281       \str_set:Nn \__spinttent_command_name_str { spinterval }
3282       \__spinttent_unknown_keys_cmd:n {#1}
3283     },
3284 }

```

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

15.38.4 Internal functions for `\spinterval`

`\__spinttent_spinterval_set_reason:`

The internal function `\__spinttent_spinterval_set_reason:` stores in `\__spinttent_spinterval_reason_str` the text that describes the error code returned by the Lua module, or `invalid` argument when the code is not known.

```

3285 \cs_new_protected:Nn \__spinttent_spinterval_set_reason:
3286 {
3287   \str_case:VnF \__spinttent_spinterval_luaset_error_code_str
3288   {
3289     { delim }
3290     {
3291       \str_set:Nn \__spinttent_spinterval_reason_str
3292       { it~must~start~with~'(' ,~'['~or~'{'~and~end~with~')' ,~'}'~or~']' }
3293     }
3294     { vacio }
3295     { \str_set:Nn \__spinttent_spinterval_reason_str { an~endpoint~is~empty } }
3296     { sep }
3297     {
3298       \str_set:Nn \__spinttent_spinterval_reason_str
3299       { there~is~no~separator:~use~a~comma~or~a~semicolon }
3300     }
3301     { ambiguo }
3302     {
3303       \str_set:Nn \__spinttent_spinterval_reason_str
3304       {
3305         more~than~one~separator:~use~a~single~semicolon~and~the~commas~are~
3306         decimal
3307       }
3308     }
3309     { llaves }
3310     { \str_set:Nn \__spinttent_spinterval_reason_str { unbalanced~braces } }
3311     { inf-orden }
3312     {
3313       \str_set:Nn \__spinttent_spinterval_reason_str
3314       { minus~infinity~goes~on~the~left~and~infinity~on~the~right }
3315     }
3316     { inf-cerrado }
3317     {
3318       \str_set:Nn \__spinttent_spinterval_reason_str
3319       {
3320         an~infinite~endpoint~cannot~be~included:~use~'('~or~'{'~on~the~left,~
3321         '~or~'['~on~the~right
3322       }
3323     }
3324   }
3325   { \str_set:Nn \__spinttent_spinterval_reason_str { invalid~argument } }
3326 }

```

(End of definition for `\__spinttent_spinterval_set_reason:`)

`\__spinttent_spinterval_set_delims:`

The internal function `\__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.

```

3327 \cs_new_protected:Nn \__spinttent_spinterval_set_delims:
3328 {
3329   \str_case:Vn \__spinttent_spinterval_luaset_left_sym_str
3330   {
3331     { ( ) } { \tl_set:Nn \__spinttent_spinterval_sym_l_tl { \lparen } }
3332     { [ ] } { \tl_set:Nn \__spinttent_spinterval_sym_l_tl { \lbrack } }
3333     { } ] } { \tl_set:Nn \__spinttent_spinterval_sym_l_tl { \rbrack } }
3334   }
3335   \str_case:Vn \__spinttent_spinterval_luaset_right_sym_str
3336   {
3337     { ) } { \tl_set:Nn \__spinttent_spinterval_sym_r_tl { \rparen } }
3338     { [ ] } { \tl_set:Nn \__spinttent_spinterval_sym_r_tl { \lbrack } }

```

```

3339     { } } { \tl_set:Nn \l__spinttent_spinterval_sym_r_tl { \rbrack } }
3340   }
3341   \tl_clear:N \l__spinttent_spinterval_size_l_tl
3342   \tl_clear:N \l__spinttent_spinterval_size_r_tl
3343   \str_case:Vn \l__spinttent_spinterval_size_str
3344   {
3345     { big }
3346     {
3347       \tl_set:Nn \l__spinttent_spinterval_size_l_tl { \bigl }
3348       \tl_set:Nn \l__spinttent_spinterval_size_r_tl { \bigr }
3349     }
3350     { Big }
3351     {
3352       \tl_set:Nn \l__spinttent_spinterval_size_l_tl { \Bigl }
3353       \tl_set:Nn \l__spinttent_spinterval_size_r_tl { \Bigr }
3354     }
3355     { bigg }
3356     {
3357       \tl_set:Nn \l__spinttent_spinterval_size_l_tl { \biggl }
3358       \tl_set:Nn \l__spinttent_spinterval_size_r_tl { \biggr }
3359     }
3360     { Bigg }
3361     {
3362       \tl_set:Nn \l__spinttent_spinterval_size_l_tl { \Biggl }
3363       \tl_set:Nn \l__spinttent_spinterval_size_r_tl { \Biggr }
3364     }
3365     { auto }
3366     {
3367       \str_if_eq:VnF \l__spinttent_spinterval_luaset_tall_str { none }
3368       {
3369         \int_case:nnF { \mathstyle }
3370         {
3371           { 0 }
3372           {
3373             \tl_set:Nn \l__spinttent_spinterval_size_l_tl { \Bigl }
3374             \tl_set:Nn \l__spinttent_spinterval_size_r_tl { \Bigr }
3375           }
3376           { 1 }
3377           {
3378             \tl_set:Nn \l__spinttent_spinterval_size_l_tl { \Bigl }
3379             \tl_set:Nn \l__spinttent_spinterval_size_r_tl { \Bigr }
3380           }
3381           { 2 }
3382           {
3383             \tl_set:Nn \l__spinttent_spinterval_size_l_tl { \bigl }
3384             \tl_set:Nn \l__spinttent_spinterval_size_r_tl { \bigr }
3385           }
3386           { 3 }
3387           {
3388             \tl_set:Nn \l__spinttent_spinterval_size_l_tl { \bigl }
3389             \tl_set:Nn \l__spinttent_spinterval_size_r_tl { \bigr }
3390           }
3391         }
3392       }
3393       \str_if_eq:VnT \l__spinttent_spinterval_luaset_tall_str { dfraction }
3394       {
3395         \tl_set:Nn \l__spinttent_spinterval_size_l_tl { \Bigl }
3396         \tl_set:Nn \l__spinttent_spinterval_size_r_tl { \Bigr }
3397       }
3398     }
3399   }
3400 }
3401 \tl_if_empty:NTF \l__spinttent_spinterval_size_l_tl
3402 {
3403   \tl_set:Nn \l__spinttent_spinterval_delim_l_tl
3404   { \exp_not:N \mathopen { \exp_not:V \l__spinttent_spinterval_sym_l_tl } }
3405   \tl_set:Nn \l__spinttent_spinterval_delim_r_tl
3406   { \exp_not:N \mathclose { \exp_not:V \l__spinttent_spinterval_sym_r_tl } }
3407 }
3408 {
3409   \tl_set:Nn \l__spinttent_spinterval_delim_l_tl

```

```

3410     {
3411         \exp_not:V \l__spinttent_spinterval_size_l_tl \exp_not:V
3412         \l__spinttent_spinterval_sym_l_tl
3413     }
3414     \tl_set:Ne \l__spinttent_spinterval_delim_r_tl
3415     {
3416         \exp_not:V \l__spinttent_spinterval_size_r_tl \exp_not:V
3417         \l__spinttent_spinterval_sym_r_tl
3418     }
3419 }
3420 }

```

(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`.

```

3421 \cs_new_protected:Nn \l__spinttent_spinterval_set_intents:
3422 {
3423     \str_case:Nn \l__spinttent_spinterval_luaset_type_str
3424     {
3425         { open }
3426         {
3427             \tl_set:Nn \l__spinttent_spinterval_word_w_tl { abierto-de }
3428             \tl_set:Nn \l__spinttent_spinterval_word_g_tl { a }
3429         }
3430         { closed }
3431         {
3432             \tl_set:Nn \l__spinttent_spinterval_word_w_tl { cerrado-de }
3433             \tl_set:Nn \l__spinttent_spinterval_word_g_tl { a }
3434         }
3435         { open-closed }
3436         {
3437             \tl_set:Nn \l__spinttent_spinterval_word_w_tl { abierto-en }
3438             \tl_set:Nn \l__spinttent_spinterval_word_g_tl { y-cerrado-en }
3439         }
3440         { closed-open }
3441         {
3442             \tl_set:Nn \l__spinttent_spinterval_word_w_tl { cerrado-en }
3443             \tl_set:Nn \l__spinttent_spinterval_word_g_tl { y-abierto-en }
3444         }
3445     }
3446     \tl_set:Ne \l__spinttent_spinterval_intent_group_tl
3447     { _ \l__spinttent_spinterval_word_g_tl :infix($a,$b) }
3448     \bool_if:NTF \l__spinttent_spinterval_silent_bool
3449     { \tl_set:Nn \l__spinttent_spinterval_intent_arg_tl { _- } }
3450     {
3451         \tl_if_empty:NTF \l__spinttent_spinterval_read_arg_tl
3452         {
3453             \tl_set:Ne \l__spinttent_spinterval_intent_arg_tl
3454             {
3455                 -
3456                 \bool_if:NT \l__spinttent_spinterval_concept_bool
3457                 {
3458                     \tl_if_empty:NF \l__spinttent_spinterval_prefix_tl
3459                     { \tl_use:N \l__spinttent_spinterval_prefix_tl - }
3460                     intervalo-
3461                 }
3462                 \l__spinttent_spinterval_word_w_tl :prefix($ab)
3463             }
3464         }
3465         {
3466             \tl_set:Ne \l__spinttent_spinterval_intent_arg_tl
3467             { _ \l__spinttent_spinterval_read_arg_tl }
3468         }
3469     }
3470 }

```

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

`\__spinttent_spinterval_inner_sep:n`

The internal function `\__spinttent_spinterval_inner_sep:n` calls `\__spinttent_inner_sep:` when the endpoint a or b passed in ‘#1’ is composite.

```

3471 \cs_new_protected:Nn \__spinttent_spinterval_inner_sep:n
3472 {
3473   \str_if_eq:eeT
3474     { \str_use:c { l__spinttent_spinterval_luaset_ #1 _composite_str } } { true }
3475     { \__spinttent_inner_sep: }
3476 }

```

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

`\__spinttent_spinterval_plain_number:n`

The internal function `\__spinttent_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.

```

3477 \cs_new_protected:Nn \__spinttent_spinterval_plain_number:n
3478 {
3479   \str_if_eq:eeTF { \str_use:c { l__spinttent_spinterval_luaset_ #1 _integer_str } }
3480     { true }
3481     % keep "a,b" apart in mathml-AF
3482     { \MathMLintent { :silent } { \c__spinttent_invisible_sep_int } }
3483     { \__spinttent_inner_sep: }
3484   \str_use:c { l__spinttent_spinterval_luaset_ #1 _braced_str }
3485 }

```

(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.

```

3486 \cs_new_protected:Nn \__spinttent_spinterval_value:n
3487 {
3488   \str_case:enF { \str_use:c { l__spinttent_spinterval_luaset_ #1 _type_str } }
3489     {
3490       { infinity } { \infty }
3491       { number }
3492       {
3493         \str_if_eq:VnTF \l__spinttent_spinterval_mode_str { school }
3494           {
3495             \tl_set:Nc \l__spinttent_spinterval_tmp_tl
3496               { \str_use:c { l__spinttent_spinterval_luaset_ #1 _value_str } }
3497             \str_if_eq:eeTF
3498               { \str_use:c { l__spinttent_spinterval_luaset_ #1 _short_str } }
3499               { true }
3500               { \__spinttent_spinterval_plain_number:n {#1} }
3501             {
3502               \__spinttent_inner_sep:
3503               \exp_args:NV \spnum \l__spinttent_spinterval_tmp_tl
3504             }
3505           }
3506         { \__spinttent_spinterval_plain_number:n {#1} }
3507       }
3508     }
3509   { \str_use:c { l__spinttent_spinterval_luaset_ #1 _value_str } }
3510 }

```

(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.

```

3511 \cs_new_protected:Nn \__spinttent_spinterval_print:n

```

```

3512 {
3513   \str_if_eq:VnTF \l__spinttent_spinterval_mode_str { school }
3514   {
3515     \str_case:enF { \str_use:c { l__spinttent_spinterval_luaset_ #1 _sign_str } }
3516     {
3517       { + }
3518       {
3519         \MathMLintent { _más:prefix($vp) }
3520         {
3521           +
3522           \MathMLarg { vp }
3523           {
3524             \__spinttent_spinterval_inner_sep:n {#1}
3525             \__spinttent_spinterval_value:n {#1}
3526           }
3527         }
3528       }
3529     { - }
3530     {
3531       \MathMLintent { _menos:prefix($vn) }
3532       {
3533         -
3534         \MathMLarg { vn }
3535         {
3536           \__spinttent_spinterval_inner_sep:n {#1}
3537           \__spinttent_spinterval_value:n {#1}
3538         }
3539       }
3540     }
3541   }
3542   { \__spinttent_spinterval_value:n {#1} }
3543 }
3544 {
3545   \str_use:c { l__spinttent_spinterval_luaset_ #1 _sign_str }
3546   \__spinttent_spinterval_value:n {#1}
3547 }
3548 }

```

(End of definition for __spinttent_spinterval_print:n.)

__spinttent_spinterval_mathml: The internal function `\__spinttent_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 `\__spinttent_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 `\__spinttent_spinterval_print:n`, separated by the silenced separator.

```

3549 \cs_new_protected:Nn \__spinttent_spinterval_mathml:
3550 {
3551   \MathMLintent { \l__spinttent_spinterval_intent_arg_tl }
3552   {
3553     \MathMLintent { :silent } { \l__spinttent_spinterval_delim_l_tl }
3554     \MathMLarg { ab }
3555     {
3556       \__spinttent_inner_sep:
3557       \MathMLintent { \l__spinttent_spinterval_intent_group_tl }
3558       {
3559         \MathMLarg { a }
3560         {
3561           \__spinttent_spinterval_inner_sep:n { a }
3562           \__spinttent_spinterval_print:n { a }
3563         }
3564         \MathMLintent { :silent } { \l__spinttent_spinterval_luaset_sep_str }
3565         \MathMLarg { b }
3566         {
3567           \__spinttent_spinterval_inner_sep:n { b }
3568           \__spinttent_spinterval_print:n { b }
3569         }
3570       }
3571     }
3572     \MathMLintent { :silent } { \l__spinttent_spinterval_delim_r_tl }
3573   }
3574 }

```

(End of definition for `\__spinttent_spinterval_mathml:`)

`\__spinttent_spinterval_exec:nn` The internal function `\__spinttent_spinterval_exec:nn` receives the optional $\langle keys \rangle$ passed in ‘#1’ and the $\langle argument \rangle$ passed in ‘#2’. It calls `\__spinttent_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 $\langle argument \rangle$ is always rejected. When there is no error, it calls `\__spinttent_spinterval_set_delims:` and `\__spinttent_spinterval_set_intents:`, places `\__spinttent_invisible_sep:` and delegates the rendering to `\__spinttent_spinterval_mathml:`.

```

3575 \cs_new_protected:Nn \__spinttent_spinterval_exec:nn
3576 {
3577   \keys_set:nn { spinttent / spinterval } {#1}
3578   \__spinttent_luafun_spinterval_parse_and_set:n { \exp_not:n {#2} }
3579   \str_if_eq:VnTF \__spinttent_spinterval_luaerror_str { true }
3580   {
3581     \__spinttent_spinterval_set_reason:
3582     \msg_error:nnee { spinttent } { spinterval-invalid-arg }
3583     { \exp_not:n {#2} } { \__spinttent_spinterval_reason_str }
3584   }
3585   {
3586     \__spinttent_spinterval_set_delims:
3587     \__spinttent_spinterval_set_intents:
3588     \__spinttent_invisible_sep:
3589     \__spinttent_spinterval_mathml:
3590   }
3591 }

```

(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`.

```

3592 \NewDocumentCommand \spinterval { 0{} m }
3593 {
3594   \mode_if_math:TF
3595   {
3596     \group_begin:
3597     \attribute \__spinttent_scope_attr_int = 1
3598     \__spinttent_spinterval_exec:nn {#1} {#2}
3599     \group_end:
3600   }
3601   { \msg_error:nnn { spinttent } { need-math-mode } { \spinterval } }
3602 }

```

(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 $\langle argument \rangle$ into the function name, which is a letter with an optional subscript or primes left in the shared `\__spinttent_geo_luaerror_*` variables, and the list of arguments, cut at the top-level commas, each one stored in `\__spinttent_spfun_luaerror_arg<i>_tl` together with a flag `\__spinttent_spfun_luaerror_simple<i>_str` telling whether it is a single node. The whole call is tagged with a native `\MathMLintent` whose `intent` is `$f($a1,$a2,...)`; the name is tagged `arg="f"` and each argument `arg="a<i>"`, while the parentheses and the commas are silenced. The name may carry the inverse exponent `^-1`, which sets the flag `\__spinttent_spfun_luaerror_inv_str`; in that case the literal `intent` of the name ends in `-inversa`, the exponent is typeset silenced and the parentheses may be omitted, with no arguments and `$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`

The variable `\__spinttent_spfun_mode_str` stores the value set by the key `read-cmd` and `\__spinttent_spfun_prefix_tl` the one set by `prefix`. The boolean variables `\__spinttent_spfun_silent_bool` and `\__spinttent_spfun_concept_bool` store the keys `silent` and `concept`. The variable `\__spinttent_spfun_call_tl` stores the `$f($a1,...)` text of the `intent` and `\__spinttent_spfun_depth_int` the nesting level of `\__spinttent_spfun_nested:n`. The three `luaerror` variables are set from Lua.

```

3603 \str_new:N \__spinttent_spfun_mode_str
3604 \tl_new:N \__spinttent_spfun_prefix_tl

```

©2026 by Pablo González L

```

3605 \tl_new:N \l__spinttent_sfun_intent_arg_tl
3606 \tl_new:N \l__spinttent_sfun_kv_tl
3607 \tl_new:N \l__spinttent_sfun_refs_tl
3608 \str_new:N \l__spinttent_sfun_tmp_str
3609 \int_new:N \l__spinttent_sfun_nargs_int
3610 \bool_new:N \l__spinttent_sfun_silent_bool
3611 \bool_new:N \l__spinttent_sfun_concept_bool
3612 \str_new:N \l__spinttent_sfun_luaset_error_str
3613 \str_new:N \l__spinttent_sfun_luaset_nargs_str
3614 \str_new:N \l__spinttent_sfun_luaset_inv_str
3615 \tl_new:N \l__spinttent_sfun_call_tl
3616 \int_new:N \l__spinttent_sfun_depth_int

```

(End of definition for `\l__spinttent_sfun_mode_str` 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`.

```

3617 \msg_new:nnnn { spinttent } { spfun-invalid-arg }
3618 {
3619   Invalid~argument~'#1'~for~'\token_to_str:N \spfun'\msg_line_context:.
3620 }
3621 {
3622   The~argument~must~have~the~form~'f(x)'\or~'f(x,y,...)',~where~'f'~is~a~letter~with~
3623   optional~primes~or~subscript.
3624 }

```

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

15.39.3 Definition of keys for `\spfun`

`read-cmd`

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

`prefix`

`concept`

`silent`

`unknown`

```

3625 \keys_define:nn { spinttent / spfun }
3626 {
3627   read-cmd .choices:nn = { school , mathcat }
3628   {
3629     \int_case:nn { \l_keys_choice_int }
3630     {
3631       {1} { \str_set:Nn \l__spinttent_sfun_mode_str { school } }
3632       {2} { \str_set:Nn \l__spinttent_sfun_mode_str { mathcat } }
3633     }
3634   },
3635   read-cmd .initial:n = school,
3636   read-cmd .value_required:n = true,
3637   read-cmd / unknown .code:n =
3638     \msg_error:nnnee { spinttent } { unknown-choice }
3639     { read-cmd } { school,~mathcat } { \exp_not:n {#1} },
3640   prefix .code:n =
3641     { \__spinttent_sanitizize_affix:Nn \l__spinttent_sfun_prefix_tl {#1} },
3642   prefix .initial:n = ,
3643   prefix .value_required:n = true,
3644   silent .bool_set:N = \l__spinttent_sfun_silent_bool,
3645   silent .initial:n = false,
3646   silent .value_required:n = true,
3647   concept .bool_set:N = \l__spinttent_sfun_concept_bool,
3648   concept .initial:n = false,
3649   concept .value_required:n = true,
3650   unknown .code:n =
3651     {
3652       \str_set:Nn \l__spinttent_command_name_str { spfun }
3653       \__spinttent_unknown_keys_cmd:n {#1}
3654     },
3655 }

```

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

15.39.4 Internal functions for `\spfun`

`\__spinttent_sfun_fname:`

Names of the `arg` attributes at the current nesting depth `\l__spinttent_sfun_depth_int`. At depth 1 they are `f` and `a<i>`; at a deeper level, `f<d>` and `a<d>x<i>`.

```

3656 \cs_new:Nn \__spinttent_sfun_fname:

```

```

3657 {
3658   f
3659   \int_compare:nNnF { \l__spinttent_spfun_depth_int } = { 1 }
3660   { \int_use:N \l__spinttent_spfun_depth_int }
3661 }
3662 \cs_new:Nn \l__spinttent_spfun_aname:n
3663 {
3664   a
3665   \int_compare:nNnF { \l__spinttent_spfun_depth_int } = { 1 }
3666   { \int_use:N \l__spinttent_spfun_depth_int x }
3667   #1
3668 }

```

(End of definition for `\l__spinttent_spfun_fname:` and `\l__spinttent_spfun_aname:n`.)

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

```

3669 \cs_new_protected:Nn \l__spinttent_spfun_name_plain:
3670 {
3671   \l__spinttent_geo_luaset_letra_str
3672   \str_case:VnF \l__spinttent_geo_luaset_tipo_str
3673   {
3674     { sub } { \c_math_subscript_token { \l__spinttent_geo_luaset_sub_str } }
3675     { prima } { \l__spinttent_geo_luaset_sub_str }
3676   }
3677   { }
3678   \str_if_eq:VnT \l__spinttent_spfun_luaset_inv_str { true }
3679   { \c_math_superscript_token { -1 } }
3680 }

```

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

`\l__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.

```

3681 \cs_new_protected:Nn \l__spinttent_spfun_name_school:
3682 {
3683   \tl_set:Ne \l__spinttent_spfun_kv_tl
3684   {
3685     arg = " \l__spinttent_spfun_fname: ", intent =
3686     "_ \l__spinttent_geo_luaset_intent_str -"
3687   }
3688   \l__spinttent_luafun_mrow_intent_arg:nn { } { \l__spinttent_spfun_kv_tl }
3689   \str_case:VnF \l__spinttent_geo_luaset_tipo_str
3690   {
3691     { sub }
3692     {
3693       \l__spinttent_geo_luaset_letra_str
3694       \c_math_subscript_token
3695       { \MathMLintent { :silent } { \l__spinttent_geo_luaset_sub_str } }
3696     }
3697     { prima }
3698     { \l__spinttent_geo_luaset_letra_str \l__spinttent_geo_luaset_sub_str }
3699   }
3700   { \l__spinttent_geo_luaset_letra_str }
3701   \str_if_eq:VnT \l__spinttent_spfun_luaset_inv_str { true }
3702   {
3703     \c_math_superscript_token
3704     { \MathMLintent { :silent } { -1 } }
3705   }
3706 }

```

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

`\l__spinttent_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>`.

```

3707 \cs_new_protected:Nn \l__spinttent_spfun_name_mathcat:

```

```

3708 {
3709   \bool_lazy_and:nnTF
3710   { \str_if_eq_p:Vn \l__spinttent_geo_luaset_tipo_str { } }
3711   { \str_if_eq_p:Vn \l__spinttent_sfun_luaset_inv_str { false } }
3712   { \MathMLarg { \l__spinttent_sfun_fname: } { \l__spinttent_sfun_name_plain: } }
3713   {
3714     \MathMLarg { \l__spinttent_sfun_fname: }
3715     { \l__spinttent_inner_sep: \l__spinttent_sfun_name_plain: }
3716   }
3717 }

```

(End of definition for `\l__spinttent_sfun_name_mathcat:`)

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

```

3718 \cs_new_protected:Nn \l__spinttent_sfun_arg:n
3719 {
3720   \str_set_eq:Nc \l__spinttent_sfun_tmp_str
3721   { \l__spinttent_sfun_luaset_simple_ #1 _str }
3722   \str_case:VnF \l__spinttent_sfun_tmp_str
3723   {
3724     { true }
3725     {
3726       \MathMLarg { \l__spinttent_sfun_aname:n {#1} }
3727       { \tl_use:c { \l__spinttent_sfun_luaset_arg_ #1 _tl } }
3728     }
3729     { call }
3730     {
3731       \MathMLarg { \l__spinttent_sfun_aname:n {#1} }
3732       { \l__spinttent_inner_sep: \l__spinttent_sfun_nested:n {#1} }
3733     }
3734   }
3735   {
3736     \MathMLarg { \l__spinttent_sfun_aname:n {#1} }
3737     { \l__spinttent_inner_sep: \tl_use:c { \l__spinttent_sfun_luaset_arg_ #1 _tl } }
3738   }
3739 }

```

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

`\l__spinttent_sfun_args:` Walks the arguments, silencing the comma between them.

```

3740 \cs_new_protected:Nn \l__spinttent_sfun_args:
3741 {
3742   \int_step_inline:nn { \l__spinttent_sfun_nargs_int }
3743   {
3744     \int_compare:nNnF { ##1 } = { 1 }
3745     { \MathMLintent { :silent } { , } }
3746     \l__spinttent_sfun_arg:n {##1}
3747   }
3748 }

```

(End of definition for `\l__spinttent_sfun_args:`)

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

```

3749 \cs_new_protected:Nn \l__spinttent_sfun_args_plain:
3750 {
3751   \int_step_inline:nn { \l__spinttent_sfun_nargs_int }
3752   {
3753     \int_compare:nNnF { ##1 } = { 1 } { , }
3754     \tl_use:c { \l__spinttent_sfun_luaset_arg_ ##1 _tl }
3755   }
3756 }

```

(End of definition for `\l__spinttent_sfun_args_plain:`)

`\l__spinttent_sfun_set_intent:` Builds the concept `intent` in `\l__spinttent_sfun_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`.

```

3757 \cs_new_protected:Nn \l__spinttent_sfun_set_intent:

```

```

3758 {
3759   \tl_clear:N \l__spinttent_spfun_refs_tl
3760   \int_step_inline:nn { \l__spinttent_spfun_nargs_int }
3761   {
3762     \tl_put_right:Ne \l__spinttent_spfun_refs_tl
3763     { \int_compare:nNnF { ##1 } = { 1 } { , } $ \l__spinttent_spfun_aname:n {##1} }
3764   }
3765   \int_compare:nNnTF { \l__spinttent_spfun_nargs_int } = { 0 }
3766   { \tl_set:Ne \l__spinttent_spfun_call_tl { $ \l__spinttent_spfun_fname: } }
3767   {
3768     \tl_set:Ne \l__spinttent_spfun_call_tl
3769     { $ \l__spinttent_spfun_fname: ( \l__spinttent_spfun_refs_tl ) }
3770   }
3771   \bool_if:NTF \l__spinttent_spfun_concept_bool
3772   {
3773     \tl_set:Ne \l__spinttent_spfun_intent_arg_tl
3774     {
3775       -
3776       \tl_if_empty:NF \l__spinttent_spfun_prefix_tl
3777       { \tl_use:N \l__spinttent_spfun_prefix_tl - }
3778       función:prefix( \l__spinttent_spfun_call_tl )
3779     }
3780   }
3781   {
3782     \tl_set_eq:NN \l__spinttent_spfun_intent_arg_tl \l__spinttent_spfun_call_tl
3783   }
3784 }

```

(End of definition for `\l__spinttent_spfun_set_intent:`)

`\l__spinttent_spfun_mathml:` Calls `\l__spinttent_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.

```

3785 \cs_new_protected:Nn \l__spinttent_spfun_mathml:
3786 {
3787   \l__spinttent_invisible_sep:
3788   \l__spinttent_spfun_body:
3789 }

```

(End of definition for `\l__spinttent_spfun_mathml:`)

`\l__spinttent_spfun_body:` The call itself, without the outer separator: used by the top-level call and by every call nested in an argument.

```

3790 \cs_new_protected:Nn \l__spinttent_spfun_body:
3791 {
3792   \bool_if:NTF \l__spinttent_spfun_silent_bool
3793   {
3794     \MathMLintent { _- }
3795     {
3796       \l__spinttent_inner_sep:
3797       \l__spinttent_spfun_name_plain:
3798       \int_compare:nNnF { \l__spinttent_spfun_nargs_int } = { 0 }
3799       {
3800         \lparen
3801         \l__spinttent_spfun_args_plain:
3802         \rparen
3803       }
3804     }
3805   }
3806   {
3807     \l__spinttent_spfun_set_intent:
3808     \MathMLintent { \l__spinttent_spfun_intent_arg_tl }
3809     {
3810       \int_compare:nNnT { \l__spinttent_spfun_nargs_int } = { 0 }
3811       { \l__spinttent_inner_sep: }
3812       \str_if_eq:VnTF \l__spinttent_spfun_mode_str { school }
3813       { \l__spinttent_spfun_name_school: }
3814       { \l__spinttent_spfun_name_mathcat: }
3815       \int_compare:nNnF { \l__spinttent_spfun_nargs_int } = { 0 }
3816       {
3817         \MathMLintent { :silent } { \lparen }
3818         \l__spinttent_spfun_args:

```

```

3819         \MathMLintent { :silent } { \rparen }
3820     }
3821 }
3822 }
3823 }

```

(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.

```

3824 \cs_new_protected:Nn \__spinttent_spfun_nested:n
3825 {
3826     \group_begin:
3827     \__spinttent_luafun_spfun_parse_and_set:v { \__spinttent_spfun_luaaset_arg_ #1 _tl }
3828     \int_incr:N \__spinttent_spfun_depth_int
3829     \bool_set_false:N \__spinttent_spfun_concept_bool
3830     \int_set:NV \__spinttent_spfun_nargs_int \__spinttent_spfun_luaaset_nargs_str
3831     \__spinttent_spfun_body:
3832     \group_end:
3833 }

```

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

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

```

3834 \cs_new_protected:Nn \__spinttent_spfun_exec:nn
3835 {
3836     \keys_set:nn { spinttent / spfun } {#1}
3837     \__spinttent_luafun_spfun_parse_and_set:n { \exp_not:n {#2} }
3838     \str_if_eq:VnTF \__spinttent_spfun_luaaset_error_str { true }
3839     {
3840         \msg_error:nne { spinttent } { spfun-invalid-arg } { \exp_not:n {#2} }
3841     }
3842     {
3843         \int_set:NV \__spinttent_spfun_nargs_int \__spinttent_spfun_luaaset_nargs_str
3844         \int_set:Nn \__spinttent_spfun_depth_int { 1 }
3845         \__spinttent_spfun_mathml:
3846     }
3847 }

```

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

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

```

3848 \NewDocumentCommand \spfun { O{} m }
3849 {
3850     \mode_if_math:TF
3851     {
3852         \group_begin:
3853         \attribute \__spinttent_scope_attr_int = 1
3854         \__spinttent_spfun_exec:nn {#1} {#2}
3855         \group_end:
3856     }
3857     { \msg_error:nnn { spinttent } { need-math-mode } { \spfun } }
3858 }

```

(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 `\__spinttent_luafun_spfuncomp_parse_and_set:n` splits the `{<argument>}` at the top-level `\circ`, or the Unicode character U+2218; every function is a name as in `\spfun` and gets its literal `intent`, so `MathCAT` neither announces a capital letter nor reads the letter `h`, U+210E, as Planck’s constant. The arguments of the application, if any, are left in the `\spfun` variables and typeset with its own functions, so an argument may be a call. The whole is tagged with a native `\MathMLintent` whose `intent` is

`_compuesta-con:infix($k1,$k2,...)`; the application adds `$c($a1,...)` around it, with the composition tagged `arg="c"`. With two or more arguments they are grouped under their own `intent`, `_coma:infix($a1,$a2,...)`, tagged `arg="l"`, because MathCAT drops the commas of a call whose head is not a single letter. The `\circ` and the parentheses are silenced.

15.40.1 Definition of variables for `\spfuncomp`

Number of functions, the references and the `intent` being built, the key-value list and the kind of the function being typeset, and the values that Lua leaves: error flag, number of functions, whether the composition is in parentheses and whether it is applied to arguments.

```

3859 \int_new:N \l__spintent_spfuncomp_n_int
3860 \tl_new:N \l__spintent_spfuncomp_refs_tl
3861 \tl_new:N \l__spintent_spfuncomp_intent_tl
3862 \tl_new:N \l__spintent_spfuncomp_kv_tl
3863 \str_new:N \l__spintent_spfuncomp_tipo_str
3864 \str_new:N \l__spintent_spfuncomp_luaset_error_str
3865 \str_new:N \l__spintent_spfuncomp_luaset_n_str
3866 \str_new:N \l__spintent_spfuncomp_luaset_paren_str
3867 \str_new:N \l__spintent_spfuncomp_luaset_apply_str

```

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

15.40.2 Definition of error messages for `\spfuncomp`

`spfuncomp-invalid-arg`

The “error” `spfuncomp-invalid-arg` is raised when the argument is not a composition of at least two function names, optionally in parentheses and optionally applied to arguments written in parentheses, whatever the value of `silent`.

```

3868 \msg_new:nnnn { spintent } { spfuncomp-invalid-arg }
3869 {
3870   Invalid~argument~'#1'~for~'\token_to_str:N \spfuncomp'~'\msg_line_context:.
3871 }
3872 {
3873   The~argument~must~be~two~or~more~functions~joined~by~'\token_to_str:N \circ'.
3874 }

```

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

15.40.3 Definition of keys for `\spfuncomp`

`silent`
`unknown`

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

```

3875 \keys_define:nn { spintent / spfuncomp }
3876 {
3877   silent .bool_set:N = \l__spintent_spfuncomp_silent_bool,
3878   silent .initial:n = false,
3879   silent .value_required:n = true,
3880   unknown .code:n =
3881     {
3882       \str_set:Nn \l__spintent_command_name_str { spfuncomp }
3883       \__spintent_unknown_keys_cmd:n {#1}
3884     },
3885 }

```

(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.

```

3886 \cs_new_protected:Nn \__spintent_spfuncomp_fun:n
3887 {
3888   \tl_set:Nc \l__spintent_spfuncomp_kv_tl
3889     {
3890       arg = " k #1 ",
3891       intent = "_ \use:c { \l__spintent_spfuncomp_luaset_intent_ #1 _str } -"
3892     }
3893   \__spintent_luafun_mrow_intent_arg:nn { } { \l__spintent_spfuncomp_kv_tl }
3894   \str_set_eq:Nc \l__spintent_spfuncomp_tipo_str
3895     { \l__spintent_spfuncomp_luaset_tipo_ #1 _str }
3896   \str_case:VnF \l__spintent_spfuncomp_tipo_str
3897     {

```

```

3898     { sub }
3899     {
3900       \use:c { l__spintent_sfuncomp_luaset_letra_ #1 _str }
3901       \c_math_subscript_token
3902       {
3903         \MathMLintent { :silent }
3904         { \use:c { l__spintent_sfuncomp_luaset_sub_ #1 _str } }
3905       }
3906     }
3907   { prima }
3908   {
3909     \use:c { l__spintent_sfuncomp_luaset_letra_ #1 _str }
3910     \use:c { l__spintent_sfuncomp_luaset_sub_ #1 _str }
3911   }
3912 }
3913 { \use:c { l__spintent_sfuncomp_luaset_letra_ #1 _str } }
3914 \str_set_eq:Nc \l__spintent_sfuncomp_tipo_str
3915 { l__spintent_sfuncomp_luaset_inv_ #1 _str }
3916 \str_if_eq:VnT \l__spintent_sfuncomp_tipo_str { true }
3917 {
3918   \c_math_superscript_token
3919   { \MathMLintent { :silent } { -1 } }
3920 }
3921 }

```

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

`\__spintent_sfuncomp_group:` The composition, with its own `intent`, in parentheses when the argument was written in parentheses.

```

3922 \cs_new_protected:Nn \__spintent_sfuncomp_group:
3923 {
3924   \tl_clear:N \l__spintent_sfuncomp_refs_tl
3925   \int_step_inline:nn { \l__spintent_sfuncomp_n_int }
3926   {
3927     \tl_put_right:Ne \l__spintent_sfuncomp_refs_tl
3928     { \int_compare:nNnF { ##1 } = { 1 } { , } $k ##1 }
3929   }
3930   \tl_set:Ne \l__spintent_sfuncomp_intent_tl
3931   { _compuesta-con:infix( \l__spintent_sfuncomp_refs_tl ) }
3932   \MathMLintent { \l__spintent_sfuncomp_intent_tl }
3933   {
3934     \str_if_eq:VnT \l__spintent_sfuncomp_luaset_paren_str { true }
3935     { \MathMLintent { :silent } { \lparen } }
3936     \int_step_inline:nn { \l__spintent_sfuncomp_n_int }
3937     {
3938       \int_compare:nNnF { ##1 } = { 1 }
3939       { \MathMLintent { :silent } { \circ } }
3940       \__spintent_sfuncomp_fun:n { ##1 }
3941     }
3942     \str_if_eq:VnT \l__spintent_sfuncomp_luaset_paren_str { true }
3943     { \MathMLintent { :silent } { \rparen } }
3944   }
3945 }

```

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

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

```

3946 \cs_new_protected:Nn \__spintent_sfuncomp_plain:
3947 {
3948   \str_if_eq:VnT \l__spintent_sfuncomp_luaset_paren_str { true } { \lparen }
3949   \int_step_inline:nn { \l__spintent_sfuncomp_n_int }
3950   {
3951     \int_compare:nNnF { ##1 } = { 1 } { \circ }
3952     \tl_use:c { l__spintent_sfuncomp_luaset_print_ ##1 _tl }
3953   }
3954   \str_if_eq:VnT \l__spintent_sfuncomp_luaset_paren_str { true } { \rparen }
3955   \str_if_eq:VnT \l__spintent_sfuncomp_luaset_apply_str { true }
3956   {
3957     \lparen \__spintent_sfun_args_plain: \rparen
3958   }
3959 }

```

(End of definition for `\__spintent_sfuncomp_plain:`)

`\__spintent_sfuncomp_mathml:` Calls `\__spintent_invisible_sep:` first. With `silent` the `intent` is ‘`-`’. Otherwise an application wraps the composition with `arg="c"`, and the arguments in `$c($a1,...)`; the arguments are processed as in `\spfun`, including calls nested in them.

```

3960 \cs_new_protected:Nn \__spintent_sfuncomp_mathml:
3961 {
3962   \__spintent_invisible_sep:
3963   \str_if_eq:VnT \l__spintent_sfuncomp_luaset_apply_str { true }
3964   {
3965     \int_set:NV \l__spintent_sfun_nargs_int \l__spintent_sfun_luaset_nargs_str
3966   }
3967   \int_set:Nn \l__spintent_sfun_depth_int { 1 }
3968   \str_set:Nn \l__spintent_sfun_mode_str { school }
3969   \bool_set_false:N \l__spintent_sfun_silent_bool
3970   \bool_set_false:N \l__spintent_sfun_concept_bool
3971   \bool_if:NTF \l__spintent_sfuncomp_silent_bool
3972   {
3973     \MathMLintent { - }
3974     { \__spintent_inner_sep: \__spintent_sfuncomp_plain: }
3975   }
3976   {
3977     \str_if_eq:VnTF \l__spintent_sfuncomp_luaset_apply_str { true }
3978     {
3979       \tl_clear:N \l__spintent_sfun_refs_tl
3980       \int_step_inline:nn { \l__spintent_sfun_nargs_int }
3981       {
3982         \tl_put_right:Ne \l__spintent_sfun_refs_tl
3983         {
3984           \int_compare:nNnF { ##1 } = { 1 } { , } $ \__spintent_sfun_aname:n
3985           {##1}
3986         }
3987       }
3988       \int_compare:nNnTF { \l__spintent_sfun_nargs_int } > { 1 }
3989       {
3990         \tl_set:Ne \l__spintent_sfun_call_tl { $c ( $l ) }
3991         \MathMLintent { \l__spintent_sfun_call_tl }
3992         {
3993           \MathMLarg { c }
3994           { \__spintent_inner_sep: \__spintent_sfuncomp_group: }
3995           \MathMLintent { :silent } { \lparen }
3996           \MathMLarg { l }
3997           {
3998             \__spintent_inner_sep:
3999             \MathMLintent
4000             { _coma:infix ( \l__spintent_sfun_refs_tl ) }
4001             { \__spintent_sfun_args: }
4002           }
4003           \MathMLintent { :silent } { \rparen }
4004         }
4005       }
4006       {
4007         \tl_set:Ne \l__spintent_sfun_call_tl
4008         { $c ( \l__spintent_sfun_refs_tl ) }
4009         \MathMLintent { \l__spintent_sfun_call_tl }
4010         {
4011           \MathMLarg { c }
4012           { \__spintent_inner_sep: \__spintent_sfuncomp_group: }
4013           \MathMLintent { :silent } { \lparen }
4014           \__spintent_sfun_args:
4015           \MathMLintent { :silent } { \rparen }
4016         }
4017       }
4018     }
4019     { \__spintent_sfuncomp_group: }
4020   }
4021 }

```

(End of definition for `\__spintent_sfuncomp_mathml:`)

`\__spintent_sfuncomp_exec:nn` Receives the optional `<keys>` in ‘`#1`’ and the `{<argument>}` in ‘`#2`’. Parses first, reporting

`spfuncomp-invalid-arg` on error whatever `silent` is.

```

4022 \cs_new_protected:Nn \__spinttent_spfuncomp_exec:nn
4023 {
4024   \keys_set:nn { spinttent / spfuncomp } {#1}
4025   \__spinttent_luafun_spfuncomp_parse_and_set:n { \exp_not:n {#2} }
4026   \str_if_eq:VnTF \__spinttent_spfuncomp_luaset_error_str { true }
4027   {
4028     \msg_error:nne { spinttent } { spfuncomp-invalid-arg }
4029     { \exp_not:n {#2} }
4030   }
4031   {
4032     \int_set:NV \l__spinttent_spfuncomp_n_int \l__spinttent_spfuncomp_luaset_n_str
4033     \__spinttent_spfuncomp_mathml:
4034   }
4035 }

```

(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`.

```

4036 \NewDocumentCommand \spfuncomp { O{} m }
4037 {
4038   \mode_if_math:TF
4039   {
4040     \group_begin:
4041     \attribute \l__spinttent_scope_attr_int = 1
4042     \__spinttent_spfuncomp_exec:nn {#1} {#2}
4043     \group_end:
4044   }
4045   { \msg_error:nnn { spinttent } { need-math-mode } { \spfuncomp } }
4046 }

```

(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, `\__spinttent_spdom_exec:nnnnn`, which receives the command name, the operator, the spoken word, the *keys* and the *argument*. The Lua function `\__spinttent_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 `^{\text{-}1}`. It leaves it in the shared `\l__spinttent_geo_luaset_*` variables. The whole is tagged with a native `\MathMLintent` whose *intent* is `[_prefix-]dominio-de:prefix($f)`, with *recorrido* or *codominio* in place of *dominio* for the other two commands; the operator, the parentheses are silenced and the name carries `arg="f"` wrapped in its own literal *intent*, such as `_f-`, `_F-doble-prima-` or `_f-inversa-`, so `MathCAT` neither announces a capital letter nor reads the inverse as a power. No `read-cmd` nor `concept` keys exist: the native reading of `Dom(f)` is not useful, and the concept is already the spoken word.

15.41.1 Definition of variables for `\spdom`

```

\l__spinttent_spdom_cmd_str
\l__spinttent_spdom_op_tl
\l__spinttent_spdom_word_tl
\l__spinttent_spdom_prefix_tl
\l__spinttent_spdom_intent_arg_tl
\l__spinttent_spdom_luaset_error_str

```

Command name, operator and word, which each public command sets, the value of the key `prefix`, the *intent* being built and the error flag set from Lua.

```

4047 \str_new:N \l__spinttent_spdom_cmd_str
4048 \tl_new:N \l__spinttent_spdom_op_tl
4049 \tl_new:N \l__spinttent_spdom_word_tl
4050 \tl_new:N \l__spinttent_spdom_prefix_tl
4051 \tl_new:N \l__spinttent_spdom_intent_arg_tl
4052 \str_new:N \l__spinttent_spdom_luaset_error_str

```

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

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`.

