

# new minivas

in luametateX

## Introduction

Here I will discuss some of the new primitives in Lua $\TeX$  and LuaMeta $\TeX$ , the later being a successor that permits the Con $\TeX$ t folks to experiment with new features. The order is arbitrary. When you compare Lua $\TeX$  with pdf $\TeX$ , there are actually quite some differences. Some primitives that pdf $\TeX$  introduced have been dropped in Lua $\TeX$  because they can be done better in Lua. Others have been promoted to core primitives that no longer have a pdf prefix. Then there are lots of new primitives, some introduce new concepts, some are a side effect of for instance new math font technologies, and then there are those that are handy extensions to the macro language. The LuaMeta $\TeX$  engine drops quite some primitives, like those related to pdf $\TeX$  specific f(r)ont or backend features. It also adds some new primitives, mostly concerning the macro language.

We also discuss the primitives that fit into the macro programming scope that are present in traditional  $\TeX$  and  $\varepsilon\text{-}\TeX$  but there are for sure better of explanations out there already. Primitives that relate to typesetting, like those controlling math, fonts, boxes, attributes, directions, catcodes, Lua (functions) etc are not discussed or discussed in less detail here.

There are for instance primitives to create aliases to low level registers like counters and dimensions, as well as other (semi-numeric) quantities like characters, but normally these are wrapped into high level macros so that definitions can't clash too much. Numbers, dimensions etc can be advanced, multiplied and divided and there is a simple expression mechanism to deal with them. We don't go into these details here: it's mostly an overview of what the engine provides. If you are new to  $\TeX$ , you need to play a while with its mixed bag of typesetting and programming features in order to understand the difference between this macro language and other languages you might be familiar with.

1	<code>\&lt;space&gt;</code> .....	11	26	<code>\aligntab</code> .....	15
2	<code>\-</code> .....	11	27	<code>\allcrampedstyles</code> .....	15
3	<code>\/</code> .....	11	28	<code>\alldisplaystyles</code> .....	15
4	<code>\above</code> .....	11	29	<code>\allmainstyles</code> .....	15
5	<code>\abovedisplayshortskip</code> .....	11	30	<code>\allmathstyles</code> .....	15
6	<code>\abovedisplayskip</code> .....	11	31	<code>\allscriptscriptstyles</code> .....	15
7	<code>\abovewithdelims</code> .....	12	32	<code>\allscriptstyles</code> .....	15
8	<code>\accent</code> .....	12	33	<code>\allsplitstyles</code> .....	15
9	<code>\additionalpageskip</code> .....	12	34	<code>\alltextstyles</code> .....	16
10	<code>\adjdemerits</code> .....	12	35	<code>\alluncrampedstyles</code> .....	16
11	<code>\adjustspacing</code> .....	12	36	<code>\allunsplitstyles</code> .....	16
12	<code>\adjustspacingshrink</code> .....	12	37	<code>\amcode</code> .....	16
13	<code>\adjustspacingstep</code> .....	12	38	<code>\associateunit</code> .....	16
14	<code>\adjustspacingstretch</code> .....	12	39	<code>\atendoffile</code> .....	17
15	<code>\advance</code> .....	12	40	<code>\atendoffiled</code> .....	17
16	<code>\advanceby</code> .....	13	41	<code>\atendofgroup</code> .....	17
17	<code>\afterassigned</code> .....	13	42	<code>\atendofgrouped</code> .....	17
18	<code>\afterassignment</code> .....	13	43	<code>\atop</code> .....	18
19	<code>\aftergroup</code> .....	13	44	<code>\atopwithdelims</code> .....	18
20	<code>\aftergrouped</code> .....	13	45	<code>\attribute</code> .....	18
21	<code>\aliased</code> .....	14	46	<code>\attributedef</code> .....	18
22	<code>\aligncontent</code> .....	14	47	<code>\automaticdiscretionary</code> .....	18
23	<code>\alignmark</code> .....	14	48	<code>\automatichyphenpenalty</code> .....	18
24	<code>\alignmentcellsource</code> .....	15	49	<code>\automigrationmode</code> .....	18
25	<code>\alignmentwrapsource</code> .....	15	50	<code>\autoparagraphmode</code> .....	19

51	<code>\badness</code> .....	19	100	<code>\clearmarks</code> .....	28
52	<code>\baselineskip</code> .....	19	101	<code>\clubpenalties</code> .....	28
53	<code>\batchmode</code> .....	19	102	<code>\clubpenalty</code> .....	28
54	<code>\begincsname</code> .....	19	103	<code>\constant</code> .....	29
55	<code>\begingroup</code> .....	19	104	<code>\constrained</code> .....	29
56	<code>\beginlocalcontrol</code> .....	20	105	<code>\copy</code> .....	29
57	<code>\beginmathgroup</code> .....	20	106	<code>\copymathatomrule</code> .....	29
58	<code>\beginsimplegroup</code> .....	20	107	<code>\copymathparent</code> .....	29
59	<code>\belowdisplayshortskip</code> .....	21	108	<code>\copymathspacing</code> .....	29
60	<code>\belowdisplayskip</code> .....	21	109	<code>\count</code> .....	29
61	<code>\binoppenalty</code> .....	21	110	<code>\countdef</code> .....	29
62	<code>\botmark</code> .....	21	111	<code>\cr</code> .....	30
63	<code>\botmarks</code> .....	21	112	<code>\crampeddisplaystyle</code> .....	30
64	<code>\boundary</code> .....	22	113	<code>\crampedscriptscriptstyle</code> .....	30
65	<code>\box</code> .....	22	114	<code>\crampedscriptstyle</code> .....	30
66	<code>\boxadapt</code> .....	22	115	<code>\crampedtextstyle</code> .....	30
67	<code>\boxanchor</code> .....	22	116	<code>\crrcr</code> .....	30
68	<code>\boxanchors</code> .....	22	117	<code>\csactive</code> .....	30
69	<code>\boxattribute</code> .....	22	118	<code>\csname</code> .....	31
70	<code>\boxdirection</code> .....	23	119	<code>\csstring</code> .....	31
71	<code>\boxfinalize</code> .....	23	120	<code>\currentgrouplevel</code> .....	31
72	<code>\boxfreeze</code> .....	24	121	<code>\currentgrouptype</code> .....	31
73	<code>\boxgeometry</code> .....	24	122	<code>\currentifbranch</code> .....	32
74	<code>\boxlimit</code> .....	24	123	<code>\currentiflevel</code> .....	32
75	<code>\boxlimitate</code> .....	24	124	<code>\currentifttype</code> .....	32
76	<code>\boxlimitmode</code> .....	24	125	<code>\currentloopiterator</code> .....	33
77	<code>\boxmaxdepth</code> .....	24	126	<code>\currentloopnesting</code> .....	33
78	<code>\boxorientation</code> .....	25	127	<code>\currentmarks</code> .....	33
79	<code>\boxrepack</code> .....	25	128	<code>\currentstacksize</code> .....	33
80	<code>\boxshift</code> .....	25	129	<code>\day</code> .....	34
81	<code>\boxshrink</code> .....	25	130	<code>\dbox</code> .....	34
82	<code>\boxsource</code> .....	25	131	<code>\deadcycles</code> .....	35
83	<code>\boxstretch</code> .....	26	132	<code>\def</code> .....	35
84	<code>\boxtarget</code> .....	26	133	<code>\defaultthyphenchar</code> .....	35
85	<code>\boxtotal</code> .....	26	134	<code>\defaultskewchar</code> .....	35
86	<code>\boxvadjust</code> .....	26	135	<code>\defcsname</code> .....	35
87	<code>\boxxmove</code> .....	26	136	<code>\deferred</code> .....	35
88	<code>\boxxoffset</code> .....	26	137	<code>\delcode</code> .....	36
89	<code>\boxymove</code> .....	27	138	<code>\delimiter</code> .....	36
90	<code>\boxyoffset</code> .....	27	139	<code>\delimiterfactor</code> .....	36
91	<code>\brokenpenalty</code> .....	27	140	<code>\delimitershortfall</code> .....	36
92	<code>\catcode</code> .....	27	141	<code>\detokened</code> .....	36
93	<code>\catcodetable</code> .....	27	142	<code>\detokenize</code> .....	37
94	<code>\cdef</code> .....	27	143	<code>\detokenized</code> .....	37
95	<code>\cdefcsname</code> .....	28	144	<code>\dimen</code> .....	37
96	<code>\cfcode</code> .....	28	145	<code>\dimendef</code> .....	37
97	<code>\char</code> .....	28	146	<code>\dimensiondef</code> .....	38
98	<code>\chardef</code> .....	28	147	<code>\dimexpr</code> .....	38
99	<code>\cleaders</code> .....	28	148	<code>\dimexpression</code> .....	38

149	<code>\directlua</code> .....	38	198	<code>\everydisplay</code> .....	46
150	<code>\discretionary</code> .....	38	199	<code>\everyeof</code> .....	46
151	<code>\discretionaryoptions</code> .....	38	200	<code>\everyhbox</code> .....	46
152	<code>\displayindent</code> .....	39	201	<code>\everyjob</code> .....	46
153	<code>\displaylimits</code> .....	39	202	<code>\everymath</code> .....	46
154	<code>\displaystyle</code> .....	39	203	<code>\everymathatom</code> .....	46
155	<code>\displaywidowpenalties</code> .....	39	204	<code>\everypar</code> .....	46
156	<code>\displaywidowpenalty</code> .....	39	205	<code>\everytab</code> .....	46
157	<code>\displaywidth</code> .....	39	206	<code>\everyvbox</code> .....	46
158	<code>\divide</code> .....	39	207	<code>\exceptionpenalty</code> .....	47
159	<code>\divideby</code> .....	40	208	<code>\exhyphenchar</code> .....	47
160	<code>\doubleadjdemerits</code> .....	40	209	<code>\exhyphenpenalty</code> .....	47
161	<code>\doublehyphendemerits</code> .....	40	210	<code>\expand</code> .....	47
162	<code>\dp</code> .....	40	211	<code>\expandactive</code> .....	47
163	<code>\dpack</code> .....	40	212	<code>\expandafter</code> .....	47
164	<code>\dsplit</code> .....	40	213	<code>\expandafterpars</code> .....	48
165	<code>\dump</code> .....	40	214	<code>\expandafterspaces</code> .....	48
166	<code>\edef</code> .....	40	215	<code>\expandcstoken</code> .....	48
167	<code>\edefcsname</code> .....	41	216	<code>\expanded</code> .....	49
168	<code>\edivide</code> .....	41	217	<code>\expandedafter</code> .....	49
169	<code>\edivideby</code> .....	41	218	<code>\expandeddetokenize</code> .....	50
170	<code>\efcode</code> .....	41	219	<code>\expandedendless</code> .....	50
171	<code>\else</code> .....	42	220	<code>\expandedloop</code> .....	50
172	<code>\emergencyextrastretch</code> .....	42	221	<code>\expandedrepeat</code> .....	50
173	<code>\emergencyleftskip</code> .....	42	222	<code>\expandparameter</code> .....	51
174	<code>\emergencyrightskip</code> .....	42	223	<code>\expandtoken</code> .....	51
175	<code>\emergencystretch</code> .....	42	224	<code>\expandtoks</code> .....	52
176	<code>\end</code> .....	42	225	<code>\explicitdiscretionary</code> .....	52
177	<code>\endcsname</code> .....	42	226	<code>\explicithyphenpenalty</code> .....	52
178	<code>\endgroup</code> .....	42	227	<code>\explicititaliccorrection</code> .....	52
179	<code>\endinput</code> .....	42	228	<code>\explicitspace</code> .....	52
180	<code>\endlinechar</code> .....	43	229	<code>\fam</code> .....	53
181	<code>\endlocalcontrol</code> .....	43	230	<code>\fi</code> .....	53
182	<code>\endmathgroup</code> .....	43	231	<code>\finalhyphendemerits</code> .....	53
183	<code>\endsimplegroup</code> .....	43	232	<code>\firstmark</code> .....	53
184	<code>\enforced</code> .....	44	233	<code>\firstmarks</code> .....	53
185	<code>\eofinput</code> .....	44	234	<code>\firstvalidlanguage</code> .....	53
186	<code>\eqno</code> .....	44	235	<code>\float</code> .....	53
187	<code>\errhelp</code> .....	44	236	<code>\floatdef</code> .....	54
188	<code>\errmessage</code> .....	44	237	<code>\floatexpr</code> .....	54
189	<code>\errorcontextlines</code> .....	44	238	<code>\floatingpenalty</code> .....	55
190	<code>\errorstopmode</code> .....	44	239	<code>\flushmarks</code> .....	55
191	<code>\escapechar</code> .....	44	240	<code>\font</code> .....	55
192	<code>\etoks</code> .....	45	241	<code>\fontcharba</code> .....	55
193	<code>\etoksapp</code> .....	45	242	<code>\fontchardp</code> .....	55
194	<code>\etokspre</code> .....	45	243	<code>\fontcharht</code> .....	55
195	<code>\eufactor</code> .....	45	244	<code>\fontcharic</code> .....	56
196	<code>\everybeforepar</code> .....	45	245	<code>\fontcharta</code> .....	56
197	<code>\everycr</code> .....	45	246	<code>\fontcharwd</code> .....	56

247	<code>\fontdimen</code> .....	56	296	<code>\glyptextscale</code> .....	66
248	<code>\fontid</code> .....	56	297	<code>\glyphweight</code> .....	66
249	<code>\fontmathcontrol</code> .....	57	298	<code>\glyphxoffset</code> .....	66
250	<code>\fontname</code> .....	57	299	<code>\glyphxscale</code> .....	66
251	<code>\fontspecdef</code> .....	57	300	<code>\glyphxscaled</code> .....	66
252	<code>\fontspecid</code> .....	58	301	<code>\glyphyoffset</code> .....	66
253	<code>\fontspecifiedname</code> .....	58	302	<code>\glyphyscale</code> .....	66
254	<code>\fontspecifiedsize</code> .....	58	303	<code>\glyphyscaled</code> .....	66
255	<code>\fontspecscale</code> .....	59	304	<code>\gtoksapp</code> .....	66
256	<code>\fontspecslant</code> .....	59	305	<code>\gtokspre</code> .....	67
257	<code>\fontspecweight</code> .....	59	306	<code>\halign</code> .....	67
258	<code>\fontspecxscale</code> .....	59	307	<code>\hangafter</code> .....	67
259	<code>\fontspecyscale</code> .....	59	308	<code>\hangindent</code> .....	67
260	<code>\fonttextcontrol</code> .....	59	309	<code>\hbadness</code> .....	67
261	<code>\forcedleftcorrection</code> .....	59	310	<code>\hbox</code> .....	67
262	<code>\forcedrightcorrection</code> .....	59	311	<code>\hccode</code> .....	67
263	<code>\formatname</code> .....	59	312	<code>\hfil</code> .....	68
264	<code>\frozen</code> .....	60	313	<code>\hfill</code> .....	68
265	<code>\futurecsname</code> .....	60	314	<code>\hfilneg</code> .....	68
266	<code>\futuredef</code> .....	60	315	<code>\hfuzz</code> .....	68
267	<code>\futureexpand</code> .....	60	316	<code>\hjcode</code> .....	68
268	<code>\futureexpandis</code> .....	61	317	<code>\hkern</code> .....	68
269	<code>\futureexpandisap</code> .....	61	318	<code>\hmcode</code> .....	68
270	<code>\futurelet</code> .....	61	319	<code>\holdinginserts</code> .....	69
271	<code>\gdef</code> .....	62	320	<code>\holdingmigrations</code> .....	69
272	<code>\gdefcsname</code> .....	62	321	<code>\hpack</code> .....	69
273	<code>\givenmathstyle</code> .....	62	322	<code>\hpenalty</code> .....	69
274	<code>\gleaders</code> .....	62	323	<code>\hrule</code> .....	69
275	<code>\glet</code> .....	63	324	<code>\hsize</code> .....	70
276	<code>\gletcsname</code> .....	63	325	<code>\hskip</code> .....	70
277	<code>\glettonothing</code> .....	63	326	<code>\hss</code> .....	70
278	<code>\global</code> .....	63	327	<code>\ht</code> .....	71
279	<code>\globaldefs</code> .....	63	328	<code>\hyphenation</code> .....	71
280	<code>\glueexpr</code> .....	63	329	<code>\hyphenationmin</code> .....	71
281	<code>\glueshrink</code> .....	64	330	<code>\hyphenchar</code> .....	71
282	<code>\glueshrinkorder</code> .....	64	331	<code>\hyphenpenalty</code> .....	71
283	<code>\gluespecdef</code> .....	64	332	<code>\if</code> .....	71
284	<code>\gluestretch</code> .....	64	333	<code>\ifabsdim</code> .....	72
285	<code>\gluestretchorder</code> .....	64	334	<code>\ifabsfloat</code> .....	72
286	<code>\gluetomu</code> .....	64	335	<code>\ifabsnum</code> .....	72
287	<code>\glyph</code> .....	64	336	<code>\ifarguments</code> .....	73
288	<code>\glyphdatafield</code> .....	65	337	<code>\ifboolean</code> .....	73
289	<code>\glyphoptions</code> .....	65	338	<code>\ifcase</code> .....	73
290	<code>\glyphscale</code> .....	65	339	<code>\ifcat</code> .....	73
291	<code>\glyphscriptfield</code> .....	65	340	<code>\ifchkdim</code> .....	73
292	<code>\glyphscriptscale</code> .....	65	341	<code>\ifchkdimension</code> .....	74
293	<code>\glyphscriptscriptscale</code> .....	65	342	<code>\ifchknum</code> .....	74
294	<code>\glyphslant</code> .....	65	343	<code>\ifchknumber</code> .....	74
295	<code>\glyphstatefield</code> .....	66	344	<code>\ifcmpdim</code> .....	74

345	<code>\ifcmpnum</code> .....	75	394	<code>\ignorenestedupto</code> .....	86
346	<code>\ifcondition</code> .....	75	395	<code>\ignorepars</code> .....	87
347	<code>\ifcramped</code> .....	76	396	<code>\ignorereset</code> .....	87
348	<code>\ifcsname</code> .....	76	397	<code>\ignorespaces</code> .....	87
349	<code>\ifcstok</code> .....	76	398	<code>\ignoreupto</code> .....	88
350	<code>\ifdefined</code> .....	77	399	<code>\immediate</code> .....	88
351	<code>\ifdim</code> .....	77	400	<code>\immutable</code> .....	88
352	<code>\ifdimexpression</code> .....	77	401	<code>\indent</code> .....	88
353	<code>\ifdimval</code> .....	77	402	<code>\indexofcharacter</code> .....	88
354	<code>\ifempty</code> .....	78	403	<code>\indexofregister</code> .....	89
355	<code>\iffalse</code> .....	78	404	<code>\inherited</code> .....	89
356	<code>\ifflags</code> .....	78	405	<code>\initcatcodetable</code> .....	89
357	<code>\iffloat</code> .....	79	406	<code>\initialpageskip</code> .....	89
358	<code>\iffontchar</code> .....	79	407	<code>\initialtopskip</code> .....	89
359	<code>\ifhaschar</code> .....	79	408	<code>\input</code> .....	89
360	<code>\ifhastok</code> .....	79	409	<code>\inputlineno</code> .....	89
361	<code>\ifhastoks</code> .....	79	410	<code>\insert</code> .....	89
362	<code>\ifhasxtoks</code> .....	79	411	<code>\insertbox</code> .....	90
363	<code>\ifhbox</code> .....	80	412	<code>\insertcopy</code> .....	90
364	<code>\ifhmode</code> .....	80	413	<code>\insertdepth</code> .....	90
365	<code>\iffinalignment</code> .....	80	414	<code>\insertdistance</code> .....	90
366	<code>\ifincsname</code> .....	81	415	<code>\insertheight</code> .....	90
367	<code>\ifinner</code> .....	81	416	<code>\insertheights</code> .....	90
368	<code>\ifinsert</code> .....	81	417	<code>\insertlimit</code> .....	90
369	<code>\ifintervaldim</code> .....	81	418	<code>\insertmaxdepth</code> .....	90
370	<code>\ifintervalfloat</code> .....	81	419	<code>\insertmode</code> .....	90
371	<code>\ifintervalnum</code> .....	81	420	<code>\insertmultiplier</code> .....	91
372	<code>\iflastnamedcs</code> .....	82	421	<code>\insertpenalties</code> .....	91
373	<code>\ifmathparameter</code> .....	82	422	<code>\insertpenalty</code> .....	91
374	<code>\ifmathstyle</code> .....	82	423	<code>\insertprogress</code> .....	91
375	<code>\ifmmode</code> .....	82	424	<code>\insertstorage</code> .....	91
376	<code>\ifnum</code> .....	82	425	<code>\insertstoring</code> .....	91
377	<code>\ifnumexpression</code> .....	83	426	<code>\insertunbox</code> .....	91
378	<code>\ifnumval</code> .....	83	427	<code>\insertuncopy</code> .....	91
379	<code>\ifodd</code> .....	83	428	<code>\insertwidth</code> .....	91
380	<code>\ifparameter</code> .....	84	429	<code>\instance</code> .....	91
381	<code>\ifparameters</code> .....	84	430	<code>\integerdef</code> .....	92
382	<code>\ifrelax</code> .....	84	431	<code>\interactionmode</code> .....	92
383	<code>\iftok</code> .....	84	432	<code>\interlinepenalties</code> .....	92
384	<code>\iftrue</code> .....	85	433	<code>\interlinepenalty</code> .....	92
385	<code>\ifvbox</code> .....	85	434	<code>\jobname</code> .....	92
386	<code>\ifvmode</code> .....	85	435	<code>\kern</code> .....	93
387	<code>\ifvoid</code> .....	85	436	<code>\language</code> .....	93
388	<code>\ifx</code> .....	85	437	<code>\lastarguments</code> .....	93
389	<code>\ifzerodim</code> .....	85	438	<code>\lastatomclass</code> .....	93
390	<code>\ifzerofloat</code> .....	86	439	<code>\lastboundary</code> .....	93
391	<code>\ifzeronum</code> .....	86	440	<code>\lastbox</code> .....	93
392	<code>\ignorearguments</code> .....	86	441	<code>\lastchkdimension</code> .....	93
393	<code>\ignoredepthcriterion</code> .....	86	442	<code>\lastchknumber</code> .....	94

443	<code>\lastkern</code> .....	94	492	<code>\long</code> .....	102
444	<code>\lastleftclass</code> .....	94	493	<code>\looseness</code> .....	102
445	<code>\lastlinefit</code> .....	94	494	<code>\lower</code> .....	102
446	<code>\lastloopiterator</code> .....	94	495	<code>\lowercase</code> .....	102
447	<code>\lastnamedcs</code> .....	94	496	<code>\lrcode</code> .....	103
448	<code>\lastnodesubtype</code> .....	95	497	<code>\luabytecode</code> .....	103
449	<code>\lastnodetype</code> .....	95	498	<code>\luabytecodecall</code> .....	103
450	<code>\lastpageextra</code> .....	95	499	<code>\luacopyinputnodes</code> .....	103
451	<code>\lastparcontext</code> .....	95	500	<code>\luaodef</code> .....	103
452	<code>\lastpartrigger</code> .....	95	501	<code>\luaescapestring</code> .....	104
453	<code>\lastpenalty</code> .....	95	502	<code>\luafunction</code> .....	104
454	<code>\lastrightclass</code> .....	95	503	<code>\luafunctioncall</code> .....	104
455	<code>\lastskip</code> .....	95	504	<code>\luatexbanner</code> .....	104
456	<code>\lccode</code> .....	96	505	<code>\luatexrevision</code> .....	104
457	<code>\leaders</code> .....	96	506	<code>\luatexversion</code> .....	104
458	<code>\left</code> .....	96	507	<code>\mark</code> .....	105
459	<code>\lefthyphenmin</code> .....	96	508	<code>\marks</code> .....	105
460	<code>\leftmarginkern</code> .....	96	509	<code>\mathaccent</code> .....	105
461	<code>\leftskip</code> .....	96	510	<code>\mathatom</code> .....	105
462	<code>\legno</code> .....	96	511	<code>\mathatomglue</code> .....	105
463	<code>\let</code> .....	96	512	<code>\mathatomskip</code> .....	105
464	<code>\letcharcode</code> .....	96	513	<code>\mathbackwardpenalties</code> .....	106
465	<code>\letcsname</code> .....	97	514	<code>\mathbeginclass</code> .....	106
466	<code>\letfrozen</code> .....	97	515	<code>\mathbin</code> .....	106
467	<code>\letmathatomrule</code> .....	97	516	<code>\mathboundary</code> .....	106
468	<code>\letmathparent</code> .....	97	517	<code>\mathchar</code> .....	106
469	<code>\letmathspacing</code> .....	97	518	<code>\mathcharclass</code> .....	106
470	<code>\letprotected</code> .....	98	519	<code>\mathchardef</code> .....	106
471	<code>\lettolastnamedcs</code> .....	98	520	<code>\mathcharfam</code> .....	106
472	<code>\lettonothing</code> .....	98	521	<code>\mathcharslot</code> .....	107
473	<code>\limits</code> .....	98	522	<code>\mathcheckfencesmode</code> .....	107
474	<code>\linebreakcriterion</code> .....	99	523	<code>\mathchoice</code> .....	107
475	<code>\linebreakoptional</code> .....	99	524	<code>\mathclass</code> .....	107
476	<code>\linebreakpasses</code> .....	99	525	<code>\mathclose</code> .....	108
477	<code>\linedirection</code> .....	99	526	<code>\mathcode</code> .....	108
478	<code>\linepenalty</code> .....	99	527	<code>\mathdictgroup</code> .....	108
479	<code>\lineskip</code> .....	99	528	<code>\mathdictionary</code> .....	108
480	<code>\lineskiplimit</code> .....	100	529	<code>\mathdictproperties</code> .....	108
481	<code>\localcontrol</code> .....	100	530	<code>\mathdirection</code> .....	108
482	<code>\localcontrolled</code> .....	100	531	<code>\mathdisplaymode</code> .....	108
483	<code>\localcontrolledendless</code> .....	101	532	<code>\mathdisplaypenaltyfactor</code> .....	109
484	<code>\localcontrolledloop</code> .....	101	533	<code>\mathdisplayskipmode</code> .....	109
485	<code>\localcontrolledrepeat</code> .....	101	534	<code>\mathdoublescriptmode</code> .....	109
486	<code>\lcalleftbox</code> .....	101	535	<code>\mathendclass</code> .....	109
487	<code>\lcalleftboxbox</code> .....	101	536	<code>\matheqnogapstep</code> .....	109
488	<code>\lcalmiddlebox</code> .....	102	537	<code>\mathfontcontrol</code> .....	109
489	<code>\lcalmiddleboxbox</code> .....	102	538	<code>\mathforwardpenalties</code> .....	110
490	<code>\lcalrightbox</code> .....	102	539	<code>\mathgluemode</code> .....	110
491	<code>\lcalrightboxbox</code> .....	102	540	<code>\mathgroupingmode</code> .....	111



541	<code>\mathinlinepenaltyfactor</code> .....	111	590	<code>\multiplyby</code> .....	119
542	<code>\mathinner</code> .....	111	591	<code>\muskip</code> .....	119
543	<code>\mathleftclass</code> .....	111	592	<code>\muskipdef</code> .....	119
544	<code>\mathlimitsmode</code> .....	111	593	<code>\mutable</code> .....	119
545	<code>\mathmainstyle</code> .....	112	594	<code>\mutoglua</code> .....	119
546	<code>\mathnolimitsmode</code> .....	112	595	<code>\nestedloopiterator</code> .....	119
547	<code>\mathop</code> .....	113	596	<code>\newlinechar</code> .....	120
548	<code>\mathopen</code> .....	113	597	<code>\noalign</code> .....	120
549	<code>\mathord</code> .....	113	598	<code>\noaligned</code> .....	120
550	<code>\mathparentstyle</code> .....	113	599	<code>\noatomruling</code> .....	120
551	<code>\mathpenaltiesmode</code> .....	113	600	<code>\noboundary</code> .....	120
552	<code>\mathpretolerance</code> .....	114	601	<code>\noexpand</code> .....	120
553	<code>\mathpunct</code> .....	114	602	<code>\nohrule</code> .....	121
554	<code>\mathrel</code> .....	114	603	<code>\noindent</code> .....	121
555	<code>\mathrightclass</code> .....	114	604	<code>\nolimits</code> .....	121
556	<code>\mathrulesfam</code> .....	114	605	<code>\nonscript</code> .....	121
557	<code>\mathrulesmode</code> .....	114	606	<code>\nonstopmode</code> .....	121
558	<code>\mathscale</code> .....	114	607	<code>\norelax</code> .....	121
559	<code>\mathscriptsmode</code> .....	114	608	<code>\normalizelinemode</code> .....	122
560	<code>\mathslackmode</code> .....	115	609	<code>\normalizeparammode</code> .....	122
561	<code>\mathspacingmode</code> .....	115	610	<code>\nospaces</code> .....	122
562	<code>\mathstack</code> .....	115	611	<code>\nosubprescript</code> .....	123
563	<code>\mathstackstyle</code> .....	116	612	<code>\nosubscript</code> .....	123
564	<code>\mathstyle</code> .....	116	613	<code>\nosuperprescript</code> .....	123
565	<code>\mathstylefontid</code> .....	116	614	<code>\nosuperscript</code> .....	123
566	<code>\mathsurround</code> .....	116	615	<code>\novrule</code> .....	123
567	<code>\mathsurroundmode</code> .....	116	616	<code>\nulldelimiterspace</code> .....	123
568	<code>\mathsurroundskip</code> .....	116	617	<code>\nullfont</code> .....	123
569	<code>\maththreshold</code> .....	116	618	<code>\number</code> .....	123
570	<code>\mathtolerance</code> .....	116	619	<code>\numericsscale</code> .....	124
571	<code>\maxdeadcycles</code> .....	117	620	<code>\numericsscaled</code> .....	124
572	<code>\maxdepth</code> .....	117	621	<code>\numexpr</code> .....	124
573	<code>\meaning</code> .....	117	622	<code>\numexpression</code> .....	124
574	<code>\meaningasis</code> .....	117	623	<code>\omit</code> .....	125
575	<code>\meaningful</code> .....	117	624	<code>\optionalboundary</code> .....	125
576	<code>\meaningfull</code> .....	117	625	<code>\or</code> .....	126
577	<code>\meaningles</code> .....	117	626	<code>\orelse</code> .....	126
578	<code>\meaningless</code> .....	118	627	<code>\orphanpenalties</code> .....	127
579	<code>\medmuskip</code> .....	118	628	<code>\orphanpenalty</code> .....	127
580	<code>\message</code> .....	118	629	<code>\orunless</code> .....	127
581	<code>\middle</code> .....	118	630	<code>\outer</code> .....	127
582	<code>\mkern</code> .....	118	631	<code>\output</code> .....	127
583	<code>\month</code> .....	118	632	<code>\outputbox</code> .....	128
584	<code>\moveleft</code> .....	118	633	<code>\outputpenalty</code> .....	128
585	<code>\moveright</code> .....	118	634	<code>\over</code> .....	128
586	<code>\mskip</code> .....	118	635	<code>\overfullrule</code> .....	128
587	<code>\muexpr</code> .....	119	636	<code>\overline</code> .....	128
588	<code>\mugluespecdef</code> .....	119	637	<code>\overloaded</code> .....	129
589	<code>\multiply</code> .....	119	638	<code>\overloadmode</code> .....	129



639	<code>\overshoot</code>	129	688	<code>\pettymuskip</code>	135
640	<code>\overwithdelims</code>	130	689	<code>\positdef</code>	135
641	<code>\pageboundary</code>	130	690	<code>\postdisplaypenalty</code>	135
642	<code>\pagedepth</code>	130	691	<code>\postexhyphenchar</code>	135
643	<code>\pagediscards</code>	130	692	<code>\posthyphenchar</code>	135
644	<code>\pageexcess</code>	130	693	<code>\postinlinepenalty</code>	135
645	<code>\pageextragoal</code>	130	694	<code>\postshortinlinepenalty</code>	136
646	<code>\pagefilllstretch</code>	130	695	<code>\prebinoppenalty</code>	136
647	<code>\pagefillstretch</code>	130	696	<code>\predisplaydirection</code>	136
648	<code>\pagefilstretch</code>	130	697	<code>\predisplaygapfactor</code>	136
649	<code>\pagefistretch</code>	131	698	<code>\predisplaypenalty</code>	136
650	<code>\pagegoal</code>	131	699	<code>\predisplaysize</code>	136
651	<code>\pagelastdepth</code>	131	700	<code>\preexhyphenchar</code>	136
652	<code>\pagelastfilllstretch</code>	131	701	<code>\prehyphenchar</code>	136
653	<code>\pagelastfillstretch</code>	131	702	<code>\preinlinepenalty</code>	136
654	<code>\pagelastfilstretch</code>	131	703	<code>\prerelpenalty</code>	136
655	<code>\pagelastfistretch</code>	131	704	<code>\preshortinlinepenalty</code>	137
656	<code>\pagelastheight</code>	131	705	<code>\pretolerance</code>	137
657	<code>\pagelastshrink</code>	131	706	<code>\prevdepth</code>	137
658	<code>\pagelaststretch</code>	131	707	<code>\prevgraf</code>	137
659	<code>\pageshrink</code>	132	708	<code>\previousloopiterator</code>	137
660	<code>\pagestretch</code>	132	709	<code>\primescript</code>	137
661	<code>\pagetotal</code>	132	710	<code>\protected</code>	138
662	<code>\pagevsize</code>	132	711	<code>\protecteddetokenize</code>	138
663	<code>\par</code>	132	712	<code>\protectedexpandeddetokenize</code>	138
664	<code>\parametercount</code>	132	713	<code>\protrudechars</code>	138
665	<code>\parameterdef</code>	132	714	<code>\protrusionboundary</code>	138
666	<code>\parameterindex</code>	132	715	<code>\pxdimen</code>	138
667	<code>\parametermark</code>	133	716	<code>\quitloop</code>	138
668	<code>\parametermode</code>	133	717	<code>\quitloopnow</code>	138
669	<code>\parattribute</code>	133	718	<code>\quitvmode</code>	139
670	<code>\pardirection</code>	133	719	<code>\radical</code>	139
671	<code>\parfillleftskip</code>	133	720	<code>\raise</code>	139
672	<code>\parfillrightskip</code>	133	721	<code>\rdivide</code>	139
673	<code>\parfillskip</code>	133	722	<code>\rdivideby</code>	139
674	<code>\parindent</code>	133	723	<code>\realign</code>	140
675	<code>\parinitleftskip</code>	133	724	<code>\relax</code>	140
676	<code>\parinitrightskip</code>	133	725	<code>\relpenalty</code>	140
677	<code>\parpasses</code>	133	726	<code>\resetmathspacing</code>	140
678	<code>\parshape</code>	134	727	<code>\restorecatcodetable</code>	140
679	<code>\parshapedimen</code>	134	728	<code>\retained</code>	142
680	<code>\parshapeindent</code>	134	729	<code>\retokenized</code>	143
681	<code>\parshapelength</code>	134	730	<code>\right</code>	144
682	<code>\parshapewidth</code>	134	731	<code>\righthyphenmin</code>	144
683	<code>\parskip</code>	134	732	<code>\rightmarginkern</code>	144
684	<code>\patterns</code>	134	733	<code>\rightskip</code>	144
685	<code>\pausing</code>	134	734	<code>\romannumeral</code>	144
686	<code>\penalty</code>	134	735	<code>\rptide</code>	144
687	<code>\permanent</code>	134	736	<code>\savecatcodetable</code>	144