```

4053 \msg_new:nnnn { spinttent } { spdom-invalid-arg }
4054 {
4055   Invalid~argument~'#2'~for~'\c_backslash_str #1'~\msg_line_context:.
4056 }
4057 {

```

```

4058     The~argument~must~be~a~function~name.
4059 }

```

(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
4060 \keys_define:nn { spintent / spdom }
4061 {
4062   prefix .code:n =
4063     { \__spintent_sanitise_affix:Nn \__spintent_spdom_prefix_tl {#1} },
4064   prefix .initial:n = ,
4065   prefix .value_required:n = true,
4066   silent .bool_set:N = \__spintent_spdom_silent_bool,
4067   silent .initial:n = false,
4068   silent .value_required:n = true,
4069   unknown .code:n =
4070     {
4071       \str_set_eq:NN \__spintent_command_name_str \__spintent_spdom_cmd_str
4072       \__spintent_unknown_keys_cmd:n {#1}
4073     },
4074 }

```

(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.

```

4075 \cs_new_protected:Nn \__spintent_spdom_name:
4076 {
4077   \MathMLarg { f }
4078   {
4079     \__spintent_inner_sep:
4080     \MathMLintent { _ \__spintent_geo_luaset_intent_str - }
4081     { \__spintent_inner_sep: \__spintent_sfun_name_plain: }
4082   }
4083 }

```

(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.

```

4084 \cs_new_protected:Nn \__spintent_spdom_mathml:
4085 {
4086   \__spintent_invisible_sep:
4087   \bool_if:NTF \__spintent_spdom_silent_bool
4088   {
4089     \MathMLintent { _- }
4090     {
4091       \__spintent_inner_sep:
4092       \operatorname { \__spintent_spdom_op_tl }
4093       \lparen \__spintent_sfun_name_plain: \rparen
4094     }
4095   }
4096   {
4097     \tl_set:Nc \__spintent_spdom_intent_arg_tl
4098     {
4099       -
4100       \tl_if_empty:NF \__spintent_spdom_prefix_tl
4101       { \tl_use:N \__spintent_spdom_prefix_tl - }
4102       \__spintent_spdom_word_tl -de:prefix( $f )
4103     }
4104     \MathMLintent { \__spintent_spdom_intent_arg_tl }
4105     {
4106       \MathMLintent { :silent } { \operatorname { \__spintent_spdom_op_tl } }
4107       \MathMLintent { :silent } { \lparen }
4108       \__spintent_spdom_name:
4109       \MathMLintent { :silent } { \rparen }

```

```

4110     }
4111   }
4112 }

```

(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 $\langle keys \rangle$ in ‘#4’ and the $\{ \langle argument \rangle \}$ in ‘#5’. Parses first, reporting `spdom-invalid-arg` on error whatever `silent` is.

```

4113 \cs_new_protected:Nn \__spintent_spdom_exec:nnnnn
4114 {
4115   \str_set:Nn \l__spintent_spdom_cmd_str {#1}
4116   \tl_set:Nn \l__spintent_spdom_op_tl {#2}
4117   \tl_set:Nn \l__spintent_spdom_word_tl {#3}
4118   \keys_set:nn { spintent / spdom } {#4}
4119   \__spintent_luafun_spdom_parse_and_set:n { \exp_not:n {#5} }
4120   \str_if_eq:VnTF \l__spintent_spdom_luaset_error_str { true }
4121   {
4122     \msg_error:nnee { spintent } { spdom-invalid-arg }
4123     {#1} { \exp_not:n {#5} }
4124   }
4125   { \__spintent_spdom_mathml: }
4126 }

```

(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`

```

4127 \NewDocumentCommand \spdom { O{} m }
4128 {
4129   \mode_if_math:TF
4130   {
4131     \group_begin:
4132     \__spintent_spdom_exec:nnnnn { spdom } { Dom } { dominio } {#1} {#2}
4133     \group_end:
4134   }
4135   { \msg_error:nnn { spintent } { need-math-mode } { \spdom } }
4136 }
4137 \NewDocumentCommand \sprec { O{} m }
4138 {
4139   \mode_if_math:TF
4140   {
4141     \group_begin:
4142     \__spintent_spdom_exec:nnnnn { sprec } { Rec } { recorrido } {#1} {#2}
4143     \group_end:
4144   }
4145   { \msg_error:nnn { spintent } { need-math-mode } { \sprec } }
4146 }
4147 \NewDocumentCommand \spcod { O{} m }
4148 {
4149   \mode_if_math:TF
4150   {
4151     \group_begin:
4152     \__spintent_spdom_exec:nnnnn { spcod } { Cod } { codominio } {#1} {#2}
4153     \group_end:
4154   }
4155   { \msg_error:nnn { spintent } { need-math-mode } { \spcod } }
4156 }

```

(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`.

```

4157 \tl_new:N \l__spintent_spabs_read_arg_tl

```

```

4158 \str_new:N \l__spinttent_spabs_size_str
4159 \str_new:N \l__spinttent_spabs_close_mode_str

```

(End of definition for `\l__spinttent_spabs_read_arg_tl`, `\l__spinttent_spabs_size_str`, and `\l__spinttent_spabs_close_mode_str`.)

```

\l__spinttent_spabs_delim_l_tl
\l__spinttent_spabs_delim_r_tl

```

The opening and closing bars, with their size commands.

```

4160 \tl_new:N \l__spinttent_spabs_delim_l_tl
4161 \tl_new:N \l__spinttent_spabs_delim_r_tl

```

(End of definition for `\l__spinttent_spabs_delim_l_tl` and `\l__spinttent_spabs_delim_r_tl`.)

```

\l__spinttent_spabs_luaset_close_str
\l__spinttent_spabs_close_bool

```

The string variable `\l__spinttent_spabs_luaset_close_str` is set by the function `\__spinttent_luafun_spabs_` and is `true` when the argument, without spaces and without a leading sign, has more than one token; the boolean variable `\l__spinttent_spabs_close_bool` tells whether the closing is read.

```

4162 \str_new:N \l__spinttent_spabs_luaset_close_str
4163 \bool_new:N \l__spinttent_spabs_close_bool

```

(End of definition for `\l__spinttent_spabs_luaset_close_str` and `\l__spinttent_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*.

```

4164 \msg_new:nnnn { spinttent } { spabs-empty-arg }
4165 {
4166   Empty~argument~for~'\token_to_str:N \spabs'\msg_line_context:.
4167 }
4168 {
4169   The~expression~inside~the~absolute~value~is~required.
4170 }

```

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

15.42.3 Definition of keys for `\spabs`

`silent`
`read-arg`
`size`
`close`
`unknown`

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

```

4171 \keys_define:nn { spinttent / spabs }
4172 {
4173   silent .bool_set:N = \l__spinttent_spabs_silent_bool,
4174   silent .initial:n = false,
4175   silent .value_required:n = true,
4176   read-arg .code:n =
4177     { \__spinttent_sanitizize_affix:Nn \l__spinttent_spabs_read_arg_tl {#1} },
4178   read-arg .initial:n = ,
4179   read-arg .value_required:n = true,
4180   size .choices:nn = { none , big , Big , bigg , Bigg }
4181     {
4182       \int_case:nn { \l_keys_choice_int }
4183       {
4184         {1} { \str_set:Nn \l__spinttent_spabs_size_str { none } }
4185         {2} { \str_set:Nn \l__spinttent_spabs_size_str { big } }
4186         {3} { \str_set:Nn \l__spinttent_spabs_size_str { Big } }
4187         {4} { \str_set:Nn \l__spinttent_spabs_size_str { bigg } }
4188         {5} { \str_set:Nn \l__spinttent_spabs_size_str { Bigg } }
4189       }
4190     },
4191   size .initial:n = none,
4192   size .value_required:n = true,
4193   size / unknown .code:n =
4194     \msg_error:nneee { spinttent } { unknown-choice }
4195     { size } { none,~big,~Big,~bigg,~Bigg } { \exp_not:n {#1} },
4196   close .choices:nn = { auto , true , false }
4197     {
4198       \int_case:nn { \l_keys_choice_int }
4199       {
4200         {1} { \str_set:Nn \l__spinttent_spabs_close_mode_str { auto } }
4201         {2} { \str_set:Nn \l__spinttent_spabs_close_mode_str { true } }
4202         {3} { \str_set:Nn \l__spinttent_spabs_close_mode_str { false } }
4203       }
4204     },

```

```

4205     close      .initial:n = auto,
4206     close      .value_required:n = true,
4207     close / unknown .code:n =
4208       \msg_error:nneee { spintent } { unknown-choice }
4209       { close } { auto,~true,~false } { \exp_not:n {#1} },
4210     unknown .code:n =
4211       {
4212         \str_set:Nn \l__spintent_command_name_str { spabs }
4213         \__spintent_unknown_keys_cmd:n {#1}
4214       },
4215   }

```

(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.

```

4216 \cs_new_protected:Nn \__spintent_spabs_set_delims:
4217 {
4218   \str_case:Vn \l__spintent_spabs_size_str
4219   {
4220     { none }
4221     {
4222       \tl_set:Nn \l__spintent_spabs_delim_l_tl { \lvert }
4223       \tl_set:Nn \l__spintent_spabs_delim_r_tl { \rvert }
4224     }
4225     { big }
4226     {
4227       \tl_set:Nn \l__spintent_spabs_delim_l_tl { \bigl \lvert }
4228       \tl_set:Nn \l__spintent_spabs_delim_r_tl { \bigr \rvert }
4229     }
4230     { Big }
4231     {
4232       \tl_set:Nn \l__spintent_spabs_delim_l_tl { \Bigl \lvert }
4233       \tl_set:Nn \l__spintent_spabs_delim_r_tl { \Bigr \rvert }
4234     }
4235     { bigg }
4236     {
4237       \tl_set:Nn \l__spintent_spabs_delim_l_tl { \biggl \lvert }
4238       \tl_set:Nn \l__spintent_spabs_delim_r_tl { \biggr \rvert }
4239     }
4240     { Bigg }
4241     {
4242       \tl_set:Nn \l__spintent_spabs_delim_l_tl { \Biggl \lvert }
4243       \tl_set:Nn \l__spintent_spabs_delim_r_tl { \Biggr \rvert }
4244     }
4245   }
4246 }

```

(End of definition for `\__spintent_spabs_set_delims:`)

`\__spintent_spabs_set_close:n`

The function `\__spintent_spabs_set_close:n` sets `\l__spintent_spabs_close_bool` from `close`: always `true` with `true`, always `false` with `false`, and with `auto true` when Lua reports that the argument ‘`#1`’, without spaces and without a leading `+` or `-`, has more than one token.

```

4247 \cs_new_protected:Nn \__spintent_spabs_set_close:n
4248 {
4249   \bool_set_false:N \l__spintent_spabs_close_bool
4250   \str_case:Vn \l__spintent_spabs_close_mode_str
4251   {
4252     { true } { \bool_set_true:N \l__spintent_spabs_close_bool }
4253     { auto }
4254     {
4255       \__spintent_luafun_spabs_close:n { \exp_not:n {#1} }
4256       \str_if_eq:VnT \l__spintent_spabs_lua_set_close_str { true }
4257       { \bool_set_true:N \l__spintent_spabs_close_bool }
4258     }
4259   }
4260 }

```

(End of definition for `\__spintent_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’.

```

4261 \cs_new_protected:Nn \__spinttent_spabs_exec:nn
4262 {
4263   \keys_set:nn { spinttent / spabs } {#1}
4264   \__spinttent_spabs_set_delims:
4265   \__spinttent_spabs_mathml:n {#2}
4266 }

```

(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 $_(\textit{read-arg}):nofix$; 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.

```

4267 \cs_new_protected:Nn \__spinttent_spabs_mathml:n
4268 {
4269   \bool_if:NTF \l__spinttent_spabs_silent_bool
4270   {
4271     \__spinttent_invisible_sep:
4272     \MathMLintent { _- }
4273     {
4274       \l__spinttent_spabs_delim_l_tl \__spinttent_inner_sep: #1
4275       \l__spinttent_spabs_delim_r_tl
4276     }
4277   }
4278   {
4279     \tl_if_empty:NTF \l__spinttent_spabs_read_arg_tl
4280     {
4281       \__spinttent_spabs_set_close:n {#1}
4282       \bool_if:NF \l__spinttent_spabs_close_bool { \__spinttent_invisible_sep: }
4283       \MathMLintent { _valor-absoluto(\$x) }
4284       {
4285         \l__spinttent_spabs_delim_l_tl
4286         \MathMLarg { x } { \__spinttent_inner_sep: #1 }
4287         \l__spinttent_spabs_delim_r_tl
4288       }
4289       \bool_if:NT \l__spinttent_spabs_close_bool
4290       {
4291         \MathMLintent { _fin-valor-absoluto }
4292         { \__spinttent_inner_sep: \c__spinttent_invisible_sep_int }
4293       }
4294     }
4295     {
4296       \__spinttent_invisible_sep:
4297       \MathMLintent { _ \tl_use:N \l__spinttent_spabs_read_arg_tl :nofix }
4298       {
4299         \l__spinttent_spabs_delim_l_tl \__spinttent_inner_sep: #1
4300         \l__spinttent_spabs_delim_r_tl
4301       }
4302     }
4303   }
4304 }

```

(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`.

```

4305 \NewDocumentCommand \spabs { O{} m }
4306 {
4307   \mode_if_math:TF
4308   {
4309     \group_begin:
4310     \attribute \l__spinttent_scope_attr_int = 1
4311     \tl_if_blank:nTF {#2}

```

```

4312         { \msg_error:nn { spintent } { spabs-empty-arg } }
4313         { \__spintent_spabs_exec:nn {#1} {#2} }
4314     \group_end:
4315 }
4316 { \msg_error:nnn { spintent } { need-math-mode } { \spabs } }
4317 }

```

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

15.43 The function `\__spintent_mathmlarg_point:nnn`

Scratch variables for `\__spintent_mathmlarg_point:nnn`: the reference name in ‘#1’, and the calling command’s own `read-cmd/silent` state, looked up by the name in ‘#2’.

```

4318 \tl_new:N \__spintent_mathmlarg_point_ref_tl
4319 \str_new:N \__spintent_mathmlarg_point_mode_str
4320 \tl_new:N \__spintent_mathmlarg_point_kv_tl
4321 \bool_new:N \__spintent_mathmlarg_point_silent_bool

```

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

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 `\__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`.

```

4322 \cs_new_protected:Nn \__spintent_mathmlarg_point:nnn
4323 {
4324     \tl_set:Nn \__spintent_mathmlarg_point_ref_tl {#1}
4325     \__spintent_luafun_geo_point_parse_and_set:n { \exp_not:n {#3} }
4326     \bool_set_eq:Nc \__spintent_mathmlarg_point_silent_bool
4327         { \__spintent_#2_silent_bool }
4328     \str_set_eq:Nc \__spintent_mathmlarg_point_mode_str { \__spintent_#2_mode_str }
4329     \__spintent_invisible_sep:
4330     \str_case:VnF \__spintent_geo_luaset_tipo_str
4331     {
4332         { sub }
4333         {
4334             \str_if_eq:VnTF \__spintent_mathmlarg_point_mode_str { school }
4335             {
4336                 \tl_set:Nc \__spintent_mathmlarg_point_kv_tl
4337                 {
4338                     arg = "\__spintent_mathmlarg_point_ref_tl",
4339                     intent = "_ \__spintent_geo_luaset_intent_str -"
4340                 }
4341                 \__spintent_luafun_mrow_intent_arg:nn
4342                 { } { \__spintent_mathmlarg_point_kv_tl }
4343                 \int_compare:nNnTF { \str_count:N \__spintent_geo_luaset_sub_str } >
4344                 { 1 }
4345                 {
4346                     \int_step_inline:nnnn { 1 } { 1 }
4347                     { \str_count:N \__spintent_geo_luaset_sub_str }
4348                     {
4349                         \__spintent_luafun_mrow_intent_arg:nn
4350                         { } { intent = ":silent" }
4351                     }
4352                 }
4353                 \__spintent_geo_luaset_letra_str
4354                 \c_math_subscript_token { \__spintent_geo_luaset_sub_str }
4355             }
4356         }
4357         {
4358             \__spintent_geo_luaset_letra_str
4359             \c_math_subscript_token
4360             { \MathMLintent { :silent } { \__spintent_geo_luaset_sub_str } }
4361         }
4362     }
4363 }
4364 }

```

```

4365         {
4366         {
4367             \__spintent_inner_sep:
4368             \MathMLarg { \__spintent_mathmlarg_point_ref_tl }
4369             {
4370             {
4371                 \__spintent_geo_luaset_letra_str
4372                 \c_math_subscript_token { \__spintent_geo_luaset_sub_str }
4373             }
4374             }
4375         }
4376     }
4377 }

```

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`.

```

4378 { prima }
4379 {
4380     \str_if_eq:VnTF \__spintent_mathmlarg_point_mode_str { school }
4381     {
4382         \tl_set:Ne \__spintent_mathmlarg_point_kv_tl
4383         {
4384             arg = "\__spintent_mathmlarg_point_ref_tl",
4385             intent = "_ \__spintent_geo_luaset_intent_str -"
4386         }
4387         \__spintent_luafun_mrow_intent_arg:nn
4388         { } { \__spintent_mathmlarg_point_kv_tl }
4389         { \__spintent_geo_luaset_letra_str \__spintent_geo_luaset_sub_str }
4390     }
4391     {
4392     {
4393         \__spintent_inner_sep:
4394         \MathMLarg { \__spintent_mathmlarg_point_ref_tl }
4395         {
4396         {
4397             \__spintent_geo_luaset_letra_str
4398             \__spintent_geo_luaset_sub_str
4399         }
4400         }
4401     }
4402     }
4403 }
4404 }

```

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

```

4405 {
4406     \bool_if:NTF \__spintent_mathmlarg_point_silent_bool
4407     {
4408         \tl_set:Ne \__spintent_mathmlarg_point_kv_tl
4409         { arg = "\__spintent_mathmlarg_point_ref_tl", intent = "-_" }
4410         \__spintent_luafun_mrow_intent_arg:nn
4411         { } % must be empty
4412         { \__spintent_mathmlarg_point_kv_tl }
4413         { \__spintent_geo_luaset_print_tl }
4414     }
4415     {
4416         \str_if_eq:VnTF \__spintent_mathmlarg_point_mode_str { school }
4417         {
4418             \tl_set:Ne \__spintent_mathmlarg_point_kv_tl
4419             {
4420                 arg = "\__spintent_mathmlarg_point_ref_tl",
4421                 intent = "_ \__spintent_geo_luaset_intent_str -"
4422             }
4423             \__spintent_luafun_mrow_intent_arg:nn
4424             { } % must be empty
4425             { \__spintent_mathmlarg_point_kv_tl }
4426             { \__spintent_geo_luaset_print_tl }

```

```

4427         }
4428     {
4429         \MathMLarg { \__spintent_mathmlarg_point_ref_tl }
4430         { \__spintent_geo_luaset_print_tl }
4431     }
4432 }
4433 }
4434 }
4435 \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`

Internal variables holding the values set by `read-cmd` and `prefix`.

```

\__spintent_spPoint_mode_str
\__spintent_spPoint_prefix_tl
\__spintent_spPoint_intent_arg_tl
4436 \str_new:N \__spintent_spPoint_mode_str
4437 \tl_new:N \__spintent_spPoint_prefix_tl
4438 \tl_new:N \__spintent_spPoint_intent_arg_tl

```

(End of definition for `\__spintent_spPoint_mode_str`, `\__spintent_spPoint_prefix_tl`, and `\__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.

```

4439 \msg_new:nnnn { spintent } { spPoint-invalid-point }
4440 {
4441     Invalid~argument~'~#1'~for~'\token_to_str:N \spPoint'~\msg_line_context:.
4442 }
4443 {
4444     The~argument~must~be~a~single~uppercase~letter~with~an~optional~subscript~(\_{...})~
4445     or~prime~(').
4446 }

```

(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`.

```

4447 \keys_define:nn { spintent / spPoint }
4448 {
4449     read-cmd .choices:nn = { school , mathcat }
4450     {
4451         \int_case:nn { \_keys_choice_int }
4452         {
4453             {1} { \str_set:Nn \__spintent_spPoint_mode_str { school } }
4454             {2} { \str_set:Nn \__spintent_spPoint_mode_str { mathcat } }
4455         }
4456     },
4457     read-cmd .initial:n = school,
4458     read-cmd .value_required:n = true,
4459     read-cmd / unknown .code:n =
4460         \msg_error:nneee { spintent } { unknown-choice }
4461         { read-cmd } { school,~mathcat } { \exp_not:n {#1} },
4462     prefix .code:n =
4463         { \__spintent_sanitize_affix:Nn \__spintent_spPoint_prefix_tl {#1} },
4464     prefix .initial:n = ,
4465     prefix .value_required:n = true,
4466     silent .bool_set:N = \__spintent_spPoint_silent_bool,
4467     silent .initial:n = false,
4468     silent .value_required:n = true,
4469     concept .bool_set:N = \__spintent_spPoint_concept_bool,
4470     concept .initial:n = true,
4471     concept .value_required:n = true,
4472     unknown .code:n =
4473     {

```

```

4474         \str_set:Nn \l__spinttent_command_name_str { spPoint }
4475         \l__spinttent_unknown_keys_cmd:n {#1}
4476     },
4477 }

```

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

15.44.4 Internal functions for \spPoint

__spinttent_spPoint_exec:nnn

The internal function `\__spinttent_spPoint_exec:nnn` processes the $\langle argument \rangle$ passed in ‘#2’. It calls `\__spinttent_luafun_geo_point_parse_and_set:n`, reporting **spPoint-invalid-point** on error. If `silent` is `true` or `concept` is `false`, it sets `\l__spinttent_spPoint_intent_arg_tl` to `_ $\langle$pt\rangle$` , or to `_-` when `silent` is `true`, whatever `read-cmd` is. Otherwise it sets it to `\punto:prefix( $\langle$pt\rangle$ )`, with `prefix` placed after the leading `_` when given. Finally it calls `\__spinttent_spPoint_mathml:n`.

Before processing the $\langle argument \rangle$, it sets the optional $\langle keys \rangle$ passed in ‘#1’.

```

4478 \cs_new_protected:Nn \__spinttent_spPoint_exec:nnn
4479 {
4480     \keys_set:nn { spinttent / spPoint } {#1}
4481     \__spinttent_luafun_geo_point_parse_and_set:n { \exp_not:n {#2} }
4482     \str_if_eq:VnT \l__spinttent_geo_luaset_error_str { true }
4483     {
4484         \msg_error:nne { spinttent } { spPoint-invalid-point } { \exp_not:n {#2} }
4485     }
4486     \bool_if:nTF
4487     {
4488         \bool_lazy_or:p:nn
4489         { \l__spinttent_spPoint_silent_bool }
4490         { ! \l__spinttent_spPoint_concept_bool }
4491     }
4492     {
4493         \tl_set:Nn \l__spinttent_spPoint_intent_arg_tl { _( $\langle$pt\rangle$ ) }
4494         \bool_if:NT \l__spinttent_spPoint_silent_bool
4495         {
4496             \tl_set:Nn \l__spinttent_spPoint_intent_arg_tl { _- }
4497         }
4498     }
4499     {
4500         \tl_set:Ne \l__spinttent_spPoint_intent_arg_tl
4501         {
4502             -
4503             \tl_if_empty:NF \l__spinttent_spPoint_prefix_tl
4504             {
4505                 \tl_use:N \l__spinttent_spPoint_prefix_tl -
4506             }
4507             punto:prefix( $\langle$pt\rangle$ )
4508         }
4509     }
4510     \__spinttent_spPoint_mathml:n {#2}
4511     \tl_if_novalue:nF {#3}
4512     {
4513         \MathMLintent { :silent } { \c__spinttent_invisible_sep_int } % prevent "por"
4514         \tl_set:Nn \l__spinttent_spcoord_raw_arg_tl {#3}
4515         \__spinttent_luafun_spcoord_parse_and_set:n { \exp_not:n {#3} }
4516         \str_if_eq:VnT \l__spinttent_spcoord_luaset_error_str { true }
4517         { \msg_error:nne { spinttent } { spcoord-invalid-arg } { \exp_not:n {#3} } }
4518         \__spinttent_spcoord_set_wrap_parens_size:
4519         \str_if_eq:VnF \l__spinttent_spcoord_luaset_error_str { true }
4520         { \__spinttent_spcoord_mathml:N \l__spinttent_spPoint_silent_bool }
4521     }
4522 }

```

(End of definition for __spinttent_spPoint_exec:nnn.)

__spinttent_spPoint_mathml:n

The function `\__spinttent_spPoint_mathml:n` calls `\__spinttent_invisible_sep:` and then wraps `\__spinttent_mathmlarg_point:nnn`, which parses ‘#1’, tags it `arg="pt"` and silences a subscript or prime, inside a native `\MathMLintent` carrying `\l__spinttent_spPoint_intent_arg_tl` as the concept intent. Inside that group it calls `\__spinttent_inner_sep:` first.

```

4523 \cs_new_protected:Nn \__spinttent_spPoint_mathml:n
4524 {
4525     \__spinttent_invisible_sep:

```

```

4526 \MathMLintent { \l__spintent_spPoint_intent_arg_tl }
4527 {
4528   \__spintent_inner_sep:
4529   \__spintent_mathmlarg_point:nnn { pt } { spPoint } {#1}
4530 }
4531 }

```

(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`.

```

4532 \NewDocumentCommand \spPoint { 0{} m d() }
4533 {
4534   \mode_if_math:TF
4535   {
4536     \group_begin:
4537     \attribute \l__spintent_scope_attr_int = 1
4538     \__spintent_spPoint_exec:nnn {#1} {#2} {#3}
4539     \group_end:
4540   }
4541   { \msg_error:nnn { spintent } { need-math-mode } { \spPoint } }
4542 }

```

(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 $\langle \textit{argument} \rangle$: 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 $\langle \textit{argument} \rangle$.

15.45.1 Definition of variables for `\spvec`

The variable `\l__spintent_spvec_mode_str` stores the value set by the key `read-cmd` and the variable `\l__spintent_spvec_prefix_tl` stores the value set by the key `prefix`.

```

4543 \str_new:N \l__spintent_spvec_mode_str
4544 \tl_new:N \l__spintent_spvec_prefix_tl
4545 \tl_new:N \l__spintent_spvec_intent_arg_tl
4546 \tl_new:N \l__spintent_spvec_kv_tl
4547 \seq_new:N \l__spintent_spvec_pt_seq
4548 \tl_new:N \l__spintent_spvec_refs_tl
4549 \int_new:N \l__spintent_spvec_pt_idx_int

```

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

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.

```

4550 \msg_new:nnnn { spintent } { spvec-invalid-arg }
4551 {
4552   Invalid~argument~'~#1'~for~'\token_to_str:N \spvec'~'\msg_line_context:.
4553 }
4554 {
4555   The~argument~must~be~a~lowercase~letter~or~two~points~separated~by~a~comma.
4556 }

```

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

15.45.3 Definition of keys for `\spvec`

`silent`
`concept`
`autofit`
`read-cmd`
`prefix`
`unknown`

We define the $\langle \textit{keys} \rangle$ `silent`, `concept`, `autofit`, `read-cmd` and `prefix`.

```

4557 \keys_define:nn { spintent / spvec }
4558 {
4559   silent .bool_set:N = \l__spintent_spvec_silent_bool,
4560   silent .initial:n = false,
4561   silent .value_required:n = true,
4562   concept .bool_set:N = \l__spintent_spvec_concept_bool,
4563   concept .initial:n = true,
4564   concept .value_required:n = true,
4565   autofit .bool_set:N = \l__spintent_spvec_autofit_bool,

```

```

4566 autofit .initial:n = false,
4567 autofit .value_required:n = true,
4568 read-cmd .choices:nn = { school , mathcat }
4569 {
4570   \int_case:nn { \l_keys_choice_int }
4571   {
4572     {1} { \str_set:Nn \l__spintent_svec_mode_str { school } }
4573     {2} { \str_set:Nn \l__spintent_svec_mode_str { mathcat } }
4574   }
4575 },
4576 read-cmd .initial:n = school,
4577 read-cmd .value_required:n = true,
4578 read-cmd / unknown .code:n =
4579   \msg_error:nnee { spintent } { unknown-choice }
4580   { read-cmd } { school,~mathcat } { \exp_not:n {##1} },
4581 prefix .code:n =
4582   { \__spintent_sanitizet_affix:Nn \l__spintent_svec_prefix_tl {#1} },
4583 prefix .initial:n = { },
4584 prefix .value_required:n = true,
4585 unknown .code:n =
4586   {
4587     \str_set:Nn \l__spintent_command_name_str { svec }
4588     \__spintent_unknown_keys_cmd:n {#1}
4589   },
4590 }

```

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

15.45.4 Internal functions for \svec

__spintent_svec_exec:nnn

The internal function `\__spintent_svec_exec:nnn` processes the $\langle argument \rangle$ passed in ‘#2’. Calls `\__spintent_luafun_geo_parse_and_set:n`, reporting **svec-invalid-arg** on error, on a proper name with `\__spintent_geo_luaset_tipo_str` equal to *prima*, since a one-letter vector cannot carry a prime, or on a point list that is not exactly two points. Otherwise calls `\__spintent_svec_mathml:n`.

Before processing the $\langle argument \rangle$, it sets the optional $\langle keys \rangle$ passed in ‘#1’.

```

4591 \cs_new_protected:Nn \__spintent_svec_exec:nnn
4592 {
4593   \keys_set:nn { spintent / svec } {#1}
4594   \__spintent_luafun_geo_parse_and_set:n { \exp_not:n {#2} }
4595   \str_if_eq:VnT \l__spintent_geo_luaset_is_name_str { true }
4596   {
4597     \str_if_eq:VnT \l__spintent_geo_luaset_tipo_str { prima }
4598     { \str_set:Nn \l__spintent_geo_luaset_error_str { true } }
4599   }
4600   \str_if_eq:VnT \l__spintent_geo_luaset_is_name_str { false }
4601   {
4602     \int_compare:nNf { \clist_count:N \l__spintent_geo_luaset_points_clist } = { 2 }
4603     { \str_set:Nn \l__spintent_geo_luaset_error_str { true } }
4604   }
4605   \str_if_eq:VnTF \l__spintent_geo_luaset_error_str { true }
4606   { \msg_error:nne { spintent } { svec-invalid-arg } { \exp_not:n {#2} } }
4607   { \__spintent_svec_mathml:n {#2} }
4608   \tl_if_novalue:nF {#3}
4609   {
4610     \bool_if:Ntf \l__spintent_svec_silent_bool
4611     { \MathMLintent { :silent } { \equal } }
4612     { \equal } % vector notation =( , )
4613     % prevent "intervalo"
4614     \MathMLintent { :silent } { \c__spintent_invisible_sep_int }
4615     \tl_set:Nn \l__spintent_spcord_raw_arg_tl {#3}
4616     \__spintent_luafun_spcord_parse_and_set:n { \exp_not:n {#3} }
4617     \str_if_eq:VnT \l__spintent_spcord_luaset_error_str { true }
4618     { \msg_error:nne { spintent } { spcoord-invalid-arg } { \exp_not:n {#3} } }
4619     \__spintent_spcord_set_wrap_parens_size:
4620     \str_if_eq:VnF \l__spintent_spcord_luaset_error_str { true }
4621     { \__spintent_spcord_mathml:N \l__spintent_svec_silent_bool }
4622   }
4623 }

```

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

`\__spintent_sptvec_mathml:n`

The function `\__spintent_sptvec_mathml:n` branches on `\l__spintent_geo_luaset_is_name_str`. For a proper name that is a single letter with no prime, the concept `intent` is ‘`_`’ when `silent` is `true`, whatever `read-cmd` is, `_{ $\$pt$ }` when `concept` is `false`, and otherwise `_vector:prefix( $\$pt$ )`, with `prefix` placed after the leading `_` when given. With `\l__spintent_geo_luaset_tipo_str` equal to `sub`, `read-cmd=school` tags `\vec{letra}` with `\__spintent_luafun_mrow_intent_arg:nn` and a literal `kv`, and appends the subscript silenced; `read-cmd=mathcat` tags `\vec{...}` with a native `\MathMLarg` and silences the subscript only under `silent`. The key `autofit` inserts a `\vphantom{b}` inside `\vec{...}` to normalize its height. Otherwise, the same `kv` logic wraps `\vec{print\_tl}` as a whole.

```

4624 \cs_new_protected:Nn \__spintent_sptvec_mathml:n
4625 {
4626   \str_if_eq:VnTF \l__spintent_geo_luaset_is_name_str { true }
4627   {
4628     \bool_if:nTF
4629     {
4630       \bool_lazy_or_p:nn
4631       { \l__spintent_sptvec_silent_bool }
4632       { ! \l__spintent_sptvec_concept_bool }
4633     }
4634     {
4635       \tl_set:Nn \l__spintent_sptvec_intent_arg_tl { _( $\$pt$ ) }
4636       \bool_if:NT \l__spintent_sptvec_silent_bool
4637       { \tl_set:Nn \l__spintent_sptvec_intent_arg_tl { _- } }
4638     }
4639     {
4640       \tl_set:Ne \l__spintent_sptvec_intent_arg_tl
4641       {
4642         -
4643         \tl_if_empty:NF \l__spintent_sptvec_prefix_tl
4644         { \tl_use:N \l__spintent_sptvec_prefix_tl - }
4645         vector:prefix( $\$pt$ )
4646       }
4647     }
4648   \str_case:VnF \l__spintent_geo_luaset_tipo_str
4649   {
4650     { sub }
4651     {
4652       \str_if_eq:VnTF \l__spintent_sptvec_mode_str { school }
4653       {
4654         \bool_if:NTF \l__spintent_sptvec_silent_bool
4655         {
4656           \tl_set:Nn \l__spintent_sptvec_kv_tl { arg = "pt", intent = "_-" }
4657         }
4658         {
4659           \tl_set:Ne \l__spintent_sptvec_kv_tl
4660           {
4661             arg = "pt", intent = "_ \l__spintent_geo_luaset_intent_str -"
4662           }
4663         }
4664       \__spintent_invisible_sep:
4665       \MathMLintent { \l__spintent_sptvec_intent_arg_tl }
4666       {
4667         \__spintent_inner_sep:
4668         \__spintent_luafun_mrow_intent_arg:nn
4669         { } { \l__spintent_sptvec_kv_tl }
4670       }
4671       \vec
4672       {
4673         \bool_if:NT \l__spintent_sptvec_autofit_bool
4674         { \vphantom{ b } }
4675         \l__spintent_geo_luaset_letra_str
4676       }
4677       \c_math_subscript_token
4678       {
4679         \MathMLintent { :silent }
4680         { \l__spintent_geo_luaset_sub_str }
4681       }
4682     }
4683   }
4684 }
4685 {

```

```

4686         \__spintent_invisible_sep:
4687         \MathMLintent { \__spintent_svec_intent_arg_tl }
4688         {
4689             \__spintent_inner_sep:
4690             \vec
4691             {
4692                 \bool_if:NT \__spintent_svec_autofit_bool
4693                 { \vphantom{ b } }
4694                 \MathMLarg{ pt } { \__spintent_geo_luaset_letra_str }
4695             }
4696             \bool_if:NT \__spintent_svec_silent_bool
4697             { \c_math_subscript_token { \__spintent_geo_luaset_sub_str } }
4698         }
4699         \bool_if:NF \__spintent_svec_silent_bool
4700         { \c_math_subscript_token { \__spintent_geo_luaset_sub_str } }
4701     }
4702 }
4703 }
4704 {
4705     \bool_if:NTF \__spintent_svec_silent_bool
4706     { \tl_set:Nn \__spintent_svec_kv_tl { arg = "pt", intent = "_-" } }
4707     {
4708         \tl_set:Ne \__spintent_svec_kv_tl
4709         { arg = "pt", intent = "_ \__spintent_geo_luaset_intent_str -" }
4710     }
4711     \__spintent_invisible_sep:
4712     \MathMLintent { \__spintent_svec_intent_arg_tl }
4713     {
4714         \__spintent_inner_sep:
4715         \__spintent_luafun_mrow_intent_arg:nn
4716         { } { \__spintent_svec_kv_tl }
4717     }
4718     \vec
4719     {
4720         \bool_if:NT \__spintent_svec_autofit_bool
4721         { \vphantom{ b } }
4722         \__spintent_geo_luaset_print_tl
4723     }
4724 }
4725 }
4726 }
4727 }

```

For a pair of points, `\__spintent_geo_luaset_points_clist` is split into `\__spintent_svec_pt_seq` and walked to build `\__spintent_svec_refs_tl`, the `$pts1`, `$pts2` reference list. The concept `intent` is ‘`_-`’ when `silent` is `true`, ‘`_(refs)`’ when `concept` is `false` or `_vector:prefix(refs)` otherwise, applied with a native `\MathMLintent` wrapping `\overrightarrow`. The sequence is walked again, calling `\__spintent_mathmlarg_point:nnn` once per point.

```

4728     {
4729         \seq_set_from_clist:NN \__spintent_svec_pt_seq
4730         \__spintent_geo_luaset_points_clist
4731         \tl_clear:N \__spintent_svec_refs_tl
4732         \int_zero:N \__spintent_svec_pt_idx_int
4733         \seq_map_inline:Nn \__spintent_svec_pt_seq
4734         {
4735             \int_incr:N \__spintent_svec_pt_idx_int
4736             \tl_if_empty:NF \__spintent_svec_refs_tl
4737             { \tl_put_right:Nn \__spintent_svec_refs_tl { , } }
4738             \tl_put_right:Ne \__spintent_svec_refs_tl
4739             { $ pts \int_use:N \__spintent_svec_pt_idx_int }
4740         }
4741         \bool_if:nTF
4742         {
4743             \bool_lazy_or_p:nn
4744             { \__spintent_svec_silent_bool }
4745             { ! \__spintent_svec_concept_bool }
4746         }
4747         {
4748             \tl_set:Ne \__spintent_svec_intent_arg_tl
4749             { _ ( \__spintent_svec_refs_tl ) }
4750             \bool_if:NT \__spintent_svec_silent_bool

```

```

4751         { \tl_set:Nn \l__spintent_svec_intent_arg_tl { _- } }
4752     }
4753     {
4754         \tl_set:Nn \l__spintent_svec_intent_arg_tl
4755         {
4756             -
4757             \tl_if_empty:NF \l__spintent_svec_prefix_tl
4758             { \tl_use:N \l__spintent_svec_prefix_tl - }
4759             vector:prefix( \l__spintent_svec_refs_tl )
4760         }
4761     }
4762     \__spintent_invisible_sep:
4763     \MathMLintent { \l__spintent_svec_intent_arg_tl }
4764     {
4765         \__spintent_inner_sep:
4766         \overrightarrow
4767         {
4768             \int_zero:N \l__spintent_svec_pt_idx_int
4769             \seq_map_inline:Nn \l__spintent_svec_pt_seq
4770             {
4771                 \int_incr:N \l__spintent_svec_pt_idx_int
4772                 \__spintent_mathmlarg_point:nnn
4773                 { pts \int_use:N \l__spintent_svec_pt_idx_int }
4774                 { svec }
4775                 { ##1 }
4776             }
4777         }
4778     }
4779 }
4780 }

```

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

\spvec The public command `\spvec` enforces *math mode*, opens a local group, and delegates execution to `\__spintent_svec_exec:nnn`.

```

4781 \NewDocumentCommand \spvec { O{} m d() }
4782 {
4783     \mode_if_math:TF
4784     {
4785         \group_begin:
4786         \attribute \l__spintent_scope_attr_int = 1
4787         \__spintent_svec_exec:nnn {#1} {#2} {#3}
4788         \group_end:
4789     }
4790     { \msg_error:nnn { spintent } { need-math-mode } { \spvec } }
4791 }

```

(End of definition for `\spvec`. This function is documented on page 25.)

15.46 The command `\sprecta`

The user command `\sprecta` supports two types of notation for the input $\langle argument \rangle$: 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`

The variable `\l__spintent_sprecta_mode_str` store the values set by the key `read-cmd` and the variable `\l__spintent_sprayo_prefix_tl` store the value set by the key `prefix`.

```

4792 \str_new:N \l__spintent_sprecta_mode_str
4793 \tl_new:N \l__spintent_sprecta_prefix_tl
4794 \seq_new:N \l__spintent_sprecta_pt_seq
4795 \tl_new:N \l__spintent_sprecta_refs_tl
4796 \tl_new:N \l__spintent_sprecta_intent_arg_tl
4797 \tl_new:N \l__spintent_sprecta_kv_tl
4798 \int_new:N \l__spintent_sprecta_pt_idx_int

```

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

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.

```

4799 \msg_new:nnnn { spintent } { sprecta-invalid-points }
4800 {
4801   Invalid~argument~'#1'~for~'\token_to_str:N \sprecta'\msg_line_context:.
4802 }
4803 {
4804   The~argument~must~be~a~letter~or~two~points~separated~by~a~comma.
4805 }
```

(End of definition for sprecta-invalid-points.)

15.46.3 Definition of keys for \sprecta

read-cmd

We define the $\langle keys \rangle$ read-cmd, prefix, silent and concept.

prefix
silent
concept
unknown

```

4806 \keys_define:nn { spintent / sprecta }
4807 {
4808   read-cmd .choices:nn = { school , mathcat }
4809   {
4810     \int_case:nn { \l_keys_choice_int }
4811     {
4812       {1} { \str_set:Nn \l__spintent_sprecta_mode_str { school } }
4813       {2} { \str_set:Nn \l__spintent_sprecta_mode_str { mathcat } }
4814     }
4815   },
4816   read-cmd .initial:n = school,
4817   read-cmd .value_required:n = true,
4818   read-cmd / unknown .code:n =
4819     \msg_error:nneee { spintent } { unknown-choice }
4820     { read-cmd } { school,~mathcat } { \exp_not:n {#1} },
4821   prefix .code:n =
4822     { \__spintent_sanitise_affix:Nn \l__spintent_sprecta_prefix_tl {#1} },
4823   prefix .initial:n = ,
4824   prefix .value_required:n = true,
4825   silent .bool_set:N = \l__spintent_sprecta_silent_bool,
4826   silent .initial:n = false,
4827   silent .value_required:n = true,
4828   concept .bool_set:N = \l__spintent_sprecta_concept_bool,
4829   concept .initial:n = true,
4830   concept .value_required:n = true,
4831   unknown .code:n =
4832     {
4833       \str_set:Nn \l__spintent_command_name_str { sprecta }
4834       \__spintent_unknown_keys_cmd:n {#1}
4835     },
4836 }
```

(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’.

```

4837 \cs_new_protected:Nn \__spintent_sprecta_exec:nn
4838 {
4839   \keys_set:nn { spintent / sprecta } {#1}
4840   \__spintent_luafun_geo_parse_and_set:n { \exp_not:n {#2} }
4841   \str_if_eq:VnT \l__spintent_geo_luasets_error_str { true }
4842   {
4843     \msg_error:nne { spintent } { sprecta-invalid-points } { \exp_not:n {#2} }
4844   }
4845   \__spintent_sprecta_mathml:n {#2}
4846 }
```

(End of definition for __spintent_sprecta_exec:nn.)

`\__spintent_spectra_mathml:n`

The function `\__spintent_spectra_mathml:n` branches on `\l__spintent_geo_luaset_is_name_str`. For a proper name, such as `m` or `l`, the concept `intent` is `_( $\$pt$ )` when `silent` is `true` or `concept` is `false`, and `_recta:prefix( $\$pt$ )` otherwise, with `prefix` placed after the leading `_` when given; with `silent = true` and `read-cmd` not `school` it is `'_-'`. With `read-cmd=school`, the sub and prima cases tag the letter with a literal kv that is `'_-'` when `silent` is `true`.

With `\l__spintent_geo_luaset_tipo_str` equal to `sub`, `read-cmd=school` tags the base through `\__spintent_luafun_mrow_intent_arg:nn` with a literal kv, the subscript appended and silenced; `read-cmd=mathcat` tags base and subscript together through a native `\MathMLarg` instead.

```

4947 \cs_new_protected:Nn \__spintent_spectra_mathml:n
4948 {
4949   \str_if_eq:NnTF \l__spintent_geo_luaset_is_name_str { true }
4950   {
4951     \__spintent_invisible_sep:
4952     \bool_if:NTF
4953     {
4954       \bool_lazy_or_p:nn
4955       { \l__spintent_spectra_silent_bool }
4956       { ! \l__spintent_spectra_concept_bool }
4957     }
4958     {
4959       \tl_set:Nn \l__spintent_spectra_intent_arg_tl { _( $\$pt$ ) }
4960     \bool_if:NT \l__spintent_spectra_silent_bool
4961     {
4962       \str_if_eq:NnF \l__spintent_spectra_mode_str { school }
4963       { \tl_set:Nn \l__spintent_spectra_intent_arg_tl { _- } }
4964     }
4965   }
4966   {
4967     \tl_set:Ne \l__spintent_spectra_intent_arg_tl
4968     {
4969       _
4970       \tl_if_empty:NF \l__spintent_spectra_prefix_tl
4971       { \tl_use:N \l__spintent_spectra_prefix_tl - }
4972       recta:prefix( $\$pt$ )
4973     }
4974   }
4975   \str_case:NnF \l__spintent_geo_luaset_tipo_str
4976   {
4977     { sub }
4978     {
4979       \str_if_eq:NnTF \l__spintent_spectra_mode_str { school }
4980       {
4981         \bool_if:NTF \l__spintent_spectra_silent_bool
4982         {
4983           \tl_set:Nn \l__spintent_spectra_kv_tl
4984           { arg = "pt", intent = "_-" }
4985         }
4986         {
4987           \tl_set:Ne \l__spintent_spectra_kv_tl
4988           {
4989             arg = "pt", intent = "_ \l__spintent_geo_luaset_intent_str -"
4990           }
4991         }
4992         \MathMLintent { \l__spintent_spectra_intent_arg_tl }
4993         {
4994           \__spintent_inner_sep:
4995           \__spintent_luafun_mrow_intent_arg:nn
4996           { } { \l__spintent_spectra_kv_tl }
4997         {
4998           \l__spintent_geo_luaset_letra_str
4999           \c_math_subscript_token
5000           {
5001             \MathMLintent { :silent }
5002             { \l__spintent_geo_luaset_sub_str }
5003           }
5004         }
5005       }
5006     }

```

For `read-cmd=mathcat`, the same sub case tags the base and the subscript together with a native `\MathMLarg`.

without using the shared queue.

```

4907         {
4908             \MathMLintent { \l__spintent_sprecta_intent_arg_tl }
4909             {
4910                 {
4911                     \__spintent_inner_sep:
4912                     \MathMLarg { pt }
4913                     {
4914                         {
4915                             \l__spintent_geo_luaset_letra_str
4916                             \c_math_subscript_token
4917                             { \l__spintent_geo_luaset_sub_str }
4918                         }
4919                     }
4920                 }
4921             }
4922         }
4923     }

```

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`.

```

4924     { prima }
4925     {
4926         \str_if_eq:VnTF \l__spintent_sprecta_mode_str { school }
4927         {
4928             \bool_if:NTF \l__spintent_sprecta_silent_bool
4929             {
4930                 \tl_set:Nn \l__spintent_sprecta_kv_tl
4931                 { arg = "pt", intent = "_" }
4932             }
4933             {
4934                 \tl_set:Ne \l__spintent_sprecta_kv_tl
4935                 {
4936                     arg = "pt", intent = "_ \l__spintent_geo_luaset_intent_str -"
4937                 }
4938             }
4939             \MathMLintent { \l__spintent_sprecta_intent_arg_tl }
4940             {
4941                 \__spintent_inner_sep:
4942                 \__spintent_luafun_mrow_intent_arg:nn
4943                 { } { \l__spintent_sprecta_kv_tl }
4944                 {
4945                     \l__spintent_geo_luaset_letra_str
4946                     \l__spintent_geo_luaset_sub_str
4947                 }
4948             }
4949         }
4950     {
4951         \MathMLintent { \l__spintent_sprecta_intent_arg_tl }
4952         {
4953             {
4954                 \__spintent_inner_sep:
4955                 \MathMLarg { pt }
4956                 {
4957                     {
4958                         \l__spintent_geo_luaset_letra_str
4959                         \l__spintent_geo_luaset_sub_str
4960                     }
4961                 }
4962             }
4963         }
4964     }
4965 }
4966

```

Otherwise, the same kv logic wraps the combined `\l__spintent_geo_luaset_print_tl` as a whole: with the shared queue for `school` and with its own `arg` alone for `mathcat`.

```

4967     {

```

```

4968     \bool_if:NTF \l__spinttent_sprecta_silent_bool
4969     {
4970       \tl_set:Nn \l__spinttent_sprecta_kv_tl { arg = "pt", intent = "-" }
4971     }
4972     {
4973       \str_if_eq:VnTF \l__spinttent_sprecta_mode_str { school }
4974       {
4975         \tl_set:Ne \l__spinttent_sprecta_kv_tl
4976         { arg = "pt", intent = "_ \l__spinttent_geo_luaset_intent_str -" }
4977       }
4978       {
4979         \tl_set:Nn \l__spinttent_sprecta_kv_tl { arg = "pt" }
4980       }
4981     }
4982     \__spinttent_luafun_mrow_intent_arg:nn
4983     { \l__spinttent_sprecta_intent_arg_tl }
4984     { \l__spinttent_sprecta_kv_tl }
4985     { \l__spinttent_geo_luaset_print_tl }
4986   }
4987 }

```

For a list of points, `\l__spinttent_geo_luaset_points_clist` is split into `\l__spinttent_sprecta_pt_seq` and walked to build `\l__spinttent_sprecta_refs_tl`, the `$pt1, $pt2, \dots` 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 `\overleftrightharrow`. The sequence is walked again, calling `\__spinttent_mathmlarg_point:nnn` once per point.

```

4988   {
4989     \seq_set_from_clist:NN \l__spinttent_sprecta_pt_seq
4990     \l__spinttent_geo_luaset_points_clist
4991     \tl_clear:N \l__spinttent_sprecta_refs_tl
4992     \int_zero:N \l__spinttent_sprecta_pt_idx_int
4993     \seq_map_inline:Nn \l__spinttent_sprecta_pt_seq
4994     {
4995       \int_incr:N \l__spinttent_sprecta_pt_idx_int
4996       \tl_if_empty:NF \l__spinttent_sprecta_refs_tl
4997       { \tl_put_right:Nn \l__spinttent_sprecta_refs_tl { , } }
4998       \tl_put_right:Ne \l__spinttent_sprecta_refs_tl
4999       { $ pt \int_use:N \l__spinttent_sprecta_pt_idx_int }
5000     }
5001     \bool_if:nTF
5002     {
5003       \bool_lazy_or:p:nn
5004       { \l__spinttent_sprecta_silent_bool }
5005       { ! \l__spinttent_sprecta_concept_bool }
5006     }
5007     {
5008       \tl_set:Ne \l__spinttent_sprecta_intent_arg_tl
5009       { _ ( \l__spinttent_sprecta_refs_tl ) }
5010       \bool_if:NT \l__spinttent_sprecta_silent_bool
5011       { \tl_set:Nn \l__spinttent_sprecta_intent_arg_tl { _- } }
5012     }
5013     {
5014       \tl_set:Ne \l__spinttent_sprecta_intent_arg_tl
5015       {
5016         -
5017         \tl_if_empty:NF \l__spinttent_sprecta_prefix_tl
5018         { \tl_use:N \l__spinttent_sprecta_prefix_tl - }
5019         recta:prefix( \l__spinttent_sprecta_refs_tl )
5020       }
5021     }
5022     \__spinttent_invisible_sep:
5023     \MathMLintent { \l__spinttent_sprecta_intent_arg_tl }
5024     {
5025       \__spinttent_inner_sep:
5026       \overleftrightharrow
5027       {
5028         \int_zero:N \l__spinttent_sprecta_pt_idx_int
5029         \seq_map_inline:Nn \l__spinttent_sprecta_pt_seq
5030         {
5031           \int_incr:N \l__spinttent_sprecta_pt_idx_int
5032           \__spinttent_mathmlarg_point:nnn

```

```

5033         { pt \int_use:N \__spintent_sprecta_pt_idx_int }
5034         { sprecta }
5035         { ##1 }
5036     }
5037 }
5038 }
5039 }
5040 }

```

(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`.

```

5041 \NewDocumentCommand \sprecta { 0{} m }
5042 {
5043     \mode_if_math:TF
5044     {
5045         \group_begin:
5046         \attribute \__spintent_scope_attr_int = 1
5047         \__spintent_sprecta_exec:nn {##1} {##2}
5048         \group_end:
5049     }
5050     { \msg_error:nnn { spintent } { need-math-mode } { \sprecta } }
5051 }

```

(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 `\__spintent_sprayo_mode_str` store the values set by the key `read-cmd`; the variable `\__spintent_sprayo_prefix_tl` store the value set by the key `prefix` and the variable `\__spintent_sprayo_read_sty_str` stores the value set by the key `read-sty`.

The variables defined here are assigned using `.choices:nn` or `.code:n` within `\keys_define:nn`.

```

5052 \str_new:N \__spintent_sprayo_mode_str
5053 \tl_new:N \__spintent_sprayo_prefix_tl
5054 \str_new:N \__spintent_sprayo_read_sty_str

```

(End of definition for `\__spintent_sprayo_mode_str`, `\__spintent_sprayo_prefix_tl`, and `\__spintent_sprayo_read_sty_str`.)

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.

```

5055 \msg_new:nnnn { spintent } { sprayo-invalid-points }
5056 {
5057     Invalid~argument~'##1'~for~'\token_to_str:N \sprayo'~\msg_line_context:.
5058 }
5059 {
5060     The~argument~must~be~two~points~separated~by~a~comma.
5061 }

```

(End of definition for `sprayo-invalid-points`.)

15.47.3 Definition of keys for `\sprayo`

We define the `(keys)` `read-cmd`, `prefix`, `silent`, `concept` and `read-sty`.

```

5062 \keys_define:nn { spintent / sprayo }
5063 {
5064     read-cmd .choices:nn = { school , mathcat }
5065     {
5066         \int_case:nn { \l_keys_choice_int }
5067         {
5068             {1} { \str_set:Nn \__spintent_sprayo_mode_str { school } }
5069             {2} { \str_set:Nn \__spintent_sprayo_mode_str { mathcat } }

```

```

5070     }
5071   },
5072   read-cmd .initial:n = school,
5073   read-cmd .value_required:n = true,
5074   read-cmd / unknown .code:n =
5075     \msg_error:nneee { spintent } { unknown-choice }
5076     { read-cmd } { school,~mathcat } { \exp_not:n {#1} },
5077   prefix .code:n =
5078     { \__spintent_sanitizize_affix:Nn \__spintent_sprayo_prefix_tl {#1} },
5079   prefix .initial:n = ,
5080   prefix .value_required:n = true,
5081   silent .bool_set:N = \__spintent_sprayo_silent_bool,
5082   silent .initial:n = false,
5083   silent .value_required:n = true,
5084   concept .bool_set:N = \__spintent_sprayo_concept_bool,
5085   concept .initial:n = true,
5086   concept .value_required:n = true,
5087   read-sty .choices:nn = { rayo , semirrecta }
5088   {
5089     \int_case:nn { \_keys_choice_int }
5090     {
5091       {1} { \str_set:Nn \__spintent_sprayo_read_sty_str { rayo } }
5092       {2} { \str_set:Nn \__spintent_sprayo_read_sty_str { semirrecta } }
5093     }
5094   },
5095   read-sty .initial:n = rayo,
5096   read-sty .value_required:n = true,
5097   read-sty / unknown .code:n =
5098     \msg_error:nneee { spintent } { unknown-choice }
5099     { read-sty } { rayo,~semirrecta } { \exp_not:n {#1} },
5100   unknown .code:n =
5101     {
5102       \str_set:Nn \__spintent_command_name_str { sprayo }
5103       \__spintent_unknown_keys_cmd:n {#1}
5104     },
5105   }

```

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

15.47.4 Internal functions for \sprayo

`\__spintent_sprayo_pt_seq` Auxiliary variables used by `\__spintent_sprayo_mathml:` to split `\__spintent_geo_luaset_points_clist` and build the `$pts1`, `$pts2`, ... reference list.

```

\__spintent_sprayo_refs_tl
  \__spintent_sprayo_intent_arg_tl
    \__spintent_sprayo_pt_idx_int
5106 \seq_new:N \__spintent_sprayo_pt_seq
5107 \tl_new:N \__spintent_sprayo_refs_tl
5108 \tl_new:N \__spintent_sprayo_intent_arg_tl
5109 \int_new:N \__spintent_sprayo_pt_idx_int

```

(End of definition for __spintent_sprayo_pt_seq and others.)

`\__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 `\sprayo` 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’.

```

5110 \cs_new_protected:Nn \__spintent_sprayo_exec:nn
5111 {
5112   \keys_set:nn { spintent / sprayo } {#1}
5113   \__spintent_luafun_geo_parse_and_set:n { \exp_not:n {#2} }
5114   \bool_if:nT
5115   {
5116     \str_if_eq_p:Vn \__spintent_geo_luaset_error_str { true } ||
5117     \str_if_eq_p:Vn \__spintent_geo_luaset_is_name_str { true }
5118   }
5119   {
5120     \msg_error:nne { spintent } { sprayo-invalid-points } { \exp_not:n {#2} }
5121   }
5122   \__spintent_sprayo_mathml:
5123 }

```

(End of definition for __spintent_sprayo_exec:nn.)

`\__spinttent_sprayo_mathml:` The function `\__spinttent_sprayo_mathml:` splits `\l__spinttent_geo_luaset_points_clist`, which Lua sets with one raw point per element, into `\l__spinttent_sprayo_pt_seq` and walks it once to build `\l__spinttent_sprayo_refs_tl`, the comma-separated `$pts1,$pts2,...` reference list, naming each point by position.

The concept `intent` is ‘`-`’ when `silent` is `true`, `_(refs)` when `concept` is `false`, and otherwise `_\l__spinttent_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 `\__spinttent_inner_sep:`. The sequence is walked a second time, calling `\__spinttent_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.

```

5124 \cs_new_protected:Nn \__spinttent_sprayo_mathml:
5125 {
5126   \seq_set_from_clist:NN \l__spinttent_sprayo_pt_seq
5127     \l__spinttent_geo_luaset_points_clist
5128   \tl_clear:N \l__spinttent_sprayo_refs_tl
5129   \int_zero:N \l__spinttent_sprayo_pt_idx_int
5130   \seq_map_inline:Nn \l__spinttent_sprayo_pt_seq
5131     {
5132       \int_incr:N \l__spinttent_sprayo_pt_idx_int
5133       \tl_if_empty:NF \l__spinttent_sprayo_refs_tl
5134         { \tl_put_right:Nn \l__spinttent_sprayo_refs_tl { , } }
5135       \tl_put_right:Ne \l__spinttent_sprayo_refs_tl
5136         { $ pts \int_use:N \l__spinttent_sprayo_pt_idx_int }
5137     }
5138   \bool_if:nTF
5139     {
5140       \bool_lazy_or_p:nn
5141         { \l__spinttent_sprayo_silent_bool }
5142         { ! \l__spinttent_sprayo_concept_bool }
5143     }
5144     {
5145       \tl_set:Ne \l__spinttent_sprayo_intent_arg_tl
5146         { _ ( \l__spinttent_sprayo_refs_tl ) }
5147       \bool_if:NT \l__spinttent_sprayo_silent_bool
5148         { \tl_set:Nn \l__spinttent_sprayo_intent_arg_tl { _- } }
5149     }
5150     {
5151       \tl_set:Ne \l__spinttent_sprayo_intent_arg_tl
5152         {
5153           -
5154           \tl_if_empty:NF \l__spinttent_sprayo_prefix_tl
5155             { \tl_use:N \l__spinttent_sprayo_prefix_tl - }
5156           \l__spinttent_sprayo_read_sty_str :prefix( \l__spinttent_sprayo_refs_tl )
5157         }
5158     }
5159   \__spinttent_invisible_sep:
5160   \MathMLintent { \l__spinttent_sprayo_intent_arg_tl }
5161   {
5162     \__spinttent_inner_sep:
5163     \overrightarrow
5164     {
5165       \int_zero:N \l__spinttent_sprayo_pt_idx_int
5166       \seq_map_inline:Nn \l__spinttent_sprayo_pt_seq
5167         {
5168           \int_incr:N \l__spinttent_sprayo_pt_idx_int
5169           \__spinttent_mathmlarg_point:nnn
5170             { pts \int_use:N \l__spinttent_sprayo_pt_idx_int }
5171             { sprayo }
5172             { ##1 }
5173         }
5174     }
5175   }
5176 }

```

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

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

```

5177 \NewDocumentCommand \sprayo { 0{} m }

```

```

5178 {
5179   \mode_if_math:TF
5180   {
5181     \group_begin:
5182       \attribute \l__spintent_scope_attr_int = 1
5183       \__spintent_sprayo_exec:nn {#1} {#2}
5184     \group_end:
5185   }
5186   { \msg_error:nnn { spintent } { need-math-mode } { \sprayo } }
5187 }

```

(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 `\l__spintent_spside_mode_str` and `\l__spintent_spmside_mode_str` store the values set by the key `read-cmd`; the variables `\l__spintent_spside_prefix_tl` and `\l__spintent_spmside_prefix_tl` store the values set by the key `prefix`. The variables `\l__spintent_spside_read_sty_str` and `\l__spintent_spmside_read_sty_str` store the value set by the key `read-sty`. The variables `\l__spintent_spside_read_arg_tl` and `\l__spintent_spmside_read_arg_tl` store the value set by the key `read-arg`.

- The variables are defined here as they are assigned using `.choices:nn` or `.code:n` rather than a direct property within `\keys_define:nn`.

```

5188 \clist_map_inline:nn { side , mside }
5189 {
5190   \str_new:c { l__spintent_sp #1 _mode_str }
5191   \tl_new:c { l__spintent_sp #1 _prefix_tl }
5192   \str_new:c { l__spintent_sp #1 _read_sty_str }
5193   \tl_new:c { l__spintent_sp #1 _read_arg_tl }
5194 }

```

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

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 $\langle \textit{argument} \rangle$ passed in ‘#1’ to `\spside` or `\spmside`.

```

5195 \clist_map_inline:nn { side , mside }
5196 {
5197   \msg_new:nnnn { spintent } { sp #1 -invalid-points }
5198   {
5199     Invalid~argument~'##2'~for~'\tl_trim_spaces:n {##1}'~\msg_line_context:.
5200   }
5201   {
5202     The~argument~must~be~two~points~separated~by~a~comma.
5203   }
5204 }

```

(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 $\langle \textit{keys} \rangle$ `read-cmd`, `prefix`, `silent`, `concept`, `read-sty`, `read-arg` and `print-sym`.
`prefix`
`silent`
`concept`
`read-sty`
`read-arg`
`print-sym`
`unknown`

```

5205 \clist_map_inline:nn { side , mside }
5206 {
5207   \keys_define:nn { spintent / sp #1 }
5208   {
5209     read-cmd .choices:nn = { school , mathcat }
5210     {
5211       \int_case:nn { \l_keys_choice_int }
5212       {

```

```

5213         {1} { \str_set:cn { l__spintent_sp #1 _mode_str } { school } }
5214         {2} { \str_set:cn { l__spintent_sp #1 _mode_str } { mathcat } }
5215     }
5216 },
5217 read-cmd .initial:n = school,
5218 read-cmd .value_required:n = true,
5219 read-cmd / unknown .code:n =
5220     \msg_error:nneee { spintent } { unknown-choice }
5221     { read-cmd } { school,~mathcat } { \exp_not:n {##1} },
5222 prefix .code:n =
5223     { \__spintent_sanitize_affix:cn { l__spintent_sp #1 _prefix_tl } {##1} },
5224 prefix .initial:n = ,
5225 prefix .value_required:n = true,
5226 silent .bool_set:c = { l__spintent_sp #1 _silent_bool },
5227 silent .initial:n = false,
5228 silent .value_required:n = true,
5229 concept .bool_set:c = { l__spintent_sp #1 _concept_bool },
5230 concept .initial:n = true,
5231 concept .value_required:n = true,
5232 read-sty .choices:nn = { lado , segmento , trazo }
5233 {
5234     \int_case:nn { \l_keys_choice_int }
5235     {
5236         {1} { \str_set:cn { l__spintent_sp #1 _read_sty_str } { lado } }
5237         {2} { \str_set:cn { l__spintent_sp #1 _read_sty_str } { segmento } }
5238         {3} { \str_set:cn { l__spintent_sp #1 _read_sty_str } { trazo } }
5239     }
5240 },
5241 read-sty .initial:n = lado,
5242 read-sty .value_required:n = true,
5243 read-sty / unknown .code:n =
5244     \msg_error:nneee { spintent } { unknown-choice }
5245     { read-sty } { lado,~segmento,~trazo } { \exp_not:n {##1} },
5246 read-arg .code:n =
5247     {
5248         \__spintent_sanitize_affix:cn { l__spintent_sp #1 _read_arg_tl } {##1}
5249     },
5250 read-arg .initial:n = ,
5251 read-arg .value_required:n = true,
5252 print-sym .bool_set:c = { l__spintent_sp #1 _print_sym_bool },
5253 print-sym .initial:n = false,
5254 print-sym .value_required:n = true,
5255 unknown .code:n =
5256     {
5257         \str_set:Nn \l__spintent_command_name_str { sp #1 }
5258         \__spintent_unknown_keys_cmd:n {##1}
5259     },
5260 }
5261 }

```

🌱 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.)

```

\l__spintent_spside_intent_tl
\l__spintent_spside_pt_seq
\l__spintent_spside_refs_tl
\l__spintent_spside_intent_arg_tl
\l__spintent_spside_pt_idx_int

```

The variable `\l__spintent_spside_intent_tl` holds the concept word, taken from `read-arg` or `read-sty`. The rest are scratch variables shared by both commands to split `\l__spintent_geo_luasets_points_clist` and build the `$pts1, $pts2, ...` reference list.

```

5262 \tl_new:N \l__spintent_spside_intent_tl
5263 \seq_new:N \l__spintent_spside_pt_seq
5264 \tl_new:N \l__spintent_spside_refs_tl
5265 \tl_new:N \l__spintent_spside_intent_arg_tl
5266 \int_new:N \l__spintent_spside_pt_idx_int

```

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

```
\__spintent_spside_exec:nnn
```

The internal function `\__spintent_spside_exec:nnn` receives the short name of the command, `side` or `mside`, in `#1` and the `{(argument)}` in `#3`. It calls `\__spintent_luafun_geo_parse_and_set:n` to process `#3`, returning an “error” if invalid, then builds `\l__spintent_spside_intent_tl`.

When `read-arg` is empty, it is set from `read-sty`, prefixed with ‘medida-del-’ for `\spmside`. When `read-arg` is given, it is used as is, and ‘medida-del-’ is prepended only if `concept` is `true`:

with `concept=false`, `read-arg` is the verbatim escape hatch for a full user-written phrase such as `read-arg={medida de la hipotenusa}`, with gender and article left to the user. No `intent` is built here: it needs the reference list, which `\__spintent_spside_mathml:n` builds after splitting the points.

Before processing the $\langle argument \rangle$, it sets the optional $\langle keys \rangle$ passed in ‘#2’.

```

5267 \cs_new_protected:Nn \__spintent_spside_exec:nnn
5268 {
5269   \keys_set:nn { spintent / sp #1 } {#2}
5270   \__spintent_luafun_geo_parse_and_set:n { \exp_not:n {#3} }
5271   \bool_if:nT
5272     {
5273       \str_if_eq_p:Vn \__spintent_geo_luaset_error_str { true } ||
5274       \str_if_eq_p:Vn \__spintent_geo_luaset_is_name_str { true }
5275     }
5276     {
5277       \msg_error:nnee { spintent } { sp #1 -invalid-points }
5278       { \c_backslash_str sp #1 } { \exp_not:n {#3} }
5279     }
5280   \tl_if_empty:cTF { \__spintent_sp #1 _read_arg_tl }
5281     {
5282       \tl_set:Ne \__spintent_spside_intent_tl
5283         { \str_use:c { \__spintent_sp #1 _read_sty_str } }
5284       \str_if_eq:eeT {#1} { mside }
5285       {
5286         \tl_put_left:Nn \__spintent_spside_intent_tl { medida-del- }
5287       }
5288     }
5289     {
5290       \tl_set:Ne \__spintent_spside_intent_tl
5291         { \tl_use:c { \__spintent_sp #1 _read_arg_tl } }
5292       \bool_if:cT { \__spintent_sp #1 _concept_bool }
5293       {
5294         \str_if_eq:eeT {#1} { mside }
5295         {
5296           \tl_put_left:Nn \__spintent_spside_intent_tl { medida-del- }
5297         }
5298       }
5299     }
5300   \__spintent_spside_mathml:n {#1}
5301 }

```

(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 `\__spintent_geo_luaset_points_clist` into `\__spintent_spside_pt_seq` and walks it once to build `\__spintent_spside_refs_tl`, the comma-separated `$pts1,$pts2,...` reference list, naming each point by position.

```

5302 \cs_new_protected:Nn \__spintent_spside_mathml:n
5303 {
5304   \seq_set_from_clist:NN \__spintent_spside_pt_seq
5305     \__spintent_geo_luaset_points_clist
5306   \tl_clear:N \__spintent_spside_refs_tl
5307   \int_zero:N \__spintent_spside_pt_idx_int
5308   \seq_map_inline:Nn \__spintent_spside_pt_seq
5309     {
5310       \int_incr:N \__spintent_spside_pt_idx_int
5311       \tl_if_empty:NF \__spintent_spside_refs_tl
5312         { \tl_put_right:Nn \__spintent_spside_refs_tl { , } }
5313       \tl_put_right:Ne \__spintent_spside_refs_tl
5314         { $ pts \int_use:N \__spintent_spside_pt_idx_int }
5315     }

```

The concept `intent` is ‘_’ when `silent` is `true`. Otherwise, if `read-arg` is empty, it is `_`(refs) when `concept` is `false` and `\__spintent_spside_intent_tl:prefix(refs)` when `true`; if `read-arg` is given, it is always `\__spintent_spside_intent_tl:prefix(refs)`. The `:prefix(...)` form is prefixed with `prefix` when given.

```

5316   \bool_if:cTF { \__spintent_sp #1 _silent_bool }
5317     {
5318       \tl_set:Nn \__spintent_spside_intent_arg_tl { _- }
5319     }

```

```

5320 {
5321   \tl_if_empty:cTF { l__spintent_sp #1 _read_arg_tl }
5322   {
5323     \bool_if:cTF { l__spintent_sp #1 _concept_bool }
5324     {
5325       \tl_set:Ne \l__spintent_spside_intent_arg_tl
5326       {
5327         -
5328         \tl_if_empty:cF { l__spintent_sp #1 _prefix_tl }
5329         { \tl_use:c { l__spintent_sp #1 _prefix_tl } - }
5330         \l__spintent_spside_intent_tl :prefix( \l__spintent_spside_refs_tl )
5331       }
5332     }
5333     {
5334       \tl_set:Ne \l__spintent_spside_intent_arg_tl
5335       { _ ( \l__spintent_spside_refs_tl ) }
5336     }
5337   }
5338   {
5339     \tl_set:Ne \l__spintent_spside_intent_arg_tl
5340     {
5341       -
5342       \tl_if_empty:cF { l__spintent_sp #1 _prefix_tl }
5343       { \tl_use:c { l__spintent_sp #1 _prefix_tl } - }
5344       \l__spintent_spside_intent_tl :prefix( \l__spintent_spside_refs_tl )
5345     }
5346   }
5347 }

```

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.

```

5348 \__spintent_invisible_sep: % (luamml issues #26)
5349 \MathMLintent { \l__spintent_spside_intent_arg_tl }
5350 {
5351   \__spintent_inner_sep:
5352   \str_if_eq:eeTF {#1} { side }
5353   {
5354     \overline { \__spintent_spside_points_loop:n {#1} }
5355   }
5356   {
5357     \bool_if:cTF { l__spintent_sp #1 _print_sym_bool }
5358     {
5359       \__spintent_luafun_mrow_intent_arg:nn
5360       { } { intent = ":silent" }
5361       { \operatorname { m } } }
5362     \overline { \__spintent_spside_points_loop:n {#1} }
5363   }
5364   {
5365     \__spintent_spside_points_loop:n {#1}
5366   }
5367 }
5368 }
5369 }

```

(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`.

```

5370 \cs_new_protected:Nn \__spintent_spside_points_loop:n
5371 {
5372   \int_zero:N \l__spintent_spside_pt_idx_int
5373   \seq_map_inline:Nn \l__spintent_spside_pt_seq
5374   {
5375     \int_incr:N \l__spintent_spside_pt_idx_int

```

```

5376         \__spintent_mathmlarg_point:nnn
5377         { pts \int_use:N \__spintent_spside_pt_idx_int }
5378         { sp #1 }
5379         { ##1 }
5380     }
5381 }

```

(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`.

```

5382 \NewDocumentCommand \spside { 0{} m }
5383 {
5384     \mode_if_math:TF
5385     {
5386         \group_begin:
5387         \attribute \__spintent_scope_attr_int = 1
5388         \__spintent_spside_exec:nnn { side } {#1} {#2}
5389         \group_end:
5390     }
5391     { \msg_error:nnn { spintent } { need-math-mode } { \spside } }
5392 }
5393 \NewDocumentCommand \spmside { 0{} m }
5394 {
5395     \mode_if_math:TF
5396     {
5397         \group_begin:
5398         \attribute \__spintent_scope_attr_int = 1
5399         \__spintent_spside_exec:nnn { mside } {#1} {#2}
5400         \group_end:
5401     }
5402     { \msg_error:nnn { spintent } { need-math-mode } { \spmside } }
5403 }

```

(End of definition for `\spside` and `\spmside`. These functions are documented on page 26.)

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`

Each pair of variables is declared once for both `\spangle` and `\spmangle` via `\clist_map_inline:nn`. The variables `\__spintent_spangle_mode_str` and `\__spintent_spmangle_mode_str` store the values set by the key `read-cmd`; the variables `\__spintent_spangle_prefix_tl` and `\__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`.

```

5404 \clist_map_inline:nn { angle , mangle }
5405 {
5406     \str_new:c { \__spintent_sp #1 _mode_str }
5407     \tl_new:c { \__spintent_sp #1 _prefix_tl }
5408 }

```

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

15.49.2 Definition of error messages for `\spangle` and `\spmangle`

`spangle-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`.

```

5409 \clist_map_inline:nn { angle , mangle }
5410 {
5411     \msg_new:nnnn { spintent } { sp #1 -invalid-arg }
5412     {
5413         Invalid~argument~'##2'~for~'\tl_trim_spaces:n {##1}'~\msg_line_context:.
5414     }
5415     {
5416         Use~a~point,~a~Greek~letter,~a~number~or~three~points~separated~by~

```

```

5417         commas.
5418     }
5419 }

```

(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`,

```

read-cmd 5420 \clist_map_inline:nn { angle , mangle }
prefix   5421 {
recto     5422   \keys_define:nn { spintent / sp #1 }
print-sym 5423   {
silent    5424     read-cmd .choices:nn = { school , mathcat }
concept   5425     {
unknown   5426       \int_case:nn { \l_keys_choice_int }
           5427       {
           5428         {1} { \str_set:cn { l__spintent_sp #1 _mode_str } { school } }
           5429         {2} { \str_set:cn { l__spintent_sp #1 _mode_str } { mathcat } }
           5430       }
           5431     },
           5432     read-cmd .initial:n = school,
           5433     read-cmd .value_required:n = true,
           5434     read-cmd / unknown .code:n =
           5435       \msg_error:nneee { spintent } { unknown-choice }
           5436       { read-cmd } { school,~mathcat } { \exp_not:n {##1} },
           5437     prefix .code:n =
           5438       { \__spintent_sanitize_affix:cn { l__spintent_sp #1 _prefix_tl } {##1} },
           5439     prefix .initial:n = ,
           5440     prefix .value_required:n = true,
           5441     recto .bool_set:c = { l__spintent_sp #1 _recto_bool },
           5442     recto .initial:n = false,
           5443     recto .value_required:n = true,
           5444     print-sym .bool_set:c = { l__spintent_sp #1 _print_sym_bool },
           5445     print-sym .initial:n = false,
           5446     print-sym .value_required:n = true,
           5447     silent .bool_set:c = { l__spintent_sp #1 _silent_bool },
           5448     silent .initial:n = false,
           5449     silent .value_required:n = true,
           5450     concept .bool_set:c = { l__spintent_sp #1 _concept_bool },
           5451     concept .initial:n = true,
           5452     concept .value_required:n = true,
           5453     unknown .code:n =
           5454       {
           5455         \str_set:Nn \l__spintent_command_name_str { sp #1 }
           5456         \__spintent_unknown_keys_cmd:n {##1}
           5457       },
           5458     }
           5459   }

```

🔗 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`.

```

5460 \cs_new_protected:Nn \__spintent_spangle_sym:n
5461 {
5462   \str_if_eq:eeTF {#1} { mangle }
5463   {
5464     \bool_if:cTF { l__spintent_sp #1 _print_sym_bool }
5465     {
5466       \bool_if:cTF { l__spintent_sp #1 _recto_bool }
5467       { \__spintent_right_angle_sqr: } { \angle }
5468     }
5469     {
5470       \bool_if:cTF { l__spintent_sp #1 _recto_bool }
5471       { \measuredrightangle } { \measuredangle }
5472     }

```

```

5473     }
5474     {
5475         \bool_if:cTF { l__spintent_sp #1 _recto_bool }
5476         { \__spintent_right_angle_sqr: } { \angle }
5477     }
5478 }

```

(End of definition for __spintent_spangle_sym:n.)

The variable __spintent_spangle_intent_tl holds the concept word, ángulo or medida-del-ángulo, with the suffix -recto when `recto` is `true`. The other variables are used by both commands to split __spintent_geo_luaset_points_clist and build the reference list, in the tres-puntos and un-punto cases of __spintent_spangle_mathml:n. The variable __spintent_spangle_kv_tl stores the key-value list passed to __spintent_luafun_mrow_intent_arg:nn.

```

5479 \tl_new:N \__spintent_spangle_intent_tl
5480 \seq_new:N \__spintent_spangle_pt_seq
5481 \tl_new:N \__spintent_spangle_refs_tl
5482 \tl_new:N \__spintent_spangle_intent_arg_tl
5483 \int_new:N \__spintent_spangle_pt_idx_int
5484 \tl_new:N \__spintent_spangle_kv_tl
5485 \bool_new:N \__spintent_spangle_print_sym_now_bool

```

(End of definition for __spintent_spangle_intent_tl and others.)

The internal function __spintent_spangle_exec:nnn receives the short name of the command, `angle` or `mangle`, in ‘#1’ and the {*argument*} in ‘#3’. It calls __spintent_luafun_geo_angulo_parse_and_set:n, reporting `spangle-invalid-arg` or `spmangle-invalid-arg` on error. For a proper name that is a single letter, not numeric, it reports the same error if the letter is lowercase, since an angle name takes an uppercase letter, a number or a Greek letter; the length check excludes a Greek letter command such as `\alpha` from this comparison. Then it sets __spintent_spangle_intent_tl to `ángulo`, or `medida-del-ángulo` for `\spmangle`, suffixed with -recto when `recto` is `true`, and calls __spintent_spangle_mathml:n.

Before processing the {*argument*}, it sets the optional {*keys*} passed in ‘#2’.

```

5486 \cs_new_protected:Nn \__spintent_spangle_exec:nnn
5487 {
5488     \keys_set:nn { spintent / sp #1 } {#2}
5489     \__spintent_luafun_geo_angulo_parse_and_set:n { \exp_not:n {#3} }
5490     \str_if_eq:VnT \__spintent_geo_luaset_error_str { true }
5491     {
5492         \msg_error:nnee { spintent } { sp #1 -invalid-arg }
5493         { \c_backslash_str sp #1 } { \exp_not:n {#3} }
5494     }
5495     \str_if_eq:VnT \__spintent_geo_luaset_angulo_case_str { nombre }
5496     {
5497         \str_if_eq:VnF \__spintent_geo_luaset_nombre_es_num_str { true }
5498         {
5499             \int_compare:nNnT { \str_count:N \__spintent_geo_luaset_letra_str } = { 1 }
5500             {
5501                 \str_if_eq:eeF { \__spintent_geo_luaset_letra_str }
5502                 { \str_uppercase:n { \__spintent_geo_luaset_letra_str } }
5503                 {
5504                     \msg_error:nnee { spintent } { sp #1 -invalid-arg }
5505                     { \c_backslash_str sp #1 } { \exp_not:n {#3} }
5506                 }
5507             }
5508         }
5509     }
5510     \str_if_eq:eeTF {#1} { mangle }
5511     {
5512         \tl_set:Nn \__spintent_spangle_intent_tl { medida-del-ángulo }
5513     }
5514     {
5515         \tl_set:Nn \__spintent_spangle_intent_tl { ángulo }
5516     }
5517     \bool_if:cT { l__spintent_sp #1 _recto_bool }
5518     {
5519         \tl_put_right:Nn \__spintent_spangle_intent_tl { -recto }
5520     }
5521     \__spintent_spangle_mathml:n {#1}
5522 }

```

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

`\__spintent_spangle_mathml:n`

The function `\__spintent_spangle_mathml:n` branches on `\__spintent_geo_luaset_angulo_case_str`. For `nombre`, the symbol `\__spintent_spangle_sym:n` is tagged with `\__spintent_luafun_mrow_intent_arg:nn` on its own, with `\__spintent_spangle_intent_tl` alone, without a $\$$ -reference, or with ‘`-`’ when `silent` is `true` or `concept` is `false`. With `silent=true`, the name that follows is silenced as a whole.

```

5523 \cs_new_protected:Nn \__spintent_spangle_mathml:n
5524 {
5525   \__spintent_invisible_sep: % (luamml issues #26)
5526   \str_if_eq:VnTF \__spintent_geo_luaset_angulo_case_str { nombre }
5527   {
5528     \bool_if:cTF { \__spintent_sp #1 _silent_bool }
5529     {
5530       \tl_set:Nn \__spintent_spangle_intent_arg_tl { _- }
5531     }
5532     {
5533       \bool_if:cTF { \__spintent_sp #1 _concept_bool }
5534       {
5535         \tl_set:Ne \__spintent_spangle_intent_arg_tl
5536         {
5537           -
5538           \tl_if_empty:cF { \__spintent_sp #1 _prefix_tl }
5539           { \tl_use:c { \__spintent_sp #1 _prefix_tl } - }
5540           \__spintent_spangle_intent_tl
5541         }
5542       }
5543       {
5544         \tl_set:Nn \__spintent_spangle_intent_arg_tl { _- }
5545       }
5546     }
5547     \tl_set:Ne \__spintent_spangle_kv_tl
5548     { intent = "\__spintent_spangle_intent_arg_tl" }
5549     \__spintent_luafun_mrow_intent_arg:nn
5550     { } { \__spintent_spangle_kv_tl }
5551     { \__spintent_spangle_sym:n {#1} }
5552     \bool_if:cTF { \__spintent_sp #1 _silent_bool }
5553     {
5554       \__spintent_luafun_mrow_intent_arg:nn { } { intent = "- " }
5555       { \__spintent_geo_luaset_print_tl }
5556     }

```

Otherwise, a numeric name is tagged as a whole, through `\__spintent_luafun_mrow_intent_arg:nn`, with a literal `intent` built from `\__spintent_geo_luaset_intent_str`.

```

5557 {
5558   \str_if_eq:VnTF \__spintent_geo_luaset_nombre_es_num_str { true }
5559   {
5560     \tl_set:Ne \__spintent_spangle_kv_tl
5561     { intent = "_ \__spintent_geo_luaset_intent_str -" }
5562     \__spintent_luafun_mrow_intent_arg:nn
5563     { } { \__spintent_spangle_kv_tl }
5564     { \__spintent_geo_luaset_print_tl }
5565   }

```

With `\__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.

```

5566 {
5567   \str_case:VnF \__spintent_geo_luaset_tipo_str
5568   {
5569     { sub }
5570     {
5571       \str_if_eq:eeTF { \use:c { \__spintent_sp #1 _mode_str } }
5572       { school }
5573       {
5574         \tl_set:Ne \__spintent_spangle_kv_tl
5575         { intent = "_ \__spintent_geo_luaset_intent_str -" }
5576         \__spintent_luafun_mrow_intent_arg:nn
5577         { } { \__spintent_spangle_kv_tl }
5578       }

```

```

5579         \l__spintent_geo_luaset_letra_str
5580         \c_math_subscript_token
5581         {
5582             \MathMLintent { :silent }
5583             { \l__spintent_geo_luaset_sub_str }
5584         }
5585     }
5586 }
5587 {
5588     \l__spintent_geo_luaset_letra_str
5589     \c_math_subscript_token { \l__spintent_geo_luaset_sub_str }
5590 }
5591 }

```

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`.

```

5592     { prima }
5593     {
5594         \str_if_eq:eeTF { \use:c { l__spintent_sp #1 _mode_str } }
5595         { school }
5596         {
5597             \tl_set:Nx \l__spintent_spangle_kv_tl
5598             { intent = "_ \l__spintent_geo_luaset_intent_str -" }
5599             \__spintent_luafun_mrow_intent_arg:nn
5600             { } { \l__spintent_spangle_kv_tl }
5601             {
5602                 \l__spintent_geo_luaset_letra_str
5603                 \l__spintent_geo_luaset_sub_str
5604             }
5605         }
5606         {
5607             \l__spintent_geo_luaset_letra_str
5608             \l__spintent_geo_luaset_sub_str
5609         }
5610     }
5611 }
5612 { \l__spintent_geo_luaset_print_tl }
5613 }
5614 }
5615 }

```

For `tres-puntos` and `un-punto`, with the suffix `-en` added to the concept word in the latter, where a single point reads “*ángulo en A*”, `\l__spintent_geo_luaset_points_clist` is split into `\l__spintent_spangle_pt_seq` and walked to build `\l__spintent_spangle_refs_tl`, the reference list `$pt1, $pt2, ...`. A single native `\MathMLintent` wraps the symbol and the points together, with `\l__spintent_spangle_intent_tl:prefix(refs)`, or `'_'` when `silent` is `true` and `_(refs)` when `concept` is `false`. For `\spmangle` with `print-sym=true`, a silent `\operatorname{m}` precedes `\__spintent_spangle_sym:n` inside the same `\MathMLintent`. The sequence is walked again, calling `\__spintent_mathmlarg_point:nnn` once per point.

```

5616     {
5617         \str_if_eq:VnT \l__spintent_geo_luaset_angulo_case_str { un-punto }
5618         { \tl_put_right:Nn \l__spintent_spangle_intent_tl { -en } }
5619         \seq_set_from_clist:NN \l__spintent_spangle_pt_seq
5620         \l__spintent_geo_luaset_points_clist
5621         \tl_clear:N \l__spintent_spangle_refs_tl
5622         \int_zero:N \l__spintent_spangle_pt_idx_int
5623         \seq_map_inline:Nn \l__spintent_spangle_pt_seq
5624         {
5625             \int_incr:N \l__spintent_spangle_pt_idx_int
5626             \tl_if_empty:NF \l__spintent_spangle_refs_tl
5627             { \tl_put_right:Nn \l__spintent_spangle_refs_tl { , } }
5628             \tl_put_right:Nx \l__spintent_spangle_refs_tl
5629             { $ pt \int_use:N \l__spintent_spangle_pt_idx_int }
5630         }
5631         \bool_if:cTF { l__spintent_sp #1 _silent_bool }
5632         {
5633             \tl_set:Nn \l__spintent_spangle_intent_arg_tl { _- }
5634         }
5635         {
5636             \bool_if:cTF { l__spintent_sp #1 _concept_bool }
5637             {

```

```

5638         \tl_set:Nx \l__spintent_spangle_intent_arg_tl
5639         {
5640             -
5641             \tl_if_empty:cF { \l__spintent_sp #1 _prefix_tl }
5642             { \tl_use:c { \l__spintent_sp #1 _prefix_tl } - }
5643             \l__spintent_spangle_intent_tl :prefix( \l__spintent_spangle_refs_tl
5644             )
5645         }
5646     }
5647     {
5648         \tl_set:Nx \l__spintent_spangle_intent_arg_tl
5649         { _ ( \l__spintent_spangle_refs_tl ) }
5650     }
5651 }
5652 \bool_set_false:N \l__spintent_spangle_print_sym_now_bool
5653 \str_if_eq:eeT {#1} { mangle }
5654 {
5655     \bool_if:cT { \l__spintent_sp #1 _print_sym_bool }
5656     { \bool_set_true:N \l__spintent_spangle_print_sym_now_bool }
5657 }
5658 \MathMLintent { \l__spintent_spangle_intent_arg_tl }
5659 {
5660     \__spintent_inner_sep:
5661     \bool_if:NT \l__spintent_spangle_print_sym_now_bool
5662     {
5663         \__spintent_luafun_mrow_intent_arg:nn
5664         { } { intent = ":silent" }
5665         { \operatorname { m } }
5666     }
5667     \__spintent_luafun_mrow_intent_arg:nn
5668     { } { intent = ":silent" }
5669     { \__spintent_spangle_sym:n {#1} }
5670     \int_zero:N \l__spintent_spangle_pt_idx_int
5671     \seq_map_inline:Nn \l__spintent_spangle_pt_seq
5672     {
5673         \int_incr:N \l__spintent_spangle_pt_idx_int
5674         \__spintent_mathmlarg_point:nnn
5675         { pt \int_use:N \l__spintent_spangle_pt_idx_int
5676         { sp#1 }
5677         { ##1 }
5678         }
5679     }
5680 }
5681 }

```

(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`.

```

5682 \NewDocumentCommand \spangle { 0{} m }
5683 {
5684     \mode_if_math:TF
5685     {
5686         \group_begin:
5687         \attribute \l__spintent_scope_attr_int = 1
5688         \__spintent_spangle_exec:nnn { angle } {#1} {#2}
5689         \group_end:
5690     }
5691     { \msg_error:nnn { spintent } { need-math-mode } { \spangle } }
5692 }
5693 \NewDocumentCommand \spmangle { 0{} m }
5694 {
5695     \mode_if_math:TF
5696     {
5697         \group_begin:
5698         \attribute \l__spintent_scope_attr_int = 1
5699         \__spintent_spangle_exec:nnn { mangle } {#1} {#2}
5700         \group_end:
5701     }
5702     { \msg_error:nnn { spintent } { need-math-mode } { \spmangle } }
5703 }

```

(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`.

```
5704 \clist_map_inline:nn { arc , marc }
5705 {
5706   \str_new:c { l__spintent_sp #1 _mode_str }
5707   \tl_new:c { l__spintent_sp #1 _prefix_tl }
5708 }
```

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

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 $\langle\textit{argument}\rangle$ passed in ‘#1’ to `\sparc` or `\spmarc`.

```
5709 \clist_map_inline:nn { arc , marc }
5710 {
5711   \msg_new:nnnn { spintent } { sp #1 -invalid-points }
5712   {
5713     Invalid~argument~'##2'~for~'\tl_trim_spaces:n {##1}'~\msg_line_context:.
5714   }
5715   {
5716     The~argument~must~be~two~or~three~points~separated~by~commas.
5717   }
5718 }
```

(End of definition for `sparc-invalid-points` and `spmarc-invalid-points`.)

15.50.3 Definition of keys for `\sparc` and `\spmarc`

We define the shared $\langle\textit{keys}\rangle$ `read-cmd`, `prefix`, `silent` and `concept`.

```
read-cmd 5719 \clist_map_inline:nn { arc , marc }
prefix   5720 {
silent   5721   \keys_define:nn { spintent / sp #1 }
concept  5722   {
unknown  5723     read-cmd .choices:nn = { school , mathcat }
          5724     {
          5725       \int_case:nn { \l_keys_choice_int }
          5726       {
          5727         {1} { \str_set:cn { l__spintent_sp #1 _mode_str } { school } }
          5728         {2} { \str_set:cn { l__spintent_sp #1 _mode_str } { mathcat } }
          5729       }
          5730     },
          5731     read-cmd .initial:n = school,
          5732     read-cmd .value_required:n = true,
          5733     read-cmd / unknown .code:n =
          5734       \msg_error:nnee { spintent } { unknown-choice }
          5735       { read-cmd } { school,~mathcat } { \exp_not:n {##1} },
          5736     prefix .code:n =
          5737       { \__spintent_sanitize_affix:cn { l__spintent_sp #1 _prefix_tl } {##1} },
          5738     prefix .initial:n = ,
          5739     prefix .value_required:n = true,
          5740     silent .bool_set:c = { l__spintent_sp #1 _silent_bool },
          5741     silent .initial:n = false,
          5742     silent .value_required:n = true,
          5743     concept .bool_set:c = { l__spintent_sp #1 _concept_bool },
          5744     concept .initial:n = true,
          5745     concept .value_required:n = true,
          5746     unknown .code:n =
          5747     {
```

```

5748         \str_set:Nn \l__spintenc_command_name_str { sp #1 }
5749         \__spintenc_unknown_keys_cmd:n {##1}
5750     },
5751 }
5752 }

```

(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` acts as a fallback when the mathematical font used in the document does not provide the command `\widearc`. The definition of this function is a direct adaptation of a response given by Clea F. Rees (@cfr) to a query in the chat room of [TeX-SX](#).

```

5753 \cs_new_protected:Nn \__spintenc_sparc_wide_arc:n
5754 {
5755     \Umathaccent 7 \symoperators "023DC \scan_stop: {#1}
5756 }

```

(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`.

```

5757 \cs_new_protected:Nn \__spintenc_sparc_print_arc:n
5758 {
5759     \cs_if_exist:NTF \widearc
5760     { \widearc {#1} }
5761     { \__spintenc_sparc_wide_arc:n {#1} }
5762 }

```

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

`\__spintenc_sparc_intent_tl`

The variable `\__spintenc_sparc_intent_tl` holds the concept word, arco or medida-del-arco. The rest are scratch variables to split `\__spintenc_geo_luaset_points_clist` and build the `$pt1, $pt2, ...` reference list used in the concept intent.

```

5763 \tl_new:N \__spintenc_sparc_intent_tl
5764 \seq_new:N \__spintenc_sparc_pt_seq
5765 \tl_new:N \__spintenc_sparc_refs_tl
5766 \tl_new:N \__spintenc_sparc_intent_arg_tl
5767 \int_new:N \__spintenc_sparc_pt_idx_int

```

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

`\__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 `\__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`.

```

5768 \cs_new_protected:Nn \__spintenc_sparc_exec:nnn
5769 {
5770     \keys_set:nn { spintenc / sp #1 } {#2}
5771     \__spintenc_luafun_geo_parse_and_set:n { \exp_not:n {#3} }
5772     \bool_if:nT
5773     {
5774         \str_if_eq_p:Vn \__spintenc_geo_luaset_error_str { true } ||
5775         \str_if_eq_p:Vn \__spintenc_geo_luaset_is_name_str { true }
5776     }
5777     {
5778         \msg_error:nnee { spintenc } { sp #1 -invalid-points }
5779         { \c_backslash_str sp #1 } { \exp_not:n {#3} }
5780     }
5781     \str_if_eq:eeTF {#1} { marc }
5782     { \tl_set:Nn \__spintenc_sparc_intent_tl { medida-del-arco } }
5783     { \tl_set:Nn \__spintenc_sparc_intent_tl { arco } }
5784     \__spintenc_sparc_mathml:n {#1}
5785 }

```

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

`\__spintent_sparc_mathml:n`

The function `\__spintent_sparc_mathml:n` receives the short command name in ‘`#1`’. It splits `\l__spintent_geo_luaset_points_clist` into `\l__spintent_sparc_pt_seq` and walks it once to build `\l__spintent_sparc_refs_tl`, the comma-separated `$pt1,$pt2,...` reference list, naming each point by position.

```

5786 \cs_new_protected:Nn \__spintent_sparc_mathml:n
5787 {
5788   \seq_set_from_clist:NN \l__spintent_sparc_pt_seq
5789   \l__spintent_geo_luaset_points_clist
5790
5791   \tl_clear:N \l__spintent_sparc_refs_tl
5792   \int_zero:N \l__spintent_sparc_pt_idx_int
5793   \seq_map_inline:Nn \l__spintent_sparc_pt_seq
5794   {
5795     \int_incr:N \l__spintent_sparc_pt_idx_int
5796     \tl_if_empty:NF \l__spintent_sparc_refs_tl
5797     { \tl_put_right:Nn \l__spintent_sparc_refs_tl { , } }
5798     \tl_put_right:Ne \l__spintent_sparc_refs_tl
5799     { $ pt \int_use:N \l__spintent_sparc_pt_idx_int }
5800   }

```

The concept `intent` is ‘`-`’ when `silent` is `true`, ‘`_(refs)`’ when `concept` is `false`; otherwise `_\l__spintent_sparc_intent_tl:prefix(refs)`, with `prefix` placed after the leading ‘`_`’ when given.

```

5801   \bool_if:cTF { \l__spintent_sp #1 _silent_bool }
5802   {
5803     \tl_set:Nn \l__spintent_sparc_intent_arg_tl { _- }
5804   }
5805   {
5806     \bool_if:cTF { \l__spintent_sp #1 _concept_bool }
5807     {
5808       \tl_set:Ne \l__spintent_sparc_intent_arg_tl
5809       {
5810         -
5811         \tl_if_empty:cF { \l__spintent_sp #1 _prefix_tl }
5812         { \tl_use:c { \l__spintent_sp #1 _prefix_tl } - }
5813         \l__spintent_sparc_intent_tl :prefix( \l__spintent_sparc_refs_tl )
5814       }
5815     }
5816     {
5817       \tl_set:Ne \l__spintent_sparc_intent_arg_tl
5818       { _ ( \l__spintent_sparc_refs_tl ) }
5819     }
5820   }

```

It is applied with a single native `\MathMLintent` wrapping `\__spintent_sparc_print_arc:n` directly, with the same shape as `\__spintent_spside_mathml:n`, preceded by `\__spintent_inner_sep:`, a second sibling so the sole `<mrow>` does not collapse into `<math>`, `luamml` issue #26. A silent `\operatorname{m}` always precedes it for `\spmarc`, since there is no `print-sym` key here. The sequence is walked a second time, calling `\__spintent_mathmlarg_point:nnn` once per point with its `ptn` reference name, looked up under `sparc` or `spmarc`.

```

5821   \__spintent_invisible_sep: % need by MathCAT (bug in luamml issues #26)
5822   \MathMLintent { \l__spintent_sparc_intent_arg_tl }
5823   {
5824     \__spintent_inner_sep:
5825     \str_if_eq:eeT {#1} { marc }
5826     {
5827       \__spintent_luafun_mrow_intent_arg:nn
5828       { } { intent = ":silent" }
5829       { \operatorname { m } } }
5830   }
5831   \__spintent_sparc_print_arc:n
5832   {
5833     \int_zero:N \l__spintent_sparc_pt_idx_int
5834     \seq_map_inline:Nn \l__spintent_sparc_pt_seq
5835     {
5836       \int_incr:N \l__spintent_sparc_pt_idx_int
5837       \__spintent_mathmlarg_point:nnn
5838       { pt \int_use:N \l__spintent_sparc_pt_idx_int }
5839       { sp #1 }
5840       { ##1 }

```

```

5841         }
5842     }
5843 }
5844 }

```

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

`\sparc` Both public commands `\sparc` and `\spmarc` enforces *math mode*, opens a local group, and delegates execution to `\__spintent_sparc_exec:nnn`.

```

5845 \NewDocumentCommand \sparc { O{} m }
5846 {
5847     \mode_if_math:TF
5848     {
5849         \group_begin:
5850         \attribute \__spintent_scope_attr_int = 1
5851         \__spintent_sparc_exec:nnn { arc } {#1} {#2}
5852         \group_end:
5853     }
5854     { \msg_error:nnn { spintent } { need-math-mode } { \sparc } }
5855 }
5856 \NewDocumentCommand \spmarc { O{} m }
5857 {
5858     \mode_if_math:TF
5859     {
5860         \group_begin:
5861         \attribute \__spintent_scope_attr_int = 1
5862         \__spintent_sparc_exec:nnn { marc } {#1} {#2}
5863         \group_end:
5864     }
5865     { \msg_error:nnn { spintent } { need-math-mode } { \spmarc } }
5866 }

```

(End of definition for `\sparc` and `\spmarc`. These functions are documented on page 29.)

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: `\__spintent_geo_luaset_intent_str`, which holds the full *intent* with the figure name.

15.51.1 Definition of variables for `\spPoly`

`\__spintent_spPoly_mode_str` The variable `\__spintent_spPoly_mode_str` stores the value set by the key `read-cmd`; the variable `\__spintent_spPoly_prefix_tl` `\__spintent_spPoly_prefix_tl` stores the value set by the key `prefix`.

```

5867 \str_new:N \__spintent_spPoly_mode_str
5868 \tl_new:N \__spintent_spPoly_prefix_tl

```

(End of definition for `\__spintent_spPoly_mode_str` and `\__spintent_spPoly_prefix_tl`.)

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.

```

5869 \msg_new:nnnn { spintent } { spPoly-invalid-points }
5870 {
5871     Invalid~argument~'!#1'~for~'\token_to_str:N \spPoly'~\msg_line_context:.
5872 }
5873 {
5874     The~argument~must~be~a~list~of~at~least~three~points~separated~by~commas.
5875 }

```

(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
5876 \keys_define:nn { spintent / spPoly }
5877 {
5878     read-cmd .choices:nn = { school , mathcat }
5879     {

```

```

5880         \int_case:nn { \l_keys_choice_int }
5881         {
5882             {1} { \str_set:Nn \l__spinttent_spPoly_mode_str { school } }
5883             {2} { \str_set:Nn \l__spinttent_spPoly_mode_str { mathcat } }
5884         }
5885     },
5886     read-cmd .initial:n = school,
5887     read-cmd .value_required:n = true,
5888     read-cmd / unknown .code:n =
5889         \msg_error:nneee { spinttent } { unknown-choice }
5890         { read-cmd } { school,~mathcat } { \exp_not:n {#1} },
5891     prefix .code:n =
5892         { \__spinttent_sanitizize_affix:Nn \l__spinttent_spPoly_prefix_tl {#1} },
5893     prefix .initial:n = ,
5894     prefix .value_required:n = true,
5895     print-sym .bool_set:N = \l__spinttent_spPoly_print_sym_bool,
5896     print-sym .initial:n = true,
5897     print-sym .value_required:n = true,
5898     silent .bool_set:N = \l__spinttent_spPoly_silent_bool,
5899     silent .initial:n = false,
5900     silent .value_required:n = true,
5901     concept .bool_set:N = \l__spinttent_spPoly_concept_bool,
5902     concept .initial:n = true,
5903     concept .value_required:n = true,
5904     unknown .code:n =
5905         {
5906             \str_set:Nn \l__spinttent_command_name_str { spPoly }
5907             \__spinttent_unknown_keys_cmd:n {#1}
5908         },
5909     }

```

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

15.51.4 Internal functions for `\spPoly`

`\l__spinttent_spPoly_pt_seq`
`\l__spinttent_spPoly_refs_tl`
`\l__spinttent_spPoly_intent_arg_tl`
`\l__spinttent_spPoly_pt_idx_int`

Scratch variables to split `\l__spinttent_geo_luaset_points_clist` and build the `$pt1, $pt2, \dots` reference list, and to hold the final concept intent once built.

```

5910 \seq_new:N \l__spinttent_spPoly_pt_seq
5911 \tl_new:N \l__spinttent_spPoly_refs_tl
5912 \tl_new:N \l__spinttent_spPoly_intent_arg_tl
5913 \int_new:N \l__spinttent_spPoly_pt_idx_int

```

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

`\__spinttent_spPoly_sym:`

Unchanged: reads `\l__spinttent_geo_luaset_poly_sym_str` and typesets the corresponding symbol.

```

5914 \cs_new_protected:Nn \__spinttent_spPoly_sym:
5915 {
5916     \str_case:Vn \l__spinttent_geo_luaset_poly_sym_str
5917     {
5918         { triangle } { \triangle }
5919         { square } { \mdlgwhtsquare }
5920     }
5921 }

```

(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.

```

5922 \cs_new_protected:Nn \__spinttent_spPoly_build_refs:
5923 {
5924     \seq_set_from_clist:NN \l__spinttent_spPoly_pt_seq
5925     \l__spinttent_geo_luaset_points_clist
5926     \tl_clear:N \l__spinttent_spPoly_refs_tl
5927     \int_zero:N \l__spinttent_spPoly_pt_idx_int
5928     \seq_map_inline:Nn \l__spinttent_spPoly_pt_seq
5929     {
5930         \int_incr:N \l__spinttent_spPoly_pt_idx_int
5931         \tl_if_empty:NF \l__spinttent_spPoly_refs_tl
5932         { \tl_put_right:Nn \l__spinttent_spPoly_refs_tl { , } }

```

```

5933         \tl_put_right:Ne \l__spintent_spPoly_refs_tl
5934         { $ pt \int_use:N \l__spintent_spPoly_pt_idx_int }
5935     }
5936 }

```

(End of definition for `\__spintent_spPoly_build_refs:`)

`\__spintent_spPoly_build_concept:` Builds `\l__spintent_spPoly_intent_arg_tl`, independently of `print-sym: ‘_-’` when `silent` is `true`, `_`(`refs`) when `concept` is `false`; otherwise `\l__spintent_geo_luaset_concept_str:prefix(refs)`, with `prefix` placed after the leading `_` when given. Must be called after `\__spintent_spPoly_build_refs:`.

```

5937 \cs_new_protected:Nn \__spintent_spPoly_build_concept:
5938 {
5939     \bool_if:NTF \l__spintent_spPoly_silent_bool
5940     {
5941         \tl_set:Nn \l__spintent_spPoly_intent_arg_tl { _- }
5942     }
5943     {
5944         \bool_if:NTF \l__spintent_spPoly_concept_bool
5945         {
5946             \tl_set:Ne \l__spintent_spPoly_intent_arg_tl
5947             {
5948                 -
5949                 \tl_if_empty:NF \l__spintent_spPoly_prefix_tl
5950                 { \tl_use:N \l__spintent_spPoly_prefix_tl - }
5951                 \l__spintent_geo_luaset_concept_str :prefix( \l__spintent_spPoly_refs_tl
5952                 )
5953             }
5954         }
5955         {
5956             \tl_set:Ne \l__spintent_spPoly_intent_arg_tl
5957             { _ ( \l__spintent_spPoly_refs_tl ) }
5958         }
5959     }
5960 }

```

(End of definition for `\__spintent_spPoly_build_concept:`)

`\__spintent_spPoly_points_loop:` It walks `\l__spintent_spPoly_pt_seq` a second time, calling `\__spintent_mathmlarg_point:nnn` once per vertex with its `ptn` reference name, looked up under `spPoly`. This call decides `arg` or the literal `intent` per vertex.

```

5961 \cs_new_protected:Nn \__spintent_spPoly_points_loop:
5962 {
5963     \int_zero:N \l__spintent_spPoly_pt_idx_int
5964     \seq_map_inline:Nn \l__spintent_spPoly_pt_seq
5965     {
5966         \int_incr:N \l__spintent_spPoly_pt_idx_int
5967         \__spintent_mathmlarg_point:nnn
5968         { pt \int_use:N \l__spintent_spPoly_pt_idx_int }
5969         { spPoly }
5970         { ##1 }
5971     }
5972 }

```

(End of definition for `\__spintent_spPoly_points_loop:`)

`\__spintent_spPoly_exec:nn` Calls `\__spintent_luafun_geo_poly_parse_and_set:n`, reports `spPoly-invalid-points` on failure, then delegates to `\__spintent_spPoly_mathml:`. No `intent` is built here: it needs the reference list, which `\__spintent_spPoly_mathml:` builds after splitting the vertices.