737	<code>\savingshyphcodes</code> .....	144	786	<code>\shiftedsuperscript</code> .....	152
738	<code>\savingsdiscards</code> .....	145	787	<code>\shipout</code> .....	152
739	<code>\scaledemwidth</code> .....	145	788	<code>\shortinlinemaththreshold</code> .....	152
740	<code>\scaledexheight</code> .....	145	789	<code>\shortinlineorphanpenalty</code> .....	153
741	<code>\scaledextraspaces</code> .....	145	790	<code>\show</code> .....	153
742	<code>\scaledfontcharba</code> .....	145	791	<code>\showbox</code> .....	153
743	<code>\scaledfontcharbp</code> .....	145	792	<code>\showboxbreadth</code> .....	153
744	<code>\scaledfontcharht</code> .....	145	793	<code>\showboxdepth</code> .....	153
745	<code>\scaledfontcharic</code> .....	145	794	<code>\showcodestack</code> .....	153
746	<code>\scaledfontcharta</code> .....	145	795	<code>\showgroups</code> .....	153
747	<code>\scaledfontcharwd</code> .....	145	796	<code>\showifs</code> .....	153
748	<code>\scaledfontdimen</code> .....	146	797	<code>\showlists</code> .....	153
749	<code>\scaledinterwordshrink</code> .....	146	798	<code>\shownodedetails</code> .....	154
750	<code>\scaledinterwordspace</code> .....	146	799	<code>\showstack</code> .....	154
751	<code>\scaledinterwordstretch</code> .....	146	800	<code>\showthe</code> .....	155
752	<code>\scaledmathaxis</code> .....	146	801	<code>\showtokens</code> .....	155
753	<code>\scaledmathemwidth</code> .....	146	802	<code>\singlelinepenalty</code> .....	155
754	<code>\scaledmathexheight</code> .....	146	803	<code>\skewchar</code> .....	155
755	<code>\scaledmathstyle</code> .....	146	804	<code>\skip</code> .....	155
756	<code>\scaledslantperpoint</code> .....	146	805	<code>\skipdef</code> .....	155
757	<code>\scantextokens</code> .....	146	806	<code>\snapshotpar</code> .....	155
758	<code>\scantokens</code> .....	147	807	<code>\spacefactor</code> .....	156
759	<code>\scriptfont</code> .....	147	808	<code>\spacefactormode</code> .....	157
760	<code>\scriptscriptfont</code> .....	147	809	<code>\spacefactorshrinklimit</code> .....	157
761	<code>\scriptscriptstyle</code> .....	147	810	<code>\spacefactorstretchlimit</code> .....	157
762	<code>\scriptspace</code> .....	147	811	<code>\spaceskip</code> .....	157
763	<code>\scriptstyle</code> .....	147	812	<code>\span</code> .....	157
764	<code>\scrollmode</code> .....	147	813	<code>\splitbotmark</code> .....	157
765	<code>\semiexpand</code> .....	148	814	<code>\splitbotmarks</code> .....	157
766	<code>\semiexpanded</code> .....	148	815	<code>\splitdiscards</code> .....	157
767	<code>\semiprotected</code> .....	148	816	<code>\splitfirstmark</code> .....	157
768	<code>\setbox</code> .....	148	817	<code>\splitfirstmarks</code> .....	158
769	<code>\setdefaultmathcodes</code> .....	148	818	<code>\splitmaxdepth</code> .....	158
770	<code>\setfontid</code> .....	148	819	<code>\splittopskip</code> .....	158
771	<code>\setlanguage</code> .....	149	820	<code>\srule</code> .....	158
772	<code>\setmathatomrule</code> .....	149	821	<code>\string</code> .....	158
773	<code>\setmathdisplaypostpenalty</code> .....	149	822	<code>\subprescript</code> .....	158
774	<code>\setmathdisplayprepenalty</code> .....	149	823	<code>\subscript</code> .....	158
775	<code>\setmathignore</code> .....	150	824	<code>\superprescript</code> .....	158
776	<code>\setmathoptions</code> .....	150	825	<code>\superscript</code> .....	158
777	<code>\setmathpostpenalty</code> .....	150	826	<code>\supmarkmode</code> .....	158
778	<code>\setmathprepenalty</code> .....	150	827	<code>\swapsvalues</code> .....	159
779	<code>\setmathspacing</code> .....	151	828	<code>\tabsize</code> .....	159
780	<code>\sfcode</code> .....	151	829	<code>\tabskip</code> .....	159
781	<code>\shapingpenaltiesmode</code> .....	151	830	<code>\textdirection</code> .....	160
782	<code>\shapingpenalty</code> .....	151	831	<code>\textfont</code> .....	160
783	<code>\shiftedsuperscript</code> .....	151	832	<code>\textstyle</code> .....	160
784	<code>\shiftedsuperscript</code> .....	152	833	<code>\the</code> .....	160
785	<code>\shiftedsuperscript</code> .....	152	834	<code>\thewithoutunit</code> .....	160

835	<code>\thickmuskip</code> .....	160	884	<code>\tsplit</code> .....	167
836	<code>\thinmuskip</code> .....	160	885	<code>\uccode</code> .....	167
837	<code>\time</code> .....	161	886	<code>\uchyph</code> .....	167
838	<code>\tinymuskip</code> .....	161	887	<code>\uleaders</code> .....	167
839	<code>\tocharacter</code> .....	161	888	<code>\unboundary</code> .....	167
840	<code>\todimension</code> .....	161	889	<code>\undent</code> .....	168
841	<code>\tohexadecimal</code> .....	161	890	<code>\underline</code> .....	168
842	<code>\tointeger</code> .....	161	891	<code>\unexpanded</code> .....	168
843	<code>\tokenized</code> .....	161	892	<code>\unexpandedendless</code> .....	168
844	<code>\toks</code> .....	162	893	<code>\unexpandedloop</code> .....	168
845	<code>\toksapp</code> .....	162	894	<code>\unexpandedrepeat</code> .....	169
846	<code>\toksdef</code> .....	162	895	<code>\unhbox</code> .....	169
847	<code>\tokspre</code> .....	162	896	<code>\unhcopy</code> .....	169
848	<code>\tolerance</code> .....	162	897	<code>\unhpack</code> .....	169
849	<code>\tolerant</code> .....	163	898	<code>\unkern</code> .....	169
850	<code>\tomathstyle</code> .....	163	899	<code>\unless</code> .....	169
851	<code>\topmark</code> .....	163	900	<code>\unletfrozen</code> .....	170
852	<code>\topmarks</code> .....	163	901	<code>\unletprotected</code> .....	170
853	<code>\topskip</code> .....	163	902	<code>\unpenalty</code> .....	170
854	<code>\toscaled</code> .....	164	903	<code>\unskip</code> .....	170
855	<code>\tosparsedimension</code> .....	164	904	<code>\untraced</code> .....	170
856	<code>\tosparsescaled</code> .....	164	905	<code>\unvbox</code> .....	171
857	<code>\tpack</code> .....	164	906	<code>\unvcopy</code> .....	171
858	<code>\tracingadjusts</code> .....	164	907	<code>\unvpack</code> .....	171
859	<code>\tracingalignments</code> .....	164	908	<code>\uppercase</code> .....	171
860	<code>\tracingassigns</code> .....	164	909	<code>\vadjust</code> .....	171
861	<code>\tracingcommands</code> .....	164	910	<code>\valign</code> .....	171
862	<code>\tracingexpressions</code> .....	164	911	<code>\variablefam</code> .....	171
863	<code>\tracingfullboxes</code> .....	165	912	<code>\vbadness</code> .....	172
864	<code>\tracinggroups</code> .....	165	913	<code>\vbox</code> .....	172
865	<code>\tracinghyphenation</code> .....	165	914	<code>\vcenter</code> .....	172
866	<code>\tracingifs</code> .....	165	915	<code>\vfil</code> .....	172
867	<code>\tracinginserts</code> .....	165	916	<code>\vfill</code> .....	172
868	<code>\tracinglevels</code> .....	165	917	<code>\vfilneg</code> .....	172
869	<code>\tracinglists</code> .....	165	918	<code>\vfuzz</code> .....	172
870	<code>\tracinglostchars</code> .....	165	919	<code>\virtualhrule</code> .....	172
871	<code>\tracingmacros</code> .....	165	920	<code>\virtualvrule</code> .....	172
872	<code>\tracingmarks</code> .....	165	921	<code>\vkern</code> .....	173
873	<code>\tracingmath</code> .....	166	922	<code>\vpack</code> .....	173
874	<code>\tracingnesting</code> .....	166	923	<code>\vpenalty</code> .....	173
875	<code>\tracingnodes</code> .....	166	924	<code>\vrule</code> .....	173
876	<code>\tracingonline</code> .....	166	925	<code>\vsize</code> .....	173
877	<code>\tracingoutput</code> .....	166	926	<code>\vskip</code> .....	173
878	<code>\tracingpages</code> .....	166	927	<code>\vsplit</code> .....	173
879	<code>\tracingparagraphs</code> .....	166	928	<code>\vss</code> .....	173
880	<code>\tracingpasses</code> .....	166	929	<code>\vtop</code> .....	173
881	<code>\tracingpenalties</code> .....	166	930	<code>\wd</code> .....	173
882	<code>\tracingrestores</code> .....	167	931	<code>\widowpenalties</code> .....	174
883	<code>\tracingstats</code> .....	167	932	<code>\widowpenalty</code> .....	174

933	<code>\wordboundary</code>	174	938	<code>\xspaceskip</code>	174
934	<code>\wrapuppar</code>	174	939	<code>\xtoks</code>	175
935	<code>\xdef</code>	174	940	<code>\xtoksapp</code>	175
936	<code>\xdefcsname</code>	174	941	<code>\xtokspre</code>	175
937	<code>\xleaders</code>	174	942	<code>\year</code>	175

In this document the section titles that discuss the original  $\text{\TeX}$  and  $\varepsilon\text{\TeX}$  primitives have a different color those explaining the  $\text{\LuaTeX}$  and  $\text{\LuaMetaTeX}$  primitives.

Primitives that extend typesetting related functionality, provide control over subsystems (like math), allocate additional data types and resources, deal with fonts and languages, manipulate boxes and glyphs, etc. are hardly discussed here, only mentioned. Math for instance is a topic of its own. In this document we concentrate on the programming aspects.

Most of the new primitives are discussed in specific manuals and often also original primitives are covered there but the best explanations of the traditional primitives can be found in The  $\text{\TeX}$ book by Donald Knuth and  $\text{\TeX}$  by Topic from Victor Eijkhout. I see no need to try to improve on those.

## Primitives

### 1 `\<space>`

This original  $\text{\TeX}$  primitive is equivalent to the more verbose `\explicitSPACE`.

### 2 `\-`

This original  $\text{\TeX}$  primitive is equivalent to the more verbose `\explicitdiscretionary`.

### 3 `\/`

This original  $\text{\TeX}$  primitive is equivalent to the more verbose `\explicititaliccorrection`.

### 4 `\above`

This is a variant of `\over` that doesn't put a rule in between.

### 5 `\abovedisplayshortskip`

The glue injected before a display formula when the line above it is not overlapping with the formula. Watch out for interference with `\baselineskip`. It can be controlled by `\displayskipmode`.

### 6 `\abovedisplayskip`

The glue injected before a display formula. Watch out for interference with `\baselineskip`. It can be controlled by `\displayskipmode`.

## 7 `\abovewithdelims`

This is a variant of `\atop` but with delimiters. It has a more advanced upgrade in `\Uabovewithdelims`.

## 8 `\accent`

This primitive is kind of obsolete in wide engines and takes two arguments: the indexes of an accent and a base character.

## 9 `\additionalpageskip`

This quantity will be added to the current page goal, stretch and shrink after which it will be set to zero.

## 10 `\adjdemerits`

When  $\TeX$  considers two lines to be incompatible it will add this penalty to its verdict when considering this breakpoint.

## 11 `\adjustspacing`

This parameter controls expansion (hz). A value 2 expands glyphs and font kerns and a value of 3 only glyphs. Expansion of kerns can have side effects when they are used for positioning by OpenType features.

## 12 `\adjustspacingshrink`

When set to a non zero value this overloads the shrink maximum in a font when expansion is applied. This is then the case for all fonts.

## 13 `\adjustspacingstep`

When set to a non zero value this overloads the expansion step in a font when expansion is applied. This is then the case for all fonts.

## 14 `\adjustspacingstretch`

When set to a non zero value this overloads the stretch maximum in a font when expansion is applied. This is then the case for all fonts.

## 15 `\advance`

Advances the given register by an also given value:

```
\advance\scratchdimen      10pt
\advance\scratchdimen      by 3pt
\advance\scratchcounterone \zerocount
\advance\scratchcounterone \scratchcountertwo
```

The `by` keyword is optional.

## 16 `\advanceby`

This is slightly more efficient variant of `\advance` that doesn't look for `by` and therefore, if one is missing, doesn't need to push back the last seen token. Using `\advance` with `by` is nearly as efficient but takes more tokens.

## 17 `\afterassigned`

The `\afterassignment` primitive stores a token to be injected (and thereby expanded) after an assignment has happened. Unlike `\aftergroup`, multiple calls are not accumulated, and changing that would be too incompatible. This is why we have `\afterassigned`, which can be used to inject a bunch of tokens. But in order to be consistent this one is also not accumulative.

```
\afterassigned{done}%
\afterassigned{{\bf done}}%
\scratchcounter=123
```

results in: **done** being typeset.

## 18 `\afterassignment`

The token following `\afterassignment`, a traditional  $\text{\TeX}$  primitive, is saved and gets injected (and then expanded) after a following assignment took place.

```
\afterassignment !\def\MyMacro {} \quad
\afterassignment !\let\MyMacro ? \quad
\afterassignment !\scratchcounter 123 \quad
\afterassignment !%
\afterassignment ?\advance\scratchcounter by 1
```

The `\afterassignments` are not accumulated, the last one wins:

! ! ! ?

## 19 `\aftergroup`

The traditional  $\text{\TeX}$  `\aftergroup` primitive stores the next token and expands that after the group has been closed.

Multiple `\aftergroups` are combined:

```
before{ ! \aftergroup a\aftergroup f\aftergroup t\aftergroup e\aftergroup r}
before ! after
```

## 20 `\aftergrouped`

The in itself powerful `\aftergroup` primitives works quite well, even if you need to do more than one thing: you can either use it multiple times, or you can define a macro that does multiple things and

apply that after the group. However, you can avoid that by using this primitive which takes a list of tokens.

```
regular
\bgrou
\aftergrouped{regular}%
\bf bold
\egrou
```

Because it happens after the group, we're no longer typesetting in bold.

regular **bold** regular

## 21 \aliased

This primitive is part of the overload protection subsystem where control sequences can be tagged.

```
\permanent\def\foo{F00}
        \let\of\foo
\aliased \let\oof\foo

\meaningasis\foo
\meaningasis\of
\meaningasis\oof
```

gives:

```
\permanent \def \foo {F00}
\def \of {F00}
\permanent \def \oof {F00}
```

When a something is \let the ‘permanent’, ‘primitive’ and ‘immutable’ flags are removed but the \aliased prefix retains them.

```
\let\relaxed\relax

\meaningasis\relax
\meaningasis\relaxed
```

So in this example the \relaxed alias is not flagged as primitive:

```
\primitive \relax
\relax
```

## 22 \aligncontent

This is equivalent to a hash in an alignment preamble. Contrary to \alignmark there is no need to duplicate inside a macro definition.

## 23 \alignmark

When you have the # not set up as macro parameter character cq. align mark, you can use this primitive instead. The same rules apply with respect to multiple such tokens in (nested) macros and alignments.



## 24 `\alignmentcellsource`

This sets the source id (a box property) of the current alignment cell.

## 25 `\alignmentwrapsource`

This sets the source id (a box property) of the current alignment row (in a `\halign`) or column (in a `\valign`).

## 26 `\aligntab`

When you have the `&` not set up as align tab, you can use this primitive instead. The same rules apply with respect to multiple such tokens in (nested) macros and alignments.

## 27 `\allcrampedstyles`

A symbolic representation of `\crampeddisplaystyle`, `\crampedtextstyle`, `\crampedscriptstyle` and `\crampedscriptscriptstyle`; integer representation: 17.

## 28 `\alldisplaystyles`

A symbolic representation of `\displaystyle` and `\crampeddisplaystyle`; integer representation: 8.

## 29 `\allmainstyles`

A symbolic representation of `\displaystyle`, `\crampeddisplaystyle`, `\textstyle` and `\crampedtextstyle`; integer representation: 13.

## 30 `\allmathstyles`

A symbolic representation of `\displaystyle`, `\crampeddisplaystyle`, `\textstyle`, `\crampedtextstyle`, `\scriptstyle`, `\crampedscriptstyle`, `\scriptscriptstyle` and `\crampedscriptscriptstyle`; integer representation: 12.

## 31 `\allscriptscriptstyles`

A symbolic representation of `\scriptscriptstyle` and `\crampedscriptscriptstyle`; integer representation: 11.

## 32 `\allscriptstyles`

A symbolic representation of `\scriptstyle` and `\crampedscriptstyle`; integer representation: 10.

## 33 `\allsplitstyles`

A symbolic representation of `\displaystyle` and `\textstyle` but not `\scriptstyle` and `\scriptscriptstyle`: set versus reset; integer representation: 14.

### 34 `\alltextstyles`

A symbolic representation of `\textstyle` and `\crampedtextstyle`; integer representation: 9.

### 35 `\alluncrampedstyles`

A symbolic representation of `\displaystyle`, `\textstyle`, `\scriptstyle` and `\scriptscriptstyle`; integer representation: 16.

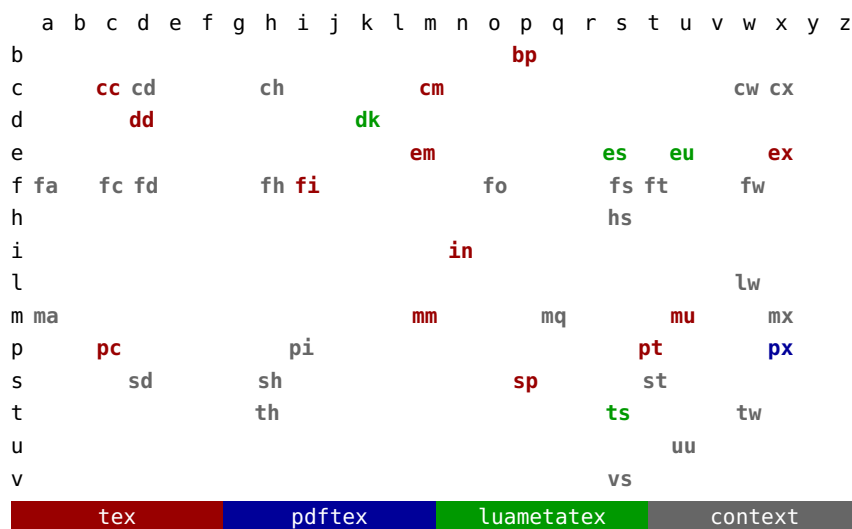
### 36 `\allunsplitstyles`

A symbolic representation of `\scriptstyle` and `\scriptscriptstyle`; integer representation: 15.