Before processing the $\langle argument \rangle$, it sets the optional $\langle keys \rangle$ passed in ‘`#1`’.

```

5973 \cs_new_protected:Nn \__spintent_spPoly_exec:nn
5974 {
5975     \keys_set:nn { spintent / spPoly } {#1}
5976     \__spintent_luafun_geo_poly_parse_and_set:n { \exp_not:n {#2} }
5977     \str_if_eq:VnT \l__spintent_geo_luaset_error_str { true }
5978     {
5979         \msg_error:nne { spintent } { spPoly-invalid-points }
5980         { \exp_not:n {#2} }
5981     }
5982     \__spintent_spPoly_mathml:
5983 }

```

(End of definition for `\__spintent_spPoly_exec:nn`)

`\__spintent_spPoly_mathml:` Builds the reference list with `\__spintent_spPoly_build_refs:` and the concept `intent` with `\__spintent_spPoly_build_concept:`, then wraps everything in a single native `\MathMLintent`, preceded by `\__spintent_inner_sep:`, a second sibling so the sole `<mrow>` does not collapse into `<math>`, [luamml](#) issue #26.

The keys `print-sym` and `concept/silent` are independent. The key `print-sym` only decides whether the glyph from `\__spintent_spPoly_sym:`, silenced with `\__spintent_luafun_mrow_intent_arg:nn`, is attempted, and the glyph is skipped when `\l__spintent_geo_luaset_poly_sym_str` is empty, which happens for a pentagon and above because they have no symbol. It does not affect whether the concept word is spoken.

```

5984 \cs_new_protected:Nn \__spintent_spPoly_mathml:
5985 {
5986   \__spintent_invisible_sep: % (luamml issues #26)
5987   \__spintent_spPoly_build_refs:
5988   \__spintent_spPoly_build_concept:
5989   \MathMLintent { \__spintent_spPoly_intent_arg_tl }
5990   {
5991     \__spintent_inner_sep:
5992     \bool_if:NT \l__spintent_spPoly_print_sym_bool
5993     {
5994       \str_if_empty:NF \l__spintent_geo_luaset_poly_sym_str
5995       {
5996         \__spintent_luafun_mrow_intent_arg:nn
5997         { } { intent = ":silent" }
5998         { \__spintent_spPoly_sym: }
5999       }
6000     }
6001     \__spintent_spPoly_points_loop:
6002   }
6003 }

```

(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`.

```

6004 \NewDocumentCommand \spPoly { 0{ } m }
6005 {
6006   \mode_if_math:TF
6007   {
6008     \group_begin:
6009     \attribute \l__spintent_scope_attr_int = 1
6010     \__spintent_spPoly_exec:nn {#1} {#2}
6011     \group_end:
6012   }
6013   { \msg_error:nnn { spintent } { need-math-mode } { \spPoly } }
6014 }

```

(End of definition for `\spPoly`. This function is documented on page 30.)

15.52 The commands `\spAfig` and `\spPfig`

The user commands `\spAfig` and `\spPfig` typeset the area and the perimeter of a polygonal figure, using the delimited-optional argument syntax `d()`. The key `print-sym` is never sent to Lua: the symbol type is returned unconditionally in `\l__spintent_geo_luaset_poly_sym_str`, and `expl3` decides whether to wrap it in `\MathMLintent{<:silent>}`.

15.52.1 Definition of variables for `\spAfig` and `\spPfig`

`\l__spintent_geo_final_print_tl` Scratch variables used by `\__spintent_spfig_exec:nnn` to assemble the visual and the `intent`. The variable `\l__spintent_geo_final_print_tl` holds the final visual placed under `\mathcal`'s subscript. The final literal `intent` and the concept forwarded when `read-cmd` is `mathcat`.

```

6015 \tl_new:N \l__spintent_geo_final_print_tl

```

(End of definition for `\l__spintent_geo_final_print_tl`)

`\l__spintent_spAfig_mode_str` 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`.

```

6016 \clist_map_inline:nn { Afig , Pfig }

```

©2026 by Pablo González L

```

6017 {
6018   \str_new:c { l__spintent_sp #1 _mode_str      }
6019   \str_new:c { l__spintent_sp #1 _print_sym_str }
6020   \tl_new:c { l__spintent_sp #1 _prefix_tl      }
6021   \tl_new:c { l__spintent_sp #1 _read_arg_tl    }
6022 }

```

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

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 `l__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.

```

6023 \clist_map_inline:nn { Afig , Pfig }
6024 {
6025   \msg_new:nnnn { spintent } { sp #1 -invalid-points }
6026   {
6027     Invalid~argument~'##2'~for~'\tl_trim_spaces:n {##1}'~\msg_line_context:.
6028   }
6029   {
6030     The~argument~must~be~a~list~of~points~separated~by~commas.
6031   }
6032   \msg_new:nnnn { spintent } { sp #1 -invalid-arg }
6033   {
6034     Invalid~argument~'##2'~for~'\tl_trim_spaces:n {##1}'~\msg_line_context:.
6035   }
6036   {
6037     See~the~manual~for~the~accepted~argument~forms.
6038   }
6039   \msg_new:nnnn { spintent } { sp #1 -sym-letter-conflict }
6040   {
6041     The~key~'print-sym'~cannot~be~combined~with~a~lettered~argument~in~'##1'~\msg_line_context:.
6042   }
6043   {
6044     See~the~documentation~of~'##1'~for~details~on~the~key~'print-sym'.
6045   }
6046   \msg_new:nnnn { spintent } { sp #1 -sym-cardinality-conflict }
6047   {
6048     The~key~'print-sym'~does~not~match~the~vertex~count~in~'##1'~\msg_line_context:.
6049   }
6050   {
6051     See~the~documentation~of~'##1'~for~details~on~the~key~'print-sym'.
6052   }
6053   \msg_new:nnnn { spintent } { sp #1 -read-arg-key }
6054   {
6055     The~key~'read-arg'~has~no~effect~with~a~single~argument~in~'##1'~\msg_line_context:.
6056   }
6057   {
6058     See~the~documentation~of~'##1'~for~details~on~the~key~'read-arg'.
6059   }
6060   }
6061 }
6062 }

```

(End of definition for `spAfig-invalid-points` and others.)

15.52.3 Definition of keys for `\spAfig` and `\spPfig`

`read-cmd` `prefix` `print-sym` `read-arg` `read-sub` `silent` `concept` `concept*` `unknown`

We define the shared `<keys>` `read-cmd`, `prefix`, `print-sym`, `read-arg`, `read-sub`, `silent` and `concept`.

```

6063 \clist_map_inline:nn { Afig , Pfig }
6064 {
6065   \keys_define:nn { spintent / sp #1 }
6066   {
6067     read-cmd .choices:nn = { school , mathcat }
6068     {
6069       \int_case:nn { \l_keys_choice_int }
6070       {

```

```

6071         {1} { \str_set:cn { l__spintent_sp #1 _mode_str } { school } }
6072         {2} { \str_set:cn { l__spintent_sp #1 _mode_str } { mathcat } }
6073     }
6074 },
6075 read-cmd .initial:n = school,
6076 read-cmd .value_required:n = true,
6077 read-cmd / unknown .code:n =
6078     \msg_error:nneee { spintent } { unknown-choice }
6079     { read-cmd } { school,~mathcat } { \exp_not:n {##1} },
6080 prefix .code:n =
6081     { \__spintent_sanitiz_affix:cn { l__spintent_sp #1 _prefix_tl } {##1} },
6082 prefix .initial:n = ,
6083 prefix .value_required:n = true,
6084 print-sym .choices:nn = { auto , triangle , square , none }
6085 {
6086     \int_case:nn { \l_keys_choice_int }
6087     {
6088         {1} { \str_set:cn { l__spintent_sp #1 _print_sym_str } { auto } }
6089         {2} { \str_set:cn { l__spintent_sp #1 _print_sym_str } { triangle } }
6090         {3} { \str_set:cn { l__spintent_sp #1 _print_sym_str } { square } }
6091         {4} { \str_set:cn { l__spintent_sp #1 _print_sym_str } { none } }
6092     }
6093 },
6094 print-sym .initial:n = auto,
6095 print-sym .value_required:n = true,
6096 print-sym / unknown .code:n =
6097     \msg_error:nneee { spintent } { unknown-choice }
6098     { print-sym } { auto,~triangle,~square,~none } { \exp_not:n {##1} },
6099 read-arg .code:n =
6100     { \__spintent_sanitiz_affix:cn { l__spintent_sp #1 _read_arg_tl } {##1} },
6101 read-arg .initial:n = ,
6102 read-arg .value_required:n = true,
6103 read-sub .bool_set:c = { l__spintent_sp #1 _read_sub_bool },
6104 read-sub .initial:n = false,
6105 read-sub .value_required:n = true,
6106 silent .bool_set:c = { l__spintent_sp #1 _silent_bool },
6107 silent .initial:n = false,
6108 silent .value_required:n = true,
6109 concept .bool_set:c = { l__spintent_sp #1 _concept_bool },
6110 concept .initial:n = true,
6111 concept .value_required:n = true,
6112 concept* .meta:n =
6113     {
6114         concept = false,
6115         read-arg = { \str_if_eq:eeTF {#1} { Afig } { _área } { _perímetro } },
6116     },
6117 concept* .value_forbidden:n = true,
6118 unknown .code:n =
6119     {
6120         \str_set:Nn \l__spintent_command_name_str { sp #1 }
6121         \__spintent_unknown_keys_cmd:n {##1}
6122     },
6123 }
6124 }

```

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

15.52.4 Internal functions for \spAfig and \spPfig

`\__spintent_spfig_sym:` The function `\__spintent_spfig_sym:` reads `\l__spintent_geo_print_sym_str` and typesets the corresponding symbol: `\triangle` for the value `triangle`, `\mdlgwhtsquare` for the value `square`.

```

6125 \cs_new_protected:Nn \__spintent_spfig_sym:
6126 {
6127     \str_case:Vn \l__spintent_geo_print_sym_str
6128     {
6129         { triangle } { \triangle }
6130         { square } { \mdlgwhtsquare }
6131     }
6132 }

```

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

```

\__spintent_sfig_op_tl
\__spintent_sfig_intent_tl
\__spintent_sfig_desc_tl
\__spintent_sfig_symbol_tl
\__spintent_sfig_pt_seq
\__spintent_sfig_refs_tl
\__spintent_sfig_intent_arg_tl
\__spintent_sfig_pt_idx_int
\__spintent_geo_print_sym_str
\__spintent_geo_temp_tl

```

The variable `\__spintent_sfig_op_tl` holds «*área*» or «*perímetro*». The variable `\__spintent_sfig_intent_tl` holds the concept word for the vertices case, taken from `read-arg` or the auto-detected figure name. The rest are scratch variables to split `\__spintent_geo_luaset_points_clist` and build the `$pt1, $pt2, ...` reference list.

```

6133 \tl_new:N \__spintent_sfig_op_tl
6134 \tl_new:N \__spintent_sfig_intent_tl
6135 \tl_new:N \__spintent_sfig_desc_tl
6136 \tl_new:N \__spintent_sfig_symbol_tl
6137 \seq_new:N \__spintent_sfig_pt_seq
6138 \tl_new:N \__spintent_sfig_refs_tl
6139 \tl_new:N \__spintent_sfig_intent_arg_tl
6140 \int_new:N \__spintent_sfig_pt_idx_int
6141 \str_new:N \__spintent_geo_print_sym_str
6142 \tl_new:N \__spintent_geo_temp_tl

```

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

```
\__spintent_sfig_exec:nnn
```

Receives the short command name, Afig or Pfig, in ‘#1’ and the $\langle argument \rangle$ in ‘#3’. Calls `\__spintent_luafun_geo_afig_parse_and_set:n` or `\__spintent_luafun_geo_pfig_parse_and_set:n`, reports `spAfig-invalid-arg` or `spPfig-invalid-arg` on failure, and warns with `spAfig-read-arg-key` or `spPfig-read-arg-key` when `read-arg` is given for a bare number/letter argument that has no vertices, where it has no effect and is ignored. Sets `\__spintent_sfig_op_tl` to «*área*»/«*perímetro*», then delegates to `\__spintent_sfig_mathml:n`.

Before processing the $\langle argument \rangle$, it sets the optional $\langle keys \rangle$ passed in ‘#2’.

```

6143 \cs_new_protected:Nn \__spintent_sfig_exec:nnn
6144 {
6145   \keys_set:nn { spintent / sp #1 } { #2 }
6146   \str_if_eq:eeTF { #1 } { Afig }
6147   { \__spintent_luafun_geo_afig_parse_and_set:n { \exp_not:n { #3 } } }
6148   { \__spintent_luafun_geo_pfig_parse_and_set:n { \exp_not:n { #3 } } }
6149   \str_if_eq:VnT \__spintent_geo_luaset_error_str { true }
6150   {
6151     \msg_error:nnee { spintent } { sp #1 -invalid-arg }
6152     { \c_backslash_str sp #1 } { \exp_not:n { #3 } }
6153   }
6154   \clist_if_empty:NT \__spintent_geo_luaset_points_clist
6155   {
6156     \tl_if_empty:cF { \__spintent_sp #1_read_arg_tl }
6157     {
6158       \msg_warning:nne { spintent } { sp #1 -read-arg-key }
6159       { \c_backslash_str sp #1 }
6160     }
6161   }
6162   \str_if_eq:eeTF { #1 } { Afig }
6163   { \tl_set:Nn \__spintent_sfig_op_tl { área } }
6164   { \tl_set:Nn \__spintent_sfig_op_tl { perímetro } }
6165   \str_set:Ne \__spintent_geo_print_sym_str
6166   { \str_use:c { \__spintent_sp #1_print_sym_str } }
6167   \__spintent_sfig_mathml:n { #1 }
6168 }

```

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

```
\__spintent_sfig_build_simple:n
```

Handles the argument-less, table-letter and bare number/letter cases, none of which has vertices. Builds `\__spintent_geo_final_print_tl`, the visual placed under `\mathcal`’s subscript, and `\__spintent_sfig_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.

```

6169 \cs_new_protected:Nn \__spintent_sfig_build_simple:n
6170 {
6171   \tl_if_empty:NTF \__spintent_geo_luaset_print_tl
6172   {
6173     \str_case:VnF \__spintent_geo_print_sym_str
6174     {
6175       { auto }
6176       {
6177         \tl_clear:N \__spintent_geo_final_print_tl

```

```

6178         \tl_clear:N \l__spinttent_spfig_desc_tl
6179     }
6180     { none }
6181     {
6182         \tl_clear:N \l__spinttent_geo_final_print_tl
6183         \tl_clear:N \l__spinttent_spfig_desc_tl
6184     }
6185 }
6186 {
6187     \tl_set:Nn \l__spinttent_geo_final_print_tl { \__spinttent_spfig_sym: }
6188     \tl_set:Ne \l__spinttent_spfig_desc_tl
6189     {
6190         \str_if_eq:VnTF \l__spinttent_geo_print_sym_str { triangle }
6191         { triángulo } { cuadrado }
6192     }
6193 }
6194 }

```

With a bare number/letter, for which `\l__spinttent_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`.

```

6195     {
6196         \str_if_empty:NTF \l__spinttent_geo_luaset_letra_word_str
6197         {
6198             \str_case:VnF \l__spinttent_geo_print_sym_str
6199             {
6200                 { auto }
6201                 {
6202                     \tl_set:NV \l__spinttent_geo_final_print_tl
6203                     \l__spinttent_geo_luaset_print_tl
6204                 }
6205                 { none }
6206                 {
6207                     \tl_set:NV \l__spinttent_geo_final_print_tl
6208                     \l__spinttent_geo_luaset_print_tl
6209                 }
6210             }
6211         {
6212             \tl_set:Nn \l__spinttent_geo_final_print_tl
6213             {
6214                 \__spinttent_spfig_sym:
6215                 \c_math_subscript_token { \l__spinttent_geo_luaset_print_tl }
6216             }
6217         }
6218         \str_case:VnF \l__spinttent_geo_print_sym_str
6219         {
6220             { auto }
6221             {
6222                 \tl_set:Ne \l__spinttent_spfig_desc_tl
6223                 {
6224                     \bool_if:cT { \l__spinttent_sp #1 _read_sub_bool } { sub - }
6225                     \l__spinttent_geo_luaset_sub_spoken_str
6226                 }
6227             }
6228             { none }
6229             {
6230                 \tl_set:Ne \l__spinttent_spfig_desc_tl
6231                 {
6232                     \bool_if:cT { \l__spinttent_sp #1 _read_sub_bool } { sub - }
6233                     \l__spinttent_geo_luaset_sub_spoken_str
6234                 }
6235             }
6236         }
6237     {
6238         \tl_set:Ne \l__spinttent_spfig_desc_tl
6239         {
6240             del-
6241             \str_if_eq:VnTF \l__spinttent_geo_print_sym_str { triangle }
6242             { triángulo } { cuadrado }
6243         }

```

```

6244         \bool_if:cT { l__spintent_sp #1 _read_sub_bool } { sub - }
6245         \l__spintent_geo_luaset_sub_spoken_str
6246     }
6247 }
6248 }

```

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.

```

6249 {
6250     \str_case:VnF \l__spintent_geo_print_sym_str
6251     {
6252         { auto } { }
6253         { none } { }
6254     }
6255     {
6256         \msg_error:nne { spintent } { sp #1 -sym-letter-conflict }
6257         { \c_backslash_str sp #1 }
6258     }
6259     \tl_set:NV \l__spintent_geo_final_print_tl \l__spintent_geo_luaset_print_tl
6260     \tl_set:Ne \l__spintent_spfig_desc_tl
6261     {
6262         \str_if_empty:NF \l__spintent_geo_luaset_letra_conector_str
6263         { \l__spintent_geo_luaset_letra_conector_str - }
6264         \l__spintent_geo_luaset_letra_word_str
6265         \str_if_empty:NF \l__spintent_geo_luaset_sub_spoken_str
6266         {
6267             -
6268             \bool_if:cT { l__spintent_sp #1 _read_sub_bool } { sub - }
6269             \l__spintent_geo_luaset_sub_spoken_str
6270         }
6271     }
6272 }
6273 }

```

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`.

```

6274     \tl_set:Ne \l__spintent_spfig_intent_tl
6275     {
6276         \tl_if_empty:NTF \l__spintent_spfig_desc_tl
6277         { \l__spintent_spfig_op_tl - }
6278         { \l__spintent_spfig_op_tl - \l__spintent_spfig_desc_tl - }
6279     }
6280     \bool_if:cTF { l__spintent_sp #1 _silent_bool }
6281     { \tl_set:Nn \l__spintent_spfig_intent_arg_tl { _- } }
6282     {
6283         \bool_if:cTF { l__spintent_sp #1 _concept_bool }
6284         {
6285             \tl_set:Ne \l__spintent_spfig_intent_arg_tl
6286             {
6287                 -
6288                 \tl_if_empty:cF { l__spintent_sp #1 _prefix_tl }
6289                 { \tl_use:c { l__spintent_sp #1 _prefix_tl } - }
6290                 \l__spintent_spfig_intent_tl
6291             }
6292         }
6293         {
6294             \tl_set_eq:NN \l__spintent_geo_temp_tl \l__spintent_spfig_desc_tl
6295             \str_if_empty:NF \l__spintent_geo_luaset_letra_conector_str
6296             {
6297                 \tl_replace_once:Nee \l__spintent_geo_temp_tl
6298                 { \l__spintent_geo_luaset_letra_conector_str - } { }
6299             }
6300             \tl_set:Ne \l__spintent_spfig_intent_arg_tl { _ \l__spintent_geo_temp_tl - }

```

```

6301     }
6302   }
6303 }

```

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

`\__spintent_sfig_build_refs:`

Splits `\l__spintent_geo_luaset_points_clist` into `\l__spintent_sfig_pt_seq` and walks it once to build `\l__spintent_sfig_refs_tl`, the comma-separated `$pt1, $pt2, \dots` reference list, naming each vertex by position.

```

6304 \cs_new_protected:Nn \__spintent_sfig_build_refs:
6305 {
6306   \seq_set_from_clist:NN \l__spintent_sfig_pt_seq
6307     \l__spintent_geo_luaset_points_clist
6308   \tl_clear:N \l__spintent_sfig_refs_tl
6309   \int_zero:N \l__spintent_sfig_pt_idx_int
6310   \seq_map_inline:Nn \l__spintent_sfig_pt_seq
6311     {
6312       \int_incr:N \l__spintent_sfig_pt_idx_int
6313       \tl_if_empty:NF \l__spintent_sfig_refs_tl
6314         { \tl_put_right:Nn \l__spintent_sfig_refs_tl { , } }
6315       \tl_put_right:Ne \l__spintent_sfig_refs_tl
6316         { $ pt \int_use:N \l__spintent_sfig_pt_idx_int }
6317     }
6318 }

```

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

`\__spintent_sfig_build_concept_vertices:n`

Handles the vertices case. Receives the short command name in ‘#1’. Must be called after `\__spintent_sfig_build_refs:`.

The key `read-arg`, when given, overrides the auto-detected figure name in `\l__spintent_sfig_intent_tl`: with `concept=true` it is still prefixed with `área-del-/perímetro-del-`, with `concept=false` it is used verbatim. With `read-arg` empty, `\l__spintent_sfig_intent_tl` is `\l__spintent_geo_luaset_concept_str`.

```

6319 \cs_new_protected:Nn \__spintent_sfig_build_concept_vertices:n
6320 {
6321   \tl_if_empty:cTF { \l__spintent_sp #1 _read_arg_tl }
6322   {
6323     \tl_set:Ne \l__spintent_sfig_intent_tl { \l__spintent_geo_luaset_concept_str }
6324   }
6325   {
6326     \bool_if:cTF { \l__spintent_sp #1 _concept_bool }
6327     {
6328       \tl_set:Ne \l__spintent_sfig_intent_tl
6329       {
6330         \l__spintent_sfig_op_tl -del-
6331         \tl_use:c { \l__spintent_sp #1 _read_arg_tl }
6332       }
6333     }
6334     {
6335       \tl_set:Ne \l__spintent_sfig_intent_tl
6336       { \tl_use:c { \l__spintent_sp #1 _read_arg_tl } }
6337     }
6338   }

```

Determines `\l__spintent_sfig_symbol_tl`, which is empty when none applies: for `print-sym=auto`, the symbol matching `\l__spintent_geo_luaset_poly_sym_str`, `\triangle`/`\mdlgwhtsquare`, or none when that is empty; `print-sym=none` always clears it; a forced `print-sym` requires it to match the actual vertex count, reporting `spAfig-sym-cardinality-conflict` or `spPfig-sym-cardinality-conflict` otherwise.

```

6339 \str_case:VnF \l__spintent_geo_print_sym_str
6340 {
6341   { auto }
6342   {
6343     \str_case:Vn \l__spintent_geo_luaset_poly_sym_str
6344     {
6345       { triangle }
6346       { \tl_set:Nn \l__spintent_sfig_symbol_tl { \triangle } }
6347       { square }
6348       { \tl_set:Nn \l__spintent_sfig_symbol_tl { \mdlgwhtsquare } }
6349     }

```

```

6350         \str_if_empty:NT \l__spintent_geo_luaset_poly_sym_str
6351         { \tl_clear:N \l__spintent_spfig_symbol_tl }
6352     }
6353     { none }
6354     { \tl_clear:N \l__spintent_spfig_symbol_tl }
6355 }
6356 {
6357     \str_if_eq:VUTF \l__spintent_geo_print_sym_str
6358     \l__spintent_geo_luaset_poly_sym_str
6359     { \tl_set:Nn \l__spintent_spfig_symbol_tl { \__spintent_spfig_sym: } }
6360     {
6361         \msg_error:nne { spintent } { sp #1 -sym-cardinality-conflict }
6362         { \c_backslash_str sp #1 }
6363     }
6364 }

```

The key `silent` sets `\l__spintent_spfig_intent_arg_tl` to ‘-’. Otherwise, with `read-arg` empty, `concept` chooses between `\l__spintent_spfig_intent_tl:prefix(refs)`, with `prefix` placed after the leading `_` when given, and `_(refs)` alone; with `read-arg` given, it is always `\l__spintent_spfig_intent_tl:prefix(refs)`.

```

6365     \bool_if:cTF { l__spintent_sp #1 _silent_bool }
6366     {
6367         \tl_set:Nn \l__spintent_spfig_intent_arg_tl { _- }
6368     }
6369     {
6370         \tl_if_empty:cTF { l__spintent_sp #1 _read_arg_tl }
6371         {
6372             \bool_if:cTF { l__spintent_sp #1 _concept_bool }
6373             {
6374                 \tl_set:Ne \l__spintent_spfig_intent_arg_tl
6375                 {
6376                     -
6377                     \tl_if_empty:cF { l__spintent_sp #1 _prefix_tl }
6378                     { \tl_use:c { l__spintent_sp #1 _prefix_tl } - }
6379                     \l__spintent_spfig_intent_tl :prefix( \l__spintent_spfig_refs_tl )
6380                 }
6381             }
6382             {
6383                 \tl_set:Ne \l__spintent_spfig_intent_arg_tl
6384                 { _ ( \l__spintent_spfig_refs_tl ) }
6385             }
6386         }
6387         {
6388             \tl_set:Ne \l__spintent_spfig_intent_arg_tl
6389             {
6390                 -
6391                 \tl_if_empty:cF { l__spintent_sp #1 _prefix_tl }
6392                 { \tl_use:c { l__spintent_sp #1 _prefix_tl } - }
6393                 \l__spintent_spfig_intent_tl :prefix( \l__spintent_spfig_refs_tl )
6394             }
6395         }
6396     }
6397 }

```

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

`\__spintent_spfig_points_loop:n`

Walks `\l__spintent_spfig_pt_seq` a second time, calling `\__spintent_mathmlarg_point:nnn` once per vertex with its `ptn` reference name, looked up under `spAfig` or `spPfig`. Receives the short command name in ‘#1’.

```

6398 \cs_new_protected:Nn \__spintent_spfig_points_loop:n
6399 {
6400     \int_zero:N \l__spintent_spfig_pt_idx_int
6401     \seq_map_inline:Nn \l__spintent_spfig_pt_seq
6402     {
6403         \int_incr:N \l__spintent_spfig_pt_idx_int
6404         \__spintent_mathmlarg_point:nnn
6405         { pt \int_use:N \l__spintent_spfig_pt_idx_int }
6406         { sp #1 }
6407         { ##1 }
6408     }
6409 }

```

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

`\__spintent_spfig_mathml:n` Receives the short command name in ‘#1’. Branches on whether `\l__spintent_geo_luaset_points_clist` is empty.

Without vertices, `\__spintent_spfig_build_simple:n` builds the content. A `\MathMLintent` wraps it only when `concept` is `true`; with `concept=false` the content is typeset with no `intent`.

With vertices, `\__spintent_spfig_build_refs:` builds the reference list and `\__spintent_spfig_build_concept` builds the concept `intent`, applied with a single native `\MathMLintent`. Inside, `\__spintent_mathcal:n{⟨A | P⟩}` is silenced with `\__spintent_luafun_mrow_intent_arg:nn`; the same is done with `\l__spintent_spfig_symbol_tl` when it is non-empty; the sequence is then walked by `\__spintent_spfig_points_loop:n`. All of this is nested inside a subscript, `\c_math_subscript_token`, which the recursive lookup of the shared `mathml_filter` callback reaches through the sub field of the node.

The function `\__spintent_invisible_sep:` precedes both branches. It adds a second sibling so that the sole `<mrow>` does not collapse into `<math>`.

```

6410 \cs_new_protected:Nn \__spintent_spfig_mathml:n
6411 {
6412   \__spintent_invisible_sep: % (luamml issues #26)
6413   \clist_if_empty:NTF \l__spintent_geo_luaset_points_clist
6414   {
6415     \__spintent_spfig_build_simple:n {#1}
6416     \MathMLintent { \l__spintent_spfig_intent_arg_tl }
6417     {
6418       \__spintent_inner_sep:
6419       \str_if_eq:eeTF {#1} { Afig } { \__spintent_mathcal:n { A } }
6420       { \__spintent_mathcal:n { P } }
6421       \tl_if_empty:NF \l__spintent_geo_final_print_tl
6422       {
6423         \c_math_subscript_token { \l__spintent_geo_final_print_tl }
6424       }
6425     }
6426   }
6427   {
6428     \__spintent_spfig_build_refs:
6429     \__spintent_spfig_build_concept_vertices:n {#1}
6430     \MathMLintent { \l__spintent_spfig_intent_arg_tl }
6431     {
6432       \__spintent_inner_sep:
6433       \__spintent_luafun_mrow_intent_arg:nn
6434       { } { intent = ":silent" }
6435       {
6436         \str_if_eq:eeTF {#1} { Afig } { \__spintent_mathcal:n { A } }
6437         { \__spintent_mathcal:n { P } }
6438       }
6439       \c_math_subscript_token
6440       {
6441         \tl_if_empty:NF \l__spintent_spfig_symbol_tl
6442         {
6443           \__spintent_luafun_mrow_intent_arg:nn
6444           { } { intent = ":silent" }
6445           { \tl_use:N \l__spintent_spfig_symbol_tl }
6446         }
6447         \__spintent_spfig_points_loop:n {#1}
6448       }
6449     }
6450   }
6451 }

```

(End of definition for `\__spintent_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`.

```

6452 \NewDocumentCommand \spAfig { 0{} d() }
6453 {
6454   \mode_if_math:TF
6455   {
6456     \group_begin:
6457     \attribute \l__spintent_scope_attr_int = 1
6458     \tl_if_no_value:NTF {#2}

```

```

6459         { \__spintent_spfig_exec:nnn { Afig } {#1} { } }
6460         { \__spintent_spfig_exec:nnn { Afig } {#1} {#2} }
6461     \group_end:
6462 }
6463 { \msg_error:nnn { spintent } { need-math-mode } { \spAfig } }
6464 }
6465 \NewDocumentCommand \spPfig { 0{} d() }
6466 {
6467     \mode_if_math:TF
6468     {
6469         \group_begin:
6470         \attribute \__spintent_scope_attr_int = 1
6471         \tl_if_novalue:nTF {#2}
6472         { \__spintent_spfig_exec:nnn { Pfig } {#1} { } }
6473         { \__spintent_spfig_exec:nnn { Pfig } {#1} {#2} }
6474         \group_end:
6475     }
6476     { \msg_error:nnn { spintent } { need-math-mode } { \spPfig } }
6477 }

```

(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`

The variables `\l__spintent_sp_rad_diam_letra_tl` and `\l__spintent_sp_rad_diam_word_tl` are scratch variables used by `\__spintent_sp_rad_diam_mathml:n`. `\l__spintent_sp_rad_diam_read_arg_full_tl` is used by `\__spintent_sp_rad_diam_read_arg_print:n` to hold the text to voice when `read-arg` is given.

```

6478 \tl_new:N \l__spintent_sp_rad_diam_letra_tl
6479 \tl_new:N \l__spintent_sp_rad_diam_word_tl
6480 \tl_new:N \l__spintent_sp_rad_diam_read_arg_full_tl

```

(End of definition for `\l__spintent_sp_rad_diam_letra_tl`, `\l__spintent_sp_rad_diam_word_tl`, and `\l__spintent_sp_rad_diam_read_arg_full_tl`.)

```

\l__spintent_sprad_mode_str
\l__spintent_sprad_sub_index_str
\l__spintent_sprad_read_arg_tl
\l__spintent_spdiam_mode_str
\l__spintent_spdiam_sub_index_str
\l__spintent_spdiam_read_arg_tl

```

Each command has its own copy of the string variables `_mode_str` and `_sub_index_str` and of the token list variable `_read_arg_tl`, which hold the values of the keys `read-cmd`, `sub-index` and `read-arg`.

```

6481 \clist_map_inline:nn { rad , diam }
6482 {
6483     \str_new:c { l__spintent_sp #1 _mode_str }
6484     \str_new:c { l__spintent_sp #1 _sub_index_str }
6485     \tl_new:c { l__spintent_sp #1 _read_arg_tl }
6486 }

```

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

15.53.2 Definition of keys for `\sprad` and `\spdiam`

We define the shared *(keys)* `read-cmd`, `upper-case`, `sub-index`, `read-sub`, `read-arg`, `concept`, `silent` and `print-sym`.

```

6487 \clist_map_inline:nn { rad , diam }
6488 {
6489     \keys_define:nn { spintent / sp #1 }
6490     {
6491         read-cmd .choices:nn = { school , mathcat }
6492         {
6493             \int_case:nn { \l_keys_choice_int }
6494             {
6495                 {1} { \str_set:cn { l__spintent_sp #1 _mode_str } { school } }
6496                 {2} { \str_set:cn { l__spintent_sp #1 _mode_str } { mathcat } }
6497             }
6498         },
6499         read-cmd .initial:n = school,
6500         read-cmd .value_required:n = true,
6501         read-cmd / unknown .code:n =
6502         \msg_error:nnee { spintent } { unknown-choice }

```

```

6503         { read-cmd } { school,~mathcat } { \exp_not:n {##1} },
6504         upper-case .bool_set:c = { l__spintent_sp #1 _upper_case_bool },
6505         upper-case .initial:n = false,
6506         upper-case .value_required:n = true,
6507         sub-index .code:n =
6508         { \str_set:cn { l__spintent_sp #1 _sub_index_str } {##1} },
6509         sub-index .initial:n = ,
6510         sub-index .value_required:n = true,
6511         read-sub .bool_set:c = { l__spintent_sp #1 _read_sub_bool },
6512         read-sub .initial:n = false,
6513         read-sub .value_required:n = true,
6514         read-arg .code:n =
6515         { \__spintent_sanitize_affix:cn { l__spintent_sp #1 _read_arg_tl } {##1} },
6516         read-arg .initial:n = ,
6517         read-arg .value_required:n = true,
6518         concept .bool_set:c = { l__spintent_sp #1 _concept_bool },
6519         concept .initial:n = true,
6520         concept .value_required:n = true,
6521         silent .bool_set:c = { l__spintent_sp #1 _silent_bool },
6522         silent .initial:n = false,
6523         silent .value_required:n = true,
6524         print-sym .bool_set:c = { l__spintent_sp #1 _print_sym_bool },
6525         print-sym .initial:n = false,
6526         print-sym .value_required:n = true,
6527         unknown .code:n =
6528         {
6529             \str_set:Nn \l__spintent_command_name_str { sp #1 }
6530             \__spintent_unknown_keys_cmd:n {##1}
6531         },
6532     }
6533 }

```

🔑 The key `print-sym` is defined for both commands for simplicity, but has no effect on `\sprad`.

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

15.53.3 Internal functions for `\sprad` and `\spdiam`

`\__spintent_sp_rad_diam_mathml:n`

The function `\__spintent_sp_rad_diam_mathml:n` sets `\l__spintent_sp_rad_diam_letra_tl` to the bare letter, `r` or `R` for `\sprad` and `d` or `D` for `\spdiam`, according to `upper-case`; for `\spdiam` with `print-sym=true`, it silently typesets `\diameter` first. Receives the short command name in ‘`#1`’: `rad` or `diam`.

```

6534 \cs_new_protected:Nn \__spintent_sp_rad_diam_mathml:n
6535 {
6536     \bool_if:cTF { l__spintent_sp #1 _upper_case_bool }
6537     {
6538         \str_if_eq:eeTF {##1} { rad }
6539         { \tl_set:Nn \l__spintent_sp_rad_diam_letra_tl { R } }
6540         { \tl_set:Nn \l__spintent_sp_rad_diam_letra_tl { D } }
6541     }
6542     {
6543         \str_if_eq:eeTF {##1} { rad }
6544         { \tl_set:Nn \l__spintent_sp_rad_diam_letra_tl { r } }
6545         { \tl_set:Nn \l__spintent_sp_rad_diam_letra_tl { d } }
6546     }
6547     \__spintent_invisible_sep:
6548     \str_if_eq:eeT {##1} { diam }
6549     {
6550         \bool_if:NT \l__spintent_spdiam_print_sym_bool
6551         { \MathMLintent { :silent } { \diameter } }
6552     }

```

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.

```

6553     \bool_if:cTF { l__spintent_sp #1 _silent_bool }
6554     {
6555         \str_if_empty:cTF { l__spintent_sp #1 _sub_index_str }
6556         {
6557             \__spintent_luafun_mrow_intent_arg:nn
6558             { } { intent = "- " }
6559             { \l__spintent_sp_rad_diam_letra_tl }
6560         }
6561         {

```

```

6562 \MathMLintent { _- }
6563 {
6564   \__spintent_inner_sep:
6565   \l__spintent_sp_rad_diam_letra_tl
6566   \c_math_subscript_token
6567   {
6568     \MathMLintent { :silent }
6569     { \str_use:c { l__spintent_sp #1 _sub_index_str } }
6570   }
6571 }
6572 }
6573 }

```

Otherwise, with `concept=true`, the default, it verbalizes “*radio*” or “*diámetro*”, or the text of `read-arg` instead of that word when given, followed by the `sub-index` if present, as `sub-⟨value⟩` or `⟨value⟩` alone according to `read-sub`; the letter is not read. With no `sub-index`, the shared queue tags the letter alone. With one, for `read-cmd=school` the letter and the subscript are wrapped together in one native `\MathMLintent` with the subscript silenced, and for `read-cmd=mathcat` the shared queue tags the letter alone and the subscript is left bare.

```

6574 {
6575   \tl_if_empty:cTF { l__spintent_sp #1 _read_arg_tl }
6576   {
6577     \bool_if:cTF { l__spintent_sp #1 _concept_bool }
6578     {
6579       \str_if_eq:eeTF {#1} { rad }
6580       { \tl_set:Nn \l__spintent_sp_rad_diam_word_tl { radio } }
6581       { \tl_set:Nn \l__spintent_sp_rad_diam_word_tl { diámetro } }
6582       \str_if_empty:cTF { l__spintent_sp #1 _sub_index_str }
6583       {
6584         \__spintent_luafun_mrow_intent_arg:nn
6585         { } { intent = "_ \l__spintent_sp_rad_diam_word_tl :nofix" }
6586         { \l__spintent_sp_rad_diam_letra_tl }
6587       }
6588       {
6589         \str_if_eq:eeTF
6590         { \str_use:c { l__spintent_sp #1 _mode_str } } { school }
6591         {
6592           \MathMLintent
6593           {
6594             _ \l__spintent_sp_rad_diam_word_tl -
6595             \bool_if:cT { l__spintent_sp #1 _read_sub_bool } { sub- }
6596             \str_use:c { l__spintent_sp #1 _sub_index_str }
6597           }
6598           {
6599             \__spintent_inner_sep:
6600             \l__spintent_sp_rad_diam_letra_tl
6601             \c_math_subscript_token
6602             {
6603               \MathMLintent { :silent }
6604               { \str_use:c { l__spintent_sp #1 _sub_index_str } }
6605             }
6606           }
6607         }
6608         {
6609           \__spintent_luafun_mrow_intent_arg:nn
6610           { } { intent = "_ \l__spintent_sp_rad_diam_word_tl :nofix" }
6611           { \l__spintent_sp_rad_diam_letra_tl }
6612           \c_math_subscript_token
6613           { \str_use:c { l__spintent_sp #1 _sub_index_str } }
6614         }
6615       }
6616     }

```

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.

```

6617 {
6618   \str_if_empty:cTF { l__spintent_sp #1 _sub_index_str }
6619   {
6620     \str_if_eq:eeTF

```

```

6621         { \str_use:c { l__spintent_sp #1 _mode_str } } { school }
6622     {
6623         \__spintent_luafun_mrow_intent_arg:nn
6624         { } { intent = "_ \l__spintent_sp_rad_diam_letra_tl -" }
6625         { \l__spintent_sp_rad_diam_letra_tl }
6626     }
6627     { \l__spintent_sp_rad_diam_letra_tl }
6628 }
6629 {
6630     \str_if_eq:eeTF
6631     { \str_use:c { l__spintent_sp #1 _mode_str } } { school }
6632     {
6633         \MathMLintent
6634         {
6635             _ \l__spintent_sp_rad_diam_letra_tl -
6636             \bool_if:cT { l__spintent_sp #1 _read_sub_bool } { sub- }
6637             \str_use:c { l__spintent_sp #1 _sub_index_str }
6638         }
6639         {
6640             \__spintent_inner_sep:
6641             \l__spintent_sp_rad_diam_letra_tl
6642             \c_math_subscript_token
6643             {
6644                 \MathMLintent { :silent }
6645                 { \str_use:c { l__spintent_sp #1 _sub_index_str } }
6646             }
6647         }
6648     }
6649     {
6650         \l__spintent_sp_rad_diam_letra_tl
6651         \c_math_subscript_token
6652         { \str_use:c { l__spintent_sp #1 _sub_index_str } }
6653     }
6654 }
6655 }
6656 }
6657 { \__spintent_sp_rad_diam_read_arg_print:n {#1} }
6658 }
6659 }

```

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

`\__spintent_sp_rad_diam_read_arg_print:n`

With `read_arg` given, `sub-index/read-sub` are ignored for the *spoken* content; the subscript, if any, still displays, silenced. With `concept=true`, the default, `\l__spintent_sp_rad_diam_read_arg_full_tl` becomes “radio”/“diámetro” followed by `read_arg`’s text; with `false`, `read_arg`’s text alone. Combined natively, regardless of `read-cmd`: with no `sub-index`, the shared queue tags the letter alone; with one, letter and subscript are wrapped together in one native `\MathMLintent`.

```

6660 \cs_new_protected:Nn \__spintent_sp_rad_diam_read_arg_print:n
6661 {
6662     \bool_if:cTF { l__spintent_sp #1 _concept_bool }
6663     {
6664         \str_if_eq:eeTF {#1} { rad }
6665         {
6666             \tl_set:Nc \l__spintent_sp_rad_diam_read_arg_full_tl
6667             { radio- \tl_use:c { l__spintent_sp #1 _read_arg_tl } }
6668         }
6669         {
6670             \tl_set:Nc \l__spintent_sp_rad_diam_read_arg_full_tl
6671             { diámetro- \tl_use:c { l__spintent_sp #1 _read_arg_tl } }
6672         }
6673     }
6674     {
6675         \tl_set_eq:Nc \l__spintent_sp_rad_diam_read_arg_full_tl
6676         { l__spintent_sp #1 _read_arg_tl }
6677     }
6678     \str_if_empty:cTF { l__spintent_sp #1 _sub_index_str }
6679     {
6680         \__spintent_luafun_mrow_intent_arg:nn
6681         { } { intent = "_ \l__spintent_sp_rad_diam_read_arg_full_tl :nofix" }
6682         { \l__spintent_sp_rad_diam_letra_tl }
6683     }

```

```

6684     {
6685       \MathMLintent
6686       { _ \l__spinttent_sp_rad_diam_read_arg_full_tl :nofix }
6687       {
6688         \l__spinttent_inner_sep:
6689         \l__spinttent_sp_rad_diam_letra_tl
6690         \c_math_subscript_token
6691         {
6692           \MathMLintent { :silent }
6693           { \str_use:c { l__spinttent_sp #1 _sub_index_str } }
6694         }
6695       }
6696     }
6697   }

```

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

`\l__spinttent_sp_rad_diam_exec:nn`

The function `\l__spinttent_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 `\l__spinttent_sp_rad_diam_mathml:n`.

```

6698 \cs_new_protected:Nn \l__spinttent_sp_rad_diam_exec:nn
6699 {
6700   \keys_set:nn { spinttent / sp #1 } {#2}
6701   \l__spinttent_sp_rad_diam_mathml:n {#1}
6702 }

```

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

`\sprad`
`\spdiam`

Both public commands `\sprad` and `\spdiam` enforces *math mode*, opens a local group, and delegates execution to `\l__spinttent_sp_rad_diam_exec:nn`.

```

6703 \NewDocumentCommand \sprad { 0{} }
6704 {
6705   \mode_if_math:TF
6706   {
6707     \group_begin:
6708     \attribute \l__spinttent_scope_attr_int = 1
6709     \l__spinttent_sp_rad_diam_exec:nn { rad } {#1}
6710     \group_end:
6711   }
6712   { \msg_error:nnn { spinttent } { need-math-mode } { \sprad } }
6713 }
6714 \NewDocumentCommand \spdiam { 0{} }
6715 {
6716   \mode_if_math:TF
6717   {
6718     \group_begin:
6719     \attribute \l__spinttent_scope_attr_int = 1
6720     \l__spinttent_sp_rad_diam_exec:nn { diam } {#1}
6721     \group_end:
6722   }
6723   { \msg_error:nnn { spinttent } { need-math-mode } { \spdiam } }
6724 }

```

(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`

`\l__spinttent_geo_luaset_circ_radio_type_str`
`\l__spinttent_geo_luaset_circ_radio_raw_str`

Shared outputs of `\l__spinttent_luafun_geo_circ_parse_and_set:n`. The variable `\l__spinttent_geo_luaset_circ_radio_type_str` holds one of *number*, *measure*, *label*, *segment*, or an empty string when no radius was given. `\l__spinttent_geo_luaset_circ_radio_raw_str` holds the corresponding raw text, ready to be forwarded to the subcommand that renders it.

```

6725 \str_new:N \l__spinttent_geo_luaset_circ_radio_type_str
6726 \str_new:N \l__spinttent_geo_luaset_circ_radio_raw_str

```

(End of definition for `\l__spintent_geo_luaset_circ_radio_type_str` and `\l__spintent_geo_luaset_circ_radio_raw_str`.)

The variable `\l__spintent_spcirc_mode_str` stores the value set by the key `read-cmd`; the variable `\l__spintent_spcirc_prefix_tl` stores the value set by the key `prefix`. The variable `\l__spintent_spcirc_print_sym_str` stores the value set by the key `print-sym`, either `usual` or `odot`. The variable `\l__spintent_spcirc_read_sty_str` stores the value set by the key `read-sty`, either `circun` or `circulo`. The variable `\l__spintent_spcirc_radio_sty_str` stores the value set by the key `radio-sty`, only meaningful when the radius is a segment. The variable `\l__spintent_spcirc_sub_index_str` stores the value set by the key `sub-index`. The variable `\l__spintent_spcirc_articulo_str` is set internally by `\__spintent_spcirc_word_and_article`: to `la` or `e``l`, matching the grammatical gender of the chosen word. The variable `\l__spintent_spcirc_read_arg_tl` stores the value set by the key `read-arg`.

```
6727 \str_new:N \l__spintent_spcirc_mode_str
6728 \tl_new:N \l__spintent_spcirc_prefix_tl
6729 \str_new:N \l__spintent_spcirc_print_sym_str
6730 \str_new:N \l__spintent_spcirc_read_sty_str
6731 \str_new:N \l__spintent_spcirc_radio_sty_str
6732 \str_new:N \l__spintent_spcirc_sub_index_str
6733 \str_new:N \l__spintent_spcirc_articulo_str
6734 \tl_new:N \l__spintent_spcirc_read_arg_tl
```

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

Scratch variables used by `\__spintent_spcirc_exec:nn` to assemble the chosen word, which is `circunferencia/círculo` or the `read-arg` override, and the final literal `intent` string.

```
6735 \tl_new:N \l__spintent_spcirc_word_tl
```

(End of definition for `\l__spintent_spcirc_word_tl`.)

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.

```
6736 \msg_new:nnnn { spintent } { spcirc-invalid-arg }
6737 {
6738   Invalid~argument~'#1'~for~'\token_to_str:N \spcirc'~'\msg_line_context:.
6739 }
6740 {
6741   The~argument~must~have~the~form~'(center)'~or~'(center,~radius)'.
6742 }
```

(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`.

```
6743 \keys_define:nn { spintent / spcirc }
6744 {
6745   read-cmd .choices:nn = { school , mathcat }
6746   {
6747     \int_case:nn { \l_keys_choice_int }
6748     {
6749       {1}
6750       {
6751         \str_set:Nn \l__spintent_spcirc_mode_str { school }
6752         \keys_set:nn { spintent / spside } { read-cmd = school }
6753         \keys_set:nn { spintent / spmside } { read-cmd = school }
6754       }
6755       {2}
6756       {
6757         \str_set:Nn \l__spintent_spcirc_mode_str { mathcat }
6758         \keys_set:nn { spintent / spside } { read-cmd = mathcat }
6759         \keys_set:nn { spintent / spmside } { read-cmd = mathcat }
6760       }
6761     }
6762   },
6763   read-cmd .initial:n = school,
```

```

6764 read-cmd .value_required:n = true,
6765 read-cmd / unknown .code:n =
6766   \msg_error:nneee { spintent } { unknown-choice }
6767   { read-cmd } { school,~mathcat } { \exp_not:n {#1} },
6768 prefix .code:n =
6769   { \__spintent_sanitizetize_affix:Nn \__spintent_spcirc_prefix_tl {#1} },
6770 prefix .initial:n = ,
6771 prefix .value_required:n = true,
6772 print-sym .choices:nn = { usual , odot }
6773 {
6774   \int_case:nn { \l_keys_choice_int }
6775   {
6776     {1} { \str_set:Nn \__spintent_spcirc_print_sym_str { usual } }
6777     {2} { \str_set:Nn \__spintent_spcirc_print_sym_str { odot } }
6778   }
6779 },
6780 print-sym .initial:n = odot,
6781 print-sym .value_required:n = true,
6782 print-sym / unknown .code:n =
6783   \msg_error:nneee { spintent } { unknown-choice }
6784   { print-sym } { usual,~odot } { \exp_not:n {#1} },
6785 read-sty .choices:nn = { circun , circulo }
6786 {
6787   \int_case:nn { \l_keys_choice_int }
6788   {
6789     {1} { \str_set:Nn \__spintent_spcirc_read_sty_str { circun } }
6790     {2} { \str_set:Nn \__spintent_spcirc_read_sty_str { circulo } }
6791   }
6792 },
6793 read-sty .initial:n = circun,
6794 read-sty .value_required:n = true,
6795 read-sty / unknown .code:n =
6796   \msg_error:nneee { spintent } { unknown-choice }
6797   { read-sty } { circun,~circulo } { \exp_not:n {#1} },
6798 diameter .bool_set:N = \__spintent_spcirc_diameter_bool,
6799 diameter .initial:n = false,
6800 diameter .value_required:n = true,
6801 radio-sty .choices:nn = { side , mside }
6802 {
6803   \int_case:nn { \l_keys_choice_int }
6804   {
6805     {1} { \str_set:Nn \__spintent_spcirc_radio_sty_str { side } }
6806     {2} { \str_set:Nn \__spintent_spcirc_radio_sty_str { mside } }
6807   }
6808 },
6809 radio-sty .initial:n = mside,
6810 radio-sty .value_required:n = true,
6811 radio-sty / unknown .code:n =
6812   \msg_error:nneee { spintent } { unknown-choice }
6813   { radio-sty } { side,~mside } { \exp_not:n {#1} },
6814 sub-index .code:n =
6815   { \str_set:Nn \__spintent_spcirc_sub_index_str {#1} },
6816 sub-index .initial:n = ,
6817 sub-index .value_required:n = true,
6818 read-sub .bool_set:N = \__spintent_spcirc_read_sub_bool,
6819 read-sub .initial:n = false,
6820 read-sub .value_required:n = true,
6821 read-arg .code:n =
6822   { \__spintent_sanitizetize_affix:Nn \__spintent_spcirc_read_arg_tl {#1} },
6823 read-arg .initial:n = ,
6824 read-arg .value_required:n = true,
6825 silent .bool_set:N = \__spintent_spcirc_silent_bool,
6826 silent .initial:n = false,
6827 silent .value_required:n = true,
6828 concept .bool_set:N = \__spintent_spcirc_concept_bool,
6829 concept .initial:n = true,
6830 concept .value_required:n = true,
6831 spside .meta:nn = { spintent / spside } {#1},
6832 spside .value_required:n = true,
6833 spmside .meta:nn = { spintent / spmside } {#1},
6834 spmside .value_required:n = true,

```

```

6835     unknown     .code:n =
6836     {
6837         \str_set:Nn \l__spintent_command_name_str { spcirc }
6838         \__spintent_unknown_keys_cmd:n {#1}
6839     },
6840 }

```

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

15.54.4 Internal functions for `\spcirc`

The variable `\l__spintent_spcirc_has_arg_bool` stores whether the optional argument `d()` was given. The variable `\l__spintent_spcirc_intent_tl` stores the word `circunferencia` or `círculo`, followed by the `sub-index` suffix if present. The variable `\l__spintent_spcirc_connector_tl` stores `de-centro-en`. The variables `\l__spintent_spcirc_intent_arg_tl` and `\l__spintent_spcirc_connector_arg_tl` store the `intent` passed to `\MathMLintent` for the word and for the connector, where the center is referenced as `$pt`. The variable `\l__spintent_spcirc_radio_arg_tl` stores the `intent` of the radius, spoken on the delimiter `,`.

```

6841 \bool_new:N \l__spintent_spcirc_has_arg_bool
6842 \tl_new:N \l__spintent_spcirc_intent_tl
6843 \tl_new:N \l__spintent_spcirc_intent_arg_tl
6844 \tl_new:N \l__spintent_spcirc_connector_tl
6845 \tl_new:N \l__spintent_spcirc_connector_arg_tl
6846 \tl_new:N \l__spintent_spcirc_radio_arg_tl

```

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

`\__spintent_spcirc_sym:` Typesets `\bigodot` or the plain letter C, according to `print-sym`.

```

6847 \cs_new_protected:Nn \__spintent_spcirc_sym:
6848 {
6849     \str_if_eq:NnTF \l__spintent_spcirc_print_sym_str { odot }
6850     {
6851         \group_begin:
6852         \textstyle\bigodot
6853         \group_end:
6854     }
6855     { C }
6856 }

```

(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.

```

6857 \cs_new_protected:Nn \__spintent_spcirc_word_and_article:
6858 {
6859     \tl_if_empty:NnTF \l__spintent_spcirc_read_arg_tl
6860     {
6861         \str_if_eq:NnTF \l__spintent_spcirc_read_sty_str { circun }
6862         {
6863             \tl_set:Nn \l__spintent_spcirc_word_tl { circunferencia }
6864             \str_set:Nn \l__spintent_spcirc_articulo_str { la }
6865         }
6866         {
6867             \tl_set:Nn \l__spintent_spcirc_word_tl { círculo }
6868             \str_set:Nn \l__spintent_spcirc_articulo_str { el }
6869         }
6870     }
6871     {
6872         \tl_set:Nv \l__spintent_spcirc_word_tl \l__spintent_spcirc_read_arg_tl
6873         \str_if_eq:NnTF \l__spintent_spcirc_read_sty_str { circun }
6874         { \str_set:Nn \l__spintent_spcirc_articulo_str { la } }
6875         { \str_set:Nn \l__spintent_spcirc_articulo_str { el } }
6876     }
6877 }

```

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

`\__spintenc_spcirc_radio_print:` Branches on `\l__spintenc_geo_luaset_circ_radio_type_str`, reusing `\spnum/\spunit/\spside/\spmside` directly rather than reimplementing any parsing.

```

6878 \cs_new_protected:Nn \__spintenc_spcirc_radio_print:
6879 {
6880   \str_case:Vn \l__spintenc_geo_luaset_circ_radio_type_str
6881   {
6882     { number }
6883     { \exp_args:Ne \spnum { \l__spintenc_geo_luaset_circ_radio_raw_str } }
6884     { measure }
6885     { \exp_args:Ne \spunit { \l__spintenc_geo_luaset_circ_radio_raw_str } }
6886     { label } { \l__spintenc_geo_luaset_circ_radio_raw_str }
6887     { segment }
6888     {
6889       \str_if_eq:VnTF \l__spintenc_spcirc_radio_sty_str { side }
6890       { \exp_args:Ne \spside { \l__spintenc_geo_luaset_circ_radio_raw_str } }
6891       { \exp_args:Ne \spmside { \l__spintenc_geo_luaset_circ_radio_raw_str } }
6892     }
6893   }
6894 }

```

(End of definition for `\__spintenc_spcirc_radio_print:`)

`\__spintenc_spcirc_build_concept:` Builds the words `\l__spintenc_spcirc_intent_tl`, with `\__spintenc_spcirc_word_and_article:` and an optional `sub-index` suffix, and `\l__spintenc_spcirc_connector_tl`, de-centro-en. Both are built even when `\l__spintenc_spcirc_has_arg_bool` is `false`, although `\__spintenc_spcirc_mathml:` does not use them in that case. The center is not concatenated into them: it is read by reference with `\__spintenc_mathmlarg_point:nnn`.

```

6895 \cs_new_protected:Nn \__spintenc_spcirc_build_concept:
6896 {
6897   \__spintenc_spcirc_word_and_article:
6898   \tl_set:Ne \l__spintenc_spcirc_intent_tl { \l__spintenc_spcirc_word_tl }
6899   \str_if_empty:NF \l__spintenc_spcirc_sub_index_str
6900   {
6901     \tl_put_right:Ne \l__spintenc_spcirc_intent_tl
6902     {
6903       -
6904       \bool_if:NT \l__spintenc_spcirc_read_sub_bool { sub - }
6905       \l__spintenc_spcirc_sub_index_str
6906     }
6907   }
6908   \tl_set:Nn \l__spintenc_spcirc_connector_tl { de-centro-en }

```

The key `silent`, when `true`, overrides `\l__spintenc_spcirc_intent_arg_tl`, `\l__spintenc_spcirc_connector_arg_tl` and `\l__spintenc_spcirc_radio_arg_tl` with `'_'`, regardless of `read-arg`.

```

6909   \bool_if:NTF \l__spintenc_spcirc_silent_bool
6910   {
6911     \tl_set:Nn \l__spintenc_spcirc_intent_arg_tl { _- }
6912     \tl_set:Nn \l__spintenc_spcirc_connector_arg_tl { _- }
6913     \tl_set:Nn \l__spintenc_spcirc_radio_arg_tl { _- }
6914   }

```

With `concept=true`, `\l__spintenc_spcirc_intent_arg_tl` is `_ \l__spintenc_spcirc_intent_tl` alone, with `prefix` placed after the leading `_` when given; `\l__spintenc_spcirc_connector_arg_tl` is `_ \l__spintenc_spcirc_connector_tl:prefix($pt)`; and `\l__spintenc_spcirc_radio_arg_tl` is the literal `_y-radio-`, or `_y-diámetro-` when `diameter` is `true`, spoken on the delimiter `,` before the reading of `\__spintenc_spcirc_radio_print:`.

```

6915   {
6916     \bool_if:NTF \l__spintenc_spcirc_concept_bool
6917     {
6918       \tl_set:Ne \l__spintenc_spcirc_intent_arg_tl
6919       {
6920         -
6921         \tl_if_empty:NF \l__spintenc_spcirc_prefix_tl
6922         { \tl_use:N \l__spintenc_spcirc_prefix_tl - }
6923         \l__spintenc_spcirc_intent_tl
6924       }
6925       \tl_set:Ne \l__spintenc_spcirc_connector_arg_tl
6926       { _ \l__spintenc_spcirc_connector_tl :prefix( $ pt ) }
6927       \bool_if:NTF \l__spintenc_spcirc_diameter_bool

```

```

6928         { \tl_set:Nn \l__spintenc_spcirc_radio_arg_tl { _y-diámetro- } }
6929         { \tl_set:Nn \l__spintenc_spcirc_radio_arg_tl { _y-radio- } }
6930     }

```

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.

```

6931     {
6932       \tl_if_empty:NTF \l__spintenc_spcirc_read_arg_tl
6933       {
6934         \tl_set:Nn \l__spintenc_spcirc_intent_arg_tl { _- }
6935         \tl_set:Ne \l__spintenc_spcirc_connector_arg_tl { _ ( $ pt ) }
6936         \tl_set:Nn \l__spintenc_spcirc_radio_arg_tl { _- }
6937       }
6938       {
6939         \tl_set:Ne \l__spintenc_spcirc_intent_arg_tl
6940         { _ \l__spintenc_spcirc_intent_tl }
6941         \tl_set:Ne \l__spintenc_spcirc_connector_arg_tl
6942         { _ \l__spintenc_spcirc_connector_tl :prefix( $ pt ) }
6943         \bool_if:NTF \l__spintenc_spcirc_diameter_bool
6944         { \tl_set:Nn \l__spintenc_spcirc_radio_arg_tl { _y-diámetro- } }
6945         { \tl_set:Nn \l__spintenc_spcirc_radio_arg_tl { _y-radio- } }
6946       }
6947     }
6948   }
6949 }

```

(End of definition for `\__spintenc_spcirc_build_concept:`)

`\__spintenc_spcirc_exec:nn` Receives the `d()` argument in ‘`#2`’, which is an explicit empty group when omitted because `\spcirc` converts `-NoValue-` to `{ }` before calling `\__spintenc_spcirc_exec:nn`. When the group is empty, the function sets `\l__spintenc_spcirc_has_arg_bool` to `false` and does not call `\__spintenc_luafun_geo_circ_parse_and_set:n`. Otherwise it calls that function, reports `spcirc-invalid-arg` on failure and delegates to `\__spintenc_spcirc_mathml:`, which calls `\__spintenc_spcirc_build_concept:`.

Before processing the $\langle argument \rangle$, it sets the optional $\langle keys \rangle$ passed in ‘`#1`’.

```

6950 \cs_new_protected:Nn \__spintenc_spcirc_exec:nn
6951 {
6952   \keys_set:nn { spintenc / spcirc } {#1}
6953   \tl_if_empty:NTF {#2}
6954   {
6955     \bool_set_false:N \l__spintenc_spcirc_has_arg_bool
6956   }
6957   {
6958     \bool_set_true:N \l__spintenc_spcirc_has_arg_bool
6959     \__spintenc_luafun_geo_circ_parse_and_set:n { \exp_not:n {#2} }
6960     \str_if_eq:VnT \l__spintenc_geo_luaset_error_str { true }
6961     {
6962       \msg_error:nne { spintenc } { spcirc-invalid-arg } { \exp_not:n {#2} }
6963     }
6964   }
6965   \__spintenc_spcirc_mathml:
6966 }

```

(End of definition for `\__spintenc_spcirc_exec:nn`)

`\__spintenc_spcirc_mathml:` Calls `\__spintenc_spcirc_build_concept:` and wraps the symbol `\l__spintenc_spcirc_intent_arg_tl` in its own native `\MathMLintent`. This is the whole result when `\l__spintenc_spcirc_has_arg_bool` is `false`, that is, for `\spcirc` with no $\langle \dots \rangle$.

Otherwise, the group $\langle center \rangle$, held in `\l__spintenc_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__spintenc_spcirc_radio_arg_tl` and `\__spintenc_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 `\__spintenc_luafun_mrow_intent_arg:nn`, one queue entry per leaf. The delimiter $\langle$ is silenced in the same way but without the usual $\{ \dots \}$ 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`.

```

6967 \cs_new_protected:Nn \__spintent_spcirc_mathml:
6968 {
6969   \__spintent_invisible_sep: % (luamml issues #26)
6970   \__spintent_spcirc_build_concept:
6971   \MathMLintent { \__spintent_spcirc_intent_arg_tl }
6972   {
6973     \__spintent_inner_sep:
6974     \__spintent_luafun_mrow_intent_arg:nn
6975     { } { intent = ":silent" }
6976     { \__spintent_spcirc_sym: }
6977     \str_if_empty:NF \__spintent_spcirc_sub_index_str
6978     {
6979       \c_math_subscript_token
6980       {
6981         \__spintent_luafun_mrow_intent_arg:nn
6982         { } { intent = ":silent" }
6983         { \__spintent_spcirc_sub_index_str }
6984       }
6985     }
6986   }
6987   \bool_if:NT \__spintent_spcirc_has_arg_bool
6988   {
6989     \str_if_empty:NTF \__spintent_geo_luaset_circ_radio_type_str
6990     {
6991       \MathMLintent { \__spintent_spcirc_connector_arg_tl }
6992       {
6993         \__spintent_inner_sep:
6994         \__spintent_luafun_mrow_intent_arg:nn { } { intent = ":silent" } \lparen
6995         \__spintent_mathmlarg_point:nnV { pt } { spcirc }
6996         \__spintent_geo_luaset_points_clist
6997         \__spintent_luafun_mrow_intent_arg:nn { } { intent = ":silent" } \rparen
6998       }
6999     }
7000     {
7001       \MathMLintent { \__spintent_spcirc_connector_arg_tl }
7002       {
7003         \__spintent_inner_sep:
7004         \__spintent_luafun_mrow_intent_arg:nn { } { intent = ":silent" } \lparen
7005         \__spintent_mathmlarg_point:nnV { pt } { spcirc }
7006         \__spintent_geo_luaset_points_clist
7007       }
7008       \MathMLintent { \__spintent_spcirc_radio_arg_tl } { , }
7009       \bool_if:NTF \__spintent_spcirc_silent_bool
7010       {
7011         \MathMLintent { _- }
7012         {
7013           \__spintent_inner_sep:
7014           \__spintent_spcirc_radio_print:
7015         }
7016         { \__spintent_spcirc_radio_print: }
7017       }
7018       \MathMLintent { :silent } { \rparen }
7019     }
7020   }
7021 }

```

(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.

```

7022 \NewDocumentCommand \spcirc { 0{} d() }
7023 {
7024   \mode_if_math:TF
7025   {
7026     \group_begin:
7027     \attribute \__spintent_scope_attr_int = 1

```

```

7028         \tl_if_novalue:nTF {#2}
7029         { \__spintent_spcirc_exec:nn {#1} { } }
7030         { \__spintent_spcirc_exec:nn {#1} {#2} }
7031     \group_end:
7032 }
7033 { \msg_error:nnn { spintent } { need-math-mode } { \spcirc } }
7034 }

```

(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.

```

\__spintent_spAcirc_mode_str
\__spintent_spPcirc_mode_str
\__spintent_spAcirc_print_sym_str
\__spintent_spPcirc_print_sym_str
\__spintent_spAcirc_read_sty_str
\__spintent_spPcirc_read_sty_str
\__spintent_spAcirc_sub_index_str
\__spintent_spPcirc_sub_index_str
\__spintent_spAcirc_prefix_tl
\__spintent_spPcirc_prefix_tl
\__spintent_spAcirc_read_arg_tl
\__spintent_spPcirc_read_arg_tl

```

```

7035 \clist_map_inline:nn { Acirc , Pcirc }
7036 {
7037     \str_new:c { \__spintent_sp #1 _mode_str }
7038     \str_new:c { \__spintent_sp #1 _print_sym_str }
7039     \str_new:c { \__spintent_sp #1 _read_sty_str }
7040     \str_new:c { \__spintent_sp #1 _sub_index_str }
7041     \tl_new:c { \__spintent_sp #1 _prefix_tl }
7042     \tl_new:c { \__spintent_sp #1 _read_arg_tl }
7043 }

```

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

15.55.2 Definition of error messages for `\spAcirc` and `\spPcirc`

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.

```

7044 \clist_map_inline:nn { Acirc , Pcirc }
7045 {
7046     \msg_new:nnnn { spintent } { sp #1 -invalid-arg }
7047     {
7048         Invalid~argument~'##2'~for~'\tl_trim_spaces:n {##1}'~\msg_line_context:.
7049     }
7050     {
7051         See~the~manual~for~the~accepted~argument~forms.
7052     }

```

(End of definition for `spAcirc-invalid-arg` and `spPcirc-invalid-arg`.)

15.55.3 Definition of keys for `\spAcirc` and `\spPcirc`

We define the shared (*keys*) `read-cmd`, `prefix`, `print-sym`, `read-sty`, `concept`, `sub-index`, `read-sub`, `read-arg` and `silent`.

```

7053 \keys_define:nn { spintent / sp #1 }
7054 {
7055     read-cmd .choices:nn = { school , mathcat }
7056     {
7057         \int_case:nn { \l_keys_choice_int }
7058         {
7059             {1} { \str_set:cn { \__spintent_sp #1 _mode_str } { school } }
7060             {2} { \str_set:cn { \__spintent_sp #1 _mode_str } { mathcat } }
7061         }
7062     },
7063     read-cmd .initial:n = school,
7064     read-cmd .value_required:n = true,
7065     read-cmd / unknown .code:n =
7066         \msg_error:nnee { spintent } { unknown-choice }
7067         { read-cmd } { school,~mathcat } { \exp_not:n {##1} },
7068     prefix .code:n =

```

```

7069     { \_spintent_sanitizate_affix:cn { \_spintent_sp #1 _prefix_tl } {##1} },
7070     prefix .initial:n = ,
7071     prefix .value_required:n = true,
7072     print-sym .choices:nn = { usual , odot }
7073     {
7074         \int_case:nn { \_keys_choice_int }
7075         {
7076             {1} { \_str_set:cn { \_spintent_sp #1 _print_sym_str } { usual } }
7077             {2} { \_str_set:cn { \_spintent_sp #1 _print_sym_str } { odot } }
7078         }
7079     },
7080     print-sym .initial:n = odot,
7081     print-sym .value_required:n = true,
7082     print-sym / unknown .code:n =
7083         \msg_error:nneee { spintent } { unknown-choice }
7084         { print-sym } { usual,~odot } { \exp_not:n {##1} },
7085     read-sty .choices:nn = { circun , circulo }
7086     {
7087         \int_case:nn { \_keys_choice_int }
7088         {
7089             {1} { \_str_set:cn { \_spintent_sp #1 _read_sty_str } { circun } }
7090             {2} { \_str_set:cn { \_spintent_sp #1 _read_sty_str } { circulo } }
7091         }
7092     },
7093     read-sty .initial:n = circun,
7094     read-sty .value_required:n = true,
7095     read-sty / unknown .code:n =
7096         \msg_error:nneee { spintent } { unknown-choice }
7097         { read-sty } { circun,~circulo } { \exp_not:n {##1} },
7098     concept .bool_set:c = { \_spintent_sp #1 _concept_bool },
7099     concept .initial:n = true,
7100     concept .value_required:n = true,
7101     sub-index .code:n =
7102         { \_str_set:cn { \_spintent_sp #1 _sub_index_str } {##1} },
7103     sub-index .initial:n = ,
7104     sub-index .value_required:n = true,
7105     read-sub .bool_set:c = { \_spintent_sp #1 _read_sub_bool },
7106     read-sub .initial:n = false,
7107     read-sub .value_required:n = true,
7108     read-arg .code:n =
7109         { \_spintent_sanitizate_affix:cn { \_spintent_sp #1 _read_arg_tl } {##1} },
7110     read-arg .initial:n = ,
7111     read-arg .value_required:n = true,
7112     silent .bool_set:c = { \_spintent_sp #1 _silent_bool },
7113     silent .initial:n = false,
7114     silent .value_required:n = true,
7115     unknown .code:n =
7116         {
7117             \_str_set:Nn \_spintent_command_name_str { sp #1 }
7118             \_spintent_unknown_keys_cmd:n {##1}
7119         },
7120     }
7121 }

```

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

$\backslash_spintent_spcircfig_short_bool$
 $\backslash_spintent_spcircfig_luaset_comma_str$
 $\backslash_spintent_spcircfig_op_tl$
 $\backslash_spintent_spcircfig_del_word_tl$
 $\backslash_spintent_spcircfig_intent_tl$
 $\backslash_spintent_spcircfig_intent_arg_tl$

The variable $\backslash_spintent_spcircfig_short_bool$ stores whether the argument resolved as the short number or letter form, with $\backslash_spintent_luafun_geo_afig_parse_and_set:n$ or $\backslash_spintent_luafun_geo_pfig_parse_and_set:n$, and not as a circle. The string variable $\backslash_spintent_spcircfig_luaset_comma_str$, set by $\backslash_spintent_luafun_spcircfig_comma:n$, is true when the argument has a comma. The variable $\backslash_spintent_spcircfig_op_tl$ holds «área» or «perímetro». The variable $\backslash_spintent_spcircfig_intent_tl$ holds the spoken phrase: $\backslash_spintent_spcircfig_del_word_tl$, set by $\backslash_spintent_spcircfig_del_word:n$, an optional sub-index suffix and, when a center is given, -con-centro-en. The center is read by reference with $\backslash_spintent_mathmlarg_point:nnn$.

```

7122 \bool_new:N \_spintent_spcircfig_short_bool
7123 \str_new:N \_spintent_spcircfig_luaset_comma_str
7124 \tl_new:N \_spintent_spcircfig_op_tl
7125 \tl_new:N \_spintent_spcircfig_del_word_tl
7126 \tl_new:N \_spintent_spcircfig_intent_tl

```

```
7127 \tl_new:N \__spintent_spcircfig_intent_arg_tl
```

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

`\__spintent_spcircfig_sym:n` Typesets `\bigodot` or the plain letter C, according to `print-sym`. Receives the short command name in ‘#1’.

```
7128 \cs_new_protected:Nn \__spintent_spcircfig_sym:n
7129 {
7130   \str_if_eq:vnTF { \__spintent_sp #1 _print_sym_str } { odot }
7131   { \bigodot }
7132   { C }
7133 }
```

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

`\__spintent_spcircfig_word_and_article:n` Sets `\__spintent_spcirc_word_tl` to `read-arg` or `circunferencia/círculo` from `read-sty`, and `\__spintent_spcirc_articulo_str` to the matching article, which always follows `read-sty` even when `read-arg` overrides the word. Receives the short command name in ‘#1’.

```
7134 \cs_new_protected:Nn \__spintent_spcircfig_word_and_article:n
7135 {
7136   \str_if_eq:vnTF { \__spintent_sp #1 _read_sty_str } { circun }
7137   { \str_set:Nn \__spintent_spcirc_articulo_str { la } }
7138   { \str_set:Nn \__spintent_spcirc_articulo_str { el } }
7139   \tl_if_empty:cTF { \__spintent_sp #1 _read_arg_tl }
7140   {
7141     \str_if_eq:vnTF { \__spintent_sp #1 _read_sty_str } { circun }
7142     { \tl_set:Nn \__spintent_spcirc_word_tl { circunferencia } }
7143     { \tl_set:Nn \__spintent_spcirc_word_tl { círculo } }
7144   }
7145   {
7146     \tl_set:Ne \__spintent_spcirc_word_tl
7147     {
7148       \tl_use:c { \__spintent_sp #1 _read_arg_tl }
7149     }
7150   }
7151 }
```

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

`\__spintent_spcircfig_del_word:n` Sets `\__spintent_spcircfig_del_word_tl` to the word of `\__spintent_spcircfig_word_and_article:n`, prefixed with `del-` or `de-la-`. Receives the short command name in ‘#1’.

```
7152 \cs_new_protected:Nn \__spintent_spcircfig_del_word:n
7153 {
7154   \__spintent_spcircfig_word_and_article:n {#1}
7155   \str_if_eq:VnTF \__spintent_spcirc_articulo_str { el }
7156   {
7157     \tl_set:Ne \__spintent_spcircfig_del_word_tl
7158     { del- \__spintent_spcirc_word_tl }
7159   }
7160   {
7161     \tl_set:Ne \__spintent_spcircfig_del_word_tl
7162     { de-la- \__spintent_spcirc_word_tl }
7163   }
7164 }
```

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

`\__spintent_spcircfig_build_short:n` Handles the short number or letter case: builds `\__spintent_spcircfig_intent_arg_tl` as `_\__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’.

```
7165 \cs_new_protected:Nn \__spintent_spcircfig_build_short:n
7166 {
7167   \bool_if:cTF { \__spintent_sp #1 _silent_bool }
7168   {
7169     \tl_set:Nn \__spintent_spcircfig_intent_arg_tl { _- }
7170   }
7171   {
7172     \bool_if:cTF { \__spintent_sp #1 _concept_bool }
7173     {
```

```

7174         \l__spintent_spcircfig_del_word:n {#1}
7175         \tl_set:Nc \l__spintent_spcircfig_intent_arg_tl
7176             {
7177             -
7178             \tl_if_empty:cF { \l__spintent_sp #1 _prefix_tl }
7179             { \tl_use:c { \l__spintent_sp #1 _prefix_tl } - }
7180             \l__spintent_spcircfig_op_tl - \l__spintent_spcircfig_del_word_tl -
7181             \bool_if:cT { \l__spintent_sp #1 _read_sub_bool } { sub- }
7182             \l__spintent_geo_luaset_sub_spoken_str
7183             }
7184         }
7185     {
7186         \tl_set:Nn \l__spintent_spcircfig_intent_arg_tl { _- }
7187     }
7188 }
7189 }

```

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

`\l__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`.

```

7190 \cs_new_protected:Nn \l__spintent_spcircfig_finish_intent:nn
7191 {
7192     \str_if_empty:cF { \l__spintent_sp #1 _sub_index_str }
7193     {
7194         \tl_put_right:Nc \l__spintent_spcircfig_intent_tl
7195             {
7196             -
7197             \bool_if:cT { \l__spintent_sp #1 _read_sub_bool } { sub- }
7198             \str_use:c { \l__spintent_sp #1 _sub_index_str }
7199             }
7200     }
7201     \tl_if_empty:nF {#2}
7202     {
7203         \tl_put_right:Nn \l__spintent_spcircfig_intent_tl { -con-centro-en }
7204     }
7205 }

```

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

`\l__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 `\l__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`.

```

7206 \cs_new_protected:Nn \l__spintent_spcircfig_build_concept:nn
7207 {
7208     \bool_if:cTF { \l__spintent_sp #1 _silent_bool }
7209     {
7210         \tl_set:Nn \l__spintent_spcircfig_intent_arg_tl { _- }
7211     }
7212     {
7213         \bool_if:cTF { \l__spintent_sp #1 _concept_bool }
7214         {
7215             \l__spintent_spcircfig_del_word:n {#1}
7216             \tl_set:Nc \l__spintent_spcircfig_intent_tl
7217                 {
7218                 \l__spintent_spcircfig_op_tl - \l__spintent_spcircfig_del_word_tl
7219                 }
7220             \l__spintent_spcircfig_finish_intent:nn {#1} {#2}
7221             \tl_set:Nc \l__spintent_spcircfig_intent_arg_tl
7222                 {
7223                 -
7224                 \tl_if_empty:cF { \l__spintent_sp #1 _prefix_tl }

```

```

7225         {
7226             \tl_use:c { l__spinttent_sp #1 _prefix_tl } -
7227         }
7228         \l__spinttent_spcircfig_intent_tl
7229         \tl_if_empty:nF {#2} { :prefix ( $ pt ) }
7230     }
7231 }
7232 {
7233     \tl_if_empty:cTF { l__spinttent_sp #1 _read_arg_tl }
7234     {
7235         \tl_if_empty:nTF {#2}
7236         { \tl_set:Nn \l__spinttent_spcircfig_intent_arg_tl { _- } }
7237         { \tl_set:Ne \l__spinttent_spcircfig_intent_arg_tl { _ ( $ pt ) } }
7238     }
7239     {
7240         \tl_set:Ne \l__spinttent_spcircfig_intent_tl
7241         { \tl_use:c { l__spinttent_sp #1 _read_arg_tl } }
7242         \__spinttent_spcircfig_finish_intent:nn {#1} {#2}
7243         \tl_set:Ne \l__spinttent_spcircfig_intent_arg_tl
7244         {
7245             -
7246             \l__spinttent_spcircfig_intent_tl
7247             \tl_if_empty:nF {#2} { :prefix ( $ pt ) }
7248         }
7249     }
7250 }
7251 }
7252 }

```

(End of definition for `\__spinttent_spcircfig_build_concept:nn`)

`\__spinttent_spcircfig_exec:nnn` Receives the short command name in ‘#1’ and the argument in ‘#3’. Sets `\l__spinttent_spcircfig_op_tl` to «*área*» or «*perímetro*». With #3 empty, it sets `\l__spinttent_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 `\__spinttent_luafun_geo_circ_parse_and_set:n`; on error, it tries `\__spinttent_luafun_geo_afig_parse_and_set:n` or `\__spinttent_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 `\__spinttent_spcircfig_build_short:n` or `\__spinttent_spcircfig_build_concept:nn` and to `\__spinttent_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’.

```

7253 \cs_new_protected:Nn \__spinttent_spcircfig_exec:nnn
7254 {
7255     \keys_set:nn { spinttent / sp #1 } {#2}
7256     \str_if_eq:eeTF {#1} { Acirc }
7257     { \tl_set:Nn \l__spinttent_spcircfig_op_tl { área } }
7258     { \tl_set:Nn \l__spinttent_spcircfig_op_tl { perímetro } }
7259     \tl_if_empty:nTF {#3}
7260     {
7261         \bool_set_false:N \l__spinttent_spcircfig_short_bool
7262     }
7263     {
7264         \bool_set_false:N \l__spinttent_spcircfig_short_bool
7265         \__spinttent_luafun_spcircfig_comma:n { \exp_not:n {#3} }
7266         \str_if_eq:VnTF \l__spinttent_spcircfig_luaset_comma_str { true }
7267         {
7268             \msg_error:nnee { spinttent } { sp #1 -invalid-arg }
7269             { \c_backslash_str sp #1 } { \exp_not:n {#3} }
7270         }
7271     }
7272     {
7273         \__spinttent_luafun_geo_circ_parse_and_set:n { \exp_not:n {#3} }
7274         \str_if_eq:VnT \l__spinttent_geo_luaset_error_str { true }
7275         {
7276             \str_if_eq:eeTF {#1} { Acirc }
7277             { \__spinttent_luafun_geo_afig_parse_and_set:n { \exp_not:n {#3} } }
7278             { \__spinttent_luafun_geo_pfig_parse_and_set:n { \exp_not:n {#3} } }
7279         }
7280     }

```

```

7280         \str_if_empty:NTF \l__spintent_geo_luaset_letra_word_str
7281         { \bool_set_true:N \l__spintent_spcircfig_short_bool }
7282         { \str_set:Nn \l__spintent_geo_luaset_error_str { true } }
7283     }
7284 }
7285 \bool_if:NF \l__spintent_spcircfig_short_bool
7286 {
7287     \str_if_eq:VnT \l__spintent_geo_luaset_error_str { true }
7288     {
7289         \msg_error:nnee { spintent } { sp #1 -invalid-arg }
7290         { \c_backslash_str sp #1 } { \exp_not:n {#3} }
7291     }
7292 }
7293 }
7294 }
7295 \bool_if:NTF \l__spintent_spcircfig_short_bool
7296 { \__spintent_spcircfig_build_short:n {#1} }
7297 { \__spintent_spcircfig_build_concept:nn {#1} {#3} }
7298 \__spintent_spcircfig_mathml:nn {#1} {#3}
7299 }

```

(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`’.

```

7300 \cs_new_protected:Nn \__spintent_spcircfig_mathml:nn
7301 {
7302     \__spintent_invisible_sep: % (luamml issues #26)
7303     \MathMLintent { \l__spintent_spcircfig_intent_arg_tl }
7304     {
7305         \__spintent_inner_sep:
7306         \__spintent_luafun_mrow_intent_arg:nn
7307         { } { intent = ":silent" }
7308     }
7309     \str_if_eq:eeTF {#1} { Acirc } { \__spintent_mathcal:n { A } }
7310     { \__spintent_mathcal:n { P } }
7311 }
7312 \bool_if:NTF \l__spintent_spcircfig_short_bool
7313 {
7314     \c_math_subscript_token
7315     {
7316         \__spintent_spcircfig_sym:n {#1}
7317         \c_math_subscript_token { \l__spintent_geo_luaset_print_tl }
7318     }
7319 }
7320 {
7321     \c_math_subscript_token
7322     {
7323         \MathMLintent { :silent } { \__spintent_spcircfig_sym:n {#1} }
7324         \str_if_empty:cF { l__spintent_sp #1 _sub_index_str }
7325         {
7326             \c_math_subscript_token
7327             {
7328                 \MathMLintent { :silent }
7329                 { \str_use:c { l__spintent_sp #1 _sub_index_str } }
7330             }
7331         }
7332     }
7333     \tl_if_empty:nF {#2}
7334     {
7335         \MathMLintent { :silent } { \lparen }
7336         \__spintent_mathmlarg_point:nnV { pt } { sp #1 }
7337         \__spintent_geo_luaset_points_clist
7338         \MathMLintent { :silent } { \rparen }
7339     }
7340 }
7341 }

```

```
7342 }
```

(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`

```
7343 \NewDocumentCommand \spAcirc { 0{} d() }
7344 {
7345   \mode_if_math:TF
7346   {
7347     \group_begin:
7348     \attribute \l__spinttent_scope_attr_int = 1
7349     \tl_if_novalue:nTF {#2}
7350     { \__spinttent_spcircfig_exec:nnn { Acirc } {#1} { } }
7351     { \__spinttent_spcircfig_exec:nnn { Acirc } {#1} {#2} }
7352   \group_end:
7353   }
7354   { \msg_error:nnn { spinttent } { need-math-mode } { \spAcirc } }
7355 }
7356 \NewDocumentCommand \spPcirc { 0{} d() }
7357 {
7358   \mode_if_math:TF
7359   {
7360     \group_begin:
7361     \attribute \l__spinttent_scope_attr_int = 1
7362     \tl_if_novalue:nTF {#2}
7363     { \__spinttent_spcircfig_exec:nnn { Pcirc } {#1} { } }
7364     { \__spinttent_spcircfig_exec:nnn { Pcirc } {#1} {#2} }
7365   \group_end:
7366   }
7367   { \msg_error:nnn { spinttent } { need-math-mode } { \spPcirc } }
7368 }
```

(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`

The variable `\l__spinttent_spVol_mode_str` holds the `read-cmd` choice, `school` or `mathcat`; `\l__spinttent_spVol_sub_index_str` the raw `sub-index` text; `\l__spinttent_spVol_read_arg_tl` the sanitized `read-arg` text. `\l__spinttent_spVol_word_tl` and `\l__spinttent_spVol_suffix_tl` are scratch variables holding the word to voice, which is «*volumen*», and the text to attach after it; `\l__spinttent_spVol_native_bool` records whether that text should be combined natively with the subscript, silenced, or left for MathCAT to read on its own. The variable `\l__spinttent_spVol_read_arg_full_tl` holds the text to voice when `read-arg` is given.

```
7369 \str_new:N \l__spinttent_spVol_mode_str
7370 \str_new:N \l__spinttent_spVol_sub_index_str
7371 \tl_new:N \l__spinttent_spVol_read_arg_tl
7372 \tl_new:N \l__spinttent_spVol_word_tl
7373 \tl_new:N \l__spinttent_spVol_suffix_tl
7374 \bool_new:N \l__spinttent_spVol_native_bool
7375 \tl_new:N \l__spinttent_spVol_read_arg_full_tl
```

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

15.56.2 Definition of keys for `\spVol`

`read-cmd` We define the *(keys)* `read-cmd`, `sub-index`, `read-sub`, `read-arg`, `concept` and `silent`.

`sub-index`

`read-sub`

`read-arg`

`concept`

`silent`

`unknown`

```
7376 \keys_define:nn { spinttent / spVol }
7377 {
7378   read-cmd .choices:nn = { school , mathcat }
7379   {
7380     \int_case:nn { \l_keys_choice_int }
7381     {
7382       {1} { \str_set:Nn \l__spinttent_spVol_mode_str { school } }
7383       {2} { \str_set:Nn \l__spinttent_spVol_mode_str { mathcat } }
7384     }
7385   }
```

```

7385     },
7386     read-cmd .initial:n = school,
7387     read-cmd .value_required:n = true,
7388     read-cmd / unknown .code:n =
7389       \msg_error:nneee { spintent } { unknown-choice }
7390       { read-cmd } { school,~mathcat } { \exp_not:n {#1} },
7391     sub-index .code:n =
7392       { \str_set:Nn \l__spintent_spVol_sub_index_str {#1} },
7393     sub-index .initial:n = ,
7394     sub-index .value_required:n = true,
7395     read-sub .bool_set:N = \l__spintent_spVol_read_sub_bool,
7396     read-sub .initial:n = false,
7397     read-sub .value_required:n = true,
7398     read-arg .code:n =
7399       { \__spintent_sanitize_affix:Nn \l__spintent_spVol_read_arg_tl {#1} },
7400     read-arg .initial:n = ,
7401     read-arg .value_required:n = true,
7402     concept .bool_set:N = \l__spintent_spVol_concept_bool,
7403     concept .initial:n = true,
7404     concept .value_required:n = true,
7405     silent .bool_set:N = \l__spintent_spVol_silent_bool,
7406     silent .initial:n = false,
7407     silent .value_required:n = true,
7408     unknown .code:n =
7409       {
7410         \str_set:Nn \l__spintent_command_name_str { spVol }
7411         \__spintent_unknown_keys_cmd:n {#1}
7412       },
7413   }

```

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

15.56.3 Internal functions for \spVol

`\__spintent_spVol_mathml:` The function `\__spintent_spVol_mathml:` calls `\__spintent_spVol_silent:` when `silent` is `true`, or `\__spintent_spVol_verbal:` otherwise.

```

7414 \cs_new_protected:Nn \__spintent_spVol_mathml:
7415 {
7416   \__spintent_invisible_sep:
7417   \bool_if:NTF \l__spintent_spVol_silent_bool
7418     { \__spintent_spVol_silent: }
7419     { \__spintent_spVol_verbal: }
7420 }

```

(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.

```

7421 \cs_new_protected:Nn \__spintent_spVol_silent:
7422 {
7423   \str_if_empty:NTF \l__spintent_spVol_sub_index_str
7424     {
7425       \__spintent_luafun_mrow_intent_arg:nn
7426       { } { intent = "- " }
7427       { \mathcal{V} }
7428     }
7429     {
7430       \MathMLintent { _- }
7431       {
7432         \__spintent_inner_sep:
7433         \mathcal{V}
7434         \c_math_subscript_token
7435         {
7436           \MathMLintent { :silent }
7437           { \l__spintent_spVol_sub_index_str }
7438         }
7439       }
7440     }
7441 }

```

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

`\__spinttent_spVol_verbal:` The function `\__spinttent_spVol_verbal:` delegates to `\__spinttent_spVol_read_arg_print:` when `read-arg` is given and, otherwise, with `concept=false`, to `\__spinttent_spVol_symbol_print:`. With `concept=true` it sets `\__spinttent_spVol_word_tl` to «*volumen*» and `\__spinttent_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 `\__spinttent_spVol_attach_subscript:`.

```

7442 \cs_new_protected:Nn \__spinttent_spVol_verbal:
7443 {
7444   \tl_if_empty:NTF \__spinttent_spVol_read_arg_tl
7445   {
7446     \bool_if:NTF \__spinttent_spVol_concept_bool
7447     {
7448       \tl_set:Nn \__spinttent_spVol_word_tl { volumen }
7449       \str_if_eq:VnTF \__spinttent_spVol_mode_str { school }
7450       { \bool_set_true:N \__spinttent_spVol_native_bool }
7451       { \bool_set_false:N \__spinttent_spVol_native_bool }
7452       \tl_set:Ne \__spinttent_spVol_suffix_tl
7453       {
7454         \bool_if:NT \__spinttent_spVol_read_sub_bool { sub- }
7455         \__spinttent_spVol_sub_index_str
7456       }
7457       \__spinttent_spVol_attach_subscript:
7458     }
7459     { \__spinttent_spVol_symbol_print: }
7460   }
7461   { \__spinttent_spVol_read_arg_print: }
7462 }

```

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

`\__spinttent_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.

```

7463 \cs_new_protected:Nn \__spinttent_spVol_symbol_print:
7464 {
7465   \str_if_empty:NTF \__spinttent_spVol_sub_index_str
7466   {
7467     \str_if_eq:VnTF \__spinttent_spVol_mode_str { school }
7468     {
7469       \__spinttent_luafun_mrow_intent_arg:nn
7470       { } { intent = "_V-" }
7471       { \mathcal{V} }
7472     }
7473     { \mathcal{V} }
7474   }
7475   {
7476     \str_if_eq:VnTF \__spinttent_spVol_mode_str { school }
7477     {
7478       \MathMLintent
7479       {
7480         _V-
7481         \bool_if:NT \__spinttent_spVol_read_sub_bool { sub- }
7482         \__spinttent_spVol_sub_index_str
7483       }
7484       {
7485         \__spinttent_inner_sep:
7486         \mathcal{V}
7487         \c_math_subscript_token
7488         {
7489           \MathMLintent { :silent }
7490           { \__spinttent_spVol_sub_index_str }
7491         }
7492       }
7493     }
7494     {
7495       \mathcal{V}
7496       \c_math_subscript_token

```

```

7497         { \l__spinttent_spVol_sub_index_str }
7498     }
7499 }
7500 }

```

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

`\l__spinttent_spVol_attach_subscript:`

The function `\l__spinttent_spVol_attach_subscript:` typesets `\mathcal{V}`. An empty `sub-index` tags the symbol alone through the shared queue, and `\l__spinttent_spVol_suffix_tl` is discarded since there is no subscript to attach it to. A given one combines symbol, subscript, and `\l__spinttent_spVol_suffix_tl` in one native `\MathMLintent`, subscript silenced, when `\l__spinttent_spVol_native_bool` is `true`, or tags the symbol alone through the shared queue with the subscript left bare when `false`.

```

7501 \cs_new_protected:Nn \l__spinttent_spVol_attach_subscript:
7502 {
7503     \str_if_empty:NTF \l__spinttent_spVol_sub_index_str
7504     {
7505         \l__spinttent_luafun_mrow_intent_arg:nn
7506         { } { intent = "_ \l__spinttent_spVol_word_tl :nofix" }
7507         { \mathcal{V} }
7508     }
7509     {
7510         \bool_if:NTF \l__spinttent_spVol_native_bool
7511         {
7512             \MathMLintent
7513             { _ \l__spinttent_spVol_word_tl - \l__spinttent_spVol_suffix_tl :nofix }
7514             {
7515                 \l__spinttent_inner_sep:
7516                 \mathcal{V}
7517                 \c_math_subscript_token
7518                 {
7519                     \MathMLintent { :silent }
7520                     { \l__spinttent_spVol_sub_index_str }
7521                 }
7522             }
7523         }
7524         {
7525             \l__spinttent_luafun_mrow_intent_arg:nn
7526             { } { intent = "_ \l__spinttent_spVol_word_tl :nofix" }
7527             { \mathcal{V} }
7528             \c_math_subscript_token
7529             { \l__spinttent_spVol_sub_index_str }
7530         }
7531     }
7532 }

```

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

`\l__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`, `\l__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`.

```

7533 \cs_new_protected:Nn \l__spinttent_spVol_read_arg_print:
7534 {
7535     \bool_if:NTF \l__spinttent_spVol_concept_bool
7536     {
7537         \tl_set:Nx \l__spinttent_spVol_read_arg_full_tl
7538         { volumen- \tl_use:N \l__spinttent_spVol_read_arg_tl }
7539     }
7540     {
7541         \tl_set_eq:NN \l__spinttent_spVol_read_arg_full_tl \l__spinttent_spVol_read_arg_tl
7542     }
7543     \str_if_empty:NTF \l__spinttent_spVol_sub_index_str
7544     {
7545         \l__spinttent_luafun_mrow_intent_arg:nn
7546         { } { intent = "_ \l__spinttent_spVol_read_arg_full_tl :nofix" }
7547         { \mathcal{V} }
7548     }
7549     {

```

```

7550     \MathMLintent
7551     { _ \l__spintent_spVol_read_arg_full_tl :nofix }
7552     {
7553       \__spintent_inner_sep:
7554       \mathcal{V}
7555       \c_math_subscript_token
7556       {
7557         \MathMLintent { :silent }
7558         { \l__spintent_spVol_sub_index_str }
7559       }
7560     }
7561   }
7562 }

```

(End of definition for `\__spintent_spVol_read_arg_print:`)

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

```

7563 \NewDocumentCommand \spVol { 0{} }
7564 {
7565   \mode_if_math:TF
7566   {
7567     \group_begin:
7568     \attribute \l__spintent_scope_attr_int = 1
7569     \keys_set:nn { spintent / spVol } {#1}
7570     \__spintent_spVol_mathml:
7571     \group_end:
7572   }
7573   { \msg_error:nnn { spintent } { need-math-mode } { \spVol } }
7574 }

```

(End of definition for `\spVol`. This function is documented on page 38.)

15.57 The command `\spProm`

The user command `\spProm` typesets its $\langle \textit{argument} \rangle$ 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`

`\l__spintent_spProm_read_arg_tl` The variable `\l__spintent_spProm_read_arg_tl` holds the normalized text of the `read-arg` key, and `\l__spintent_spProm_intent_tl` holds the name of the intent used by `\__spintent_spProm_exec:nn`.

```

7575 \tl_new:N \l__spintent_spProm_read_arg_tl
7576 \tl_new:N \l__spintent_spProm_intent_tl

```

(End of definition for `\l__spintent_spProm_read_arg_tl` and `\l__spintent_spProm_intent_tl`.)