### 37 `\amcode`

### 38 `\associateunit`

The  $\text{\TeX}$  engine comes with some build in units, like pt (fixed) and em (adaptive). On top of that a macro package can add additional units, which is what we do in Con $\text{\TeX}$ t. In figure 1 we show the current repertoire.



**Figure 1** Available units

When this primitive is used in a context where a number is expected it returns the origin of the unit (in the color legend running from 1 upto 4). A new unit is defined as:

```
\newdimen\MyDimenZA \MyDimenZA=10pt

\protected\def\MyDimenAB{\dimexpr\hsize/2\relax}

\associateunit za \MyDimenZA
\associateunit zb \MyMacroZB
```

Possible associations are: macros that expand to a dimension, internal dimension registers, register dimensions (`\dimendef`), direct dimensions (`\dimensiondef`) and Lua functions that return a dimension.

One can run into scanning ahead issues where  $\text{\TeX}$  expects a unit and a user unit gets expanded. This is why for instance in  $\text{\ConTeXt}$  we define the `ma` unit as:

```
\protected\def\mathaxisunit{\scaledmathaxis\mathstyle\norelax}

\associateunit ma \mathaxisunit % or \newuserunit \mathaxisunit ma
```

So that it can be used in rule specifications that themselves look ahead for keywords and therefore are normally terminated by a `\relax`. Adding the extra `\norelax` will make the scanner see one that doesn't get fed back into the input. Of course a macro package has to manage extra units in order to avoid conflicts.

### 39 `\atendoffile`

The `\everyeof` primitive is kind of useless because you don't know if a file (which can be a tokenlist processed as pseudo file) itself includes a file, which then results in nested application of this token register. One way around this is:

```
\atendoffile\SomeCommand
```

This acts on files the same way as `\atendofgroup` does. Multiple calls will be accumulated and are bound to the current file.

### 40 `\atendoffiled`

This is the multi token variant of `\atendoffile`. Multiple invocations are accumulated and by default prepended to the existing list. As with grouping this permits proper nesting. You can force an append by the optional keyword `reverse`.

### 41 `\atendofgroup`

The token provided will be injected just before the group ends. Because these tokens are collected, you need to be aware of possible interference between them. However, normally this is managed by the macro package.

```
\bgroup
\atendofgroup\unskip
\atendofgroup )%
( but it works okay
\egroup
```

Of course these effects can also be achieved by combining (extra) grouping with `\aftergroup` calls, so this is more a convenience primitives than a real necessity: (but it works okay), as proven here.

### 42 `\atendofgrouped`

This is the multi token variant of `\atendofgroup`. Of course the next example is somewhat naive when it comes to spacing and so, but it shows the purpose.

```
\bgroup
\atendofgrouped{\bf QED}%
```

`\atendofgrouped{ (indeed) }%`

This sometimes looks nicer.

`\egroup`

Multiple invocations are accumulated: This sometimes looks nicer. **QED (indeed).**

### 43 `\atop`

This one stack two math elements on top of each other, like a fraction but with no rule. It has a more advanced upgrade in `\Uatop`.

### 44 `\atopwithdelims`

This is a variant of `\atop` but with delimiters. It has a more advanced upgrade in `\Uatopwithdelims`.

### 45 `\attribute`

The following sets an `attribute(register)` value:

`\attribute 999 = 123`

An attribute is unset by assigning -2147483647 to it. A user needs to be aware of attributes being used now and in the future of a macro package and setting them this way is very likely going to interfere.

### 46 `\attributedef`

This primitive can be used to relate a control sequence to an attribute register and can be used to implement a mechanism for defining unique ones that won't interfere. As with other registers: leave management to the macro package in order to avoid unwanted side effects!

### 47 `\automaticdiscretionary`

This is an alias for the automatic hyphen trigger `-`.

### 48 `\automatichyphenpenalty`

The penalty injected after an automatic discretionary `-`, when `\hyphenationmode` enables this.

### 49 `\automigrationmode`

This bitset determines what will bubble up to an outer level:

0x01 mark  
 0x02 insert  
 0x04 adjust  
 0x08 pre  
 0x10 post

The current value is 0xFFFF.

## 50 `\autoparagraphmode`

A paragraph can be triggered by an empty line, a `\par` token or an equivalent of it. This parameter controls how `\par` is interpreted in different scenarios:

```
0x01 text
0x02 macro
0x04 continue
```

The current value is 0x1 and setting it to a non-zero value can have consequences for mechanisms that expect otherwise. The text option uses the same code as an empty line. The macro option checks a token in a macro preamble against the frozen `\`

token. The last option ignores the `par` token.

## 51 `\badness`

This one returns the last encountered badness value.

## 52 `\baselineskip`

This is the maximum glue put between lines. The depth of the previous and height of the next line are subtracted.

## 53 `\batchmode`

This command disables (error) messages which can save some runtime in situations where  $\text{\TeX}$ 's character-by-character log output impacts runtime. It only makes sense in automated workflows where one doesn't look at the log anyway.

## 54 `\begincsname`

The next code creates a control sequence token from the given serialized tokens:

```
\csname mymacro\endcsname
```

When `\mymacro` is not defined a control sequence will be created with the meaning `\relax`. A side effect is that a test for its existence might fail because it now exists. The next sequence will *not* create a control sequence:

```
\begincsname mymacro\endcsname
```

This actually is kind of equivalent to:

```
\ifcsname mymacro\endcsname
  \csname mymacro\endcsname
\fi
```

## 55 `\begingroup`

This primitive starts a group and has to be ended with `\endgroup`. See `\beginsimplegroup` for more info.

## 56 `\beginlocalcontrol`

Once  $\text{\TeX}$  is initialized it will enter the main loop. In there certain commands trigger a function that itself can trigger further scanning and functions. In  $\text{\LuaMetaTeX}$  we can have local main loops and we can either enter it from the Lua end (which we don't discuss here) or at the  $\text{\TeX}$  end using this primitive.

```
\scratchcounter100

\edef\whatever{
  a
  \beginlocalcontrol
    \advance\scratchcounter 10
  b
  \endlocalcontrol
  \beginlocalcontrol
    c
  \endlocalcontrol
  d
  \advance\scratchcounter 10
}

\the\scratchcounter
\whatever
\the\scratchcounter
```

A bit of close reading probably gives an impression of what happens here:

```
b c

110 a d 120
```

The local loop can actually result in material being injected in the current node list. However, where normally assignments are not taking place in an `\edef`, here they are applied just fine. Basically we have a local  $\text{\TeX}$  job, be it that it shares all variables with the parent loop.

## 57 `\beginmathgroup`

In math mode grouping with `\begingroup` and `\endgroup` in some cases works as expected, but because the math input is converted in a list that gets processed later some settings can become persistent, like changes in style or family. The engine therefore provides the alternatives `\beginmathgroup` and `\endmathgroup` that restore some properties.

## 58 `\beginsimplegroup`

The original  $\text{\TeX}$  engine distinguishes two kind of grouping that at the user end show up as:

```
\begingroup \endgroup
\bgroup \egroup { }
```

where the last two pairs are equivalent unless the scanner explicitly wants to see a left and/or right brace and not an equivalent. For the sake of simplify we use the aliases here. It is not possible to mix these pairs, so:

```
\bgroup xxx\endgroup
\begingroup xxx\egroup
```

will in both cases issue an error. This can make it somewhat hard to write generic grouping macros without somewhat dirty trickery. The way out is to use the generic group opener `\beginsimplegroup`.

Internally LuaMetaTeX is aware of what group it currently is dealing with and there we distinguish:

simple group	<code>\bgroup</code>	<code>\egroup</code>
semi simple group	<code>\begingroup</code>	<code>\endgroup \endsimplegroup</code>
also simple group	<code>\beginsimplegroup</code>	<code>\egroup \endgroup \endsimplegroup</code>
math simple group	<code>\beginmathgroup</code>	<code>\endmathgroup</code>

This means that you can say:

```
\beginsimplegroup xxx\endsimplegroup
\beginsimplegroup xxx\endgroup
\beginsimplegroup xxx\egroup
```

So a group started with `\beginsimplegroup` can be finished in three ways which means that the user (or calling macro) doesn't have take into account what kind of grouping was used to start with. Normally usage of this primitive is hidden in macros and not something the user has to be aware of.

## 59 `\belowdisplayshortskip`

The glue injected after a display formula when the line above it is not overlapping with the formula (TeX can't look ahead). Watch out for interference with `\baselineskip`. It can be controlled by `\displayskipmode`.

## 60 `\belowdisplayskip`

The glue injected after a display formula. Watch out for interference with `\baselineskip`. It can be controlled by `\displayskipmode`.

## 61 `\binoppenalty`

This internal quantity is a compatibility feature because normally we will use the inter atom spacing variables.

## 62 `\botmark`

This is a reference to the last mark on the current page, it gives back tokens.

## 63 `\botmarks`

This is a reference to the last mark with the given id (a number) on the current page, it gives back tokens.



## 64 `\boundary`

Boundaries are signals added to the current list. This primitive injects a user boundary with the given (integer) value. Such a boundary can be consulted at the Lua end or with `\lastboundary`.

## 65 `\box`

This is the box register accessor. While other registers have one property a box has many, like `\wd`, `\ht` and `\dp`. This primitive returns the box and resets the register.

## 66 `\boxadapt`

Adapting will recalculate the dimensions with a scale factor for the glue:

```
\setbox 0 \hbox {test test test}
\setbox 2 \hbox {\red test test test} \boxadapt 0 200
\setbox 4 \hbox {\blue test test test} \boxadapt 0 -200
\ruledhbox{\box0} \vskip-\lineheight
\ruledhbox{\box0} \vskip-\lineheight
\ruledhbox{\box0}
```

Like `\boxfreeze` and `\boxrepack` this primitive has been introduced for experimental usage, although we do use some in production code.

test test test

## 67 `\boxanchor`

This feature is part of an (experimental) mechanism that relates boxes. The engine just tags a box and it is up to the macro package to deal with it.

```
\setbox0\hbox anchor "01010202 {test}\tohexadecimal\boxanchor0
```

This gives: 1010202. Of course this feature is very macro specific and should not be used across macro packages without coordination. An anchor has two parts each not exceeding 0x0FFF.

## 68 `\boxanchors`

This feature is part of an (experimental) mechanism that relates boxes. The engine just tags a box and it is up to the macro package to deal with it.

```
\setbox0\hbox anchors "0101 "0202 {test}\tohexadecimal\boxanchors0
```

This gives: 1010202. Of course this feature is very macro specific and should not be used across macro packages without coordination. An anchor has two parts each not exceeding 0x0FFF.

## 69 `\boxattribute`

Every node, and therefore also every box gets the attributes set that are active at the moment of creation. Additional attributes can be set too:

```
\darkred
\setbox0\hbox attr 9999 1 {whatever}
\the\boxattribute 0 \colorattribute
\the\boxattribute 0 9998
\the\boxattribute 0 9999
```

A macro package should make provide a way define attributes that don't clash the ones it needs itself, like, in ConT<sub>E</sub>Xt, the ones that can set a color

```
4
-2147483647
1
```

The number -2147483647 (-7FFFFFFF) indicates an unset attribute.

## 70 \boxdirection

The direction of a box defaults to l2r but can be explicitly set:

```
\setbox0\hbox direction 1 {this is a test}\textdirection1
\setbox2\hbox direction 0 {this is a test}\textdirection0
\the\boxdirection0: \box0
\the\boxdirection2: \box2
```

The \textdirection does not influence the box direction:

```
1: tset a si siht
0: this is a test
```

## 71 \boxfinalize

This is special version of \boxfreeze which we demonstrate with an example:

```
\boxlimitate 0 0 % don't recurse
\boxfreeze 2 0 % don't recurse
\boxfinalize 4 500 % scale glue multiplier by .50
\boxfinalize 6 250 % scale glue multiplier by .25
\boxfinalize 8 100 % scale glue multiplier by .10

\hpack\bggroup
\copy0\quad\copy2\quad\copy4\quad\copy6\quad\copy8
\egroup
```

where the boxes are populated with:

```
\setbox0\rule\vbox to 3cm{\hsize 2cm test\vskip10pt plus 10pt test}
\setbox2\copy0\setbox4\copy0\setbox6\copy0\setbox8\copy0
```

test	test	test	test	test
test		test	test	test
	test			

## 72 `\boxfreeze`

Glue in a box has a fixed component that will always be used and stretch and shrink that kicks in when needed. The effective value (width) of the glue is driven by some box parameters that are set by the packaging routine. This is why we can unbox: the original value is kept. It is the backend that calculates the effective value. The `\boxfreeze` primitive can do the same: turn the flexible glue into a fixed one.

```
\setbox 0 \hbox to 6cm {\hss frost}
\setbox 2 \hbox to 6cm {\hss frost}
\boxfreeze 2 0
\ruledhbox{\unhbox 0}
\ruledhbox{\unhbox 2}
```

The second parameter to `\boxfreeze` determines recursion. We don't recurse here so just freeze the outer level:

```
frost
 frost
```

## 73 `\boxgeometry`

A box can have an orientation, offsets and/or anchors. These are stored independently but for efficiency reasons we register if one or more of these properties is set. This primitive accesses this state; it is a bitset:

```
0x01 offset
0x02 orientation
0x04 anchor
```

## 74 `\boxlimit`

This primitive will freeze the glue in a box but only when there is glue marked with the limit option.

## 75 `\boxlimitate`

This primitive will freeze the glue in a box. It takes two arguments, a box number and an number that when set to non-zero will recurse into nested lists.

## 76 `\boxlimitmode`

This variable controls if boxes with glue marked 'limit' will be checked and frozen.

## 77 `\boxmaxdepth`

You can limit the depth of boxes being constructed. It's one of these parameters that should be used with care because when that box is filled nested boxes can be influenced.

## 78 `\boxorientation`

The orientation field can take quite some values and is discussed in one of the low level ConT<sub>E</sub>Xt manuals. Some properties are dealt with in the T<sub>E</sub>X engine because they influence dimensions but in the end it is the backend that does the work.

## 79 `\boxrepack`

When a box is too wide or too tight we can tweak it a bit with this primitive. The primitive expects a box register and a dimension, where a positive number adds and a negative subtracts from the current box width.

```
\setbox 0 \hbox {test test test}
\setbox 2 \hbox {\red test test test} \boxrepack0 +.2em
\setbox 4 \hbox {\green test test test} \boxrepack0 -.2em
\ruledhbox{\box0} \vskip-\lineheight
\ruledhbox{\box0} \vskip-\lineheight
\ruledhbox{\box0}
```

test test test

We can also use this primitive to check the natural dimensions of a box:

```
\setbox 0 \hbox spread 10pt {test test test}
\ruledhbox{\box0} (\the\boxrepack0,\the\wd0)
```

In this context only one argument is expected.

test test test

(0.0pt,0.0pt)

## 80 `\boxshift`

Returns or sets how much the box is shifted: up or down in horizontal mode, left or right in vertical mode.

## 81 `\boxshrink`

Returns the amount of shrink found (applied) in a box:

```
\setbox0\hbox to 4em {m m m m}
\the\boxshrink0
```

gives: 3.17871pt

## 82 `\boxsource`

This feature is part of an (experimental) mechanism that relates boxes. The engine just tags a box and it is up to the macro package to deal with it.

```
\setbox0\hbox source 123 {m m m m}
```

**\the\boxsource0**

This gives: 123. Of course this feature is very macro specific and should not be used across macro packages without coordination.

**83 \boxstretch**

Returns the amount of stretch found (applied) in a box:

```
\setbox0\hbox to 6em {m m m m}
\the\boxstretch0
```

gives: 4.76807pt

**84 \boxtarget**

This feature is part of an (experimental) mechanism that relates boxes. The engine just tags a box and it is up to the macro package to deal with it.

```
\setbox0\hbox source 123 {m m m m}
\the\boxsource0
```

This gives: 123. Of course this feature is very macro specific and should not be used across macro packages without coordination.

**85 \boxtotal**

Returns the total of height and depth of the given box.

**86 \boxvadjust**

When used as query this returns a bitset indicating the associated adjust and migration (marks and inserts) data:

```
0x1  pre adjusted
0x2  post adjusted
0x4  pre migrated
0x8  post migrated
```

When used as a setter it directly adds adjust data to the box and it accepts the same keywords as \vadjust.

**87 \boxxmove**

This will set the vertical offset and adapt the dimensions accordingly.

**88 \boxxoffset**

Returns or sets the horizontal offset of the given box.

## 89 `\boxymove`

This will set the vertical offset and adapt the dimensions accordingly.

## 90 `\boxyoffset`

Returns or sets the vertical offset of the given box.

## 91 `\brokenpenalty`

This penalty is added after a line that ends with a hyphen; it can help to discourage a page break (or split in a box).

## 92 `\catcode`

Every character can be put in a category, but this is typically something that the macro package manages because changes can affect behavior. Also, once passed as an argument, the catcode of a character is frozen. There are 16 different values:

<code>\escapecatcode</code>	0	<code>\begingroupcatcode</code>	1
<code>\endgroupcatcode</code>	2	<code>\mathshifcatcode</code>	3
<code>\alignmentcatcode</code>	4	<code>\endoflinecatcode</code>	5
<code>\parametercatcode</code>	6	<code>\superscriptcatcode</code>	7
<code>\subscriptcatcode</code>	8	<code>\ignorecatcode</code>	9
<code>\spacecatcode</code>	10	<code>\lettercatcode</code>	11
<code>\othercatcode</code>	12	<code>\activecatcode</code>	13
<code>\commentcatcode</code>	14	<code>\invalidcatcode</code>	15

The first column shows the constant that ConT<sub>E</sub>Xt provides and the name indicates the purpose. Here are two examples:

```
\catcode123=\begingroupcatcode
\catcode125=\endgroupcatcode
```

## 93 `\catcodetable`

The catcode table with the given index will become active.