15.57.2 Definition of keys for `\spProm`

`silent` We define the $\langle \textit{keys} \rangle$ `silent` and `read-arg`.
`read-arg`
`unknown`

```

7577 \keys_define:nn { spintent / spProm }
7578 {
7579   silent .bool_set:N = \l__spintent_spProm_silent_bool,
7580   silent .initial:n = false,
7581   silent .value_required:n = true,
7582   read-arg .code:n =
7583     { \__spintent_sanitizate_affix:Nn \l__spintent_spProm_read_arg_tl {#1} },
7584   read-arg .initial:n = ,
7585   read-arg .value_required:n = true,
7586   unknown .code:n =
7587     {
7588       \str_set:Nn \l__spintent_command_name_str { spProm }
7589       \__spintent_unknown_keys_cmd:n {#1}
7590     },
7591 }

```

(End of definition for `silent`, `read-arg`, and `unknown`.)

15.57.3 Internal functions for `\spProm`

`\__spintent_spProm_exec:nn`

The function `\__spintent_spProm_exec:nn` receives the optional $\langle keys \rangle$ in ‘#1’ and the $\{\langle argument \rangle\}$ in ‘#2’. It sets the $\langle keys \rangle$ and calls `\__spintent_spProm_mathml:n`.

```

7592 \cs_new_protected:Nn \__spintent_spProm_exec:nn
7593 {
7594   \keys_set:nn { spintent / spProm } {#1}
7595   \__spintent_spProm_mathml:n {#2}
7596 }
```

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

`\__spintent_spProm_mathml:n`

The function `\__spintent_spProm_mathml:n` receives the $\{\langle argument \rangle\}$ in ‘#1’. Under `silent` the whole expression is silenced as one unit with the intent ‘-’.

Otherwise the $\{\langle argument \rangle\}$ is tagged through a native `\MathMLarg`, `a`, inside `\bar`, and the outer `\MathMLintent` reads `_promedio-de:prefix($a)`. When `read-arg` is not empty, its text replaces `promedio-de`. The inner ghost `\__spintent_inner_sep:` keeps the content of the intent from collapsing into a single child; a second one inside the `\MathMLarg` does the same for the argument, whose `arg` attribute would otherwise be lost when the argument is a single compound element, such as a letter with a subscript.

```

7597 \cs_new_protected:Nn \__spintent_spProm_mathml:n
7598 {
7599   \__spintent_invisible_sep:
7600   \bool_if:NTF \__spintent_spProm_silent_bool
7601   {
7602     \MathMLintent { - }
7603     {
7604       \__spintent_inner_sep:
7605       \bar {#1}
7606     }
7607   }
7608   {
7609     \tl_if_empty:NTF \__spintent_spProm_read_arg_tl
7610     { \tl_set:Nn \__spintent_spProm_intent_tl { promedio-de } }
7611     {
7612       \tl_set_eq:NN \__spintent_spProm_intent_tl \__spintent_spProm_read_arg_tl
7613     }
7614     \MathMLintent { - \__spintent_spProm_intent_tl :prefix($a) }
7615     {
7616       \__spintent_inner_sep:
7617       \bar { \MathMLarg { a } { \__spintent_inner_sep: #1 } }
7618     }
7619   }
7620 }
```

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

`\spProm`

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

```

7621 \NewDocumentCommand \spProm { O{} m }
7622 {
7623   \mode_if_math:TF
7624   {
7625     \group_begin:
7626     \attribute \__spintent_scope_attr_int = 1
7627     \__spintent_spProm_exec:nn {#1} {#2}
7628     \group_end:
7629   }
7630   { \msg_error:nnn { spintent } { need-math-mode } { \spProm } }
7631 }
```

(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

```

\l__spinttent_spmedia_sub_index_str
\l__spinttent_spmedia_read_arg_tl
\l__spinttent_spmediana_sub_index_str
\l__spinttent_spmediana_read_arg_tl
\l__spinttent_spmoda_sub_index_str
\l__spinttent_spmoda_read_arg_tl
\l__spinttent_spdesv_sub_index_str
\l__spinttent_spdesv_read_arg_tl
\l__spinttent_spvarz_sub_index_str
\l__spinttent_spvarz_read_arg_tl
\l__spinttent_sprango_sub_index_str
\l__spinttent_sprango_read_arg_tl
\l__spinttent_spriq_sub_index_str
\l__spinttent_spriq_read_arg_tl
\l__spinttent_spmmin_sub_index_str
\l__spinttent_spmmin_read_arg_tl
\l__spinttent_spmmax_sub_index_str
\l__spinttent_spmmax_read_arg_tl

```

Each command of the family has its own copy of the variables `_sub_index_str` and `_read_arg_tl`, which hold the values of the keys `sub-index` and `read-arg`, so each command can be configured separately with `\setspinttent`. 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 `\__spinttent_stat_mathml:nnnnnn` reads it, and always keeps its initial value.

```

7632 \clist_map_inline:nn
7633 {
7634     spmedia , spmediana , spmoda , spdesv , spvarz , sprango , spriq ,
7635     spmin , spmax
7636 }
7637 {
7638     \str_new:c { \l__spinttent_ #1 _sub_index_str }
7639     \tl_new:c { \l__spinttent_ #1 _read_arg_tl }
7640 }

```

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

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`.

```

7641 \clist_map_inline:nn
7642 {
7643     spmedia , spmediana , spmoda , spdesv , spvarz , sprango , spriq
7644 }
7645 {
7646     \keys_define:nn { spinttent / #1 }
7647     {
7648         silent      .bool_set:c = { \l__spinttent_ #1 _silent_bool },
7649         silent      .initial:n = false,
7650         silent      .value_required:n = true,
7651         sub-index   .code:n =
7652             { \str_set:cn { \l__spinttent_ #1 _sub_index_str } {##1} },
7653         sub-index   .initial:n = ,
7654         sub-index   .value_required:n = true,
7655         read-sub    .bool_set:c = { \l__spinttent_ #1 _read_sub_bool },
7656         read-sub    .initial:n = false,
7657         read-sub    .value_required:n = true,
7658         read-arg    .code:n =
7659             { \__spinttent_sanitizate_affix:cn { \l__spinttent_ #1 _read_arg_tl } {##1} },
7660         read-arg    .initial:n = ,
7661         read-arg    .value_required:n = true,
7662         unknown     .code:n =
7663             {
7664                 \str_set:Nn \l__spinttent_command_name_str {#1}
7665                 \__spinttent_unknown_keys_cmd:n {##1}
7666             },
7667     }
7668 }

```

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

sigma The key `sigma` exists only for `\spdesv` and `\spvarz`, and it starts in `false`.

```

7669 \clist_map_inline:nn { spdesv , spvarz }
7670 {
7671     \keys_define:nn { spinttent / #1 }
7672     {
7673         sigma      .bool_set:c = { \l__spinttent_ #1 _sigma_bool },
7674         sigma      .initial:n = false,
7675         sigma      .value_required:n = true,
7676         read-sub   .initial:n = true,
7677     }
7678 }

```

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`.)

Since the symbol of `\spmin` and `\spmax` already carries a subscript, we define for them only the *(keys)* `silent` and `read-arg`.

```

7679 \clist_map_inline:nn { spmin , spmax }
7680 {
7681   \keys_define:nn { spintent / #1 }
7682   {
7683     silent .bool_set:c = { l__spintent_ #1 _silent_bool },
7684     silent .initial:n = false,
7685     silent .value_required:n = true,
7686     read-arg .code:n =
7687       { \__spintent_sanitizize_affix:cn { l__spintent_ #1 _read_arg_tl } {##1} },
7688     read-arg .initial:n = ,
7689     read-arg .value_required:n = true,
7690     unknown .code:n =
7691       {
7692         \str_set:Nn \__spintent_command_name_str {#1}
7693         \__spintent_unknown_keys_cmd:n {##1}
7694       },
7695   }
7696 }

```

(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.

```

7697 \cs_new_protected:Nn \__spintent_stat_typeset:nnn
7698 {
7699   #2
7700   \str_if_empty:cF { l__spintent_ #1 _sub_index_str }
7701   {
7702     \c_math_subscript_token
7703     {
7704       \MathMLintent { :silent }
7705       { \str_use:c { l__spintent_ #1 _sub_index_str } }
7706     }
7707   }
7708   \tl_if_blank:nF {#3}
7709   { \c_math_superscript_token {#3} }
7710 }

```

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

`\__spintent_stat_mathml:nnnnnn` The function `\__spintent_stat_mathml:nnnnnn` receives the short name of the command in ‘#1’, the symbol in ‘#2’, the exponent in ‘#3’, the name of the concept in ‘#4’, the name of the letter in ‘#5’, empty when the symbol is not a letter, and the spoken form of the exponent in ‘#6’, empty when there is none, the last three already written as they go in the intent, with hyphens. 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:`.

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`.

```

7711 \cs_new_protected:Nn \__spintent_stat_mathml:nnnnnn
7712 {
7713   \__spintent_invisible_sep:
7714   \bool_if:cTF { l__spintent_ #1 _silent_bool }
7715   {
7716     \MathMLintent { - }
7717     {
7718       \__spintent_inner_sep:
7719       \__spintent_stat_typeset:nnn {#1} {#2} {#3}
7720     }
7721   }
7722 }

```

```

7723 \tl_if_empty:cTF { l__spintent_ #1 _read_arg_tl }
7724 {
7725   \str_if_empty:cTF { l__spintent_ #1 _sub_index_str }
7726   {
7727     \MathMLintent
7728     {
7729       - #4
7730       \tl_if_blank:nF {#5} { - #5 }
7731       \tl_if_blank:nF {#6} { - #6 }
7732       :nofix
7733     }
7734     {
7735       \__spintent_inner_sep:
7736       \__spintent_stat_typeset:nnn {#1} {#2} {#3}
7737     }
7738   }
7739   {
7740     \MathMLintent
7741     {
7742       - #4
7743       \tl_if_blank:nF {#5} { - #5 }
7744       -
7745       \bool_if:cTF { l__spintent_ #1 _read_sub_bool } { sub } { de }
7746       - \str_use:c { l__spintent_ #1 _sub_index_str }
7747       \tl_if_blank:nF {#6} { - #6 }
7748     }
7749     {
7750       \__spintent_inner_sep:
7751       \__spintent_stat_typeset:nnn {#1} {#2} {#3}
7752     }
7753   }
7754 }
7755 {
7756   \MathMLintent { - \tl_use:c { l__spintent_ #1 _read_arg_tl } :nofix }
7757   {
7758     \__spintent_inner_sep:
7759     \__spintent_stat_typeset:nnn {#1} {#2} {#3}
7760   }
7761 }
7762 }
7763 }

```

(End of definition for `\__spintent_stat_mathml:nnnnnn`.)

```

\__spintent_stat_exec:nnn
\__spintent_stat_simple:nnn
\__spintent_stat_letter:nnnn

```

The function `\__spintent_stat_exec:nnn` holds what all the commands of the family share. It receives the short name of the command in ‘#1’, the optional *⟨keys⟩* in ‘#2’ and the code that prints the symbol in ‘#3’. It enforces *math mode*, opens a local group, sets the *⟨keys⟩* and only then runs ‘#3’, so that the code can depend on the *⟨keys⟩* already set, as `\sigma` does.

The function `\__spintent_stat_simple:nnn` is the short form of `\__spintent_stat_mathml:nnnnnn` 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 `\__spintent_stat_letter:nnnn` is the short form for the symbols with a letter but no exponent. It receives the same three arguments and the name of the letter in ‘#4’.

```

7764 \cs_new_protected:Nn \__spintent_stat_exec:nnn
7765 {
7766   \mode_if_math:TF
7767   {
7768     \group_begin:
7769     \attribute \l__spintent_scope_attr_int = 1
7770     \keys_set:nn { spintent / #1 } {#2}
7771     #3
7772     \group_end:
7773   }
7774   { \msg_error:nne { spintent } { need-math-mode } { \exp_not:c {#1} } }
7775 }
7776 \cs_new_protected:Nn \__spintent_stat_simple:nnn
7777 {
7778   \__spintent_stat_mathml:nnnnnn {#1} {#2} { } {#3} { } { }
7779 }
7780 \cs_new_protected:Nn \__spintent_stat_letter:nnnn

```

```

7781 {
7782   \__spintent_stat_mathml:nnnnnn {#1} {#2} { } {#3} {#4} { }
7783 }

```

(End of definition for `\__spintent_stat_exec:nnn`, `\__spintent_stat_simple:nnn`, and `\__spintent_stat_letter:nnnn`.)

`\spmedia` The public commands `\spmedia`, `\spmediana` and `\spmoda` delegate execution to `\__spintent_stat_exec:nnn`,
`\spmediana` which prints the symbols MA, MD and MO through `\__spintent_stat_simple:nnn` with the concepts
`\spmoda` media-aritmética, mediana and moda.

```

7784 \NewDocumentCommand \spmedia { 0{} }
7785 {
7786   \__spintent_stat_exec:nnn { spmedia } {#1}
7787   { \__spintent_stat_simple:nnn { spmedia } { \mathrm{MA} } { media-aritmética } }
7788 }
7789 \NewDocumentCommand \spmediana { 0{} }
7790 {
7791   \__spintent_stat_exec:nnn { spmediana } {#1}
7792   { \__spintent_stat_simple:nnn { spmediana } { \mathrm{MD} } { mediana } }
7793 }
7794 \NewDocumentCommand \spmoda { 0{} }
7795 {
7796   \__spintent_stat_exec:nnn { spmoda } {#1}
7797   { \__spintent_stat_simple:nnn { spmoda } { \mathrm{MO} } { moda } }
7798 }

```

(End of definition for `\spmedia`, `\spmediana`, and `\spmoda`. These functions are documented on page 43.)

`\spdesv` The public commands `\spdesv` and `\spvarz` work the same way, but `\spdesv` calls `\__spintent_stat_letter:nnnn`
`\spvarz` and `\spvarz` calls `\__spintent_stat_mathml:nnnnnn` 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.

```

7799 \NewDocumentCommand \spdesv { 0{} }
7800 {
7801   \__spintent_stat_exec:nnn { spdesv } {#1}
7802   {
7803     \bool_if:NTF \__spintent_spdesv_sigma_bool
7804     {
7805       \__spintent_stat_letter:nnnn { spdesv } { \sigma }
7806       { desviación-estándar } { sigma }
7807     }
7808     {
7809       \__spintent_stat_letter:nnnn { spdesv } { s }
7810       { desviación-estándar } { s }
7811     }
7812   }
7813 }
7814 \NewDocumentCommand \spvarz { 0{} }
7815 {
7816   \__spintent_stat_exec:nnn { spvarz } {#1}
7817   {
7818     \bool_if:NTF \__spintent_spvarz_sigma_bool
7819     {
7820       \__spintent_stat_mathml:nnnnnn { spvarz } { \sigma } { 2 }
7821       { varianza } { sigma } { al-cuadrado }
7822     }
7823     {
7824       \__spintent_stat_mathml:nnnnnn { spvarz } { s } { 2 }
7825       { varianza } { s } { al-cuadrado }
7826     }
7827   }
7828 }

```

(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,
`\spriq` RIQ, $x_{\min}$ and $x_{\max}$ and the concepts rango, rango-intercuartil, mínimo and máximo.

```

7829 \NewDocumentCommand \sprango { 0{} }
7830 {
7831   \__spintent_stat_exec:nnn { sprango } {#1}

```

```

7832     { \__spinttent_stat_simple:nnn { sprango } { R } { rango } }
7833   }
7834   \NewDocumentCommand \spriq { 0{} }
7835   {
7836     \__spinttent_stat_exec:nnn { spriq } { #1 }
7837     { \__spinttent_stat_simple:nnn { spriq } { \mathrm{RIQ} } { rango-intercuartil } }
7838   }
7839   \NewDocumentCommand \spmin { 0{} }
7840   {
7841     \__spinttent_stat_exec:nnn { spmin } { #1 }
7842     {
7843       \__spinttent_stat_simple:nnn { spmin }
7844       { x \c_math_subscript_token { \min } } { mínimo }
7845     }
7846   }
7847   \NewDocumentCommand \spmax { 0{} }
7848   {
7849     \__spinttent_stat_exec:nnn { spmax } { #1 }
7850     {
7851       \__spinttent_stat_simple:nnn { spmax }
7852       { x \c_math_subscript_token { \max } } { máximo }
7853     }
7854   }

```

(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

Each command of the family has its own copy of the variable `_read_arg_tl`, so each command can be configured separately with `\setspinttent`. The boolean variable `\__spinttent_stat_pos_error_bool` is set to `true` by `\__spinttent_stat_pos_check:nnn` when the index is rejected, so that nothing is typeset. The integer variable `\__spinttent_stat_pos_value_int` holds the accepted index without leading zeros.

```

7855 \clist_map_inline:nn { spcuartil , spdecil , spPerc }
7856 { \tl_new:c { \__spinttent_ #1 _read_arg_tl } }
7857 \bool_new:N \__spinttent_stat_pos_error_bool
7858 \int_new:N \__spinttent_stat_pos_value_int

```

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

15.59.2 Definition of error messages for the `stat_pos` family

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.

```

7859 \msg_new:nnnn { spinttent } { stat-pos-invalid-index }
7860 {
7861   Invalid~index~' #2 ' ~for~' \tl_trim_spaces:n { #1 } ' ~\msg_line_context:.
7862 }
7863 {
7864   Only~an~integer~without~sign~between~' 1 ' ~and~' #3 ' ~is~accepted.
7865 }

```

(End of definition for `stat-pos-invalid-index`.)

15.59.3 Definition of keys for the `stat_pos` family

We define the $\langle keys \rangle$ `silent` and `read-arg` for the three commands.

```

7866 \clist_map_inline:nn { spcuartil , spdecil , spPerc }
7867 {
7868   \keys_define:nn { spinttent / #1 }
7869   {
7870     silent .bool_set:c = { \__spinttent_ #1 _silent_bool },
7871     silent .initial:n = false,
7872     silent .value_required:n = true,
7873     read-arg .code:n =

```

```

7874         { \__spinttent_sanitizate_affix:cn { \__spinttent_ #1 _read_arg_tl } {##1} },
7875         read-arg .initial:n = ,
7876         read-arg .value_required:n = true,
7877         unknown .code:n =
7878         {
7879             \str_set:Nn \l__spinttent_command_name_str {#1}
7880             \__spinttent_unknown_keys_cmd:n {##1}
7881         },
7882     }
7883 }

```

(End of definition for `silent`, `read-arg`, and `unknown`.)

15.59.4 Internal functions for the `stat_pos` family

`\__spinttent_stat_pos_check:nnn`

The function `\__spinttent_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 `\__spinttent_luafun_clean_split_and_set:n` on the index and accepts it only if it is an integer without sign between 1 and ‘#2’. Otherwise it raises `stat-pos-invalid-index`, whatever the value of `silent`, and sets `\l__spinttent_stat_pos_error_bool` to true.

```

7884 \cs_new_protected:Nn \__spinttent_stat_pos_check:nnn
7885 {
7886     \__spinttent_luafun_clean_split_and_set:n { \exp_not:n {#3} }
7887     \bool_set_true:N \l__spinttent_stat_pos_error_bool
7888     \bool_lazy_and:nnT
7889     { \str_if_eq_p:Vn \l__spinttent_luaset_has_integer_str { true } }
7890     { \tl_if_empty_p:N \l__spinttent_luaset_sign_tl }
7891     {
7892         \bool_lazy_and:nnT
7893         { \int_compare_p:nNn { \l__spinttent_luaset_only_part_int_str } > { 0 } }
7894         { \int_compare_p:nNn { \l__spinttent_luaset_only_part_int_str } < { #2 + 1 } }
7895         { \bool_set_false:N \l__spinttent_stat_pos_error_bool }
7896     }
7897     \bool_if:NT \l__spinttent_stat_pos_error_bool
7898     {
7899         \msg_error:nneee { spinttent } { stat-pos-invalid-index }
7900         { \token_to_str:c {#1} } { \exp_not:n {#3} } {#2}
7901     }
7902 }

```

(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`.

```

7903 \cs_new_protected:Nn \__spinttent_stat_pos_typeset:nn
7904 {
7905     #1
7906     \c_math_subscript_token { \MathMLintent { :silent } {#2} }
7907 }

```

(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 `_⟨read-arg⟩:nofix`. Otherwise the spoken content is the name of the measure followed by the index, as in `_cuartil-1:nofix`.

```

7908 \cs_new_protected:Nn \__spinttent_stat_pos_mathml:nnnn
7909 {
7910     \__spinttent_invisible_sep:
7911     \bool_if:cTF { \l__spinttent_ #1 _silent_bool }
7912     {
7913         \MathMLintent { _- }
7914         {
7915             \__spinttent_inner_sep:
7916             \__spinttent_stat_pos_typeset:nn {#2} {#4}
7917         }
7918     }

```

```

7919 {
7920   \tl_if_empty:cTF { l__spintent_ #1 _read_arg_tl }
7921   {
7922     \MathMLintent { _ #3 - #4 :nofix }
7923     {
7924       \__spintent_inner_sep:
7925       \__spintent_stat_pos_typeset:nn {#2} {#4}
7926     }
7927   }
7928   {
7929     \MathMLintent { _ \tl_use:c { l__spintent_ #1 _read_arg_tl } :nofix }
7930     {
7931       \__spintent_inner_sep:
7932       \__spintent_stat_pos_typeset:nn {#2} {#4}
7933     }
7934   }
7935 }
7936 }
7937 \cs_generate_variant:Nn \__spintent_stat_pos_mathml:nnnn { nnnV }

```

(End of definition for `\__spintent_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`.

```

7938 \cs_new_protected:Nn \__spintent_stat_pos_exec:nnnn
7939 {
7940   \__spintent_stat_pos_check:nnn {#1} {#4} {#5}
7941   \bool_if:NF \__spintent_stat_pos_error_bool
7942   {
7943     \int_set:NV \l__spintent_stat_pos_value_int \l__spintent_luaset_only_part_int_str
7944     \__spintent_stat_pos_mathml:nnnV {#1} {#2} {#3} \l__spintent_stat_pos_value_int
7945   }
7946 }

```

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

`\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:nnnn` with the symbols Q, D and P, the measures cuartil, decil and percentil and the maxima 3, 9 and 99.

```

7947 \NewDocumentCommand \spcuartil { O{} m }
7948 {
7949   \mode_if_math:TF
7950   {
7951     \group_begin:
7952     \attribute \l__spintent_scope_attr_int = 1
7953     \keys_set:nn { spintent / spcuartil } {#1}
7954     \__spintent_stat_pos_exec:nnnn { spcuartil } { Q } { cuartil } { 3 } {#2}
7955     \group_end:
7956   }
7957   { \msg_error:nnn { spintent } { need-math-mode } { \spcuartil } }
7958 }
7959 \NewDocumentCommand \spdecil { O{} m }
7960 {
7961   \mode_if_math:TF
7962   {
7963     \group_begin:
7964     \attribute \l__spintent_scope_attr_int = 1
7965     \keys_set:nn { spintent / spdecil } {#1}
7966     \__spintent_stat_pos_exec:nnnn { spdecil } { D } { decil } { 9 } {#2}
7967     \group_end:
7968   }
7969   { \msg_error:nnn { spintent } { need-math-mode } { \spdecil } }
7970 }
7971 \NewDocumentCommand \spPerc { 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 / spPerc } {#1}
7978 \__spintent_stat_pos_exec:nnnn { spPerc } { P } { percentil } { 99 } {#2}
7979 \group_end:
7980 }
7981 { \msg_error:nnn { spintent } { need-math-mode } { \spPerc } }
7982 }

```

(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 $\langle argument \rangle$, 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`

The string variable `\l__spintent_spFrec_sub_index_str` and the token list variable `\l__spintent_spFrec_read_arg_tl` hold the values of the keys `sub-index` and `read-arg`.

```

7983 \str_new:N \l__spintent_spFrec_sub_index_str
7984 \tl_new:N \l__spintent_spFrec_read_arg_tl

```

(End of definition for `\l__spintent_spFrec_sub_index_str` and `\l__spintent_spFrec_read_arg_tl`.)

The string variable `\l__spintent_spFrec_symbol_str` holds the $\langle argument \rangle$ without spaces, and the token list variable `\l__spintent_spFrec_concept_tl` the name of the frequency, empty while the symbol is not in the table.

```

7985 \str_new:N \l__spintent_spFrec_symbol_str
7986 \tl_new:N \l__spintent_spFrec_concept_tl

```

(End of definition for `\l__spintent_spFrec_symbol_str` and `\l__spintent_spFrec_concept_tl`.)

15.60.2 Definition of error messages for `\spFrec`

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`.

```

7987 \msg_new:nnnn { spintent } { spFrec-invalid-symbol }
7988 {
7989   Invalid~symbol~'#1'~for~'\token_to_str:N \spFrec'~\msg_line_context:.
7990 }
7991 {
7992   Use~'f',~'F',~'f_{r}',~'F_{r}',~'f_{r}\iow_char:N \%',~or~'F_{r}\iow_char:N \%',~or~
7993   give~the~key~'read-arg'.
7994 }
7995 \msg_new:nnnn { spintent } { spFrec-invalid-sub-index }
7996 {
7997   Invalid~key~'sub-index'~for~the~symbol~'#1'~\msg_line_context:.
7998 }
7999 {
8000   Only~the~symbols~'f'~and~'F'~accept~the~key~'sub-index'.
8001 }

```

(End of definition for `spFrec-invalid-symbol` and `spFrec-invalid-sub-index`.)

15.60.3 Definition of keys for `\spFrec`

We define the $\langle keys \rangle$ `silent`, `sub-index`, `read-sub` and `read-arg`, the third one starting in `true`.

```

8002 \keys_define:nn { spintent / spFrec }
8003 {
8004   silent      .bool_set:N = \l__spintent_spFrec_silent_bool,
8005   silent      .initial:n = false,
8006   silent      .value_required:n = true,
8007   sub-index   .code:n =
8008     { \str_set:Nn \l__spintent_spFrec_sub_index_str {#1} },
8009   sub-index   .initial:n = ,
8010   sub-index   .value_required:n = true,

```

```

8011   read-sub .bool_set:N = \l__spinttent_spFrec_read_sub_bool,
8012   read-sub .initial:n = true,
8013   read-sub .value_required:n = true,
8014   read-arg .code:n =
8015     { \l__spinttent_sanitizet_affix:Nn \l__spinttent_spFrec_read_arg_tl {#1} },
8016   read-arg .initial:n = ,
8017   read-arg .value_required:n = true,
8018   unknown .code:n =
8019     {
8020       \str_set:Nn \l__spinttent_command_name_str { spFrec }
8021       \l__spinttent_unknown_keys_cmd:n {#1}
8022     },
8023   }

```

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

15.60.4 Internal functions for \spFrec

`\__spinttent_spFrec_exec:n` The function `\__spinttent_spFrec_exec:n` receives the symbol in ‘#1’. It removes the spaces and looks the result up in the table, setting `\l__spinttent_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 `\__spinttent_stat_simple:nnn`.

```

8024 \cs_new_protected:Nn \__spinttent_spFrec_exec:n
8025 {
8026   \str_set:Nn \l__spinttent_spFrec_symbol_str { \tl_to_str:n {#1} }
8027   \str_remove_all:Nn \l__spinttent_spFrec_symbol_str { ~ }
8028   \tl_clear:N \l__spinttent_spFrec_concept_tl
8029   \str_case:Vn \l__spinttent_spFrec_symbol_str
8030   {
8031     { f }
8032     { \tl_set:Nn \l__spinttent_spFrec_concept_tl { frecuencia-absoluta } }
8033     { F }
8034     { \tl_set:Nn \l__spinttent_spFrec_concept_tl { frecuencia-absoluta-acumulada } }
8035     { f_{r} }
8036     { \tl_set:Nn \l__spinttent_spFrec_concept_tl { frecuencia-relativa } }
8037     { F_{r} }
8038     { \tl_set:Nn \l__spinttent_spFrec_concept_tl { frecuencia-relativa-acumulada } }
8039     { f_{r}\% }
8040     { \tl_set:Nn \l__spinttent_spFrec_concept_tl { frecuencia-porcentual } }
8041     { F_{r}\% }
8042     {
8043       \tl_set:Nn \l__spinttent_spFrec_concept_tl { frecuencia-porcentual-acumulada }
8044     }
8045   }
8046   \bool_lazy_and:nnTF
8047   { \tl_if_empty_p:N \l__spinttent_spFrec_concept_tl }
8048   { \tl_if_empty_p:N \l__spinttent_spFrec_read_arg_tl }
8049   {
8050     \msg_error:nnn { spinttent } { spFrec-invalid-symbol } {#1}
8051   }
8052   {
8053     \bool_lazy_and:nnTF
8054     { \bool_not_p:n { \str_if_empty_p:N \l__spinttent_spFrec_sub_index_str } }
8055     {
8056       \bool_not_p:n
8057       {
8058         \bool_lazy_or_p:nn
8059         { \str_if_eq_p:Vn \l__spinttent_spFrec_symbol_str { f } }
8060         { \str_if_eq_p:Vn \l__spinttent_spFrec_symbol_str { F } }
8061       }
8062     }
8063     {
8064       \msg_error:nnn { spinttent } { spFrec-invalid-sub-index } {#1}
8065     }
8066     {
8067       \__spinttent_stat_simple:nnn { spFrec } {#1}

```

```

8068         { \l__spinttent_spFrec_concept_tl }
8069     }
8070 }
8071 }

```

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

`\spFrec` The public command `\spFrec` enforces *math mode*, opens a local group, sets the *(keys)* and delegates execution to `\l__spinttent_spFrec_exec:n`.

```

8072 \NewDocumentCommand \spFrec { O{} m }
8073 {
8074     \mode_if_math:TF
8075     {
8076         \group_begin:
8077             \attribute \l__spinttent_scope_attr_int = 1
8078             \keys_set:nn { spinttent / spFrec } {#1}
8079             \l__spinttent_spFrec_exec:n {#2}
8080         \group_end:
8081     }
8082     { \msg_error:nnn { spinttent } { need-math-mode } { \spFrec } }
8083 }

```

(End of definition for `\spFrec`. This function is documented on page 45.)

15.61 The command `\spcase`

The user command `\spcase` typesets the symbols of the cases of a probability: `C_{f}` for the favorable cases, `C_{p}` for the possible ones, and `C_{t}` for the total. The letter is the *(argument)*, and the symbol is read with the name of the cases, for example, «casos favorables».

15.61.1 Definition of variables for `\spcase`

`\l__spinttent_spcase_sub_index_str` The token list variable `\l__spinttent_spcase_read_arg_tl` holds the value of the key `read-arg`. This command has no `sub-index` key, so the string variable `\l__spinttent_spcase_sub_index_str` exists only because `\l__spinttent_stat_mathml:nnnnnn` reads it, and always keeps its initial value.

```

8084 \str_new:N \l__spinttent_spcase_sub_index_str
8085 \tl_new:N \l__spinttent_spcase_read_arg_tl

```

(End of definition for `\l__spinttent_spcase_sub_index_str` and `\l__spinttent_spcase_read_arg_tl`.)

`\l__spinttent_spcase_letter_str` The string variable `\l__spinttent_spcase_letter_str` holds the *(argument)* without spaces, and the token list variable `\l__spinttent_spcase_concept_tl` the name of the cases, empty while the letter is not in the table.

```

8086 \str_new:N \l__spinttent_spcase_letter_str
8087 \tl_new:N \l__spinttent_spcase_concept_tl

```

(End of definition for `\l__spinttent_spcase_letter_str` and `\l__spinttent_spcase_concept_tl`.)

15.61.2 Definition of error messages for `\spcase`

`spcase-invalid-letter` The “error” `spcase-invalid-letter` is raised when the *(argument)* is not one of the letters f, p or t.

```

8088 \msg_new:nnnn { spinttent } { spcase-invalid-letter }
8089 {
8090     Invalid~letter~'#1'~for~'\token_to_str:N \spcase'~\msg_line_context:.
8091 }
8092 {
8093     The~letter~must~be~'f',~'p'~or~'t'.
8094 }

```

(End of definition for `spcase-invalid-letter`.)

15.61.3 Definition of keys for `\spcase`

`silent` We define the *(keys)* `silent` and `read-arg`.
`read-arg`
`unknown`

```

8095 \keys_define:nn { spinttent / spcase }
8096 {
8097     silent .bool_set:N = \l__spinttent_spcase_silent_bool,
8098     silent .initial:n = false,
8099     silent .value_required:n = true,
8100     read-arg .code:n =

```

```

8101     { \__spintent_sanitizé_affix:Nn \__spintent_spcase_read_arg_tl {#1} },
8102     read-arg .initial:n = ,
8103     read-arg .value_required:n = true,
8104     unknown .code:n =
8105     {
8106         \str_set:Nn \__spintent_command_name_str { spcase }
8107         \__spintent_unknown_keys_cmd:n {#1}
8108     },
8109 }

```

(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 `\__spintent_spcase_concept_tl` to `casos-favorables`, `casos-posibles` or `casos-totales`. If the letter is not in the table, it is rejected with `spcase-invalid-letter` whatever the value of the keys `silent` and `read-arg`. Otherwise the function delegates the printing of the symbol, with the letter as its subscript, to `\__spintent_stat_simple:nnn`.

```

8110 \cs_new_protected:Nn \__spintent_spcase_exec:n
8111 {
8112     \str_set:Ne \__spintent_spcase_letter_str { \tl_to_str:n {#1} }
8113     \str_remove_all:Nn \__spintent_spcase_letter_str { ~ }
8114     \tl_clear:N \__spintent_spcase_concept_tl
8115     \str_case:Vn \__spintent_spcase_letter_str
8116     {
8117         { f } { \tl_set:Nn \__spintent_spcase_concept_tl { casos-favorables } }
8118         { p } { \tl_set:Nn \__spintent_spcase_concept_tl { casos-posibles } }
8119         { t } { \tl_set:Nn \__spintent_spcase_concept_tl { casos-totales } }
8120     }
8121     \tl_if_empty:NTF \__spintent_spcase_concept_tl
8122     { \msg_error:nnn { spintent } { spcase-invalid-letter } {#1} }
8123     {
8124         \__spintent_stat_simple:nnn { spcase }
8125         { C \c_math_subscript_token { \str_use:N \__spintent_spcase_letter_str } }
8126         { \__spintent_spcase_concept_tl }
8127     }
8128 }

```

(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`.

```

8129 \NewDocumentCommand \spcase { O{} m }
8130 {
8131     \mode_if_math:TF
8132     {
8133         \group_begin:
8134         \attribute \__spintent_scope_attr_int = 1
8135         \keys_set:nn { spintent / spcase } {#1}
8136         \__spintent_spcase_exec:n {#2}
8137         \group_end:
8138     }
8139     { \msg_error:nnn { spintent } { need-math-mode } { \spcase } }
8140 }

```

(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`

`\__spintent_spOmega_sub_index_str`

`\__spintent_spOmega_read_arg_tl`

`\__spintent_spOmega_read_sub_bool`

The string variable `\__spintent_spOmega_sub_index_str` and the token list variable `\__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 `\__spintent_spOmega_read_sub_bool` starts as `true` and the subscript is always spoken with «*sub*».

```

8141 \str_new:N \__spintent_spOmega_sub_index_str

```

```

8142 \tl_new:N          \l__spinttent_sp0mega_read_arg_tl
8143 \bool_new:N         \l__spinttent_sp0mega_read_sub_bool
8144 \bool_set_true:N    \l__spinttent_sp0mega_read_sub_bool

```

(End of definition for `\l__spinttent_sp0mega_sub_index_str`, `\l__spinttent_sp0mega_read_arg_tl`, and `\l__spinttent_sp0mega_read_sub_bool`.)

15.62.2 Definition of keys for `\sp0mega`

We define the *(keys)* `silent`, `sub-index`, `read-arg` and `card`, the last one starting in `false`.

```

8145 \keys_define:nn { spinttent / sp0mega }
8146 {
8147   silent      .bool_set:N = \l__spinttent_sp0mega_silent_bool,
8148   silent      .initial:n = false,
8149   silent      .value_required:n = true,
8150   sub-index   .code:n =
8151     { \str_set:Nn \l__spinttent_sp0mega_sub_index_str {#1} },
8152   sub-index   .initial:n = ,
8153   sub-index   .value_required:n = true,
8154   read-arg    .code:n =
8155     { \__spinttent_sanitizate_affix:Nn \l__spinttent_sp0mega_read_arg_tl {#1} },
8156   read-arg    .initial:n = ,
8157   read-arg    .value_required:n = true,
8158   card        .bool_set:N = \l__spinttent_sp0mega_card_bool,
8159   card        .initial:n = false,
8160   card        .value_required:n = true,
8161   unknown     .code:n =
8162     {
8163       \str_set:Nn \l__spinttent_command_name_str { sp0mega }
8164       \__spinttent_unknown_keys_cmd:n {#1}
8165     },
8166 }

```

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

15.62.3 Internal functions for `\sp0mega`

`\sp0mega` The public command `\sp0mega` 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`.

```

8167 \NewDocumentCommand \sp0mega { 0{} }
8168 {
8169   \mode_if_math:TF
8170   {
8171     \group_begin:
8172     \attribute \l__spinttent_scope_attr_int = 1
8173     \keys_set:nn { spinttent / sp0mega } {#1}
8174     \bool_if:NTF \l__spinttent_sp0mega_card_bool
8175     {
8176       \__spinttent_stat_simple:nnn { sp0mega } { \# \Omega }
8177       { cardinalidad-del-espacio-muestral }
8178     }
8179     {
8180       \__spinttent_stat_simple:nnn { sp0mega } { \Omega }
8181       { espacio-muestral }
8182     }
8183     \group_end:
8184   }
8185   { \msg_error:nnn { spinttent } { need-math-mode } { \sp0mega } }
8186 }

```

(End of definition for `\sp0mega`. This function is documented on page 45.)

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`

We define the *(keys)* `silent` and `use-dfrac`.

```

8187 \keys_define:nn { spinttent / spPval }

```

```

8188 {
8189   silent      .bool_set:N = \l__spinttent_spPval_silent_bool,
8190   silent      .initial:n = false,
8191   silent      .value_required:n = true,
8192   use-dfrac    .bool_set:N = \l__spinttent_spPval_use_dfrac_bool,
8193   use-dfrac    .initial:n = false,
8194   use-dfrac    .value_required:n = true,
8195   unknown     .code:n =
8196     {
8197       \str_set:Nn \l__spinttent_command_name_str { spPval }
8198       \__spinttent_unknown_keys_cmd:n {#1}
8199     },
8200 }

```

(End of definition for `silent`, `use-dfrac`, and `unknown`.)

`\__spinttent_spPval_exec:nnn` The function `\__spinttent_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 `\__spinttent_spPval_mathml:nn`.

```

8201 \cs_new_protected:Nn \__spinttent_spPval_exec:nnn
8202 {
8203   \keys_set:nn { spinttent / spPval } {#1}
8204   \__spinttent_spPval_mathml:nn {#2} {#3}
8205 }

```

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

`\__spinttent_spPval_mathml:nn` The function `\__spinttent_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)`.

```

8206 \cs_new_protected:Nn \__spinttent_spPval_mathml:nn
8207 {
8208   \__spinttent_invisible_sep:
8209   \bool_if:NTF \l__spinttent_spPval_silent_bool
8210     {
8211       \MathMLintent { _- }
8212       {
8213         \__spinttent_inner_sep:
8214         \bool_if:NTF \l__spinttent_spPval_use_dfrac_bool
8215           { \dfrac {#1} {#2} }
8216           { \frac {#1} {#2} }
8217       }
8218     }
8219     {
8220       \MathMLintent { _de:infix($num,$denom) }
8221       {
8222         \__spinttent_inner_sep:
8223         \bool_if:NTF \l__spinttent_spPval_use_dfrac_bool
8224           {
8225             \dfrac { \MathMLarg { num } {#1} } { \MathMLarg { denom } {#2} }
8226           }
8227           { \frac { \MathMLarg { num } {#1} } { \MathMLarg { denom } {#2} } }
8228       }
8229     }
8230 }

```

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

`\spPval` The public command `\spPval` enforces *math mode*, opens a local group and delegates execution to `\__spinttent_spPval_exec:nnn`.

```

8231 \NewDocumentCommand \spPval { O{} m m }
8232 {
8233   \mode_if_math:TF
8234     {
8235       \group_begin:
8236       \__spinttent_spPval_exec:nnn {#1} {#2} {#3}
8237       \group_end:
8238     }
8239     { \msg_error:nnn { spinttent } { need-math-mode } { \spPval } }
8240 }

```

(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`

Variables set by the function `\__spinttent_luafun_spProb_parse_and_set:n`. The string variable `\__spinttent_spProb_luaset_error_str` is `true` when the $\langle argument \rangle$ is invalid. The string variable `\__spinttent_spProb_luaset_op_str` is `none`, `cap`, `cup` or `mid`, and `\__spinttent_spProb_luaset_a_str` and `\__spinttent_spProb_luaset_b_str` hold the event before and after the operator; the second one is empty when there is no operator.

```
8241 \str_new:N \__spinttent_spProb_luaset_error_str
8242 \str_new:N \__spinttent_spProb_luaset_op_str
8243 \str_new:N \__spinttent_spProb_luaset_a_str
8244 \str_new:N \__spinttent_spProb_luaset_b_str
```

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

`\__spinttent_spProb_read_arg_tl` The token list variable `\__spinttent_spProb_read_arg_tl` stores the key `read-arg`.

```
8245 \tl_new:N \__spinttent_spProb_read_arg_tl
```

(End of definition for `\__spinttent_spProb_read_arg_tl`.)

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.

```
8246 \msg_new:nnnn { spinttent } { spProb-invalid-arg }
8247 {
8248   Invalid~argument~'#1'~for~'\token_to_str:N \spProb'~\msg_line_context:.
8249 }
8250 {
8251   Use~one~event,~or~two~joined~by~one~of~'\token_to_str:N \cap',~
8252   '\token_to_str:N \cup',~'\token_to_str:N \mid'~or~'|'~(outside~braces~
8253   and~parentheses).
8254 }
```

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

15.64.3 Definition of keys for `\spProb`

`silent` We define the $\langle keys \rangle$ `silent` and `read-arg`.

```
read-arg
unknown
8255 \keys_define:nn { spinttent / spProb }
8256 {
8257   silent .bool_set:N = \__spinttent_spProb_silent_bool,
8258   silent .initial:n = false,
8259   silent .value_required:n = true,
8260   read-arg .code:n =
8261     { \__spinttent_sanitizize_affix:Nn \__spinttent_spProb_read_arg_tl {#1} },
8262   read-arg .initial:n = ,
8263   read-arg .value_required:n = true,
8264   unknown .code:n =
8265     {
8266       \str_set:Nn \__spinttent_command_name_str { spProb }
8267       \__spinttent_unknown_keys_cmd:n {#1}
8268     },
8269 }
```

(End of definition for `silent`, `read-arg`, and `unknown`.)

15.64.4 Internal functions for `\spProb`

`\__spinttent_spProb_nested_bool` The boolean variable `\__spinttent_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 \__spinttent_spProb_nested_bool
```

(End of definition for `\__spinttent_spProb_nested_bool`.)

```

\__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:`.

```

8271 \cs_new_protected:Nn \__spintent_spProb_cap:
8272 {
8273   \mathbin { \MathMLintent { _y- } { \__spintent_spProb_cap_orig: } }
8274 }
8275 \cs_new_protected:Nn \__spintent_spProb_cup:
8276 {
8277   \mathbin { \MathMLintent { _o } { \__spintent_spProb_cup_orig: } }
8278 }
8279 \cs_new_protected:Nn \__spintent_spProb_mid:
8280 {
8281   \mathrel { \MathMLintent { _dado } { \__spintent_spProb_mid_orig: } }
8282 }
8283 \cs_new_protected:Nn \__spintent_spProb_nested_ops:
8284 {
8285   \bool_if:NF \l__spintent_spProb_nested_bool
8286   {
8287     \bool_set_true:N \l__spintent_spProb_nested_bool
8288     \cs_set_eq:NN \__spintent_spProb_cap_orig: \cap
8289     \cs_set_eq:NN \__spintent_spProb_cup_orig: \cup
8290     \cs_set_eq:NN \__spintent_spProb_mid_orig: \mid
8291     \cs_set_eq:NN \cap \__spintent_spProb_cap:
8292     \cs_set_eq:NN \cup \__spintent_spProb_cup:
8293     \cs_set_eq:NN \mid \__spintent_spProb_mid:
8294   }
8295 }
8296 \cs_new_protected:Nn \__spintent_spProb_event:N
8297 {
8298   \group_begin:
8299     \__spintent_spProb_nested_ops:
8300     \str_use:N #1
8301   \group_end:
8302 }

```

(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.

```

8303 \cs_new_protected:Nn \__spintent_spProb_content:
8304 {
8305   \str_use:N \l__spintent_spProb_luaset_a_str
8306   \str_case:Vn \l__spintent_spProb_luaset_op_str
8307   {
8308     { cap } { \cap \str_use:N \l__spintent_spProb_luaset_b_str }
8309     { cup } { \cup \str_use:N \l__spintent_spProb_luaset_b_str }
8310     { mid } { \mid \str_use:N \l__spintent_spProb_luaset_b_str }
8311   }
8312 }

```

(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 `_(\read-arg):nofix`. Otherwise the intent is `_probabilidad-de:prefix($a)` with the single event tagged as `a`; with an operator, the intent is `_probabilidad-de:prefix($p)`, where `p` is the group of the two events, which carries its own intent `_y-:infix($a,$b)`, `_o:infix($a,$b)` or `_dado:infix($a,$b)` according to the operator, with the two events tagged as `a` and `b`.

```

8313 \cs_new_protected:Nn \__spintent_spProb_mathml:
8314 {
8315   \__spintent_invisible_sep:
8316   \bool_if:NTF \l__spintent_spProb_silent_bool

```

```

8317 {
8318   \MathMLintent { _- }
8319   {
8320     \__spintent_inner_sep:
8321     P \lparen \__spintent_spProb_content: \rparen
8322   }
8323 }
8324 {
8325   \tl_if_empty:NTF \l__spintent_spProb_read_arg_tl
8326   {
8327     \str_if_eq:VnTF \l__spintent_spProb_luaset_op_str { none }
8328     {
8329       \MathMLintent { _probabilidad-de:prefix($a) }
8330       {
8331         \__spintent_inner_sep:
8332         P
8333         \lparen
8334         \MathMLarg { a }
8335         {
8336           \__spintent_inner_sep: \__spintent_spProb_event:N
8337           \l__spintent_spProb_luaset_a_str
8338         }
8339         \rparen
8340       }
8341     }
8342     {
8343       \MathMLintent { _probabilidad-de:prefix($p) }
8344       {
8345         \__spintent_inner_sep:
8346         P
8347         \lparen
8348         \MathMLarg { p }
8349         {
8350           \MathMLintent
8351           {
8352             -
8353             \str_case:Vn \l__spintent_spProb_luaset_op_str
8354             { { cap } { y- } { cup } { o } { mid } { dado } }
8355             :infix($a,$b)
8356           }
8357           {
8358             \__spintent_inner_sep:
8359             \MathMLarg { a }
8360             {
8361               \__spintent_inner_sep: \__spintent_spProb_event:N
8362               \l__spintent_spProb_luaset_a_str
8363             }
8364             \str_case:Vn \l__spintent_spProb_luaset_op_str
8365             { { cap } { \cap } { cup } { \cup } { mid } { \mid } }
8366             \MathMLarg { b }
8367             {
8368               \__spintent_inner_sep: \__spintent_spProb_event:N
8369               \l__spintent_spProb_luaset_b_str
8370             }
8371           }
8372         }
8373         \rparen
8374       }
8375     }
8376   }
8377   {
8378     \MathMLintent { _ \tl_use:N \l__spintent_spProb_read_arg_tl :nofix }
8379     {
8380       \__spintent_inner_sep:
8381       P \lparen \__spintent_spProb_content: \rparen
8382     }
8383   }
8384 }
8385 }

```

(End of definition for __spintent_spProb_mathml:.)

`\__spinttent_spProb_exec:nn` The function `\__spinttent_spProb_exec:nn` sets the *(keys)* in ‘#1’ and calls `\__spinttent_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 `\__spinttent_spProb_mathml:.`

```

8386 \cs_new_protected:Nn \__spinttent_spProb_exec:nn
8387 {
8388   \keys_set:nn { spinttent / spProb } {#1}
8389   \__spinttent_luafun_spProb_parse_and_set:n { \exp_not:n {#2} }
8390   \str_if_eq:VnTF \__spinttent_spProb_luaset_error_str { true }
8391     { \msg_error:nnn { spinttent } { spProb-invalid-arg } {#2} }
8392     { \__spinttent_spProb_mathml: }
8393 }

```

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

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

```

8394 \NewDocumentCommand \spProb { O{} m }
8395 {
8396   \mode_if_math:TF
8397   {
8398     \group_begin:
8399     \attribute \l__spinttent_scope_attr_int = 1
8400     \__spinttent_spProb_exec:nn {#1} {#2}
8401     \group_end:
8402   }
8403   { \msg_error:nnn { spinttent } { need-math-mode } { \spProb } }
8404 }

```

(End of definition for `\spProb`. This function is documented on page 46.)

15.65 The command `\spcomp`

The user command `\spcomp` typesets the complement of an event, as *(event)*^{c} or, with the key `bar`, as `\overline{(event)}`, and reads it «*complemento de*» the event.