## 94 `\cdef`

This primitive is like `\edef` but in some usage scenarios is slightly more efficient because (delayed) expansion is ignored which in turn saves building a temporary token list.

```
\edef\FooA{this is foo} \meaningfull\FooA\cr\lf
\cdef\FooB{this is foo} \meaningfull\FooB\par
```

macro:this is foo

constant macro:this is foo

## 95 `\cdefcsname`

This primitive is like `\edefcsame` but in some usage scenarios is slightly more efficient because (delayed) expansion is ignored which in turn saves building a temporary token list.

```
\edefcsname FooA\endcsname{this is foo} \meaningasis\FooA\crlf
\cdefcsname FooB\endcsname{this is foo} \meaningasis\FooB\par

\def \FooA {this is foo}
\constant \def \FooB {this is foo}
```

## 96 `\cfcode`

This primitive is a companion to `\efcode` and sets the compression factor. It takes three values: font, character code, and factor.

## 97 `\char`

This appends a character with the given index in the current font.

## 98 `\chardef`

The following definition relates a control sequence to a specific character:

```
\chardef\copyrightsign"A9
```

However, because in a context where a number is expected, such a `\chardef` is seen as valid number, there was a time when this primitive was used to define constants without overflowing the by then limited pool of count registers. In  $\varepsilon$ -TeX aware engines this was less needed, and in LuaMetaTeX we have `\integerdef` as a more natural candidate.

## 99 `\cleaders`

See `\gleaders` for an explanation.

## 100 `\clearmarks`

This primitive is an addition to the multiple marks mechanism that originates in  $\varepsilon$ -TeX and reset the mark registers of the given category (a number).

## 101 `\clubpenalties`

This is an array of penalty put before the first lines in a paragraph. High values discourage (or even prevent) a lone line at the end of a page. This command expects a count value indicating the number of entries that will follow. The first entry is ends up after the first line.

## 102 `\clubpenalty`

This is the penalty put before a club line in a paragraph. High values discourage (or even prevent) a lone line at the end of a next page.



### 103 `\constant`

This prefix tags a macro (without arguments) as being constant. The main consequence is that in some cases expansion gets delayed which gives a little performance boost and less (temporary) memory usage, for instance in `\csname` like scenarios.

### 104 `\constrained`

See previous section about `\retained`.

### 105 `\copy`

This is the box register accessor that returns a copy of the box.

### 106 `\copymathatomrule`

This copies the rule bitset from the parent class (second argument) to the target class (first argument). The bitset controls the features that apply to atoms.

### 107 `\copymathparent`

This binds the given class (first argument) to another class (second argument) so that one doesn't need to define all properties.

### 108 `\copymathspacing`

This copies an class spacing specification to another one, so in

`\copymathspacing 34 2`

class 34 (a user one) get the spacing from class 2 (binary).

### 109 `\count`

This accesses a count register by index. This is kind of ‘not done’ unless you do it local and make sure that it doesn't influence macros that you call.

`\count4023=10`

In standard T<sub>E</sub>X the first 10 counters are special because they get reported to the console, and `\count0` is then assumed to be the page counter.

### 110 `\countdef`

This primitive relates a control sequence to a count register. Compare this to the example in the previous section.

`\countdef\MyCounter4023`  
`\MyCounter=10`

However, this is also ‘not done’. Instead one should use the allocator that the macro package provides.

```
\newcount\MyCounter
\MyCounter=10
```

In LuaMetaTeX we also have integers that don't rely on registers. These are assigned by the primitive `\integerdef`:

```
\integerdef\MyCounterA 10
```

Or better `\newinteger`.

```
\newinteger\MyCounterB
\MyCounterN10
```

There is a lowlevel manual on registers.

## 111 `\cr`

This ends a row in an alignment. It also ends an alignment preamble.

## 112 `\crampeddisplaystyle`

A less spacy alternative of `\displaystyle`; integer representation: 4.

## 113 `\crampedscriptscriptstyle`

A less spacy alternative of `\scriptscriptstyle`; integer representation: 6.

## 114 `\crampedscriptstyle`

A less spacy alternative of `\scriptstyle`; integer representation: 4.

## 115 `\crampedtextstyle`

A less spacy alternative of `\textstyle`; integer representation: 2.

## 116 `\crrcr`

This ends a row in an alignment when it hasn't ended yet.

## 117 `\csactive`

Because LuaTeX (and LuaMetaTeX) are Unicode engines active characters are implemented a bit differently. They don't occupy a eight bit range of characters but are stored as control sequence with a special prefix U+FFFF which never shows up in documents. The `\csstring` primitive injects the name of a control sequence without leading escape character, the `\csactive` injects the internal name of the following (either of not active) character. As we cannot display the prefix: `\csactive~` will inject

the utf sequences for U+FFFF and U+007E, so here we get the bytes EFBFBF7E. Basically the next token is preceded by `\string`, so when you don't provide a character you are in for a surprise.

## 118 `\csname`

This original  $\TeX$  primitive starts the construction of a control sequence reference. It does a lookup and when no sequence with than name is found, it will create a hash entry and defaults its meaning to `\relax`.

`\csname` letters and other characters `\endcsname`

## 119 `\csstring`

This primitive returns the name of the control sequence given without the leading escape character (normally a backslash). Of course you could strip that character with a simple helper but this is more natural.

`\csstring\mymacro`

We get the name, not the meaning: `mymacro`.

## 120 `\currentgrouplevel`

The next example gives: [1] [2] [3] [2] [1].

```
[\the\currentgrouplevel] \bgroup
  [\the\currentgrouplevel] \bgroup
    [\the\currentgrouplevel]
      \egroup [\the\currentgrouplevel]
\egroup [\the\currentgrouplevel]
```

## 121 `\currentgrouptype`

The next example gives: [22] [1] [22] [1] [1] [23] [1] [1].

```
[\the\currentgrouptype] \bgroup
  [\the\currentgrouptype] \begingroup
    [\the\currentgrouptype]
      \endgroup [\the\currentgrouptype]
        [\the\currentgrouptype] \beginmathgroup
          [\the\currentgrouptype]
            \endmathgroup [\the\currentgrouptype]
[\the\currentgrouptype] \egroup
```

The possible values depend in the engine and for LuaMeta $\TeX$  they are:

0	bottomlevel	3	adjustedhbox	6	dbbox	9	output
1	simple	4	vbox	7	align	10	mathsubformula
2	hbox	5	vtop	8	noalign	11	mathstack

12	mathcomponent	18	mathoperator	24	mathfence	30	splitkeep
13	discretionary	19	mathradical	25	mathinline	31	preamble
14	insert	20	mathchoice	26	mathdisplay	32	alignset
15	vadjust	21	alsosimple	27	mathnumber	33	finishrow
16	vcenter	22	semisimple	28	localbox	34	lua
17	mathfraction	23	mathsimple	29	splitoff		

## 122 \currentifbranch

The next example gives:  $[0] [1] [-1] [1] [0]$ .

```
[\the\currentifbranch] \iftrue
  [\the\currentifbranch] \iffalse
    [\the\currentifbranch]
  \else
    [\the\currentifbranch]
  \fi [\the\currentifbranch]
\fi [\the\currentifbranch]
```

So when in the ‘then’ branch we get plus one and when in the ‘else’ branch we end up with a minus one.

## 123 \currentiflevel

The next example gives:  $[0] [1][2] [3] [2] [1] [0]$ .

```
[\the\currentiflevel] \iftrue
  [\the\currentiflevel]\iftrue
    [\the\currentiflevel] \iftrue
      [\the\currentiflevel]
    \fi [\the\currentiflevel]
  \fi [\the\currentiflevel]
\fi [\the\currentiflevel]
```

## 124 \currentifttype

The next example gives:  $[-1] [25][25] [25] [25] [25] [-1]$ .

```
[\the\currentifttype] \iftrue
  [\the\currentifttype]\iftrue
    [\the\currentifttype] \iftrue
      [\the\currentifttype]
    \fi [\the\currentifttype]
  \fi [\the\currentifttype]
\fi [\the\currentifttype]
```

The values are engine dependent:

0	char	2	num	4	zeronum	6	float	8	zerofloat
1	cat	3	absnum	5	intervalnum	7	absfloat	9	intervalfloat

10 dim	15 vmode	20 hbox	25 true	30 cmpnum
11 absdim	16 hmode	21 vbox	26 false	31 chkdim
12 zerodim	17 mmode	22 tok	27 chknum	32 chkdimension
13 intervaldim	18 inner	23 cstoken	28 chknumber	33 dimval
14 odd	19 void	24 x	29 numval	34 cmpdim

## 125 \currentloopiterator

Here we show the different expanded loop variants:

```
\edef\testA{\expandedloop 1 10 1{!}}
\edef\testB{\expandedrepeat 10 {!}}
\edef\testC{\expandedendless {\ifnum\currentloopiterator>10 \quitloop\else !\fi}}
\edef\testD{\expandedendless {\ifnum#I>10 \quitloop\else !\fi}}
```

All these give the same result:

```
\def \testA {!!!!!!!!!!!!}
\def \testB {!!!!!!!!!!!!}
\def \testC {!!!!!!!!!!!!}
\def \testD {!!!!!!!!!!!!}
```

The `#I` is a shortcut to the current loop iterator; other shortcuts are `#P` for the parent iterator value and `#G` for the grand parent.

## 126 \currentloopnesting

This integer reports how many nested loops are currently active. Of course in practice the value only has meaning when you know at what outer level your nested loop started.

## 127 \currentmarks

Marks only get updated when a page is split off or part of a box using `\vsplit` gets wrapped up. This primitive gives access to the current value of a mark and takes the number of a mark class.

## 128 \currentstacksize

This is more diagnostic feature than a useful one but we show it anyway. There is some basic overhead when we enter a group:

```
\bgroup [\the\currentstacksize]
  \bgroup [\the\currentstacksize]
    \bgroup [\the\currentstacksize]
      [\the\currentstacksize] \egroup
    [\the\currentstacksize] \egroup
  [\the\currentstacksize] \egroup
```

```
[62] [63] [64] [64] [63] [62]
```

As soon as we define something or change a value, the stack gets populated by information needed for recovery after the group ends.

```
\bgroup [\the\currentstacksize]
  \scratchcounter 1
  \bgroup [\the\currentstacksize]
    \scratchdimen 1pt
    \scratchdimen 2pt
    \bgroup [\the\currentstacksize]
      \scratchcounter 2
      \scratchcounter 3
      [\the\currentstacksize] \egroup
    [\the\currentstacksize] \egroup
  [\the\currentstacksize] \egroup
```

[62] [64] [66] [67] [65] [63]

The stack also keeps some state information, for instance when a box is being built. In LuaMetaT<sub>E</sub>X that is quite a bit more than in other engines but it is compensated by more efficient save stack handling elsewhere.

```
\hbox \bgroup [\the\currentstacksize]
  \hbox \bgroup [\the\currentstacksize]
    \hbox \bgroup [\the\currentstacksize]
      [\the\currentstacksize] \egroup
    [\the\currentstacksize] \egroup
  [\the\currentstacksize] \egroup
```

[70] [79] [88] [88] [79] [70]

## 129 \day

This internal number starts out with the day that the job started.

## 130 \dbox

A `\dbox` is just a `\vbox` (baseline at the bottom) but it has the property ‘dual baseline’ which means that in some cases it will behave like a `\vtop` (baseline at the top) too. Like:

<code>\dbox</code>	<code>\vbox</code>		
<code>\dbox</code>	<code>\vbox</code>		<code>\vcenter</code>
<code>\dbox</code>	<code>\vbox</code>	<code>\vtop</code>	<code>\vcenter</code>
		<code>\vtop</code>	<code>\vcenter</code>
		<code>\vtop</code>	

A `\dbox` behaves like a `\vtop` when it's appended to a vertical list which means that the height of the first box or rule determines the (base)line correction that gets applied.

XXXXXXXXXXXXXXXXXXXX

The Earth, as a habitat for animal life, is in old age and has a fatal illness. Several, in fact. It would be happening whether humans had ever evolved or not. But our presence is like the effect of an old-age patient who smokes many packs of cigarettes per day—and we humans are the cigarettes.

XXXXXXXXXXXXXXXXXXXX

**\vbox**

XXXXXXXXXXXXXXXXXXXX

The Earth, as a habitat for animal life, is in old age and has a fatal illness. Several, in fact. It would be happening whether humans had ever evolved or not. But our presence is like the effect of an old-age patient who smokes many packs of cigarettes per day—and we humans are the cigarettes.

XXXXXXXXXXXXXXXXXXXX

**\vtop**

XXXXXXXXXXXXXXXXXXXX

The Earth, as a habitat for animal life, is in old age and has a fatal illness. Several, in fact. It would be happening whether humans had ever evolved or not. But our presence is like the effect of an old-age patient who smokes many packs of cigarettes per day—and we humans are the cigarettes.

XXXXXXXXXXXXXXXXXXXX

**\dbox**

## 131 \deadcycles

This counter is incremented every time the output routine is entered. When `\maxdeadcycles` is reached  $\TeX$  will issue an error message, so you'd better reset its value when a page is done.

## 132 \def

This is the main definition command, as in:

```
\def\foo{l me}
```

with companions like `\gdef`, `\edef`, `\xdef`, etc. and variants like:

```
\def\foo#1{... #1...}
```

where the hash is used in the preamble and for referencing. More about that can be found in the low level manual about macros.

## 133 \defaultthyphenchar

When a font is loaded its hyphen character is set to this value. It can be changed afterwards. However, in  $\text{LuaMeta}\TeX$  font loading is under Lua control so these properties can be set otherwise.

## 134 \defaultskewchar

When a font is loaded its skew character is set to this value. It can be changed afterwards. However, in  $\text{LuaMeta}\TeX$  font loading is under Lua control so these properties can be set otherwise. Also, OpenType math fonts have top anchor instead.

## 135 \defcsname

We now get a series of log clutter avoidance primitives. It's fine if you argue that they are not really needed, just don't use them.

```
\expandafter\def\csname MyMacro:1\endcsname{...}
\defcsname MyMacro:1\endcsname{...}
```

The fact that  $\TeX$  has three (expanded and global) companions can be seen as a signal that less verbosity makes sense. It's just that macro packages use plenty of `\csname`'s.

## 136 \deferred

This is mostly a compatibility prefix and it can be checked at the Lua end when there is a Lua based assignment going on. It is the counterpart of `\immediate`. In the traditional engines a `\write` is



normally deferred (turned into a node) and can be handled `\immediate`, while a `\special` does the opposite.

### 137 `\delcode`

This assigns delimiter properties to an eight bit character so it has little use in an OpenType math setup. When the assigned value is hex encoded, the first byte denotes the small family, then we have two bytes for the small index, followed by three similar bytes for the large variant.

### 138 `\delimiter`

This command inserts a delimiter with the given specification. In OpenType math we use a different command so it is unlikely that this primitive is used in LuaMetaTeX. It takes a number that can best be coded hexadecimal: one byte for the class, one for the small family, two for the small index, one for the large family and two for the large index. This demonstrates that it can't handle wide fonts. Also, in OpenType math fonts the larger sizes and extensible come from the same font as the small symbol. On top of that, in LuaMetaTeX we have more classes than fit in a byte.

### 139 `\delimiterfactor`

This is one of the parameters that determines the size of a delimiter: at least this factor times the formula height divided by 1000. In OpenType math different properties and strategies are used.

### 140 `\delimitershortfall`

This is one of the parameters that determines the size of a delimiter: at least the formula height minus this parameter. In OpenType math different properties and strategies are used.

### 141 `\detokened`

The following token will be serialized into characters with category 'other'.

```
\toks0{123}
\def\foo{let's be \relax'd}
\def\oof#1{let's see #1}
\detokened\toks0
\detokened\foo
\detokened\oof
\detokened\setbox
\detokened X
```

Gives:

```
123
let's be \relax 'd
\oof
\setbox
X
```

Macros with arguments are not shown.

## 142 `\detokenize`

This  $\varepsilon$ -TeX primitive turns the content of the provides list will become characters, kind of verbatim.

```
\expandafter\let\expandafter\temp\detokenize{1} \meaning\temp
\expandafter\let\expandafter\temp\detokenize{A} \meaning\temp
```

the character U+0031 1

the character U+0041 A

## 143 `\detokenized`

The following (single) token will be serialized into characters with category ‘other’.

```
\toks0{123}
\def\foo{let's be \relax'd}
\def\oof#1{let's see #1}
\detokenized\toks0
\detokenized\foo
\detokenized\oof
\detokenized\setbox
\detokenized X
```

Gives:

```
\toks 0
\foo
\oof
\setbox
X
```

It is one of these new primitives that complement others like `\detokened` and such, and they are often mostly useful in experiments of some low level magic, which made them stay.

## 144 `\dimen`

Like `\count` this is a register accessor which is described in more detail in a low level manual.

```
\dimen0=10pt
```

While TeX has some assumptions with respect to the first ten count registers (as well as the one that holds the output, normally 255), all dimension registers are treated equal. However, you need to be aware of clashes with other usage. Therefore you can best use the predefined scratch registers or define dedicate ones with the `\newdimen` macro.

## 145 `\dimendef`

This primitive is used by the `\newdimen` macro when it relates a control sequence with a specific register. Only use it when you know what you're doing.

## 146 `\dimensiondef`

A variant of `\integerdef` is:

```
\dimensiondef\MyDimen = 1234pt
```

The properties are comparable to the ones described in the section `\integerdef`.

## 147 `\dimexpr`

This primitive is similar to `\numexpr` but operates on dimensions instead. Integer quantities are interpreted as dimensions in scaled points.

```
\the\dimexpr (1pt + 2pt - 5pt) * 10 / 2 \relax
```

gives: -10.0pt. You can mix in symbolic integers and dimensions. This doesn't work:

because the engine scans for a dimension and only for an integer (or equivalent) after a `*` or `/`.

## 148 `\dimexpression`

This command is like `\numexpression` but results in a dimension instead of an integer. Where `\dimexpr` doesn't like `2 * 10pt` this expression primitive is quite happy with it.

## 149 `\directlua`

This is the low level interface to Lua:

Gives: "Greetings from the lua end!" as expected. In Lua we have access to all kind of internals of the engine. In LuaMetaTeX the interfaces have been polished and extended compared to LuaTeX. Although many primitives and mechanisms were added to the TeX frontend, the main extension interface remains Lua. More information can be found in documents that come with ConTeXt, in presentations and in articles.

## 150 `\discretionary`

The three snippets given with this command determine the pre, post and replace component of the injected discretionary node. The `penalty` keyword permits setting a penalty with this node. The `postword` keyword indicates that this discretionary starts a word, and `preword` ends it. With `break` the line break algorithm will prefer a pre or post component over a replace, and with `nobreak` replace will win over pre. With `class` you can set a math class that will determine spacing and such for discretions used in math mode.

## 151 `\discretionaryoptions`

Processing of discretions is controlled by this bitset:

```
0x00000000 normalword
0x00000001 preword
0x00000002 postword
```

```

0x00000010 preferbreak
0x00000020 prefernobreak
0x00000040 noitaliccorrection
0x00000080 nozeroitaliccorrection
0x00010000 userfirst
0x40000000 userlast

```

These can also be set on `\discretionary` using the options key.

## 152 `\displayindent`

The `\displaywidth`, `\displayindent` and `\predisplaysize` parameters are set by the line break routine (but can be adapted by the user), so that mid-par display formula can adapt itself to hanging indentation and par shapes. In order to calculate these values and adapt the line break state afterwards such a display formula is assumed to occupy three lines, so basically a rather compact formula.

## 153 `\displaylimits`

By default in math display mode limits are placed on top while in inline mode they are placed like scripts, after the operator. Placement can be forced with the `\limits` and `\nolimits` modifiers (after the operator). Because there can be multiple of these in a row there is `\displaylimits` that forces the default placement, so effectively it acts here as a reset modifier.

## 154 `\displaystyle`

One of the main math styles; integer representation: 0.

## 155 `\displaywidowpenalties`

This is a math specific variant of `\widowpenalties`.

## 156 `\displaywidowpenalty`

This is a math specific variant of `\widowpenalty`.

## 157 `\displaywidth`

This parameter determines the width of the formula and normally defaults to the `\hsize` unless we are in the middle of a paragraph in which case it is compensated for hanging indentation or the par shape.

## 158 `\divide`

The `\divide` operation can be applied to integers, dimensions, float, attribute and glue quantities. There are subtle rounding differences between the divisions in expressions and `\divide`:

```

\scratchcounter 1049 \numexpr\scratchcounter / 10\relax : 105
\scratchcounter 1049 \numexpr\scratchcounter : 10\relax : 104
\scratchcounter 1049 \divide\scratchcounter by 10      : 104

```

The `:` divider in `\dimexpr` is something that we introduced in LuaTeX.

## 159 `\divideby`

This is slightly more efficient variant of `\divide` that doesn't look for `by`. See previous section.

## 160 `\doubleadjdemerits`

This penalty will be added to the penalty assigned to a breakpoint that results in two incompatible lines (in LuaMetaTeX we can be more granular with respect to compatible lines and this concerns a larger delta).

## 161 `\doublehyphendemerits`

This penalty will be added to the penalty assigned to a breakpoint that results in two lines ending with a hyphen.

## 162 `\dp`

Returns the depth of the given box.

## 163 `\dpack`

This does what `\dbox` does but without callback overhead.

## 164 `\dsplit`

This is the dual baseline variant of `\vsplit` (see `\dbox` for what that means).

## 165 `\dump`

This finishes an (ini) run and dumps a format (basically the current state of the engine).

## 166 `\edef`

This is the expanded version of `\def`.

```
\def \foo{foo}          \meaning\foo
\def \of{\foo\foo}      \meaning\of
\edef\oof{\foo\foo}     \meaning\oof
```

Because `\foo` is unprotected it will expand inside the body definition:

```
macro:foo
macro:\foo \foo
macro:foofoo
```

## 167 \edefcsname

This is the companion of \edef:

```
\expandafter\edef\csname MyMacro:1\endcsname{...}
      \edefcsname MyMacro:1\endcsname{...}
```

## 168 \edivide

When expressions were introduced the decision was made to round the divisions which is incompatible with the way \divide works. The expression scanners in LuaMetaT<sub>E</sub>X compensates that by providing a : for integer division. The \edivide does the opposite: it rounds the way expressions do.

```
\the\dimexpr .4999pt : 2 \relax =.24994pt
\the\dimexpr .4999pt / 2 \relax =.24995pt
\scratchdimen.4999pt \divide \scratchdimen 2 \the\scratchdimen=.24994pt
\scratchdimen.4999pt \edivide\scratchdimen 2 \the\scratchdimen=.24995pt

\the\numexpr 1001 : 2 \relax =500
\the\numexpr 1001 / 2 \relax =501
\scratchcounter1001 \divide \scratchcounter 2 \the\scratchcounter=500
\scratchcounter1001 \edivide\scratchcounter 2 \the\scratchcounter=501
```

Keep in mind that with dimensions we have a fractional part so we actually rounding applies to the fraction. For that reason we also provide \rdivide.

```
0.24994pt=.24994pt
0.24995pt=.24995pt
0.24994pt=.24994pt
0.24995pt=.24995pt
```

```
500=500
501=501
500=500
501=501
```

## 169 \edivideby

This the by-less variant of \edivide.

## 170 \efcode

This primitive originates in pdfT<sub>E</sub>X and can be used to set the expansion factor of a glyph (characters). This primitive is obsolete because the values can be set in the font specification that gets passed via Lua to T<sub>E</sub>X. Keep in mind that setting font properties at the T<sub>E</sub>X end is a global operation and can therefore influence related fonts. In LuaMetaT<sub>E</sub>X the \cf code can be used to specify the compression factor independent from the expansion factor. The primitive takes three values: font, character code, and factor.

## 171 `\else`

This traditional primitive is part of the condition testing mechanism. When a condition matches,  $\text{\TeX}$  will continue till it sees an `\else` or `\or` or `\orelse` (to be discussed later). It will then do a fast skipping pass till it sees an `\fi`.

## 172 `\emergencyextrastretch`

This is one of the extended parbuilder parameters. You can use it so temporarily increase the permitted stretch without knowing or messing with the normal value.

## 173 `\emergencyleftskip`

This is one of the extended parbuilder parameters (playground). It permits going ragged left in case of a too bad result.

## 174 `\emergencyrightskip`

This is one of the extended parbuilder parameters (playground). It permits going ragged right in case of a too bad result.

## 175 `\emergencystretch`

When set the par builder will run a third pass in order to fit the set criteria.

## 176 `\end`

This ends a  $\text{\TeX}$  run, unless of course this primitive is redefined.

## 177 `\endcsname`

This primitive is used in combination with `\csname`, `\ifcsname` and `\beginscname` where it ends the scanning for the to be constructed control sequence token.

## 178 `\endgroup`

This is the companion of the `\begingroup` primitive that opens a group. See `\beginsimplegroup` for more info.

## 179 `\endinput`

The engine can be in different input modes: reading from file, reading from a token list, expanding a macro, processing something that comes back from Lua, etc. This primitive quits reading from file:

```
this is seen
\endinput
here we're already quit
```

There is a catch. This is what the above gives:

this is seen

but how about this:

this is seen  
before `\endinput` after  
here we're already quit

Here we get:

this is seen before after

Because a token list is one line, the following works okay:

```
\def\quitrun{\ifsomething \endinput \fi}
```

but in a file you'd have to do this when you quit in a conditional:

```
\ifsomething
  \expandafter \endinput
\fi
```

While the one-liner works as expected:

```
\ifsomething \endinput \fi
```

## 180 `\endlinechar`

This is an internal integer register. When set to positive value the character with that code point will be appended to the line. The current value is 13. Here is an example:

```
\endlinechar\hyphenasciicode
line 1
line 2
```

line 1-line 2-

If the character is active, the property is honored and the command kicks in. The maximum value is 127 (the maximum character code a single byte utf character can carry.)

## 181 `\endlocalcontrol`

See `\beginlocalcontrol`.

## 182 `\endmathgroup`

This primitive is the counterpart of `\beginmathgroup`.

## 183 `\endsimplegroup`

This one ends a simple group, see `\beginsimplegroup` for an explanation about grouping primitives.



## 184 `\enforced`

The engine can be set up to prevent overloading of primitives and macros defined as `\permanent` or `\immutable`. However, a macro package might want to get around this in controlled situations, which is why we have a `\enforced` prefix. This prefix is interpreted differently in so called ‘ini’ mode when macro definitions can be dumped in the format. Internally they get an `always` flag as indicator that in these places an overload is possible.

```
\permanent\def\foo{original}

\def\oof          {\def\foo{fails}}
\def\oof{\enforced\def\foo{succeeds}}
```

Of course this only has an effect when overload protection is enabled.

## 185 `\eofinput`

This is a variant on `\input` that takes a token list as first argument. That list is expanded when the file ends. It has companion primitives `\atendoffile` (single token) and `\atendoffiled` (multiple tokens).

## 186 `\eqno`

This primitive stores the (typeset) content (presumably a number) and when the display formula is wrapped that number will end up right of the formula.

## 187 `\errhelp`

This is additional help information to `\errmessage` that triggers an error and shows a message.

## 188 `\errmessage`

This primitive expects a token list and shows its expansion on the console and/or in the log file, depending on how  $\text{\TeX}$  is configured. After that it will enter the error state and either goes on or waits for input, again depending on how  $\text{\TeX}$  is configured. For the record: we don't use this primitive in  $\text{\ConTeXt}$ .

## 189 `\errorcontextlines`

This parameter determines the number on lines shown when an error is triggered.

## 190 `\errorstopmode`

This directive stops at every opportunity to interact. In  $\text{\ConTeXt}$  we overload the actions in a callback and quit the run because we can assume that a successful outcome is unlikely.

## 191 `\escapechar`

This internal integer has the code point of the character that get prepended to a control sequence when it is serialized (for instance in tracing or messages).

## 192 \etoks

This assigns an expanded token list to a token register:

```
\def\temp{less stuff}
\etoks\scratchtoks{a bit \temp}
```

The original value of the register is lost.

## 193 \etoksapp

A variant of \toksapp is the following: it expands the to be appended content.

```
\def\temp{more stuff}
\etoksapp\scratchtoks{some \temp}
```

## 194 \etokspre

A variant of \tokspre is the following: it expands the to be prepended content.

```
\def\temp{less stuff}
\etokspre\scratchtoks{a bit \temp}
```

## 195 \eufactor

When we introduced the es (2.5cm) and ts (2.5mm) units as metric variants of the in we also added the eu factor. One eu equals one tenth of a es times the \eufactor. The ts is a convenient offset in test files, the es a convenient ones for layouts and image dimensions and the eu permits definitions that scale nicely without the need for dimensions. They also were a prelude to what later became possible with \associateunit.

## 196 \everybeforepar

This token register is expanded before a paragraph is triggered. The reason for triggering is available in \lastpartrigger.

## 197 \everycr

This token list gets expanded when a row ends in an alignment. Normally it will use \noalign as wrapper

```
{\everycr{\noalign{H}} \halign{#\cr test\cr test\cr}}
{\everycr{\noalign{V}} \hsize 4cm \valign{#\cr test\cr test\cr}}
```

Watch how the \cr ending the preamble also get this treatment:

H  
test

H  
test

H

Vtest

Vtest

V

**198 `\everydisplay`**

This token list gets expanded every time we enter display mode. It is a companion of `\everymath`.

**199 `\everyeof`**

The content of this token list is injected when a file ends but it can only be used reliably when one is really sure that no other file is loaded in the process. So in the end it is of no real use in a more complex macro package.

**200 `\everyhbox`**

This token list behaves similar to `\everyvbox` so look there for an explanation.

**201 `\everyjob`**

This token list register is injected at the start of a job, or more precisely, just before the main control loop starts.

**202 `\everymath`**

Often math needs to be set up independent from the running text and this token list can be used to do that. There is also `\everydisplay`.

**203 `\everymathatom`**

When a math atom is seen this tokenlist is expanded before content is processed inside the atom body.

**204 `\everypar`**

When a paragraph starts this tokenlist is expanded before content is processed.

**205 `\everytab`**

This token list gets expanded every time we start a table cell in `\halign` or `\valign`.

**206 `\everyvbox`**

This token list gets expanded every time we start a vertical box. Like `\everyhbox` this is not that useful unless you are certain that there are no nested boxes that don't need this treatment. Of course you can wipe this register in this expansion, like:

```
\everyvbox{\kern10pt\everyvbox{}}
```

## 207 `\exceptionpenalty`

In exceptions we can indicate a penalty by `[digit]` in which case a penalty is injected set by this primitive, multiplied by the digit.

## 208 `\exhyphenchar`

The character that is used as pre component of the related discretionary.

## 209 `\exhyphenpenalty`

The penalty injected after `-` or `\-` unless `\hyphenationmode` is set to force the dedisated penalties.

## 210 `\expand`

Beware, this is not a prefix but a directive to ignore the protected characters of the following macro.

```
\protected \def \testa{\the\scratchcounter}
      \edef\testb{\testa}
      \edef\testc{\expand\testa}
```

The meaning of the three macros is:

```
protected macro:\the \scratchcounter
macro:\testa
macro:123
```

## 211 `\expandactive`

This a bit of an outlier and mostly there for completeness.

```

\meaningasis~
\edef\foo{~} \meaningasis\foo
\edef\foo{\expandactive~} \meaningasis\foo
```

There seems to be no difference but the real meaning of the first `\foo` is ‘active character 126’ while the second `\foo` ‘protected call ’ is.

```
\protected \def ~ {\nobreakspace }
\def \foo {~}
\def \foo {~}
```

Of course the definition of the active tilde is ConT<sub>E</sub>Xt specific and situation dependent.

## 212 `\expandafter`

This original T<sub>E</sub>X primitive stores the next token, does a one level expansion of what follows it, which actually can be an not expandable token, and reinjects the stored token in the input. Like:

```
\expandafter\let\csname my weird macro name\endcsname{m w m n}
```

Without `\expandafter` the `\csname` primitive would have been let to the left brace (effectively then a begin group). Actually in this particular case the control sequence with the weird name is injected and when it didn't yet exist it will get the meaning `\relax` so we sort of have two assignments in a row then.

## 213 `\expandafterpars`

Here is another gobbler: the next token is reinjected after following spaces and par tokens have been read. So:

```
[\expandafterpars 1 2]
[\expandafterpars 3
4]
[\expandafterpars 5
6]
```

gives us: [12] [34] [56], because empty lines are like `\par` and therefore ignored.

## 214 `\expandafterspaces`

This is a gobbler: the next token is reinjected after following spaces have been read. Here is a simple example:

```
[\expandafterspaces 1 2]
[\expandafterspaces 3
4]
[\expandafterspaces 5
6]
```

We get this typeset: [12] [34] [5

6], because a newline normally is configured to be a space (and leading spaces in a line are normally being ignored anyway).

## 215 `\expandcstoken`

The rationale behind this primitive is that when we `\let` a single token like a character it is hard to compare that with something similar, stored in a macro. This primitive pushes back a single token alias created by `\let` into the input.

```
\let\tempA + \meaning\tempA
```

```
\let\tempB X \meaning\tempB \crlf
```

```
\let\tempC $ \meaning\tempC \par
```

```
\edef\temp          {\tempA} \doifelse{\temp}{+}{Y}{N} \meaning\temp \crlf
```

```
\edef\temp          {\tempB} \doifelse{\temp}{X}{Y}{N} \meaning\temp \crlf
```

```
\edef\temp          {\tempC} \doifelse{\temp}{X}{Y}{N} \meaning\temp \par
```

```
\edef\temp{\expandcstoken\tempA} \doifelse{\temp}{+}{Y}{N} \meaning\temp \crlf
```

```

\edef\temp{\expandcstoken\tempB} \doifelse{\temp}{X}{Y}{N} \meaning\temp \crlf
\edef\temp{\expandcstoken\tempC} \doifelse{\temp}{$}{Y}{N} \meaning\temp \par

\doifelse{\expandcstoken\tempA}{+}{Y}{N}
\doifelse{\expandcstoken\tempB}{X}{Y}{N}
\doifelse{\expandcstoken\tempC}{$}{Y}{N} \par

```

The meaning of the `\let` macros shows that we have a shortcut to a character with (in this case) catcode letter, other (here ‘other character’ gets abbreviated to ‘character’), math shift etc.

the character U+002B ‘plus sign’

the letter U+0058 X

math shift character U+0024 ‘dollar sign’

N macro:\tempA

N macro:\tempB

N macro:\tempC

Y macro:+

Y macro:X

Y macro:\$

Y Y Y

Here we use the ConT<sub>E</sub>Xt macro `\doifelse` which can be implemented in different ways, but the only property relevant to the user is that the expanded content of the two arguments is compared.

## 216 \expanded

This primitive complements the two expansion related primitives mentioned in the previous two sections. This time the content will be expanded and then pushed back into the input. Protected macros will not be expanded, so you can use this primitive to expand the arguments in a call. In ConT<sub>E</sub>Xt you need to use `\normalexpanded` because we already had a macro with that name. We give some examples:

```

\def\A{!}
      \def\B#1{\string#1}
      \def\B#1{\string#1} \normalexpanded{\noexpand\B{\A}}
\protected\def\B#1{\string#1}
\B{\A}

\A
!
\A

```

## 217 \expandedafter

The following two lines are equivalent:

```

\def\foo{123}
\expandafter[\expandafter[\expandafter\secondofthreearguments\foo]]
\expandedafter{[[\secondofthreearguments]\foo]]

```

In ConT<sub>E</sub>Xt MkIV the number of times that one has multiple `\expandafters` is much larger than in ConT<sub>E</sub>Xt LMTX thanks to some of the new features in LuaMetaT<sub>E</sub>X, and this primitive is not really used yet in the core code.

```
[[2]]
```

```
[[2]]
```

## 218 `\expandeddetokenize`

This is a companion to `\detokenize` that expands its argument:

```
\def\foo{12#H3}
\def\oof{\foo}
\detokenize      {\foo} \detokenize      {\oof}
\expandeddetokenize{\foo} \expandeddetokenize{\oof}
\edef\of{\expandeddetokenize{\foo}} \meaningless\of
\edef\of{\expandeddetokenize{\oof}} \meaningless\of
```

This is a bit more convenient than

```
\detokenize \expandafter {\normalexpanded {\foo}}
```

kind of solutions. We get:

```
\foo \oof
12#3 12#3
12#3
12#3
```

## 219 `\expandedendless`

This one loops forever but because the loop counter is not set you need to find a way to quit it.