15.65.1 Definition of variables for `\spcomp`

The token list variable `\l__spinttent_spcomp_read_arg_tl` stores the key `read-arg`, and the string variable `\l__spinttent_spcomp_wrap_parens_str` the key `wrap-parens`. The boolean variable `\l__spinttent_spcomp_do_paren_bool` tells whether the event is printed in parentheses once the key and the event have been examined. The boolean variable `\l__spinttent_spcomp_compound_bool` is true when the event contains `\cap`, `\cup`, `\setminus` or `\mid`, whatever the keys are; Lua decides it with the function `\__spinttent_luafun_spcomp_compound:n` and leaves the answer in the string variable `\l__spinttent_spcomp_luaset_compound_str`.

```

8405 \tl_new:N \l__spinttent_spcomp_read_arg_tl
8406 \str_new:N \l__spinttent_spcomp_wrap_parens_str
8407 \bool_new:N \l__spinttent_spcomp_do_paren_bool
8408 \bool_new:N \l__spinttent_spcomp_compound_bool
8409 \str_new:N \l__spinttent_spcomp_luaset_compound_str

```

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

15.65.2 Definition of keys for `\spcomp`

`silent` We define the *(keys)* `silent`, `bar`, `wrap-parens` and `read-arg`.
`bar`
`wrap-parens`
`read-arg`
`unknown`

```

8410 \keys_define:nn { spinttent / spcomp }
8411 {
8412   silent .bool_set:N = \l__spinttent_spcomp_silent_bool,
8413   silent .initial:n = false,
8414   silent .value_required:n = true,
8415   bar .bool_set:N = \l__spinttent_spcomp_bar_bool,
8416   bar .initial:n = false,
8417   bar .value_required:n = true,
8418   wrap-parens .choices:nn = { auto , true , false }
8419   {
8420     \int_case:nn { \l_keys_choice_int }
8421     {
8422       {1} { \str_set:Nn \l__spinttent_spcomp_wrap_parens_str { auto } }
8423       {2} { \str_set:Nn \l__spinttent_spcomp_wrap_parens_str { true } }

```

```

8424         {3} { \str_set:Nn \l__spintent_spcomp_wrap_parens_str { false } }
8425     }
8426 },
8427 wrap-parens .initial:n = auto,
8428 wrap-parens .value_required:n = true,
8429 wrap-parens / unknown .code:n =
8430     \msg_error:nneee { spintent } { unknown-choice }
8431     { wrap-parens } { auto,~true,~false } { \exp_not:n {#1} },
8432 read-arg .code:n =
8433     { \__spintent_sanitize_affix:Nn \l__spintent_spcomp_read_arg_tl {#1} },
8434 read-arg .initial:n = ,
8435 read-arg .value_required:n = true,
8436 unknown .code:n =
8437     {
8438         \str_set:Nn \l__spintent_command_name_str { spcomp }
8439         \__spintent_unknown_keys_cmd:n {#1}
8440     },
8441 }

```

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

15.65.3 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.

```

8442 \cs_new_protected:Nn \__spintent_spcomp_set_paren:n
8443 {
8444     \bool_set_false:N \l__spintent_spcomp_do_paren_bool
8445     \bool_set_false:N \l__spintent_spcomp_compound_bool
8446     \__spintent_luafun_spcomp_compound:n { \exp_not:n {#1} }
8447     \str_if_eq:VnT \l__spintent_spcomp_luaseta_compound_str { true }
8448     { \bool_set_true:N \l__spintent_spcomp_compound_bool }
8449     \bool_if:NF \l__spintent_spcomp_bar_bool
8450     {
8451         \str_case:Vn \l__spintent_spcomp_wrap_parens_str
8452         {
8453             { true } { \bool_set_true:N \l__spintent_spcomp_do_paren_bool }
8454             { auto }
8455             {
8456                 \bool_if:NT \l__spintent_spcomp_compound_bool
8457                 { \bool_set_true:N \l__spintent_spcomp_do_paren_bool }
8458             }
8459         }
8460     }
8461 }
8462 \cs_new_protected:Nn \__spintent_spcomp_event:n
8463 {
8464     \group_begin:
8465     \__spintent_spProb_nested_ops:
8466     \bool_if:NTF \l__spintent_spcomp_do_paren_bool
8467     { \lparen #1 \rparen }
8468     { #1 }
8469     \group_end:
8470 }

```

(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`.

```

8471 \cs_new_protected:Nn \__spintent_spcomp_typeset:n
8472 {
8473     \bool_if:NTF \l__spintent_spcomp_bar_bool
8474     { \overline {#1} }
8475     { {#1} ^ { c } }
8476 }

```

(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`.

```

8477 \cs_new_protected:Nn \__spintent_spcomp_exec:nn
8478 {
8479   \keys_set:nn { spintent / spcomp } {#1}
8480   \__spintent_spcomp_set_paren:n {#2}
8481   \__spintent_spcomp_mathml:n {#2}
8482 }

```

(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 `\MathMLint`, 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.

```

8483 \cs_new_protected:Nn \__spintent_spcomp_mathml:n
8484 {
8485   \__spintent_invisible_sep:
8486   \bool_if:NTF \l__spintent_spcomp_silent_bool
8487   {
8488     \MathMLint { _- }
8489     {
8490       \__spintent_inner_sep:
8491       \__spintent_spcomp_typeset:n { \__spintent_spcomp_event:n {#1} }
8492     }
8493   }
8494   {
8495     \tl_if_empty:NTF \l__spintent_spcomp_read_arg_tl
8496     {
8497       \MathMLint
8498       {
8499         \bool_lazy_and:nnTF
8500         { \bool_if_p:N \l__spintent_spcomp_bar_bool }
8501         { \bool_if_p:N \l__spintent_spcomp_compound_bool }
8502         { _complemento-de-la-expresión:prefix($a) }
8503         { _complemento-de:prefix($a) }
8504       }
8505       {
8506         \__spintent_inner_sep:
8507         \__spintent_spcomp_typeset:n
8508         {
8509           \MathMLarg { a }
8510           { \__spintent_inner_sep: \__spintent_spcomp_event:n {#1} }
8511         }
8512       }
8513     }
8514     {
8515       \MathMLint { _ \tl_use:N \l__spintent_spcomp_read_arg_tl :nofix }
8516       {
8517         \__spintent_inner_sep:
8518         \__spintent_spcomp_typeset:n { \__spintent_spcomp_event:n {#1} }
8519       }
8520     }
8521   }
8522 }

```

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

`spcomp-empty-arg` The “error” `spcomp-empty-arg` is raised when the argument of `\spcomp` is missing or *blank*.

```

8523 \msg_new:nnnn { spintent } { spcomp-empty-arg }
8524 {
8525   Empty~argument~for~'\token_to_str:N \spcomp'\msg_line_context:.
8526 }
8527 {
8528   The~event~is~required.
8529 }

```

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

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

```

8530 \NewDocumentCommand \spcomp { O{} m }
8531 {
8532   \mode_if_math:TF
8533   {
8534     \group_begin:
8535       \attribute \l__spintent_scope_attr_int = 1
8536       \tl_if_blank:nTF {#2}
8537       { \msg_error:nn { spintent } { spcomp-empty-arg } }
8538       { \__spintent_spcomp_exec:nn {#1} {#2} }
8539     \group_end:
8540   }
8541   { \msg_error:nnn { spintent } { need-math-mode } { \spcomp } }
8542 }

```

(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` The function `\__spintent_luafun_spcomb_split:n` splits the argument of the command at its first comma of top level, ignoring the commas inside braces, parentheses or brackets, and sets the string variables `\l__spintent_spcomb_luaset_raw_n_str`, what precedes the comma, and `\l__spintent_spcomb_luaset_raw_k_str`, what follows it; the string variable `\l__spintent_spcomb_luaset_n_str` is *true* when the argument has no such comma. The other variables are set by the function `\__spintent_luafun_spcomb_prepare:nnnn`. The string variable `\l__spintent_spcomb_luaset_error_str` is *true* when the arguments are invalid. The string variables `\l__spintent_spcomb_luaset_n_str` and `\l__spintent_spcomb_luaset_k_str` hold the arguments without the surrounding spaces. In the list mode, `\l__spintent_spcomb_luaset_list_tl` holds the multiplicities, each one in its own braces.

```

8543 \str_new:N \l__spintent_spcomb_luaset_single_str
8544 \str_new:N \l__spintent_spcomb_luaset_raw_n_str
8545 \str_new:N \l__spintent_spcomb_luaset_raw_k_str
8546 \str_new:N \l__spintent_spcomb_luaset_error_str
8547 \str_new:N \l__spintent_spcomb_luaset_n_str
8548 \str_new:N \l__spintent_spcomb_luaset_k_str
8549 \tl_new:N \l__spintent_spcomb_luaset_list_tl

```

(End of definition for `\l__spintent_spcomb_luaset_single_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*.

```

8550 \tl_new:N \l__spintent_spcomb_read_arg_tl
8551 \str_new:N \l__spintent_spcomb_kind_str
8552 \str_new:N \l__spintent_spcomb_print_sty_str

```

(End of definition for `\l__spintent_spcomb_read_arg_tl`, `\l__spintent_spcomb_kind_str`, and `\l__spintent_spcomb_print_sty_str`.)

`\l__spintent_spcomb_letter_tl` Variables computed from the keys. The token list variable `\l__spintent_spcomb_letter_tl` holds the letters of the symbol: C, V, A or P, followed by R and/or C; and `\l__spintent_spcomb_concept_tl` the name of the concept that is read, before the final -de. The string variable `\l__spintent_spcomb_mode_str` is one, pair or list according to the arguments the symbol takes, and `\l__spintent_spcomb_problem_str` describes an incompatible combination of keys, it is empty when there is none. The boolean variable `\l__spintent_spcomb_use_binom_bool` is true when the symbol is printed with `\binom`. The sequence `\l__spintent_spcomb_list_seq` holds the multiplicities to print, and the token list `\l__spintent_spcomb_last_tl` the last one.

```

8553 \tl_new:N \l__spintent_spcomb_letter_tl
8554 \tl_new:N \l__spintent_spcomb_concept_tl

```

```

8555 \str_new:N \l__spintent_spcomb_mode_str
8556 \str_new:N \l__spintent_spcomb_problem_str
8557 \bool_new:N \l__spintent_spcomb_use_binom_bool
8558 \seq_new:N \l__spintent_spcomb_list_seq
8559 \tl_new:N \l__spintent_spcomb_last_tl

```

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

15.66.2 Definition of error messages for `\spcomb`

spcomb-invalid-keys
spcomb-missing-arg
spcomb-extra-arg
spcomb-invalid-arg

The “error” `spcomb-invalid-keys` is raised for a combination of keys that does not exist: `circular` without `kind=permutation`, `circular` with `repeat`, or `print-sty=binom` for anything but combinations without repetition. The “error” `spcomb-missing-arg` is raised when `k` is missing and the symbol needs it. The “error” `spcomb-extra-arg` is raised when `k` is given and the symbol does not take it. The “error” `spcomb-invalid-arg` is raised when the arguments are empty, when the list of multiplicities has fewer than two elements or an empty one, or when integers are not possible, because `k` is greater than `n` or the multiplicities add more than `n`.

```

8560 \msg_new:nnnn { spintent } { spcomb-invalid-keys }
8561 {
8562   Incompatible~keys~for~'\token_to_str:N \spcomb':~'#1'~\msg_line_context:.
8563 }
8564 {
8565   'circular'~needs~'kind=permutation'~and~'repeat=false';~'print-sty=binom'~needs~
8566   'kind=combination'~and~'repeat=false'.
8567 }
8568 \msg_new:nnnn { spintent } { spcomb-missing-arg }
8569 {
8570   Missing~'k'~(after~the~comma)~for~'\token_to_str:N \spcomb'~\msg_line_context:.
8571 }
8572 {
8573   This~kind~of~symbol~needs~'k'~or~the~list~of~repetitions~after~the~
8574   comma:~'\token_to_str:N \spcomb {n,k}'.
8575 }
8576 \msg_new:nnnn { spintent } { spcomb-extra-arg }
8577 {
8578   Unexpected~comma~and~'k'~for~'\token_to_str:N \spcomb'~\msg_line_context:.
8579 }
8580 {
8581   Circular~permutations~take~only~'n':~'\token_to_str:N \spcomb {n}'.
8582 }
8583 \msg_new:nnnn { spintent } { spcomb-invalid-arg }
8584 {
8585   Invalid~arguments~'#1'~and~'#2'~for~'\token_to_str:N \spcomb'~\msg_line_context:.
8586 }
8587 {
8588   Use~'n',~'n,k'~or~'n,a,b,...'~(at~least~two~repetitions,~none~empty);~if~
8589   integers,~'k'<=~'n'~and~'a+b+...<=~'n'.
8590 }

```

(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`.

```

8591 \keys_define:nn { spintent / spcomb }
8592 {
8593   silent      .bool_set:N = \l__spintent_spcomb_silent_bool,
8594   silent      .initial:n = false,
8595   silent      .value_required:n = true,
8596   repeat      .bool_set:N = \l__spintent_spcomb_repeat_bool,
8597   repeat      .initial:n = false,
8598   repeat      .value_required:n = true,
8599   circular     .bool_set:N = \l__spintent_spcomb_circular_bool,
8600   circular     .initial:n = false,
8601   circular     .value_required:n = true,
8602   kind         .choices:nn =
8603     { combination , variation , arrangement , permutation }
8604     {
8605       \int_case:nn { \l_keys_choice_int }
8606       {
8607         {1} { \str_set:Nn \l__spintent_spcomb_kind_str { combination } }

```

```

8608         {2} { \str_set:Nn \l__spinttent_spcomb_kind_str { variation } }
8609         {3} { \str_set:Nn \l__spinttent_spcomb_kind_str { arrangement } }
8610         {4} { \str_set:Nn \l__spinttent_spcomb_kind_str { permutation } }
8611     }
8612 },
8613 kind .initial:n = combination,
8614 kind .value_required:n = true,
8615 kind / unknown .code:n =
8616     \msg_error:nneee { spinttent } { unknown-choice }
8617     { kind } { combination,~variation,~arrangement,~permutation }
8618     { \exp_not:n {#1} },
8619 print-sty .choices:nn = { auto , binom , letter }
8620 {
8621     \int_case:nn { \l_keys_choice_int }
8622     {
8623         {1} { \str_set:Nn \l__spinttent_spcomb_print_sty_str { auto } }
8624         {2} { \str_set:Nn \l__spinttent_spcomb_print_sty_str { binom } }
8625         {3} { \str_set:Nn \l__spinttent_spcomb_print_sty_str { letter } }
8626     }
8627 },
8628 print-sty .initial:n = auto,
8629 print-sty .value_required:n = true,
8630 print-sty / unknown .code:n =
8631     \msg_error:nneee { spinttent } { unknown-choice }
8632     { print-sty } { auto,~binom,~letter } { \exp_not:n {#1} },
8633 read-arg .code:n =
8634     { \__spinttent_sanitizet_affix:Nn \l__spinttent_spcomb_read_arg_tl {#1} },
8635 read-arg .initial:n = ,
8636 read-arg .value_required:n = true,
8637 unknown .code:n =
8638     {
8639         \str_set:Nn \l__spinttent_command_name_str { spcomb }
8640         \__spinttent_unknown_keys_cmd:n {#1}
8641     },
8642 }

```

(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.

```

8643 \cs_new_protected:Nn \__spinttent_spcomb_setup:
8644 {
8645     \str_clear:N \l__spinttent_spcomb_problem_str
8646     \str_case:Vn \l__spinttent_spcomb_kind_str
8647     {
8648         { combination }
8649         {
8650             \tl_set:Nn \l__spinttent_spcomb_letter_tl { C }
8651             \tl_set:Nn \l__spinttent_spcomb_concept_tl { combinación }
8652         }
8653         { variation }
8654         {
8655             \tl_set:Nn \l__spinttent_spcomb_letter_tl { V }
8656             \tl_set:Nn \l__spinttent_spcomb_concept_tl { variación }
8657         }
8658         { arrangement }
8659         {
8660             \tl_set:Nn \l__spinttent_spcomb_letter_tl { A }
8661             \tl_set:Nn \l__spinttent_spcomb_concept_tl { arreglo }
8662         }
8663         { permutation }
8664         {
8665             \tl_set:Nn \l__spinttent_spcomb_letter_tl { P }
8666             \tl_set:Nn \l__spinttent_spcomb_concept_tl { permutación }
8667         }
8668     }
8669     \bool_if:NT \l__spinttent_spcomb_circular_bool
8670     {

```

```

8671 \bool_lazy_and:nnTF
8672 { \str_if_eq_p:Vn \l__spintent_spcomb_kind_str { permutation } }
8673 { ! \bool_if_p:N \l__spintent_spcomb_repeat_bool }
8674 {
8675 \tl_put_right:Nn \l__spintent_spcomb_letter_tl { C }
8676 \tl_put_right:Nn \l__spintent_spcomb_concept_tl { -circular }
8677 }
8678 {
8679 \str_set:Nn \l__spintent_spcomb_problem_str
8680 { 'circular'~requires~'kind=permutation'~and~'repeat=false' }
8681 }
8682 }
8683 \bool_if:NT \l__spintent_spcomb_repeat_bool
8684 {
8685 \tl_put_right:Nn \l__spintent_spcomb_letter_tl { R }
8686 \tl_put_right:Nn \l__spintent_spcomb_concept_tl { -con-repetición }
8687 }
8688 \bool_set_false:N \l__spintent_spcomb_use_binom_bool
8689 \bool_lazy_and:nnTF
8690 { \str_if_eq_p:Vn \l__spintent_spcomb_kind_str { combination } }
8691 { ! \bool_if_p:N \l__spintent_spcomb_repeat_bool }
8692 {
8693 \str_if_eq:VnF \l__spintent_spcomb_print_sty_str { letter }
8694 { \bool_set_true:N \l__spintent_spcomb_use_binom_bool }
8695 }
8696 {
8697 \str_if_eq:VnT \l__spintent_spcomb_print_sty_str { binom }
8698 {
8699 \str_if_empty:NT \l__spintent_spcomb_problem_str
8700 {
8701 \str_set:Nn \l__spintent_spcomb_problem_str
8702 { 'print-sty=binom'~requires~'kind=combination'~and~'repeat=false' }
8703 }
8704 }
8705 }
8706 }

```

(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.

```

8707 \cs_new_protected:Nn \__spintent_spcomb_arg_ann:nn
8708 {
8709 \MathMLarg {#1} { \__spintent_inner_sep: #2 }
8710 }
8711 \cs_new_protected:Nn \__spintent_spcomb_arg_plain:nn
8712 {
8713 #2
8714 }
8715 \cs_new_eq:NN \__spintent_spcomb_arg:nn \__spintent_spcomb_arg_plain:nn
8716 \cs_new_protected:Nn \__spintent_spcomb_sep_ann:
8717 {
8718 \MathMLintent { :silent } { , }
8719 }
8720 \cs_new_protected:Nn \__spintent_spcomb_sep_plain:
8721 {
8722 ,
8723 }
8724 \cs_new_eq:NN \__spintent_spcomb_sep: \__spintent_spcomb_sep_plain:
8725 \cs_new_protected:Nn \__spintent_spcomb_last_ann:n
8726 {
8727 \MathMLintent { _y-:prefix($l) }
8728 { \__spintent_inner_sep: \MathMLarg { l } { \__spintent_inner_sep: #1 } }
8729 }
8730 \cs_new_protected:Nn \__spintent_spcomb_last_plain:n
8731 {
8732 #1
8733 }

```

```
8734 \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)`.

```
8735 \cs_new_protected:Nn \__spintent_spcomb_list:
8736 {
8737   \seq_clear:N \__spintent_spcomb_list_seq
8738   \tl_map_inline:Nn \__spintent_spcomb_luaset_list_tl
8739     { \seq_put_right:Nn \__spintent_spcomb_list_seq {##1} }
8740   \seq_pop_right:NN \__spintent_spcomb_list_seq \__spintent_spcomb_last_tl
8741   \seq_use:Nn \__spintent_spcomb_list_seq { \__spintent_spcomb_sep: }
8742   \__spintent_spcomb_sep:
8743   \exp_args:NV \__spintent_spcomb_last:n \__spintent_spcomb_last_tl
8744 }
```

(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.

```
8745 \cs_new_protected:Nn \__spintent_spcomb_symbol:
8746 {
8747   \bool_if:NTF \__spintent_spcomb_use_binom_bool
8748     {
8749       \int_compare:nNnTF { \mathstyle } < { 2 } { \Bigl } { \bigl } \lparen
8750       \genfrac { } { } { } { \opt } { }
8751       {
8752         \__spintent_spcomb_arg:nn { n }
8753         { \str_use:N \__spintent_spcomb_luaset_n_str }
8754       }
8755       {
8756         \__spintent_spcomb_arg:nn { k }
8757         { \str_use:N \__spintent_spcomb_luaset_k_str }
8758       }
8759       \int_compare:nNnTF { \mathstyle } < { 2 } { \Bigr } { \bigr } \rparen
8760     }
8761     {
8762       { \tl_use:N \__spintent_spcomb_letter_tl }
8763       \str_case:Vn \__spintent_spcomb_mode_str
8764       {
8765         { pair }
8766         {
8767           \c_math_subscript_token
8768           {
8769             \__spintent_spcomb_arg:nn { k }
8770             { \str_use:N \__spintent_spcomb_luaset_k_str }
8771           }
8772           ^
8773           {
8774             \__spintent_spcomb_arg:nn { n }
8775             { \str_use:N \__spintent_spcomb_luaset_n_str }
8776           }
8777         }
8778         { one }
8779         {
8780           \c_math_subscript_token
8781           {
8782             \__spintent_spcomb_arg:nn { n }
8783             { \str_use:N \__spintent_spcomb_luaset_n_str }
8784           }
8785         }
8786         { list }
8787         {
8788           \c_math_subscript_token
8789           {
8790             \__spintent_spcomb_arg:nn { n }
8791             { \str_use:N \__spintent_spcomb_luaset_n_str }
8792           }
8793         }
8794       }
8795     }
```

```

8793             ^
8794             {
8795             \_spintent_spcomb_arg:nn { m } { \_spintent_spcomb_list: }
8796             }
8797         }
8798     }
8799 }
8800 }

```

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

`\_spintent_spcomb_mathml:` The function `\_spintent_spcomb_mathml:` prints the symbol inside one native `\MathMLintent`, preceded by the outer ghost `\_spintent_invisible_sep:` and by the inner ghost `\_spintent_inner_sep:`.

With `silent=true` the intent is `'_-'`. With `read-arg` given it is `_(read-arg):nofix`. Otherwise the intent of the whole symbol is the concept followed by `-de:` with one argument it is `_(concept)-de:prefix($c)`, where `c` wraps the symbol with the intent `_elementos:postfix($n)`; with pair it is `_(concept)-de:prefix($c)`, where `c` is the symbol, which carries its own intent `_elementos-tomados-de-a:infix($n,$k)`, except for the binomial coefficient, whose single intent is `_sobre:infix($n,$k)`; with list it is `_(concept)-de:prefix($c)`, where `c` is the symbol, which carries the intent `_elementos-con-repeticiones:infix($n,$m)`, and `m` is the group of the multiplicities, typeset by `\_spintent_spcomb_list:`.

```

8801 \cs_new_protected:Nn \_spintent_spcomb_mathml:
8802 {
8803     \_spintent_invisible_sep:
8804     \bool_if:NTF \l__spintent_spcomb_silent_bool
8805     {
8806         \cs_set_eq:NN \_spintent_spcomb_arg:nn \_spintent_spcomb_arg_plain:nn
8807         \cs_set_eq:NN \_spintent_spcomb_sep: \_spintent_spcomb_sep_plain:
8808         \cs_set_eq:NN \_spintent_spcomb_last:n \_spintent_spcomb_last_plain:n
8809         \MathMLintent { _- }
8810         { \_spintent_inner_sep: \_spintent_spcomb_symbol: }
8811     }
8812     {
8813         \tl_if_empty:NTF \l__spintent_spcomb_read_arg_tl
8814         {
8815             \cs_set_eq:NN \_spintent_spcomb_arg:nn \_spintent_spcomb_arg_ann:nn
8816             \cs_set_eq:NN \_spintent_spcomb_sep: \_spintent_spcomb_sep_ann:
8817             \cs_set_eq:NN \_spintent_spcomb_last:n \_spintent_spcomb_last_ann:n
8818             \str_case:Vn \l__spintent_spcomb_mode_str
8819             {
8820                 { one }
8821                 {
8822                     \MathMLintent
8823                     { _ \tl_use:N \l__spintent_spcomb_concept_tl -de:prefix($c) }
8824                     {
8825                         \_spintent_inner_sep:
8826                         \MathMLarg { c }
8827                         {
8828                             \MathMLintent { _elementos:postfix($n) }
8829                             { \_spintent_inner_sep: \_spintent_spcomb_symbol: }
8830                         }
8831                     }
8832                 }
8833                 { pair }
8834                 {
8835                     \bool_if:NTF \l__spintent_spcomb_use_binom_bool
8836                     {
8837                         \MathMLintent { _sobre:infix($n,$k) }
8838                         { \_spintent_inner_sep: \_spintent_spcomb_symbol: }
8839                     }
8840                     {
8841                         \MathMLintent
8842                         { _ \tl_use:N \l__spintent_spcomb_concept_tl -de:prefix($c) }
8843                         {
8844                             \_spintent_inner_sep:
8845                             \MathMLarg { c }
8846                             {
8847                                 \MathMLintent { _elementos-tomados-de-a:infix($n,$k) }
8848                                 { \_spintent_inner_sep: \_spintent_spcomb_symbol: }
8849                             }
8850                         }
8851                     }
8852                 }
8853             }
8854         }
8855     }
8856 }

```

```

8850         }
8851     }
8852 }
8853 { list }
8854 {
8855     \MathMLintent
8856     { _ \tl_use:N \l__spinttent_spcomb_concept_tl -de:prefix($c) }
8857     {
8858         \__spinttent_inner_sep:
8859         \MathMLarg { c }
8860         {
8861             \MathMLintent { _elementos-con-repeticiones:infix($n,$m) }
8862             { \__spinttent_inner_sep: \__spinttent_spcomb_symbol: }
8863         }
8864     }
8865 }
8866 }
8867 }
8868 {
8869     \cs_set_eq:NN \__spinttent_spcomb_arg:nn \__spinttent_spcomb_arg_plain:nn
8870     \cs_set_eq:NN \__spinttent_spcomb_sep: \__spinttent_spcomb_sep_plain:
8871     \cs_set_eq:NN \__spinttent_spcomb_last:n \__spinttent_spcomb_last_plain:n
8872     \MathMLintent { _ \tl_use:N \l__spinttent_spcomb_read_arg_tl :nofix }
8873     { \__spinttent_inner_sep: \__spinttent_spcomb_symbol: }
8874 }
8875 }
8876 }

```

(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:.`

```

8877 \cs_new_protected:Nn \__spinttent_spcomb_exec:nn
8878 {
8879     \keys_set:nn { spinttent / spcomb } {#1}
8880     \__spinttent_spcomb_setup:
8881     \str_if_empty:NTF \l__spinttent_spcomb_problem_str
8882     {
8883         \__spinttent_luafun_spcomb_split:n { \exp_not:n {#2} }
8884         \bool_if:NTF \l__spinttent_spcomb_circular_bool
8885         {
8886             \str_if_eq:VnTF \l__spinttent_spcomb_luaset_single_str { true }
8887             { \str_set:Nn \l__spinttent_spcomb_mode_str { one } }
8888             { \str_set:Nn \l__spinttent_spcomb_mode_str { extra } }
8889         }
8890         {
8891             \bool_lazy_and:nnTF
8892             { \str_if_eq_p:Vn \l__spinttent_spcomb_kind_str { permutation } }
8893             { \bool_if_p:N \l__spinttent_spcomb_repeat_bool }
8894             {
8895                 \str_if_eq:VnTF \l__spinttent_spcomb_luaset_single_str { true }
8896                 { \str_set:Nn \l__spinttent_spcomb_mode_str { missing } }
8897                 { \str_set:Nn \l__spinttent_spcomb_mode_str { list } }
8898             }
8899         }
8900         \str_if_eq:VnTF \l__spinttent_spcomb_luaset_single_str { true }
8901         {
8902             \str_if_eq:VnTF \l__spinttent_spcomb_kind_str { permutation }
8903             { \str_set:Nn \l__spinttent_spcomb_mode_str { one } }
8904             { \str_set:Nn \l__spinttent_spcomb_mode_str { missing } }
8905         }
8906         { \str_set:Nn \l__spinttent_spcomb_mode_str { pair } }
8907     }
8908 }
8909 \str_case:VnF \l__spinttent_spcomb_mode_str
8910 {
8911     { extra }
8912     { \msg_error:nn { spinttent } { spcomb-extra-arg } }

```

```

8913         { missing }
8914         { \msg_error:nn { spintent } { spcomb-missing-arg } }
8915     }
8916     {
8917         \__spintent_luafun_spcomb_prepare:nnnn
8918         { \l__spintent_spcomb_mode_str }
8919         { \l__spintent_spcomb_luaset_raw_n_str }
8920         { \l__spintent_spcomb_luaset_raw_k_str }
8921         {
8922             \bool_if:NTF \l__spintent_spcomb_repeat_bool
8923             { false } { true }
8924         }
8925         \str_if_eq:NnTF \l__spintent_spcomb_luaset_error_str { true }
8926         {
8927             \msg_error:nnee { spintent } { spcomb-invalid-arg }
8928             { \l__spintent_spcomb_luaset_raw_n_str }
8929             { \l__spintent_spcomb_luaset_raw_k_str }
8930         }
8931         { \__spintent_spcomb_mathml: }
8932     }
8933 }
8934 {
8935     \msg_error:nne { spintent } { spcomb-invalid-keys }
8936     { \l__spintent_spcomb_problem_str }
8937 }
8938 }

```

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

`\__spintent_spcomb_begin:nn`

The function `\__spintent_spcomb_begin:nn` opens a local group, activates the scope attribute and delegates execution to `\__spintent_spcomb_exec:nn`.

```

8939 \cs_new_protected:Nn \__spintent_spcomb_begin:nn
8940 {
8941     \group_begin:
8942     \attribute \l__spintent_scope_attr_int = 1
8943     \__spintent_spcomb_exec:nn {#1} {#2}
8944     \group_end:
8945 }

```

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

`\spcomb`

The public command `\spcomb` enforces *math mode* and delegates execution to `\__spintent_spcomb_begin:nn`, which opens a local group. The mandatory argument ‘`#2`’ is `n` or `n,k`, and Lua splits it.

```

8946 \NewDocumentCommand \spcomb { O{} } m {
8947     {
8948         \mode_if_math:TF
8949         { \__spintent_spcomb_begin:nn {#1} {#2} }
8950         { \msg_error:nnn { spintent } { need-math-mode } { \spcomb } }
8951     }

```

(End of definition for `\spcomb`. This function is documented on page 47.)

15.67 The command `\spfact`

The user command `\spfact` typesets the factorial $n!$ and reads it «*n factorial*». It has the keys `silent` and `read-arg`.

15.67.1 Definition of variables for `\spfact`

`\l__spintent_spfact_read_arg_tl`

The token list variable `\l__spintent_spfact_read_arg_tl` stores the key `read-arg`.

```

8952 \tl_new:N \l__spintent_spfact_read_arg_tl

```

(End of definition for `\l__spintent_spfact_read_arg_tl`.)

15.67.2 Definition of error messages for `\spfact`

`spfact-empty-arg`

The “error” `spfact-empty-arg` is raised when the argument of `\spfact` is missing or *blank*.

```

8953 \msg_new:nnnn { spintent } { spfact-empty-arg }
8954 {
8955     Empty~argument~for~'\token_to_str:N \spfact'~\msg_line_context:.

```

```

8956   }
8957   {
8958     The~number~n~is~required.
8959   }

```

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

15.67.3 Definition of keys for `\spfact`

We define the $\langle keys \rangle$ `silent` and `read-arg`.

```

silent
read-arg
unknown
8960 \keys_define:nn { spintent / spfact }
8961 {
8962   silent .bool_set:N = \l__spintent_spfact_silent_bool,
8963   silent .initial:n = false,
8964   silent .value_required:n = true,
8965   read-arg .code:n =
8966     { \__spintent_sanitizе_affix:Nn \l__spintent_spfact_read_arg_tl {#1} },
8967   read-arg .initial:n = ,
8968   read-arg .value_required:n = true,
8969   unknown .code:n =
8970     {
8971       \str_set:Nn \l__spintent_command_name_str { spfact }
8972       \__spintent_unknown_keys_cmd:n {#1}
8973     },
8974 }

```

(End of definition for `silent`, `read-arg`, and `unknown`.)

15.67.4 Internal functions for `\spfact`

`\__spintent_spfact_exec:nn` The function `\__spintent_spfact_exec:nn` sets the $\langle keys \rangle$ in ‘#1’ and calls `\__spintent_spfact_mathml:n` with the argument in ‘#2’.

```

8975 \cs_new_protected:Nn \__spintent_spfact_exec:nn
8976 {
8977   \keys_set:nn { spintent / spfact } {#1}
8978   \__spintent_spfact_mathml:n {#2}
8979 }

```

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

`\__spintent_spfact_mathml:n` The function `\__spintent_spfact_mathml:n` prints the factorial of ‘#1’ inside one native `\MathMLintent`, preceded by the outer ghost `\__spintent_invisible_sep:` and followed by the inner ghost `\__spintent_inner_sep:`. With `silent=true` the intent is ‘-’; with `read-arg` given it is `_⟨read-arg⟩:nofix`; otherwise it is `_factorial:postfix($n)`.

```

8980 \cs_new_protected:Nn \__spintent_spfact_mathml:n
8981 {
8982   \__spintent_invisible_sep:
8983   \bool_if:NTF \l__spintent_spfact_silent_bool
8984     { \MathMLintent { _- } { \__spintent_inner_sep: #1 ! } }
8985     {
8986       \tl_if_empty:NTF \l__spintent_spfact_read_arg_tl
8987         {
8988           \MathMLintent { _factorial:postfix($n) }
8989           { \__spintent_inner_sep: \MathMLarg { n } { \__spintent_inner_sep: #1 } ! }
8990         }
8991         {
8992           \MathMLintent { _ \tl_use:N \l__spintent_spfact_read_arg_tl :nofix }
8993           { \__spintent_inner_sep: #1 ! }
8994         }
8995       }
8996 }

```

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

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

```

8997 \NewDocumentCommand \spfact { O{} m }
8998 {
8999   \mode_if_math:TF
9000   {

```

```

9001     \group_begin:
9002     \attribute \l__spintent_scope_attr_int = 1
9003     \tl_if_blank:nTF {#2}
9004     { \msg_error:nn { spintent } { spfact-empty-arg } }
9005     { \__spintent_spfact_exec:nn {#1} {#2} }
9006     \group_end:
9007   }
9008   { \msg_error:nnn { spintent } { need-math-mode } { \spfact } }
9009 }

```

(End of definition for \spfact. This function is documented on page 48.)

15.68 Finish package

Finish package implementation.

```

9010 \file_input_stop:
9011 \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>4557</u>
B	
bar	<u>8410</u>
C	
card	<u>8145</u>
cifras-sep	<u>308, 585, 872, 1960, 2035, 2687</u>
circular	<u>8591</u>
Document class:	
l3doc	<u>57</u>
close	<u>4171</u>
cmd-key-unknown	<u>215</u>
cmd-key-value-unknown	<u>215</u>
Commands provided by spanish-mathml-intent :	
\setspintent	<u>63</u>
\spAcirc	<u>207</u>
\spAfig	<u>189</u>
\spFrec	<u>222</u>
\spOmega	<u>224</u>
\spPcirc	<u>207</u>
\spPerc	<u>219</u>
\spPfig	<u>189</u>
\spPoint	<u>153</u>
\spPoly	<u>181</u>
\spProb	<u>229</u>
\spProm	<u>212</u>
\spProp	<u>113</u>
\spPval	<u>225</u>
\spVol	<u>211</u>
\spabs	<u>148</u>
\spangle	<u>174</u>
\spang	<u>115</u>
\sparc	<u>178</u>
\spcase	<u>223</u>
\spcirc	<u>200</u>
\spcod	<u>145</u>
\spcomb	<u>239</u>
\spcomp	<u>232</u>
\spcoord	<u>126</u>
\spcuartil	<u>219</u>
\spdate	<u>92</u>
\spdecil	<u>219</u>
\spdesv	<u>216</u>
\spdiam	<u>194</u>
\spdiv	<u>101</u>
\spdom	<u>145</u>
\spdsym	<u>98</u>
\spfact	<u>240</u>
\spfuncomp	<u>143</u>
\spfun	<u>139</u>
\spinterval	<u>134</u>
\spmQ	<u>122</u>
\spmangle	<u>174</u>
\spmarc	<u>178</u>
\spmmax	<u>216</u>
\spmcd	<u>107</u>
\spmcm	<u>107</u>
\spmediana	<u>216</u>
\spmedia	<u>216</u>
\spmin	<u>216</u>
\spmoda	<u>216</u>
\spmoney	<u>83</u>
\spmside	<u>169</u>
\spmsym	<u>100</u>
\spmul	<u>103</u>
\spnD	<u>111</u>
\spnM	<u>111</u>
\spnZ	<u>116</u>
\spnewread	<u>96</u>
\spnewunit	<u>76</u>
\spnum	<u>70</u>
\sprad	<u>194</u>
\sprango	<u>216</u>
\sprayo	<u>164</u>
\sprazon	<u>113</u>
\spread	<u>96</u>
\sprecta	<u>162</u>
\sprec	<u>145</u>
\spriq	<u>216</u>
\spshort	<u>86</u>
\spside	<u>169</u>
\spsiglo	<u>89</u>
\sptime	<u>91</u>
\spunitalias	<u>76</u>
\spunit	<u>75</u>
\spusep	<u>94</u>
\spvarz	<u>216</u>
\spvec	<u>157</u>
concept	<u>2115, 2352, 3233, 3625, 4447, 4557, 4806, 5062, 5205, 5420, 5719, 5876, 6063, 6487, 6743, 7053, 7376</u>
concept*	<u>6063</u>
currency	<u>872</u>
D	
diameter	<u>6743</u>
I	
inline	<u>2532</u>
int commands:	
\int_set:Nn	<u>70</u>
J	
join-word	<u>2765</u>
K	
Keys for \spabs provided by spanish-mathml-intent :	
close	<u>145-147</u>
read-arg	<u>145, 146</u>
silent	<u>145, 146</u>
size	<u>145-147</u>
Keys for \spAcirc provided by spanish-mathml-intent :	
concept	<u>201</u>
prefix	<u>201</u>
print-sym	<u>201, 203</u>
read-arg	<u>201</u>
read-cmd	<u>201</u>
read-sty	<u>201</u>
read-sub	<u>201</u>

silent	201	read-arg	190
sub-index	201	read-cmd	190
Keys for \spAfig provided by spanish-mathml-intent:		read-sub	190
concept	182	silent	190
prefix	181, 182	sub-index	190
print-sym	181, 182	upper-case	190
read-arg	181, 182, 184	Keys for \spdiv provided by spanish-mathml-intent:	
read-cmd	181, 182	cifras-sep	100
read-sub	182	notation-sym	100
silent	182	read-parens	100
Keys for \spangle provided by spanish-mathml-intent:		read-period-sep	100
concept	170	read-sign	100
prefix	169, 170	read-sym	100
print-sym	170	set-sym	100
read-cmd	169, 170	wrap-parens	100
recto	169, 170	Keys for \spdom provided by spanish-mathml-intent:	
silent	170	prefix	143, 144
Keys for \sparc provided by spanish-mathml-intent:		silent	144
concept	175	Keys for \spdsym provided by spanish-mathml-intent:	
prefix	175	prefix	97
read-cmd	175	read-sym	97
silent	175	set-sym	97
Keys for \spcase provided by spanish-mathml-intent:		suffix	97
read-arg	222, 223	Keys for \spfact provided by spanish-mathml-intent:	
silent	222, 223	read-arg	239, 240
Keys for \spcirc provided by spanish-mathml-intent:		silent	239, 240
concept	195	Keys for \spFrec provided by spanish-mathml-intent:	
diameter	195	read-arg	220, 221
prefix	195	read-sub	220
print-sym	195, 197	silent	220
radio-sty	195	sub-index	220
read-arg	195	Keys for \spfuncomp provided by spanish-mathml-intent:	
read-cmd	195	silent	140
read-sty	195	Keys for \spfun provided by spanish-mathml-intent:	
read-sub	195	concept	134, 135
silent	195	prefix	134, 135
sub-index	195	read-cmd	134, 135
Keys for \spcomb provided by spanish-mathml-intent:		silent	134, 135
circular	232, 233	Keys for \spinterval provided by spanish-mathml-intent:	
kind	232, 233	concept	128, 131
print-sty	232, 233	prefix	127, 128, 131
read-arg	232, 233	read-arg	127, 128, 131
repeat	232, 233	read-cmd	127, 128, 132
silent	233	silent	128, 131, 134
Keys for \spcomp provided by spanish-mathml-intent:		size	127–129
bar	229–231	Keys for \spmangle provided by spanish-mathml-intent:	
read-arg	229	concept	170
silent	229	prefix	169, 170
wrap-parens	229, 230	print-sym	169, 170
Keys for \spcoord provided by spanish-mathml-intent:		read-cmd	169, 170
silent	123	recto	169, 170
Keys for \spcuartil provided by spanish-mathml-intent:		silent	170
read-arg	217	Keys for \spmarc provided by spanish-mathml-intent:	
silent	217	concept	175
Keys for \spdecil provided by spanish-mathml-intent:		prefix	175
read-arg	217	read-cmd	175
silent	217	silent	175
Keys for \spdesv provided by spanish-mathml-intent:		Keys for \spmax provided by spanish-mathml-intent:	
read-arg	213	read-arg	214
read-sub	213	silent	214
sigma	213	Keys for \spmcd provided by spanish-mathml-intent:	
silent	213	concept	103
sub-index	213	prefix	103
Keys for \spdiam provided by spanish-mathml-intent:		read-sty	103
concept	190	result	103

silent	103	result-num	108
upper-case	103	result	108
Keys for \spmcn provided by spanish-mathml-intent:		silent	108
concept	103	Keys for \spnM provided by spanish-mathml-intent:	
prefix	103	concept	108
read-sty	103	prefix	108
result	103	result-num	108
silent	103	result	108
upper-case	103	silent	108
Keys for \spmediana provided by spanish-mathml-intent:		Keys for \spnum provided by spanish-mathml-intent:	
read-arg	213	cifras-sep	64, 66, 67
read-sub	213	notation-sym	65, 66
silent	213	read-period-sep	65, 66, 68
sub-index	213	read-sign	65, 66
Keys for \spmedia provided by spanish-mathml-intent:		Keys for \spnZ provided by spanish-mathml-intent:	
read-arg	213	cifras-sep	115
read-sub	213	read-parens	115
silent	213	read-sign	115
sub-index	213	wrap-parens	115
Keys for \spmin provided by spanish-mathml-intent:		Keys for \spOmega provided by spanish-mathml-intent:	
read-arg	214	card	223, 224
silent	214	read-arg	223, 224
Keys for \spmoda provided by spanish-mathml-intent:		silent	224
read-arg	213	sub-index	223, 224
read-sub	213	Keys for \spPcirc provided by spanish-mathml-intent:	
silent	213	concept	201
sub-index	213	prefix	201
Keys for \spmoney provided by spanish-mathml-intent:		print-sym	201, 203
cifras-sep	78	read-arg	201
currency	78	read-cmd	201
money-math	78	read-sty	201
sub-units	78	read-sub	201
Keys for \spmQ provided by spanish-mathml-intent:		silent	201
join-word	116, 117, 119	sub-index	201
read-parens	117, 119	Keys for \spPerc provided by spanish-mathml-intent:	
silent	117, 121, 122	read-arg	217
use-dfrac	117–121	silent	217
use-spN	117, 120, 121	Keys for \spPfig provided by spanish-mathml-intent:	
wrap-parens	116–118	concept	182
Keys for \spmside provided by spanish-mathml-intent:		prefix	181, 182
concept	165	print-sym	181, 182
prefix	165	read-arg	181, 182, 184
print-sym	165, 190	read-cmd	181, 182
read-arg	165	read-sub	182
read-cmd	165, 168	silent	182
read-sty	165	Keys for \spPoint provided by spanish-mathml-intent:	
silent	165, 168	concept	151
Keys for \spmsym provided by spanish-mathml-intent:		prefix	151
prefix	98	read-cmd	151
print-sym	98, 99	silent	151
read-sym	98	Keys for \spPoly provided by spanish-mathml-intent:	
set-sym	98	concept	178
suffix	98	prefix	178
Keys for \spmul provided by spanish-mathml-intent:		print-sym	178
cifras-sep	102	read-cmd	178
notation-sym	102	silent	178
read-parens	102	Keys for \spProb provided by spanish-mathml-intent:	
read-period-sep	102	read-arg	226
read-sign	102	silent	226
read-sym	102	Keys for \spProm provided by spanish-mathml-intent:	
set-sym	102	read-arg	211
wrap-parens	102	silent	211
Keys for \spnD provided by spanish-mathml-intent:		Keys for \spPval provided by spanish-mathml-intent:	
concept	108	silent	224
prefix	108	use-dfrac	224

©2026 by Pablo González L

S

seq commands:

<code>\seq_set_from_clist:Nn</code>	70
<code>set-sym</code>	1799 , 1874 , 1960 , 2035
<code>\setspintent</code>	6 , 62 , 202
<code>sigma</code>	7669
<code>silent</code>	1573 , 1715 , 2115 , 2352 , 2532 , 2765 , 3027 , 3233 , 3625 , 3875 , 4060 , 4171 , 4447 , 4557 , 4806 , 5062 , 5205 , 5420 , 5719 , 5876 , 6063 , 6487 , 6743 , 7053 , 7376 , 7577 , 7641 , 7679 , 7866 , 8002 , 8095 , 8145 , 8187 , 8255 , 8410 , 8591 , 8960
<code>size</code>	1573 , 3233 , 4171
<code>small-caps</code>	1162
<code>\spabs</code>	41 , 145 , 4305
<code>spabs-empty-arg</code>	4164
<code>\spAcirc</code>	35 , 201 , 7343
<code>spAcirc-invalid-arg</code>	7044
<code>\spAfig</code>	30 , 181 , 6452
<code>spAfig-invalid-arg</code>	6023
<code>spAfig-invalid-points</code>	6023
<code>spAfig-read-arg-key</code>	6023
<code>spAfig-sym-cardinality-conflict</code>	6023
<code>spAfig-sym-letter-conflict</code>	6023
<code>\spang</code>	28 , 113 , 2670
<code>spang-invalid-arg</code>	2625
<code>\spangle</code>	27 , 169 , 5682
<code>spangle-invalid-arg</code>	5409
<code>\sparc</code>	29 , 175 , 5845
<code>sparc-invalid-points</code>	5709
<code>\spcase</code>	44 , 222 , 8129
<code>spcase-invalid-letter</code>	8088
<code>\spcirc</code>	34 , 194 , 7022
<code>spcirc-invalid-arg</code>	6736
<code>\spcod</code>	40 , 143 , 4127
<code>\spcomb</code>	47 , 232 , 8946
<code>spcomb-extra-arg</code>	8560
<code>spcomb-invalid-arg</code>	8560
<code>spcomb-invalid-keys</code>	8560
<code>spcomb-missing-arg</code>	8560
<code>\spcomp</code>	46 , 229 , 8530
<code>spcomp-empty-arg</code>	8523
<code>\spcoord</code>	23 , 123 , 3178
<code>spcoord-invalid-arg</code>	3020
<code>\spcuartil</code>	43 , 7947
<code>\spdate</code>	15 , 91 , 1545
<code>spdate-invalid-arg</code>	1520
<code>\spdecil</code>	43 , 7947
<code>\spdesv</code>	42 , 7799
<code>\spdiam</code>	33 , 190 , 6703
<code>\spdiv</code>	17 , 100 , 2018
<code>spdiv-empty-arg</code>	1953
<code>\spdom</code>	40 , 143 , 4127
<code>spdom-invalid-arg</code>	4053
<code>\spdsym</code>	16 , 97 , 1859
<code>\spfact</code>	47 , 239 , 8997
<code>spfact-empty-arg</code>	8953
<code>\spFrec</code>	44 , 220 , 8072
<code>spFrec-invalid-sub-index</code>	7987
<code>spFrec-invalid-symbol</code>	7987
<code>\spfun</code>	39 , 134 , 3848
<code>spfun-invalid-arg</code>	3617

<code>\spfuncomp</code>	41 , 139 , 4036
<code>spfuncomp-invalid-arg</code>	3868
spintent internal commands:	
<code>__spintent_collapse_hyphens:N</code>	134
<code>\l__spintent_command_name_str</code>	215
<code>\l__spintent_geo_final_print_tl</code>	6015
<code>\l__spintent_geo_luaset_angulo_case_-str</code>	2992
<code>\l__spintent_geo_luaset_circ_radio_raw_-str</code>	6725
<code>\l__spintent_geo_luaset_circ_radio_type_-str</code>	6725
<code>\l__spintent_geo_luaset_concept_str</code> ...	2992
<code>\l__spintent_geo_luaset_error_str</code> ...	2992
<code>\l__spintent_geo_luaset_intent_str</code> ...	2992
<code>\l__spintent_geo_luaset_is_name_str</code> ...	2992
<code>\l__spintent_geo_luaset_letra_str</code> ...	2992
<code>\l__spintent_geo_luaset_letra_word_str</code>	2992
<code>\l__spintent_geo_luaset_nombre_es_num_-str</code>	2992
<code>\l__spintent_geo_luaset_points_clist</code> ..	2992
<code>\l__spintent_geo_luaset_poly_sym_str</code> ..	2992
<code>\l__spintent_geo_luaset_print_tl</code> ...	2992
<code>\l__spintent_geo_luaset_sub_spoken_str</code>	2992
<code>\l__spintent_geo_luaset_sub_str</code>	2992
<code>\l__spintent_geo_luaset_tipo_str</code> ...	2992
<code>\l__spintent_geo_print_sym_str</code>	6133
<code>\l__spintent_geo_temp_tl</code>	6133
<code>__spintent_inner_sep:</code>	126
<code>__spintent_invisible_sep:</code>	122
<code>\c__spintent_invisible_sep_int</code>	130
<code>__spintent_luafun_clean_split_and_-set:n</code>	252
<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>	270
<code>\l__spintent_luaset_arg_below_tl</code>	270
<code>\l__spintent_luaset_cal_fam_ok_str</code> ...	73
<code>\l__spintent_luaset_dec_is_all_zeros_-str</code>	263
<code>\l__spintent_luaset_dec_sep_tl</code>	252
<code>\l__spintent_luaset_decimal_period_str</code>	263
<code>\l__spintent_luaset_denom_has_number_-str</code>	270
<code>\l__spintent_luaset_exponent_tl</code>	263
<code>\l__spintent_luaset_format_part_dec_str</code> ..	258
<code>\l__spintent_luaset_format_part_int_str</code> ..	258
<code>\l__spintent_luaset_has_exponent_str</code> ..	263
<code>\l__spintent_luaset_has_integer_str</code> ...	252
<code>\l__spintent_luaset_has_units_str</code> ...	270
<code>\l__spintent_luaset_millions_str</code>	263
<code>\l__spintent_luaset_not_decimal_not_-period_str</code>	263
<code>\l__spintent_luaset_not_decimal_period_-str</code>	263
<code>\l__spintent_luaset_only_part_dec_str</code> .	258
<code>\l__spintent_luaset_only_part_int_str</code> .	258