## 220 `\expandedloop`

This variant of the previously introduced `\localcontrolledloop` doesn't enter a local branch but immediately does its work. This means that it can be used inside an expansion context like `\edef`.

```
\edef\whatever
  {\expandedloop 1 10 1
   {\scratchcounter=\the\currentloopiterator\relax}}
```

```
\meaningasis\whatever
```

```
\def \whatever {\scratchcounter =1\relax \scratchcounter =2\relax \scratchcounter =3\relax \scratchcounter
=4\relax \scratchcounter =5\relax \scratchcounter =6\relax \scratchcounter =7\relax \scratchcounter =8\relax
\scratchcounter =9\relax \scratchcounter =10\relax }
```

## 221 `\expandedrepeat`

This one takes one instead of three arguments which is sometimes more convenient.

## 222 \expandparameter

This primitive is a predecessor of \parameterdef so we stick to a simple example.

```
\def\foo#1#2%
  {\integerdef\MyIndexOne\parameterindex\plusone % 1
   \integerdef\MyIndexTwo\parameterindex\plustwo % 2
   \oof{P}\oof{Q}\oof{R}\norelax}

\def\oof#1%
  {<1:\expandparameter\MyIndexOne><1:\expandparameter\MyIndexOne>%
   #1%
   <2:\expandparameter\MyIndexTwo><2:\expandparameter\MyIndexTwo>}}

\foo{A}{B}
```

In principle the whole parameter stack can be accessed but often one never knows if a specific macro is called nested. The original idea behind this primitive was tracing but it can also be used to avoid passing parameters along a chain of calls.

```
<1:A><1:A>P<2:B><2:B><1:A><1:A>Q<2:B><2:B><1:A><1:A>R<2:B><2:B>
```

## 223 \expandtoken

This primitive creates a token with a specific combination of catcode and character code. Because it assumes some knowledge of T<sub>E</sub>X we can show it using some \expandafter magic:

```
\expandafter\let\expandafter\temp\expandtoken 11 `X \meaning\temp
\expandafter\let\expandafter\temp\expandtoken 12 `X \meaning\temp
```

The meanings are:

```
the letter U+0058 X
the character U+0058 X
```

Using other catcodes is possible but the results of injecting them into the input directly (or here by injecting \temp) can be unexpected because of what T<sub>E</sub>X expects. You can get messages you normally won't get, for instance about unexpected alignment interference, which is a side effect of T<sub>E</sub>X using some catcode/character combinations as signals and there is no reason to change those internals. That said:

```
\xdef\tempA{\expandtoken 9 `X} \meaning\tempA
\xdef\tempB{\expandtoken 10 `X} \meaning\tempB
\xdef\tempC{\expandtoken 11 `X} \meaning\tempC
\xdef\tempD{\expandtoken 12 `X} \meaning\tempD
```

are all valid and from the meaning you cannot really deduce what's in there:

```
macro:X
macro:X
macro:X
macro:X
```



But you can be assured that:

```
[AB: \ifx\tempA\tempB Y\else N\fi]
[AC: \ifx\tempA\tempC Y\else N\fi]
[AD: \ifx\tempA\tempD Y\else N\fi]
[BC: \ifx\tempB\tempC Y\else N\fi]
[BD: \ifx\tempB\tempD Y\else N\fi]
[CD: \ifx\tempC\tempD Y\else N\fi]
```

makes clear that they're different: [AB: N] [AC: N] [AD: N] [BC: N] [BD: N] [CD: N], and in case you wonder, the characters with catcode 10 are spaces, while those with code 9 are ignored.

## 224 \expandtoks

This is a more efficient equivalent of \the applied to a token register, so:

```
\scratchtoks{just some tokens}
\edef\TestA{[\the \scratchtoks]}
\edef\TestB{[\expandtoks\scratchtoks]}
[\the \scratchtoks] [\TestA] \meaning\TestA
[\expandtoks\scratchtoks] [\TestB] \meaning\TestB
```

does the expected:

```
[just some tokens] [[just some tokens]] macro:[just some tokens]
[just some tokens] [[just some tokens]] macro:[just some tokens]
```

The \expandtoken primitive avoid a copy into the input when there is no need for it.

## 225 \explicitdiscretionary

This is the verbose alias for one of T<sub>E</sub>X's single character control sequences: \-. .

## 226 \explicitthyphenpenalty

The penalty injected after an automatic discretionary \-, when \hyphenationmode enables this.

## 227 \explicititaliccorrection

This is the verbose alias for one of T<sub>E</sub>X's single character control sequences: \/. Italic correction is a character property specific to T<sub>E</sub>X and the concept is not present in modern font technologies. There is a callback that hooks into this command so that a macro package can provide its own solution to this (or alternatively it can assign values to the italic correction field).

## 228 \explicitspace

This is the verbose alias for one of T<sub>E</sub>X's single character control sequences: \ . A space is inserted with properties according the space related variables. There is look-back involved in order to deal with space factors.

When `\nospaces` is set to 1 no spaces are inserted, when its value is 2 a zero space is inserted.

## 229 `\fam`

In a numeric context it returns the current family number, otherwise it sets the given family. The number of families in a traditional engine is 16, in LuaTeX it is 256 and in LuaMetaTeX we have at most 64 families. A future version can lower that number when we need more classes.

## 230 `\fi`

This traditional primitive is part of the condition testing mechanism and ends a test. So, we have:

```
\ifsomething ... \else ... \fi
\ifsomething ... \or ... \or ... \else ... \fi
\ifsomething ... \orelse \ifsomething ... \else ... \fi
\ifsomething ... \or ... \orelse \ifsomething ... \else ... \fi
```

The `\orelse` is new in LuaMetaTeX and a continuation like we find in other programming languages (see later section).

## 231 `\finalhyphendemerits`

This penalty will be added to the penalty assigned to a breakpoint when that break results in a pre-last line ending with a hyphen.

## 232 `\firstmark`

This is a reference to the first mark on the (split off) page, it gives back tokens.

## 233 `\firstmarks`

This is a reference to the first mark with the given id (a number) on the (split off) page, it gives back tokens.

## 234 `\firstvalidlanguage`

Language id's start at zero, which makes it the first valid language. You can set this parameter to indicate the first language id that is actually a language. The current value is 1, so lower values will not trigger hyphenation.

## 235 `\float`

In addition to integers and dimensions, which are fixed 16.16 integer floats we also have ‘native’ floats, based on 32 bit posit unums.

```
\float0 = 123.456           \the\float0
\float2 = 123.456           \the\float0
\advance \float0 by 123.456 \the\float0
\advance \float0 by \float2 \the\float0
```

```
\divideby\float0 3          \the\float0
```

They come with the same kind of support as the other numeric data types:

```
123.45600032806396484
123.45600032806396484
246.91200065612792969
370.36800384521484375
123.45600128173828125
```

We leave the subtle differences between floats and dimensions to the user to investigate:

```
\dimen00 = 123.456pt          \the\dimen0
\dimen02 = 123.456pt          \the\dimen0
\advance \dimen0 by 123.456pt \the\dimen0
\advance \dimen0 by \dimen2    \the\dimen0
\divideby\dimen0 3            \the\dimen0
```

The nature of posits is that they are more accurate around zero (or smaller numbers in general).

```
123.456pt
123.456pt
246.91199pt
370.36798pt
123.456pt
```

This also works:

```
\float0=123.456e4
\float2=123.456    \multiply\float2 by 10000
\the\float0
\the\float2
```

The values are (as expected) the same:

```
1234560
1234560
```

## 236 \floatdef

This primitive defines a symbolic (macro) alias to a float register, just like \countdef and friends do.

## 237 \floatexpr

This is the companion of \numexpr, \dimexpr etc.

```
\scratchcounter 200
\the \floatexpr 123.456/456.123    \relax
\the \floatexpr 1.2*\scratchcounter \relax
\the \floatexpr \scratchcounter/3   \relax
\number\floatexpr \scratchcounter/3 \relax
```

Watch the difference between `\the` and `\number`:

```
0.27066383324563503265
240
66.666666984558105469
67
```

## 238 `\floatingpenalty`

When an insertion is split (across pages) this one is added to to accumulated `\insertpenalties`. In LuaMetaTeX this penalty can be stored per insertion class.

## 239 `\flushmarks`

This primitive is an addition to the multiple marks mechanism that originates in  $\epsilon$ -TeX and inserts a reset signal for the mark given category that will perform a clear operation (like `\clearmarks` which operates immediately).

## 240 `\font`

This primitive is either a symbolic reference to the current font or in the perspective of an assignment is used to trigger a font definitions with a given name (cs) and specification. In LuaMetaTeX the assignment will trigger a callback that then handles the definition; in addition to the filename an optional size specifier is checked (at or scaled).

In LuaMetaTeX *all* font loading is delegated to Lua, and there is no loading code built in the engine. Also, instead of `\font` in ConTeXt one uses dedicated and more advanced font definition commands.

## 241 `\fontcharba`

Fetches the bottom anchor of a character in the given font, so:

results in: 4.8025pt. However, this anchor is only available when it is set and it is not part of OpenType; it is something that ConTeXt provides for math fonts.

## 242 `\fontchardp`

Fetches the depth of a character in the given font, so:

results in: 2.22168pt.

## 243 `\fontcharht`

Fetches the width of a character in the given font, so:

results in: 5.33203pt.

## 244 `\fontcharic`

Fetches the italic correction of a character in the given font, but because it is not an OpenType property it is unlikely to return something useful. Although math fonts have such a property in ConT<sub>E</sub>Xt we deal with it differently.

## 245 `\fontcharta`

Fetches the top anchor of a character in the given font, so:

results in: 4.8025pt. This is a specific property of math characters because in text mark anchoring is driven by a feature.

## 246 `\fontcharwd`

Fetches the width of a character in the given font, so:

results in: 6.40137pt.

## 247 `\fontdimen`

A traditional T<sub>E</sub>X font has a couple of font specific dimensions, we only mention the seven that come with text fonts:

1. The slant (slope) is an indication that we have an italic shape. The value divided by 65.536 is a fraction that can be compared with for instance the `slanted` operator in MetaPost. It is used for positioning accents, so actually not limited to oblique fonts (just like italic correction can be a property of any character). It is not relevant in the perspective of OpenType fonts where we have glyph specific top and bottom anchors.
2. Unless is it overloaded by `\spaceskip` this determines the space between words (or actually anything separated by a space).
3. This is the stretch component of `\fontdimen 2(space)`.
4. This is the shrink component of `\fontdimen 2(space)`.
5. The so called ex-height is normally the height of the ‘x’ and is also accessible as em unit.
6. The so called em-width or in T<sub>E</sub>X speak quad width is about the with of an ‘M’ but in many fonts just matches the font size. It is also accessible as em unit.
7. This is a very T<sub>E</sub>X specific property also known as extra space. It gets *added* to the regular space after punctuation when `\spacefactor` is 2000 or more. It can be overloaded by `\xspaceskip`.

This primitive expects a a number and a font identifier. Setting a font dimension is a global operation as it directly pushes the value in the font resource.

## 248 `\fontid`

Returns the (internal) number associated with the given font:

```
{\bf \xdef\MyFontA{\the\fontid\font}}
{\sl \xdef\MyFontB{\setfontid\the\fontid\font}}
```

with:

```
test {\setfontid\MyFontA test} test {\MyFontB test} test
```

gives: test **test** test *test* test.

## 249 \fontmathcontrol

The `\fontmathcontrol` parameter controls how the engine deals with specific font related properties and possibilities. It is set at the  $\TeX$  end. It makes it possible to fine tune behavior in this mixed traditional and not perfect OpenType math font arena. One can also set this bitset when initializing (loading) the font (at the Lua end) and the value set there is available in `\fontmathcontrol`. The bits set in the font win over those in `\fontmathcontrol`. There are a few cases where we set these options in the (so called) goodie files. For instance we ignore font kerns in Libertinus, Antykwa and some more.

```
modern          0x0
pagella         0x0
antykwa        0x37EF3FF
libertinus     0x37EF3FF
```

## 250 \fontname

Depending on how the font subsystem is implemented this gives some information about the used font:

```
{\tf \fontname\font}
{\bf \fontname\font}
{\sl \fontname\font}
```

DejaVuSerif at 10.0pt

**DejaVuSerif-Bold at 10.0pt**

*DejaVuSerif-Italic at 10.0pt*

## 251 \fontspecdef

This primitive creates a reference to a specification that when triggered will change multiple parameters in one go.

```
\fontspecdef\MyFontSpec
  \fontid\font
  scale 1200
  xscale 1100
  yscale 800
  weight 200
  slant 500
\relax
```

is equivalent to:

```
\fontspecdef\MyFontSpec
  \fontid\font
  all 1200 1100 800 200 500
\relax
```

while

```
\fontspecdef\MyFontSpec
  \fontid\font
  all \glyphscale \glyphxscale \glyphyscale \glyphslant \glyphweight
\relax
```

is the same as

```
\fontspecdef\MyFontSpec
  \fontid\font
\relax
```

The engine adapts itself to these glyph parameters but when you access certain quantities you have to make sure that you use the scaled ones. The same is true at the Lua end. This is somewhat fundamental in the sense that when one uses these sort of dynamic features one also need to keep an eye on code that uses font specific dimensions.

## 252 \fontspecid

Internally a font reference is a number and this primitive returns the number of the font bound to the specification.

## 253 \fontspecifiedname

Depending on how the font subsystem is implemented this gives some information about the (original) definition of the used font:

```
{\tf \fontspecifiedname\font}
{\bf \fontspecifiedname\font}
{\sl \fontspecifiedname\font}
```

Serif sa 1

**SerifBold sa 1**

*SerifSlanted sa 1*

## 254 \fontspecifiedsize

Depending on how the font subsystem is implemented this gives some information about the (original) size of the used font:

```
{\tf \the\fontspecifiedsize\font : \the\glyphscale}
{\bfa \the\fontspecifiedsize\font : \the\glyphscale}
{\slx \the\fontspecifiedsize\font : \the\glyphscale}
```

Depending on how the font system is setup, this is not the real value that is used in the text because we can use for instance \glyphscale. So the next lines depend on what font mode this document is typeset.

10.0pt: 1000

**10.0pt: 1200**

*10.0pt: 800*

## 255 \fontspecscale

This returns the scale factor of a fontspec where as usual 1000 means scaling by 1.

## 256 \fontspecslant

This returns the slant factor of a font specification, usually between zero and 1000 with 1000 being maximum slant.

## 257 \fontspecweight

This returns the weight of the font specification. Reasonable values are between zero and 500.

## 258 \fontspecxscale

This returns the scale factor of a font specification where as usual 1000 means scaling by 1.

## 259 \fontspecyscale

This returns the scale factor of a font specification where as usual 1000 means scaling by 1.

## 260 \fonttextcontrol

This returns the text control flags that are set on the given font, here 0x8. Bits that can be set are:

- 0x01 collapsehyphens
- 0x02 baseligaturing
- 0x04 basekerning
- 0x08 noneprotected
- 0x10 hasitalics
- 0x20 autoitalics

## 261 \forcedleftcorrection

This is a callback driven left correction signal similar to italic corrections.

## 262 \forcedrightcorrection

This is a callback driven right correction signal similar to italic corrections.

## 263 \formatname

It is in the name: cont-en, but we cheat here by only showing the filename and not the full path, which in a ConT<sub>E</sub>Xt setup can span more than a line in this paragraph.



## 264 \frozen

You can define a macro as being frozen:

```
\frozen\def\MyMacro{...}
```

When you redefine this macro you get an error:

```
! You can't redefine a frozen macro.
```

This is a prefix like \global and it can be combined with other prefixes.<sup>1</sup>

## 265 \futurecsname

In order to make the repertoire of def, let and futurelet primitives complete we also have:

```
\futurecsname MyMacro:1\endcsname\MyAction
```

## 266 \futuredef

We elaborate on the example of using \futurelet in the previous section. Compare that one with the next:

```
\def\MySpecialToken{[]}  
\def\DoWhatever{\ifx\NextToken\MySpecialToken YES\else NOP\fi : }  
\futurelet\NextToken\DoWhatever [A]\crlf  
\futurelet\NextToken\DoWhatever (A)\par
```

This time we get:

```
NOP: [A]
```

```
NOP: (A)
```

It is for that reason that we now also have \futuredef:

```
\def\MySpecialToken{[]}  
\def\DoWhatever{\ifx\NextToken\MySpecialToken YES\else NOP\fi : }  
\futuredef\NextToken\DoWhatever [A]\crlf  
\futuredef\NextToken\DoWhatever (A)\par
```

So we're back to what we want:

```
YES: [A]
```

```
NOP: (A)
```

## 267 \futureexpand

This primitive can be used as an alternative to a \futurelet approach, which is where the name comes from.<sup>2</sup>

<sup>1</sup> The \outer and \long prefixes are no-ops in LuaMetaTeX and LuaTeX can be configured to ignore them.

<sup>2</sup> In the engine primitives that have similar behavior are grouped in commands that are then dealt with together, code wise.

```

\def\variantone<#1>{(#1)}
\def\varianttwo#1{[#1]}
\futureexpand<\variantone\varianttwo<one>
\futureexpand<\variantone\varianttwo{two}

```

So, the next token determines which of the two variants is taken:

(one) [two]

Because we look ahead there is some magic involved: spaces are ignored but when we have no match they are pushed back into the input. The next variant demonstrates this:

```

\def\variantone<#1>{(#1)}
\def\varianttwo{}
\def\temp{\futureexpand<\variantone\varianttwo}
[\temp <one>]
[\temp {two}]
[\expandafter\temp\space <one>]
[\expandafter\temp\space {two}]

```

This gives us:

[(one)] [two] [(one)] [ two]

## 268 \futureexpandis

We assume that the previous section is read. This variant will not push back spaces, which permits a consistent approach i.e. the user can assume that macro always gobbles the spaces.

```

\def\variantone<#1>{(#1)}
\def\varianttwo{}
\def\temp{\futureexpandis<\variantone\varianttwo}
[\temp <one>]
[\temp {two}]
[\expandafter\temp\space <one>]
[\expandafter\temp\space {two}]

```

So, here no spaces are pushed back. This is in the name of this primitive means ‘ignore spaces’, but having that added to the name would have made the primitive even more verbose (after all, we also don't have `\expandeddef` but `\edef` and no `\globalexpandeddef` but `\xdef`).

[(one)] [two] [(one)] [two]

## 269 \futureexpandisap

This primitive is like the one in the previous section but also ignores par tokens, so `isap` means ‘ignore spaces and paragraphs’.