<code>\l__spintent_luaset_only_part_period_-str</code>	258	<code>\l__spintent_spAfig_read_arg_tl</code>	6016
<code>\l__spintent_luaset_part_dec_tl</code>	252	<code>__spintent_spang_exec:n</code>	2633
<code>\l__spintent_luaset_part_int_tl</code>	252	<code>\l__spintent_spang_luaset_error_str</code> ...	2621
<code>\l__spintent_luaset_part_period_tl</code> ...	252	<code>\l__spintent_spang_luaset_grado_str</code> ...	2621
<code>\l__spintent_luaset_scope_attr_str</code>	73	<code>\l__spintent_spang_luaset_minuto_str</code> ..	2621
<code>\l__spintent_luaset_sign_tl</code>	252	<code>\l__spintent_spang_luaset_segundo_str</code> .	2621
<code>\l__spintent_luaset_status_str</code>	270	<code>__spintent_spang_mathml:</code>	2643
<code>\l__spintent_luaset_sym_fam_ok_str</code>	73	<code>__spintent_spangle_exec:nnn</code>	5486
<code>__spintent_mathcal:n</code>	100	<code>\l__spintent_spangle_intent_arg_tl</code> ...	5479
<code>__spintent_mathmlarg_point:nnn</code>	4322	<code>\l__spintent_spangle_intent_tl</code>	5479
<code>\l__spintent_mathmlarg_point_kv_tl</code> ...	4318	<code>\l__spintent_spangle_kv_tl</code>	5479
<code>\l__spintent_mathmlarg_point_mode_str</code> .	4318	<code>__spintent_spangle_mathml:n</code>	5523
<code>\l__spintent_mathmlarg_point_ref_tl</code> ...	4318	<code>\l__spintent_spangle_mode_str</code>	5404
<code>\l__spintent_mathmlarg_point_silent_-bool</code>	4318	<code>\l__spintent_spangle_prefix_tl</code>	5404
<code>\l__spintent_mc_args_seq</code>	2103	<code>\l__spintent_spangle_print_sym_now_-bool</code>	5479
<code>\l__spintent_mc_idx_int</code>	2103	<code>\l__spintent_spangle_pt_idx_int</code>	5479
<code>\l__spintent_mc_luaset_all_numeric_str</code>	2103	<code>\l__spintent_spangle_pt_seq</code>	5479
<code>\l__spintent_mc_luaset_types_clist</code> ...	2103	<code>\l__spintent_spangle_refs_tl</code>	5479
<code>__spintent_mc_mathml_args:n</code>	2176	<code>__spintent_spangle_sym:n</code>	5460
<code>__spintent_mc_process_args:nn</code>	2156	<code>__spintent_sparc_exec:nnn</code>	5768
<code>\l__spintent_mc_raw_args_seq</code>	2103	<code>\l__spintent_sparc_intent_arg_tl</code>	5763
<code>\l__spintent_mc_types_seq</code>	2103	<code>\l__spintent_sparc_intent_tl</code>	5763
<code>\l__spintent_md_out_seq</code>	2329	<code>__spintent_sparc_mathml:n</code>	5786
<code>__spintent_print_spunit_hierarchy:</code> ...	675	<code>\l__spintent_sparc_mode_str</code>	5704
<code>__spintent_right_angle_sqr:</code>	113	<code>\l__spintent_sparc_prefix_tl</code>	5704
<code>__spintent_sanitize_affix:Nn</code>	142	<code>__spintent_sparc_print_arc:n</code>	5757
<code>\l__spintent_saved_luamml_flag_int</code> ...	2737	<code>\l__spintent_sparc_pt_idx_int</code>	5763
<code>\l__spintent_scope_attr_int</code>	80	<code>\l__spintent_sparc_pt_seq</code>	5763
<code>\l__spintent_setspintent_input_arg_cmd_-tl</code>	165	<code>\l__spintent_sparc_refs_tl</code>	5763
<code>__spintent_setspintent_parse_args:nn</code> .	185	<code>__spintent_sparc_wide_arc:n</code>	5753
<code>__spintent_setspintent_set_keys:nn</code> ...	174	<code>\l__spintent_spcase_concept_tl</code>	8086
<code>__spintent_sp_rad_diam_exec:nn</code>	6698	<code>__spintent_spcase_exec:n</code>	8110
<code>\l__spintent_sp_rad_diam_letra_tl</code>	6478	<code>\l__spintent_spcase_letter_str</code>	8086
<code>__spintent_sp_rad_diam_mathml:n</code>	6534	<code>\l__spintent_spcase_read_arg_tl</code>	8084
<code>\l__spintent_sp_rad_diam_read_arg_full_-tl</code>	6478	<code>\l__spintent_spcase_sub_index_str</code>	8084
<code>__spintent_sp_rad_diam_read_arg_-print:n</code>	6660	<code>\l__spintent_spcirc_articulo_str</code>	6727
<code>\l__spintent_sp_rad_diam_word_tl</code>	6478	<code>__spintent_spcirc_build_concept:</code>	6895
<code>\l__spintent_spabs_close_bool</code>	4162	<code>\l__spintent_spcirc_connector_arg_tl</code> ..	6841
<code>\l__spintent_spabs_close_mode_str</code>	4157	<code>\l__spintent_spcirc_connector_tl</code>	6841
<code>\l__spintent_spabs_delim_l_tl</code>	4160	<code>__spintent_spcirc_exec:nn</code>	6950
<code>\l__spintent_spabs_delim_r_tl</code>	4160	<code>\l__spintent_spcirc_has_arg_bool</code>	6841
<code>__spintent_spabs_exec:nn</code>	4261	<code>\l__spintent_spcirc_intent_arg_tl</code>	6841
<code>\l__spintent_spabs_luaset_close_str</code> ...	4162	<code>\l__spintent_spcirc_intent_tl</code>	6841
<code>__spintent_spabs_mathml:n</code>	4267	<code>__spintent_spcirc_mathml:</code>	6967
<code>\l__spintent_spabs_read_arg_tl</code>	4157	<code>\l__spintent_spcirc_mode_str</code>	6727
<code>__spintent_spabs_set_close:n</code>	4247	<code>\l__spintent_spcirc_prefix_tl</code>	6727
<code>__spintent_spabs_set_delims:</code>	4216	<code>\l__spintent_spcirc_print_sym_str</code>	6727
<code>\l__spintent_spabs_size_str</code>	4157	<code>\l__spintent_spcirc_radio_arg_tl</code>	6841
<code>\l__spintent_spAcirc_mode_str</code>	7035	<code>__spintent_spcirc_radio_print:</code>	6878
<code>\l__spintent_spAcirc_prefix_tl</code>	7035	<code>\l__spintent_spcirc_radio_sty_str</code>	6727
<code>\l__spintent_spAcirc_print_sym_str</code> ...	7035	<code>\l__spintent_spcirc_read_arg_tl</code>	6727
<code>\l__spintent_spAcirc_read_arg_tl</code>	7035	<code>\l__spintent_spcirc_read_sty_str</code>	6727
<code>\l__spintent_spAcirc_read_sty_str</code>	7035	<code>\l__spintent_spcirc_sub_index_str</code>	6727
<code>\l__spintent_spAcirc_sub_index_str</code> ...	7035	<code>__spintent_spcirc_sym:</code>	6847
<code>\l__spintent_spAfig_mode_str</code>	6016	<code>__spintent_spcirc_word_and_article:</code> ..	6857
<code>\l__spintent_spAfig_prefix_tl</code>	6016	<code>\l__spintent_spcirc_word_tl</code>	6735
<code>\l__spintent_spAfig_print_sym_str</code>	6016	<code>__spintent_spcircfig_build_concept:nn</code>	7206
		<code>__spintent_spcircfig_build_short:n</code> ...	7165
		<code>__spintent_spcircfig_del_word:n</code>	7152
		<code>\l__spintent_spcircfig_del_word_tl</code> ...	7122
		<code>__spintent_spcircfig_exec:nnn</code>	7253

<code>__spintent_spcircfig_finish_intent:nn</code>	7190	<code>\l__spintent_spcircfig_intent_arg_tl ..</code>	7122	<code>\l__spintent_spcircfig_intent_tl</code>	7122	<code>\l__spintent_spcircfig_luaset_comma_-</code>		<code>\l__spintent_spcircfig_luaset_comma_-</code>		<code>str</code>	7122	<code>__spintent_spcircfig_mathml:nn</code>	7300	<code>\l__spintent_spcircfig_op_tl</code>	7122	<code>\l__spintent_spcircfig_short_bool</code>	7122	<code>__spintent_spcircfig_sym:n</code>	7128	<code>__spintent_spcircfig_word_and_-</code>		<code>article:n</code>	7134	<code>__spintent_spcomb_arg:nn</code>	8707	<code>__spintent_spcomb_arg_ann:nn</code>	8707	<code>__spintent_spcomb_arg_plain:nn</code>	8707	<code>__spintent_spcomb_begin:nn</code>	8939	<code>\l__spintent_spcomb_concept_tl</code>	8553	<code>__spintent_spcomb_exec:nn</code>	8877	<code>\l__spintent_spcomb_kind_str</code>	8550	<code>__spintent_spcomb_last:n</code>	8707	<code>__spintent_spcomb_last_ann:n</code>	8707	<code>__spintent_spcomb_last_plain:n</code>	8707	<code>\l__spintent_spcomb_last_tl</code>	8553	<code>\l__spintent_spcomb_letter_tl</code>	8553	<code>__spintent_spcomb_list:</code>	8735	<code>\l__spintent_spcomb_list_seq</code>	8553	<code>\l__spintent_spcomb_luaset_error_str ..</code>	8543	<code>\l__spintent_spcomb_luaset_k_str</code>	8543	<code>\l__spintent_spcomb_luaset_list_tl ...</code>	8543	<code>\l__spintent_spcomb_luaset_n_str</code>	8543	<code>\l__spintent_spcomb_luaset_raw_k_str ..</code>	8543	<code>\l__spintent_spcomb_luaset_raw_n_str ..</code>	8543	<code>\l__spintent_spcomb_luaset_single_str .</code>	8543	<code>__spintent_spcomb_mathml:</code>	8801	<code>\l__spintent_spcomb_mode_str</code>	8553	<code>\l__spintent_spcomb_print_sty_str</code>	8550	<code>\l__spintent_spcomb_problem_str</code>	8553	<code>\l__spintent_spcomb_read_arg_tl</code>	8550	<code>__spintent_spcomb_sep:</code>	8707	<code>__spintent_spcomb_sep_ann:</code>	8707	<code>__spintent_spcomb_sep_plain:</code>	8707	<code>__spintent_spcomb_setup:</code>	8643	<code>__spintent_spcomb_symbol:</code>	8745	<code>\l__spintent_spcomb_use_binom_bool ...</code>	8553	<code>\l__spintent_spcomp_compound_bool</code>	8405	<code>\l__spintent_spcomp_do_paren_bool</code>	8405	<code>__spintent_spcomp_event:n</code>	8442	<code>__spintent_spcomp_exec:nn</code>	8477	<code>\l__spintent_spcomp_luaset_compound_-</code>		<code>str</code>	8405	<code>__spintent_spcomp_mathml:n</code>	8483	<code>\l__spintent_spcomp_read_arg_tl</code>	8405	<code>__spintent_spcomp_set_paren:n</code>	8442	<code>__spintent_spcomp_typeset:n</code>	8471	<code>\l__spintent_spcomp_wrap_parens_str ...</code>	8405	<code>__spintent_spcoord_exec:nn</code>	3160	<code>__spintent_spcoord_inner_sep:n</code>	3123	<code>__spintent_spcoord_intent_infix:</code>	3129	<code>\l__spintent_spcoord_intent_str</code>	3016	<code>\l__spintent_spcoord_luaset_error_str .</code>	3006	<code>\l__spintent_spcoord_luaset_sep_str ...</code>	3006	<code>\l__spintent_spcoord_luaset_x_composite_-</code>		<code>str</code>	3006	<code>\l__spintent_spcoord_luaset_x_has_frac_-</code>		<code>str</code>	3006	<code>\l__spintent_spcoord_luaset_x_sign_str</code>	3006	<code>\l__spintent_spcoord_luaset_x_type_str</code>	3006	<code>\l__spintent_spcoord_luaset_x_value_-</code>		<code>str</code>	3006	<code>\l__spintent_spcoord_luaset_y_composite_-</code>		<code>str</code>	3006	<code>\l__spintent_spcoord_luaset_y_has_frac_-</code>		<code>str</code>	3006	<code>\l__spintent_spcoord_luaset_y_sign_str</code>	3006	<code>\l__spintent_spcoord_luaset_y_type_str</code>	3006	<code>\l__spintent_spcoord_luaset_y_value_-</code>		<code>str</code>	3006	<code>__spintent_spcoord_mathml:N</code>	3143	<code>__spintent_spcoord_print:n</code>	3100	<code>\l__spintent_spcoord_raw_arg_tl</code>	3016	<code>__spintent_spcoord_set_wrap_parens_-</code>		<code>size:</code>	3038	<code>__spintent_spcoord_value:n</code>	3091	<code>\l__spintent_spcoord_wrap_parens_l_tl .</code>	3016	<code>\l__spintent_spcoord_wrap_parens_r_tl .</code>	3016	<code>\l__spintent_spcuartil_read_arg_tl ...</code>	7855	<code>__spintent_spdate_exec:n</code>	1528	<code>\l__spintent_spdate_luaset_error_str ..</code>	1518	<code>\l__spintent_spdate_luaset_output_str .</code>	1518	<code>__spintent_spdate_mathml:</code>	1538	<code>\l__spintent_spdecil_read_arg_tl</code>	7855	<code>\l__spintent_spdesv_read_arg_tl</code>	7632	<code>\l__spintent_spdesv_sub_index_str</code>	7632	<code>\l__spintent_spdiam_mode_str</code>	6481	<code>\l__spintent_spdiam_read_arg_tl</code>	6481	<code>\l__spintent_spdiam_sub_index_str</code>	6481	<code>__spintent_spdiv_exec:nnn</code>	2005	<code>__spintent_spdiv_mathml:nn</code>	1989	<code>__spintent_spdiv_render_op:nn</code>	1980	<code>\l__spintent_spdom_cmd_str</code>	4047	<code>__spintent_spdom_exec:nnnnn</code>	4113	<code>\l__spintent_spdom_intent_arg_tl</code>	4047	<code>\l__spintent_spdom_luaset_error_str ...</code>	4047	<code>__spintent_spdom_mathml:</code>	4084	<code>__spintent_spdom_name:</code>	4075	<code>\l__spintent_spdom_op_tl</code>	4047	<code>\l__spintent_spdom_prefix_tl</code>	4047	<code>\l__spintent_spdom_word_tl</code>	4047	<code>__spintent_spdsym_mathml:</code>	1846	<code>\l__spintent_spdsym_prefix_tl</code>	1795	<code>\l__spintent_spdsym_read_sym_tl</code>	1795	<code>\l__spintent_spdsym_suffix_tl</code>	1795	<code>\l__spintent_spdsym_symbol_tl</code>	1795	<code>__spintent_spfact_exec:nn</code>	8975	<code>__spintent_spfact_mathml:n</code>	8980	<code>\l__spintent_spfact_read_arg_tl</code>	8952	<code>__spintent_spfig_build_concept_-</code>		<code>vertices:n</code>	6319	<code>__spintent_spfig_build_refs:</code>	6304	<code>__spintent_spfig_build_simple:n</code>	6169	<code>\l__spintent_spfig_desc_tl</code>	6133	<code>__spintent_spfig_exec:nnn</code>	6143	<code>\l__spintent_spfig_intent_arg_tl</code>	6133
---	----------------------	--	----------------------	---	----------------------	--	--	--	--	------------------------	----------------------	--	----------------------	---	----------------------	---	----------------------	--	----------------------	---	--	------------------------------	----------------------	--	----------------------	--	----------------------	--	----------------------	--	----------------------	---	----------------------	---	----------------------	---	----------------------	--	----------------------	--	----------------------	--	----------------------	--	----------------------	--	----------------------	---	----------------------	---	----------------------	--	----------------------	---	----------------------	---	----------------------	---	----------------------	--	----------------------	--	----------------------	--	----------------------	---	----------------------	---	----------------------	---	----------------------	--	----------------------	--	----------------------	--	----------------------	--	----------------------	--	----------------------	--	----------------------	---	----------------------	---	----------------------	---	----------------------	---	----------------------	---	----------------------	---	----------------------	--	--	------------------------	----------------------	--	----------------------	--	----------------------	---	----------------------	---	----------------------	--	----------------------	--	----------------------	--	----------------------	---	----------------------	--	----------------------	--	----------------------	--	----------------------	--	--	------------------------	----------------------	---	--	------------------------	----------------------	---	----------------------	---	----------------------	--	--	------------------------	----------------------	--	--	------------------------	----------------------	---	--	------------------------	----------------------	---	----------------------	---	----------------------	--	--	------------------------	----------------------	---	----------------------	--	----------------------	--	----------------------	--	--	--------------------------	----------------------	--	----------------------	--	----------------------	--	----------------------	---	----------------------	--	----------------------	--	----------------------	--	----------------------	---	----------------------	---	----------------------	--	----------------------	---	----------------------	---	----------------------	--	----------------------	---	----------------------	---	----------------------	--	----------------------	---	----------------------	---	----------------------	---	----------------------	---	----------------------	--	----------------------	--	----------------------	--	----------------------	---	----------------------	---	----------------------	---	----------------------	---	----------------------	--	----------------------	--	----------------------	--	----------------------	--	----------------------	---	----------------------	--	----------------------	--	----------------------	--	--	-------------------------------	----------------------	--	----------------------	---	----------------------	---	----------------------	---	----------------------	---	----------------------

<code>\l__spintent_spfig_intent_tl</code>	6133	<code>__spintent_spinterval_inner_sep:n</code> ...	3471
<code>__spintent_spfig_mathml:n</code>	6410	<code>\l__spintent_spinterval_intent_arg_tl</code> .	3208
<code>\l__spintent_spfig_op_tl</code>	6133	<code>\l__spintent_spinterval_intent_group_-</code> <code>tl</code>	3208
<code>__spintent_spfig_points_loop:n</code>	6398	<code>\l__spintent_spinterval_luaset_a_braced_-</code> <code>str</code>	3196
<code>\l__spintent_spfig_pt_idx_int</code>	6133	<code>\l__spintent_spinterval_luaset_a_-</code> <code>composite_str</code>	3196
<code>\l__spintent_spfig_pt_seq</code>	6133	<code>\l__spintent_spinterval_luaset_a_inf_-</code> <code>str</code>	3196
<code>\l__spintent_spfig_refs_tl</code>	6133	<code>\l__spintent_spinterval_luaset_a_integer_-</code> <code>str</code>	3196
<code>__spintent_spfig_sym:</code>	6125	<code>\l__spintent_spinterval_luaset_a_short_-</code> <code>str</code>	3196
<code>\l__spintent_spfig_symbol_tl</code>	6133	<code>\l__spintent_spinterval_luaset_a_sign_-</code> <code>str</code>	3196
<code>\l__spintent_spFrec_concept_tl</code>	7985	<code>\l__spintent_spinterval_luaset_a_tall_-</code> <code>str</code>	3196
<code>__spintent_spFrec_exec:n</code>	8024	<code>\l__spintent_spinterval_luaset_a_type_-</code> <code>str</code>	3196
<code>\l__spintent_spFrec_read_arg_tl</code>	7983	<code>\l__spintent_spinterval_luaset_a_value_-</code> <code>str</code>	3196
<code>\l__spintent_spFrec_sub_index_str</code>	7983	<code>\l__spintent_spinterval_luaset_b_braced_-</code> <code>str</code>	3196
<code>\l__spintent_spFrec_symbol_str</code>	7985	<code>\l__spintent_spinterval_luaset_b_-</code> <code>composite_str</code>	3196
<code>__spintent_spfun_aname:n</code>	3656	<code>\l__spintent_spinterval_luaset_b_inf_-</code> <code>str</code>	3196
<code>__spintent_spfun_arg:n</code>	3718	<code>\l__spintent_spinterval_luaset_b_integer_-</code> <code>str</code>	3196
<code>__spintent_spfun_args:</code>	3740	<code>\l__spintent_spinterval_luaset_b_short_-</code> <code>str</code>	3196
<code>__spintent_spfun_args_plain:</code>	3749	<code>\l__spintent_spinterval_luaset_b_sign_-</code> <code>str</code>	3196
<code>__spintent_spfun_body:</code>	3790	<code>\l__spintent_spinterval_luaset_b_tall_-</code> <code>str</code>	3196
<code>\l__spintent_spfun_call_tl</code>	3603	<code>\l__spintent_spinterval_luaset_b_type_-</code> <code>str</code>	3196
<code>\l__spintent_spfun_concept_bool</code>	3603	<code>\l__spintent_spinterval_luaset_b_value_-</code> <code>str</code>	3196
<code>\l__spintent_spfun_depth_int</code>	3603	<code>\l__spintent_spinterval_luaset_error_-</code> <code>code_str</code>	3189
<code>__spintent_spfun_exec:nn</code>	3834	<code>\l__spintent_spinterval_luaset_error_-</code> <code>str</code>	3189
<code>__spintent_spfun_fname:</code>	3656	<code>\l__spintent_spinterval_luaset_left_sym_-</code> <code>str</code>	3189
<code>\l__spintent_spfun_intent_arg_tl</code>	3603	<code>\l__spintent_spinterval_luaset_right_sym_-</code> <code>str</code>	3189
<code>\l__spintent_spfun_kv_tl</code>	3603	<code>\l__spintent_spinterval_luaset_sep_str</code>	3189
<code>\l__spintent_spfun_luaset_error_str</code> ...	3603	<code>\l__spintent_spinterval_luaset_tall_-</code> <code>str</code>	3189
<code>\l__spintent_spfun_luaset_inv_str</code>	3603	<code>\l__spintent_spinterval_luaset_type_-</code> <code>str</code>	3189
<code>\l__spintent_spfun_luaset_nargs_str</code> ...	3603	<code>__spintent_spinterval_mathml:</code>	3549
<code>__spintent_spfun_mathml:</code>	3785	<code>\l__spintent_spinterval_mode_str</code>	3208
<code>\l__spintent_spfun_mode_str</code>	3603	<code>__spintent_spinterval_plain_number:n</code> .	3477
<code>__spintent_spfun_name_mathcat:</code>	3707	<code>\l__spintent_spinterval_prefix_tl</code>	3208
<code>__spintent_spfun_name_plain:</code>	3669	<code>__spintent_spinterval_print:n</code>	3511
<code>__spintent_spfun_name_school:</code>	3681	<code>\l__spintent_spinterval_read_arg_tl</code> ...	3208
<code>\l__spintent_spfun_nargs_int</code>	3603	<code>\l__spintent_spinterval_reason_str</code> ...	3208
<code>__spintent_spfun_nested:n</code>	3824	<code>__spintent_spinterval_set_delims:</code> ...	3327
<code>\l__spintent_spfun_prefix_tl</code>	3603	<code>__spintent_spinterval_set_intents:</code> ...	3421
<code>\l__spintent_spfun_refs_tl</code>	3603	<code>__spintent_spinterval_set_reason:</code> ...	3285
<code>__spintent_spfun_set_intent:</code>	3757	<code>\l__spintent_spinterval_size_l_tl</code>	3208
<code>\l__spintent_spfun_silent_bool</code>	3603		
<code>\l__spintent_spfun_tmp_str</code>	3603		
<code>__spintent_spfuncomp_exec:nn</code>	4022		
<code>__spintent_spfuncomp_fun:n</code>	3886		
<code>__spintent_spfuncomp_group:</code>	3922		
<code>\l__spintent_spfuncomp_intent_tl</code>	3859		
<code>\l__spintent_spfuncomp_kv_tl</code>	3859		
<code>\l__spintent_spfuncomp_luaset_apply_-</code> <code>str</code>	3859		
<code>\l__spintent_spfuncomp_luaset_error_-</code> <code>str</code>	3859		
<code>\l__spintent_spfuncomp_luaset_n_str</code> ...	3859		
<code>\l__spintent_spfuncomp_luaset_paren_-</code> <code>str</code>	3859		
<code>__spintent_spfuncomp_mathml:</code>	3960		
<code>\l__spintent_spfuncomp_n_int</code>	3859		
<code>__spintent_spfuncomp_plain:</code>	3946		
<code>\l__spintent_spfuncomp_refs_tl</code>	3859		
<code>\l__spintent_spfuncomp_tipo_str</code>	3859		
<code>\l__spintent_spinterval_delim_l_tl</code> ...	3208		
<code>\l__spintent_spinterval_delim_r_tl</code> ...	3208		
<code>__spintent_spinterval_exec:nn</code>	3575		

<code>\l__spintent_spminterval_size_r_tl</code>	3208	<code>__spintent_spmoney_print_sep_con:</code>	1081
<code>\l__spintent_spminterval_size_str</code>	3208	<code>__spintent_spmoney_print_symbol:</code>	924
<code>\l__spintent_spminterval_sym_l_tl</code>	3208	<code>__spintent_spmoney_render_layout:</code>	941
<code>\l__spintent_spminterval_sym_r_tl</code>	3208	<code>__spintent_spmoney_render_number_node:</code>	953
<code>\l__spintent_spminterval_tmp_tl</code>	3208	<code>__spintent_spmoney_render_only_integer_-</code>	
<code>__spintent_spminterval_value:n</code>	3486	<code>node:</code>	953
<code>\l__spintent_spminterval_word_g_tl</code>	3208	<code>\l__spintent_spmoney_res_main_voice_str</code>	850
<code>\l__spintent_spminterval_word_w_tl</code>	3208	<code>__spintent_spmoney_set_and_print_sub_-</code>	
<code>\l__spintent_spmangle_mode_str</code>	5404	<code>units:</code>	1032
<code>\l__spintent_spmangle_prefix_tl</code>	5404	<code>__spintent_spmoney_set_currency:n</code>	890
<code>\l__spintent_spmarc_mode_str</code>	5704	<code>\l__spintent_spmoney_sub_units_bool</code>	850
<code>\l__spintent_spmarc_prefix_tl</code>	5704	<code>\l__spintent_spmoney_subnum_int</code>	850
<code>\l__spintent_spmmax_read_arg_tl</code>	7632	<code>__spintent_spmoney_validate_money:</code>	905
<code>\l__spintent_spmmax_sub_index_str</code>	7632	<code>__spintent_spmQ_check_part:nn</code>	2931
<code>__spintent_spmc_exec:nnnn</code>	2219	<code>\l__spintent_spmQ_error_bool</code>	2737
<code>__spintent_spmc_mathml:nn</code>	2241	<code>__spintent_spmQ_exec:nnnn</code>	2959
<code>__spintent_spmc_result_mathml:n</code>	2287	<code>\l__spintent_spmQ_intent_read_sign_str</code>	2737
<code>\l__spintent_spmcd_prefix_tl</code>	2103	<code>\l__spintent_spmQ_join_word_str</code>	2737
<code>\l__spintent_spmcd_print_result_bool</code>	2103	<code>__spintent_spmQ_mathml:nnn</code>	2969
<code>\l__spintent_spmcd_print_upper_bool</code>	2103	<code>__spintent_spmQ_print:nnn</code>	2923
<code>\l__spintent_spmcd_read_terse_bool</code>	2103	<code>__spintent_spmQ_print_scale_int:</code>	2867
<code>\l__spintent_spmcd_suffix_tl</code>	2103	<code>__spintent_spmQ_print_scale_int:n</code>	2876
<code>\l__spintent_spmcm_prefix_tl</code>	2103	<code>__spintent_spmQ_print_spnZ_dfrac:nn</code>	2905
<code>\l__spintent_spmcm_print_result_bool</code>	2103	<code>__spintent_spmQ_set_join_word:</code>	2858
<code>\l__spintent_spmcm_print_upper_bool</code>	2103	<code>__spintent_spmQ_set_wrap_parens_size:</code>	2803
<code>\l__spintent_spmcm_read_terse_bool</code>	2103	<code>__spintent_spmQ_start_wrap_parens:</code>	2838
<code>\l__spintent_spmcm_suffix_tl</code>	2103	<code>__spintent_spmQ_stop_wrap_parens:</code>	2838
<code>\l__spintent_spmmedia_read_arg_tl</code>	7632	<code>\l__spintent_spmQ_wrap_parens_l_tl</code>	2737
<code>\l__spintent_spmmedia_sub_index_str</code>	7632	<code>\l__spintent_spmQ_wrap_parens_r_tl</code>	2737
<code>\l__spintent_spmmediana_read_arg_tl</code>	7632	<code>\l__spintent_spm_side_mode_str</code>	5188
<code>\l__spintent_spmmediana_sub_index_str</code>	7632	<code>\l__spintent_spm_side_prefix_tl</code>	5188
<code>\l__spintent_spmmin_read_arg_tl</code>	7632	<code>\l__spintent_spm_side_read_arg_tl</code>	5188
<code>\l__spintent_spmmin_sub_index_str</code>	7632	<code>\l__spintent_spm_side_read_sty_str</code>	5188
<code>\l__spintent_spmmoda_read_arg_tl</code>	7632	<code>__spintent_spm_sym_mathml:</code>	1923
<code>\l__spintent_spmmoda_sub_index_str</code>	7632	<code>\l__spintent_spm_sym_prefix_tl</code>	1870
<code>__spintent_spmoney_build_tree:</code>	1081	<code>\l__spintent_spm_sym_read_sym_tl</code>	1870
<code>__spintent_spmoney_check_sub_units:</code>	1032	<code>\l__spintent_spm_sym_suffix_tl</code>	1870
<code>\l__spintent_spmoney_extracted_curr_str</code>	850	<code>\l__spintent_spm_sym_symbol_tl</code>	1870
<code>\l__spintent_spmoney_extracted_num_tl</code>	850	<code>__spintent_spmul_exec:nnn</code>	2080
<code>__spintent_spmoney_intent_int:</code>	991	<code>__spintent_spmul_mathml:nn</code>	2064
<code>\l__spintent_spmoney_intent_money_str</code>	850	<code>__spintent_spmul_render_op:nn</code>	2055
<code>\l__spintent_spmoney_intent_sign_str</code>	850	<code>\l__spintent_spnD_luaset_is_truncated_-</code>	
<code>\l__spintent_spmoney_intent_sub_unit_-</code>		<code>str</code>	2329
<code>str</code>	850	<code>\l__spintent_spnD_luaset_result_tl</code>	2329
<code>\l__spintent_spmoney_luaset_conde_str</code>	840	<code>\l__spintent_spnD_prefix_tl</code>	2329
<code>\l__spintent_spmoney_luaset_currency_arg_-</code>		<code>\l__spintent_spnD_print_result_num_tl</code>	2329
<code>str</code>	840	<code>\l__spintent_spnM_luaset_is_truncated_-</code>	
<code>\l__spintent_spmoney_luaset_plur_str</code>	840	<code>str</code>	2329
<code>\l__spintent_spmoney_luaset_position_-</code>		<code>\l__spintent_spnM_luaset_result_tl</code>	2329
<code>str</code>	840	<code>\l__spintent_spnM_prefix_tl</code>	2329
<code>\l__spintent_spmoney_luaset_print_iso_-</code>		<code>\l__spintent_spnM_print_result_num_tl</code>	2329
<code>str</code>	840	<code>__spintent_spnM_spnD_exec:nnnnn</code>	2434
<code>\l__spintent_spmoney_luaset_sing_str</code>	840	<code>__spintent_spnM_spnD_intent:nn</code>	2379
<code>\l__spintent_spmoney_luaset_status_str</code>	840	<code>\l__spintent_spnM_spnD_luaset_error_-</code>	
<code>\l__spintent_spmoney_luaset_subplur_str</code>	840	<code>str</code>	2329
<code>\l__spintent_spmoney_luaset_subsing_str</code>	840	<code>__spintent_spnM_spnD_mathml:nnnn</code>	2445
<code>\l__spintent_spmoney_luaset_symbol_str</code>	840	<code>__spintent_spnM_spnD_result:nn</code>	2396
<code>__spintent_spmoney_mathml:</code>	962	<code>__spintent_spnM_spnD_silent:nn</code>	2388
<code>__spintent_spmoney_normalize_key:n</code>	890	<code>\l__spintent_spnmd_luaset_is_number_-</code>	
<code>__spintent_spmoney_parse_and_route:n</code>	1122	<code>str</code>	2329
<code>__spintent_spmoney_print_dec_sub_-</code>			
<code>unit:</code>	1081		

<code>\l__spintent_spnun_intent_read_dec_sep_-str</code>	289	<code>__spintent_spProb_mid:</code>	8271
<code>\l__spintent_spnun_intent_read_sign_str</code>	289	<code>\l__spintent_spProb_nested_bool</code>	8270
<code>__spintent_spnun_mathml:</code>	471	<code>__spintent_spProb_nested_ops:</code>	8271
<code>\l__spintent_spnun_notation_sym_tl</code> ...	289	<code>\l__spintent_spProb_read_arg_tl</code>	8245
<code>__spintent_spnun_print_integer:</code>	471	<code>__spintent_spProm_exec:nn</code>	7592
<code>__spintent_spnun_print_num_start:</code> ...	471	<code>\l__spintent_spProm_intent_tl</code>	7575
<code>__spintent_spnun_print_part_dec:</code>	450	<code>__spintent_spProm_mathml:n</code>	7597
<code>__spintent_spnun_print_part_period:</code> ..	450	<code>\l__spintent_spProm_read_arg_tl</code>	7575
<code>__spintent_spnun_print_period_bar:</code> ...	378	<code>__spintent_spPval_exec:nnn</code>	8201
<code>\l__spintent_spnun_read_period_sep_str</code>	289	<code>__spintent_spPval_mathml:nn</code>	8206
<code>\l__spintent_spnun_read_terse_bool</code> ...	289	<code>\l__spintent_sprad_mode_str</code>	6481
<code>__spintent_spnun_render_dec:</code>	361	<code>\l__spintent_sprad_read_arg_tl</code>	6481
<code>__spintent_spnun_render_decimal_sep:</code> .	415	<code>\l__spintent_sprad_sub_index_str</code>	6481
<code>__spintent_spnun_render_int:</code>	361	<code>\l__spintent_sprango_read_arg_tl</code>	7632
<code>__spintent_spnun_render_only_period:</code> .	378	<code>\l__spintent_sprango_sub_index_str</code> ...	7632
<code>__spintent_spnun_render_part:n</code>	361	<code>__spintent_sprayo_exec:nn</code>	5110
<code>__spintent_spnun_render_period_sep:</code> ..	378	<code>\l__spintent_sprayo_intent_arg_tl</code>	5106
<code>__spintent_spnz_exec:nn</code>	2719	<code>__spintent_sprayo_mathml:</code>	5124
<code>__spintent_spnz_mathml:</code>	2703	<code>\l__spintent_sprayo_mode_str</code>	5052
<code>\l__spintent_sp0mega_read_arg_tl</code>	8141	<code>\l__spintent_sprayo_prefix_tl</code>	5052
<code>\l__spintent_sp0mega_read_sub_bool</code> ...	8141	<code>\l__spintent_sprayo_pt_idx_int</code>	5106
<code>\l__spintent_sp0mega_sub_index_str</code> ...	8141	<code>\l__spintent_sprayo_pt_seq</code>	5106
<code>\l__spintent_spPcirc_mode_str</code>	7035	<code>\l__spintent_sprayo_read_sty_str</code>	5052
<code>\l__spintent_spPcirc_prefix_tl</code>	7035	<code>\l__spintent_sprayo_refs_tl</code>	5106
<code>\l__spintent_spPcirc_print_sym_str</code> ...	7035	<code>\l__spintent_sprazon_cmd_str</code>	2530
<code>\l__spintent_spPcirc_read_arg_tl</code>	7035	<code>__spintent_sprazon_exec:nnn</code>	2549
<code>\l__spintent_spPcirc_read_sty_str</code>	7035	<code>__spintent_sprazon_mathml:nn</code>	2554
<code>\l__spintent_spPcirc_sub_index_str</code> ...	7035	<code>__spintent_spread_exec:nn</code>	1732
<code>\l__spintent_spPerc_read_arg_tl</code>	7855	<code>\l__spintent_spread_luaset_reading_str</code>	1711
<code>\l__spintent_spPfig_mode_str</code>	6016	<code>\l__spintent_spread_luaset_status_str</code> .	1711
<code>\l__spintent_spPfig_prefix_tl</code>	6016	<code>__spintent_spread_mathml:nn</code>	1758
<code>\l__spintent_spPfig_print_sym_str</code>	6016	<code>\l__spintent_spread_read_arg_tl</code>	1711
<code>\l__spintent_spPfig_read_arg_tl</code>	6016	<code>\l__spintent_spread_read_tmp_tl</code>	1711
<code>__spintent_spPoint_exec:nnn</code>	4478	<code>__spintent_sprecta_exec:nn</code>	4837
<code>\l__spintent_spPoint_intent_arg_tl</code> ...	4436	<code>\l__spintent_sprecta_intent_arg_tl</code> ...	4792
<code>__spintent_spPoint_mathml:n</code>	4523	<code>\l__spintent_sprecta_kv_tl</code>	4792
<code>\l__spintent_spPoint_mode_str</code>	4436	<code>__spintent_sprecta_mathml:n</code>	4847
<code>\l__spintent_spPoint_prefix_tl</code>	4436	<code>\l__spintent_sprecta_mode_str</code>	4792
<code>__spintent_spPoly_build_concept:</code>	5937	<code>\l__spintent_sprecta_prefix_tl</code>	4792
<code>__spintent_spPoly_build_refs:</code>	5922	<code>\l__spintent_sprecta_pt_idx_int</code>	4792
<code>__spintent_spPoly_exec:nn</code>	5973	<code>\l__spintent_sprecta_pt_seq</code>	4792
<code>\l__spintent_spPoly_intent_arg_tl</code>	5910	<code>\l__spintent_sprecta_refs_tl</code>	4792
<code>__spintent_spPoly_mathml:</code>	5984	<code>\l__spintent_spriq_read_arg_tl</code>	7632
<code>\l__spintent_spPoly_mode_str</code>	5867	<code>\l__spintent_spriq_sub_index_str</code>	7632
<code>__spintent_spPoly_points_loop:</code>	5961	<code>__spintent_spsshort_exec:n</code>	1201
<code>\l__spintent_spPoly_prefix_tl</code>	5867	<code>__spintent_spsshort_execute_layout:</code> ...	1236
<code>\l__spintent_spPoly_pt_idx_int</code>	5910	<code>\l__spintent_spsshort_luaset_base_str</code> ..	1149
<code>\l__spintent_spPoly_pt_seq</code>	5910	<code>\l__spintent_spsshort_luaset_expand_str</code>	1149
<code>\l__spintent_spPoly_refs_tl</code>	5910	<code>\l__spintent_spsshort_luaset_layout_str</code>	1149
<code>__spintent_spPoly_sym:</code>	5914	<code>\l__spintent_spsshort_luaset_output_str</code>	1149
<code>__spintent_spProb_cap:</code>	8271	<code>\l__spintent_spsshort_luaset_status_str</code>	1149
<code>__spintent_spProb_content:</code>	8303	<code>\l__spintent_spsshort_luaset_suffix_str</code>	1149
<code>__spintent_spProb_cup:</code>	8271	<code>__spintent_spsshort_print:nn</code>	1189
<code>__spintent_spProb_event:N</code>	8271	<code>__spintent_spsshort_render_bypass:n</code> ...	1212
<code>__spintent_spProb_exec:nn</code>	8386	<code>__spintent_spsshort_render_error:n</code> ...	1212
<code>\l__spintent_spProb_luaset_a_str</code>	8241	<code>__spintent_spsshort_render_fallback:</code> ..	1212
<code>\l__spintent_spProb_luaset_b_str</code>	8241	<code>__spintent_spsshort_render_success:</code> ...	1212
<code>\l__spintent_spProb_luaset_error_str</code> ..	8241	<code>__spintent_spsshort_tag_span:nn</code>	1176
<code>\l__spintent_spProb_luaset_op_str</code>	8241	<code>__spintent_spside_exec:nnn</code>	5267
<code>__spintent_spProb_mathml:</code>	8313	<code>\l__spintent_spside_intent_arg_tl</code>	5262
		<code>\l__spintent_spside_intent_tl</code>	5262

__spintent_spside_mathml:n	5302	\l__spintent_spvec_prefix_tl	4543
\l__spintent_spside_mode_str	5188	\l__spintent_spvec_pt_idx_int	4543
__spintent_spside_points_loop:n	5370	\l__spintent_spvec_pt_seq	4543
\l__spintent_spside_prefix_tl	5188	\l__spintent_spvec_refs_tl	4543
\l__spintent_spside_pt_idx_int	5262	__spintent_spVol_attach_subscript: ...	7501
\l__spintent_spside_pt_seq	5262	__spintent_spVol_mathml:	7414
\l__spintent_spside_read_arg_tl	5188	\l__spintent_spVol_mode_str	7369
\l__spintent_spside_read_sty_str	5188	\l__spintent_spVol_native_bool	7369
\l__spintent_spside_refs_tl	5262	\l__spintent_spVol_read_arg_full_tl ...	7369
__spintent_spsiglo_exec:n	1376	__spintent_spVol_read_arg_print: ...	7533
\l__spintent_spsiglo_luaset_arabic_str	1293	\l__spintent_spVol_read_arg_tl	7369
\l__spintent_spsiglo_luaset_error_str .	1293	__spintent_spVol_silent:	7421
\l__spintent_spsiglo_luaset_roman_min_-		\l__spintent_spVol_sub_index_str	7369
str	1293	\l__spintent_spVol_suffix_tl	7369
__spintent_spsiglo_mathml:	1376	__spintent_spVol_symbol_print:	7463
\l__spintent_spsiglo_print_era_str ...	1293	__spintent_spVol_verbal:	7442
__spintent_spsiglo_render_simple:nn ..	1327	\l__spintent_spVol_word_tl	7369
__spintent_spsiglo_render_with_-		__spintent_stat_exec:nnn	7764
era:nnn	1327	__spintent_stat_letter:nnnn	7764
__spintent_spsiglo_tag_span:nn	1363	__spintent_stat_mathml:nnnnnn	7711
__spintent_spsiglo_text:	1376	__spintent_stat_pos_check:nnn	7884
__spintent_sptime_build_seconds:nn ...	1450	\l__spintent_stat_pos_error_bool	7855
\l__spintent_sptime_luaset_base_str ...	1437	__spintent_stat_pos_exec:nnnnn	7938
\l__spintent_sptime_luaset_error_str ..	1437	__spintent_stat_pos_mathml:nnnn	7908
\l__spintent_sptime_luaset_has_seconds_-		__spintent_stat_pos_typeset:nn	7903
str	1437	\l__spintent_stat_pos_value_int	7855
\l__spintent_sptime_luaset_is_fraction_-		__spintent_stat_simple:nnn	7764
str	1437	__spintent_stat_typeset:nnn	7697
\l__spintent_sptime_luaset_seconds_str	1437	__spintent_unknown_keys_cmd:n	241
__spintent_sptime_mathml:	1468	__spintent_unknown_keys_cmd:nn	241
__spintent_spunit_exec:n	687	spintent.lua	76
\l__spintent_spunit_exponent_tl	560	\spinterval	38, 127, 3592
__spintent_spunit_format_canonical: ..	619	spinterval-invalid-arg	3224
__spintent_spunit_load_term:nn	627	\spmangle	28, 169, 5682
\l__spintent_spunit_luaset_above_found_-		spmangle-invalid-arg	5409
str	552	\spmarc	29, 175, 5845
\l__spintent_spunit_luaset_above_n_str	552	spmarc-invalid-points	5709
\l__spintent_spunit_luaset_above_sexa_-		\spmmax	43, 7829
str	552	\spmcd	19, 103, 2309
\l__spintent_spunit_luaset_below_found_-		\spmcm	20, 103, 2309
str	552	\spmedia	42, 7784
\l__spintent_spunit_luaset_below_n_str	552	\spmediana	42, 7784
\l__spintent_spunit_luaset_canonical_-		\spmmin	43, 7829
str	557	\spmmoda	42, 7784
\l__spintent_spunit_luaset_read_str ...	557	\spmoney	10, 77, 1138
\l__spintent_spunit_luaset_status_str .	557	spmoney-invalid-currency	858
__spintent_spunit_mathml:	730	spmoney-no-subunits	858
__spintent_spunit_print_product:n ...	670	\spmQ	21, 116, 2982
__spintent_spunit_print_term:nn	636	spmQ-not-integer	2743
\l__spintent_spunit_read_slash_str ...	560	spmQ-signed-part	2743
\l__spintent_spusep_arg_tl	1570	spmQ-zero-denominator	2743
__spintent_spusep_exec:n	1632	\spmside	26, 165, 5382
\l__spintent_spusep_left_tl	1570	spmside	6743
__spintent_spusep_mathml:nn	1621	spmside-invalid-points	5195
\l__spintent_spusep_right_tl	1570	\spmsym	17, 98, 1942
\l__spintent_spvarz_read_arg_tl	7632	\spmul	18, 101, 2093
\l__spintent_spvarz_sub_index_str ...	7632	spmul-empty-arg	2028
__spintent_spvec_exec:nnn	4591	\spnD	18, 107, 2510
\l__spintent_spvec_intent_arg_tl	4543	\spnewread	15, 1774
\l__spintent_spvec_kv_tl	4543	spnewread-duplicate	1682
__spintent_spvec_mathml:n	4624		
\l__spintent_spvec_mode_str	4543		

spnewread-empty-args	1682
\spnewunit	10, 75, 808
spnewunit-duplicate	786
\spnM	19, 107, 2510
\spnum	7, 65, 534
spnum-has-units	294
spnum-no-digits	294
\spnZ	21, 115, 2727
spnZ-not-integer	2680
\spOmega	45, 223, 8167
\spPcirc	36, 201, 7343
spPcirc-invalid-arg	7044
\spPerc	43, 7947
\spPfig	32, 181, 6452
spPfig-invalid-arg	6023
spPfig-invalid-points	6023
spPfig-read-arg-key	6023
spPfig-sym-cardinality-conflict	6023
spPfig-sym-letter-conflict	6023
\spPoint	23, 151, 4532
spPoint-invalid-point	4439
\spPoly	30, 178, 6004
spPoly-invalid-points	5869
\spProb	46, 226, 8394
spProb-invalid-arg	8246
\spProm	42, 211, 7621
\spProp	22, 113, 2604
\spPval	45, 224, 8231
\sprad	33, 190, 6703
\sprango	43, 7829
\sprayo	25, 162, 5177
sprayo-invalid-points	5055
\sprazon	22, 111, 2594
\spread	16, 95, 1764
spread-empty-sym	1682
spread-unknown-sym	1682
\sprec	40, 143, 4127
\sprecta	24, 157, 5041
sprecta-invalid-points	4799
\spriq	43, 7829
\spshort	12, 83, 1280
spshort-unknown-abbreviation	1155
\spside	26, 165, 5382
spside	6743

spside-invalid-points	5195
\spsiglo	14, 86, 1429
spsiglo-invalid-arg	1297
\sptime	15, 89, 1500
sptime-invalid-arg	1442
\spunit	8, 71, 774
spunit-inline-numeric-denominator	562
spunit-invalid-unit	562
spunit-no-units	562
\spunitalias	10, 75, 822
spunitalias-duplicate	786
spunitalias-invalid-expr	786
\spusep	16, 92, 1671
spusep-empty-arg	1555
spusep-invalid-arg	1555
\spvarz	42, 7799
\spvec	24, 153, 4781
spvec-invalid-arg	4550
\spVol	38, 207, 7563
stat-pos-invalid-index	7859
str commands:	
\str_lowercase:n	70
sub-index	6487, 6743, 7053, 7376, 7641, 8002, 8145
sub-units	872
suffix	1799, 1874

T

Tool provided by spanish-mathml-intent :	
spmml-i-tool	59–61

U

unknown	308, 585, 872, 1162, 1304, 1573, 1715, 1799, 1874, 1960, 2035, 2115, 2352, 2532, 2687, 2765, 3027, 3233, 3625, 3875, 4060, 4171, 4447, 4557, 4806, 5062, 5205, 5420, 5719, 5876, 6063, 6487, 6743, 7053, 7376, 7577, 7641, 7679, 7866, 8002, 8095, 8145, 8187, 8255, 8410, 8591, 8960
unknown-choice	215
unknown-command-setup	165
upper-case	2115, 6487
use-dfrac	2532, 2765, 8187
use-spnZ	2765

W

wrap-parens	1960, 2035, 2687, 2765, 8410
-------------	------------------------------