## 270 \futurelet

The original  $\text{\TeX}$  primitive `\futurelet` can be used to create an alias to a next token, push it back into the input and then expand a given token.

```

\let\MySpecialTokenL[
\let\MySpecialTokenR] % nicer for checker
\def\DoWhatever{\ifx\NextToken\MySpecialTokenL YES\else NOP\fi : }
\futurelet\NextToken\DoWhatever [A]\crlf
\futurelet\NextToken\DoWhatever (A)\par

```

This is typically the kind of primitive that most users will never use because it expects a sane follow up handler (here `\DoWhatever`) and therefore is related to user interfacing.

YES: [A]

NOP: (A)

## 271 `\gdef`

The is the global companion of `\def`.

## 272 `\gdefcsname`

As with standard  $\TeX$  we also define global ones:

```

\expandafter\gdef\csname MyMacro:1\endcsname{...}
\gdefcsname MyMacro:1\endcsname{...}

```

## 273 `\givenmathstyle`

This primitive expects a math style and returns it when valid or otherwise issues an error.

## 274 `\gleaders`

Leaders are glue with special property: a box, rule of (in  $\text{LuaMeta}\TeX$ ) glyph, like:

```

x  MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMx
xx MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM  xx

xMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMx
xx MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM  xx

xMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMx
xxMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMxx

xMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMx
xx MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM  xx

```

Leaders fill the available space. The `\leaders` command starts at the left edge and stops when there is no more space. The blobs get centered when we use `\cleaders`: excess space is distributed before and after a blob while `\xleaders` also puts space between the blobs.

When a rule is given the advance (width or height and depth) is ignored, so these are equivalent.

```

x\leaders \hrule \hfill x
x\leaders \hrule width 1cm \hfill x

```

When a box is used one will normally have some alignment in that box.

```
x\leaders \hbox {\hss.\hss} \hfill x
x\leaders \hbox {\hss.\hss} \hskip 6cm \relax x
```

The reference point is the left edge of the current (outer) box and the effective glue (when it has stretch or shrink) depends on that box. The `\gleaders` variant takes the page as reference. That makes it possible to ‘align’ across boxes.

## 275 `\glet`

This is the global companion of `\let`. The fact that it is not an original primitive is probably due to the expectation for it not being used (as) often (as in `ConTeXt`).

## 276 `\gletcsname`

Naturally `LuaMetaTeX` also provides a global variant:

```
\expandafter\global\expandafter\let\csname MyMacro:1\endcsname\relax
\expandafter \glet\csname MyMacro:1\endcsname\relax
\gletcsname MyMacro:1\endcsname\relax
```

So, here we save even more.

## 277 `\glettonothing`

This is the global companion of `\lettonothing`.

## 278 `\global`

This is one of the original prefixes that can be used when we define a macro of change some register.

```
\bgroup
\def\MyMacroA{a}
\global\def\MyMacroB{a}
\gdef\MyMacroC{a}
\egroup
```

The macro defined in the first line is forgotten when the groups is left. The second and third definition are both global and these definitions are retained.

## 279 `\globaldefs`

When set to a positive value, this internal integer will force all definitions to be global, and in a complex macro package that is not something a user will do unless it is very controlled.

## 280 `\glueexpr`

This is a more extensive variant of `\dimexpr` that also handles the optional stretch and shrink components.

## 281 `\glueshrink`

This returns the shrink component of a glue quantity. The result is a dimension so you need to apply `\the` when applicable.

## 282 `\glueshrinkorder`

This returns the shrink order of a glue quantity. The result is a integer so you need to apply `\the` when applicable.

## 283 `\gluespecdef`

A variant of `\integerdef` and `\dimensiondef` is:

```
\gluespecdef\MyGlue = 3pt plus 2pt minus 1pt
```

The properties are comparable to the ones described in the previous sections.

## 284 `\gluestretch`

This returns the stretch component of a glue quantity. The result is a dimension so you need to apply `\the` when applicable.

## 285 `\gluestretchorder`

This returns the stretch order of a glue quantity. The result is a integer so you need to apply `\the` when applicable.

## 286 `\gluetomu`

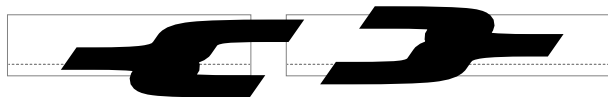
The sequence `\the\gluetomu 20pt plus 10pt minus 5pt` gives 20.0mu plus 10.0mu minus 5.0mu.

## 287 `\glyph`

This is a more extensive variant of `\char` that permits setting some properties if the injected character node.

```
\ruledhbox{\glyph
  scale 2000 xscale 9000 yscale 1200
  slant 700 weight 200
  xoffset 10pt yoffset -5pt left 10pt right 20pt
  123}
\quad
\ruledhbox{\glyph
  scale 2000 xscale 9000 yscale 1200
  slant 700 weight 200
  125}
```

In addition one can specify font (symbol), id (valid font id number), an options (bit set) and raise.



When no parameters are set, the current ones are used. More details and examples of usage can be found in the ConT<sub>E</sub>Xt distribution.

## 288 `\glyphdatafield`

The value of this parameter is assigned to data field in glyph nodes that get injected. It has no meaning in itself but can be used at the Lua end.

## 289 `\glyphoptions`

The value of this parameter is assigned to the options field in glyph nodes that get injected.

0x00000000	normal	0x00000080	nozeroitaliccorrection
0x00000001	noleftligature	0x00000100	applyxoffset
0x00000002	norightligature	0x00000200	applyyoffset
0x00000004	noleftkern	0x00000400	mathdiscretionary
0x00000008	norightkern	0x00000800	mathsitalicstoo
0x00000010	noexpansion	0x00010000	userfirst
0x00000020	noprotrusion	0x40000000	userlast
0x00000040	noitaliccorrection		

## 290 `\glyphscale`

An integer parameter defining the current glyph scale, assigned to glyphs (characters) inserted into the current list.

## 291 `\glyphscriptfield`

The value of this parameter is assigned to script field in glyph nodes that get injected. It has no meaning in itself but can be used at the Lua end.

## 292 `\glyphscriptscale`

This multiplier is applied to text font and glyph dimension properties when script style is used.

## 293 `\glyphscriptscriptscale`

This multiplier is applied to text font and glyph dimension properties when script script style is used.

## 294 `\glyphslant`

An integer parameter defining the current glyph slant, assigned to glyphs (characters) inserted into the current list.

## 295 `\glyphstatefield`

The value of this parameter is assigned to script state in glyph nodes that get injected. It has no meaning in itself but can be used at the Lua end.

## 296 `\glyphtextscale`

This multiplier is applied to text font and glyph dimension properties when text style is used.

## 297 `\glyphweight`

An integer parameter defining the current glyph weight, assigned to glyphs (characters) inserted into the current list.

## 298 `\glyphxoffset`

An integer parameter defining the current glyph x offset, assigned to glyphs (characters) inserted into the current list. Normally this will only be set when one explicitly works with glyphs and defines a specific sequence.

## 299 `\glyphxscale`

An integer parameter defining the current glyph x scale, assigned to glyphs (characters) inserted into the current list.

## 300 `\glyphxscaled`

This primitive returns the given dimension scaled by the `\glyphscale` and `\glyphxscale`.

## 301 `\glyphyoffset`

An integer parameter defining the current glyph y offset, assigned to glyphs (characters) inserted into the current list. Normally this will only be set when one explicitly works with glyphs and defines a specific sequence.

## 302 `\glyphyscale`

An integer parameter defining the current glyph y scale, assigned to glyphs (characters) inserted into the current list.

## 303 `\glyphyscaled`

This primitive returns the given dimension scaled by the `\glyphscale` and `\glyphyscale`.

## 304 `\gtoksapp`

This is the global variant of `\toksapp`.

### 305 `\gtokspre`

This is the global variant of `\tokspre`.

### 306 `\halign`

This command starts horizontally aligned material. Macro packages use this command in table mechanisms and math alignments. It starts with a preamble followed by entries (rows and columns).

### 307 `\hangafter`

This parameter tells the par builder when indentation specified with `\hangindent` starts. A negative value does the opposite and starts indenting immediately. So, a value of  $-2$  will make the first two lines indent.

### 308 `\hangindent`

This parameter relates to `\hangafter` and sets the amount of indentation. When larger than zero indentation happens left, otherwise it starts at the right edge.

### 309 `\hbadness`

This sets the threshold for reporting a horizontal badness value, its current value is 0.

### 310 `\hbox`

This constructs a horizontal box. There are a lot of optional parameters so more details can be found in dedicated manuals. When the content is packed a callback can kick in that can be used to apply for instance font features.

### 311 `\hccode`

The  $\text{\TeX}$  engine is good at hyphenating but traditionally that has been limited to hyphens. Some languages however use different characters. You can set up a different `\hyphenchar` as well as pre and post characters, but there's also a dedicated code for controlling this.

```
\hccode"2013 "2013
```

```
\hsize 50mm test\char"2013test\par
```

```
\hsize 1mm test\char"2013test\par
```

```
\hccode"2013 `!
```

```
\hsize 50mm test\char"2013test\par
```

```
\hsize 1mm test\char"2013test\par
```

This example shows that we can mark a character as hyphen-like but also can remap it to something else:

```
test-test
```



```
test-
test
test-test
test!
test
```

### 312 `\hfil`

This is a shortcut for `\hskip` plus 1 fil (first order filler).

### 313 `\hfill`

This is a shortcut for `\hskip` plus 1 fill (second order filler).

### 314 `\hfilneg`

This is a shortcut for `\hskip` plus - 1 fil so it can compensate `\hfil`.

### 315 `\hfuzz`

This dimension sets the threshold for reporting horizontal boxes that are under- or overfull. The current value is 0.1pt.

### 316 `\hjcode`

The so called lowercase code determines if a character is part of a to-be-hyphenated word. In LuaT<sub>E</sub>X we introduced the ‘hyphenation justification’ code as replacement. When a language is saved and no `\hjcode` is set the `\lccode` is used instead. This code serves a second purpose. When the assigned value is greater than 0 but less than 32 it indicated the to be used length when checking for left- and righthyphenmin. For instance it make sense to set the code to 2 for characters like œ.

### 317 `\hkern`

This primitive is like `\kern` but will force the engine into horizontal mode if it isn't yet.

### 318 `\hmcode`

The hm stands for ‘hyphenation math’. When bit 1 is set the characters will be repeated on the next line after a break. The second bit concerns italic correction but is of little relevance now that we moved to a different model in ConT<sub>E</sub>Xt. Here are some examples, we also show an example of `\mathdiscretionary` because that is what this code triggers:

```
test $ \dorecurse {50} {
  a \discretionary class 2 {$\darkred +$}{$\darkgreen +$}{$\darkblue +$}
} b$
```

```
test $ a \mathdiscretionary class 1 {-}{-}{-} b$
```

`\egroup`

test.  $a - b$

## h rule

h  
rule  
v rule  
v rule

### 324 `\hsize`

This sets (or gets) the current horizontal size.

```
\hsize 40pt \setbox0\vbox{x} hsize: \the\wd0
\setbox0\vbox{\hsize 40pt x} hsize: \the\wd0
```

In both cases we get the same size reported but the first one will also influence the current paragraph when used ungrouped.

```
hsize: 40.0pt
hsize: 40.0pt
```

### 325 `\hskip`

The given glue is injected in the horizontal list. If possible horizontal mode is entered.

### 326 `\hss`

In traditional  $\text{\TeX}$  glue specifiers are shared. This makes a lot of sense when memory has to be saved. For instance spaces in a paragraph of text are often the same and a glue specification has at least an amount, stretch, shrink, stretch order and shrink order field plus a leader pointer; in  $\text{\LuaMetaTeX}$  we have even more fields. In  $\text{\LuaTeX}$  these shared (and therefore referenced) glue spec nodes became just copies.

```
x\hbox to 0pt{\hskip 0pt plus 1 fil minus 1 fil\relax test}x
x\hbox to 0pt{\hss test}x
x\hbox to 0pt{test\hskip 0pt plus 1 fil minus 1 fil\relax}x
x\hbox to 0pt{test\hss}x
```

The `\hss` primitives injects a glue node with one order stretch and one order shrink. In traditional  $\text{\TeX}$  this is a reference to a shared specification, and in  $\text{\LuaTeX}$  just a copy of a predefined specifier. The only gain is now in tokens because one could just be explicit or use a glue register with that value because we have plenty glue registers.

```
testx
testx
xtest
xtest
```

We could have this:

```
\permanent\protected\untraced\def\hss
{\hskip0pt plus 1 fil minus 1 fil\relax}
```

or this:

```
\gluespecdef\hssglue 0pt plus 1 fil minus 1 fil
```

```
\permanent\protected\untraced\def\hss
  {\hskip\hssglue}
```

but we just keep the originals around.

### 327 \ht

Returns the height of the given box.

### 328 \hyphenation

The list passed to this primitive contains hyphenation exceptions that get bound to the current language. In LuaMetaTeX this can be managed at the Lua end. Exceptions are not stored in the format file.

### 329 \hyphenationmin

This property (that also gets bound to the current language) sets the minimum length of a word that gets hyphenated.

### 330 \hyphenchar

This is one of the font related primitives: it returns the number of the hyphen set in the given font.

### 331 \hyphenpenalty

Discretionary nodes have a related default penalty. The `\hyphenpenalty` is injected after a regular discretionary, and `\exhyphenpenalty` after `\-` or `-\`. The later case is called an automatic discretionary. In LuaMetaTeX we have two extra penalties: `\explicithyphenpenalty` and `\automatichyphenpenalty` and these are used when the related bits are set in `\hyphenationmode`.

### 332 \if

This traditional TeX conditional checks if two character codes are the same. In order to understand unexpanded results it is good to know that internally TeX groups primitives in a way that serves the implementation. Each primitive has a command code and a character code, but only for real characters the name character code makes sense. This condition only really tests for character codes when we have a character, in all other cases, the result is true.

```
\def\A{A}\def\B{B} \chardef\C=`C \chardef\D=`D \def\AA{AA}
```

```
[\if AA YES \else NOP \fi] [\if AB YES \else NOP \fi]
[\if \A\B YES \else NOP \fi] [\if \A\A YES \else NOP \fi]
[\if \C\D YES \else NOP \fi] [\if \C\C YES \else NOP \fi]
[\if \count\dimen YES \else NOP \fi] [\if \AA\A YES \else NOP \fi]
```

The last example demonstrates that the tokens get expanded, which is why we get the extra A:

[ YES ] [NOP ] [NOP ] [YES ] [YES ] [YES ] [YES ] [AYES ]

### 333 \ifabsdim

This test will negate negative dimensions before comparison, as in:

```
\def\TestA#1{\ifdim #1<2pt too small\orelse\ifdim #1>4pt too large\else okay\fi}
\def\TestB#1{\ifabsdim#1<2pt too small\orelse\ifabsdim#1>4pt too large\else okay\fi}

\TestA {1pt}\quad\TestA {3pt}\quad\TestA {5pt}\crlf
\TestB {1pt}\quad\TestB {3pt}\quad\TestB {5pt}\crlf
\TestB{-1pt}\quad\TestB{-3pt}\quad\TestB{-5pt}\par
```

So we get this:

```
too small  okay  too large
too small  okay  too large
too small  okay  too large
```

### 334 \ifabsfloat

This test will negate negative floats before comparison, as in:

```
\def\TestA#1{\iffloat #1<2.46 small\orelse\iffloat #1>4.68 large\else medium\fi}
\def\TestB#1{\ifabsfloat#1<2.46 small\orelse\ifabsfloat#1>4.68 large\else medium\fi}

\TestA {1.23}\quad\TestA {3.45}\quad\TestA {5.67}\crlf
\TestB {1.23}\quad\TestB {3.45}\quad\TestB {5.67}\crlf
\TestB{-1.23}\quad\TestB{-3.45}\quad\TestB{-5.67}\par
```

So we get this:

```
small  medium  large
small  medium  large
small  medium  large
```

### 335 \ifabsnum

This test will negate negative numbers before comparison, as in:

```
\def\TestA#1{\ifnum #1<100 too small\orelse\ifnum #1>200 too large\else okay\fi}
\def\TestB#1{\ifabsnum#1<100 too small\orelse\ifabsnum#1>200 too large\else okay\fi}

\TestA {10}\quad\TestA {150}\quad\TestA {210}\crlf
\TestB {10}\quad\TestB {150}\quad\TestB {210}\crlf
\TestB{-10}\quad\TestB{-150}\quad\TestB{-210}\par
```

Here we get the same result each time:

```
too small  okay  too large
too small  okay  too large
too small  okay  too large
```

### 336 \ifarguments

This is a variant of `\ifcase` where the selector is the number of arguments picked up. For example:

```
\def\MyMacro#1#2#3{\ifarguments\0\or1\or2\or3\else ?\fi} \MyMacro{A}{B}{C}
\def\MyMacro#1#0#3{\ifarguments\0\or1\or2\or3\else ?\fi} \MyMacro{A}{B}{C}
\def\MyMacro#1#-#2{\ifarguments\0\or1\or2\or3\else ?\fi} \MyMacro{A}{B}{C}\par
```

Watch the non counted, ignored, argument in the last case. Normally this test will be used in combination with `\ignorearguments`.

3 3 2

### 337 \ifboolean

This tests a number (register or equivalent) and any nonzero value represents true, which is nicer than using an `\unless\ifcase`.

### 338 \ifcase

This numeric  $\TeX$  conditional takes a counter (literal, register, shortcut to a character, internal quantity) and goes to the branch that matches.

```
\ifcase 3 zero\or one\or two\or three\or four\else five or more\fi
```

Indeed: three equals three. In later sections we will see some  $\text{LuaMeta}\TeX$  primitives that behave like an `\ifcase`.

### 339 \ifcat

Another traditional  $\TeX$  primitive: what happens with what gets read in depends on the catcode of a character, think of characters marked to start math mode, or alphabetic characters (letters) versus other characters (like punctuation).

```
\def\A{A}\def\B{,} \chardef\C=`C \chardef\D=` , \def\AA{AA}

[\ifcat $! YES \else NOP \fi] [\ifcat () YES \else NOP \fi]
[\ifcat AA YES \else NOP \fi] [\ifcat AB YES \else NOP \fi]
[\ifcat \A\B YES \else NOP \fi] [\ifcat \A\A YES \else NOP \fi]
[\ifcat \C\D YES \else NOP \fi] [\ifcat \C\C YES \else NOP \fi]
[\ifcat \count\dimen YES \else NOP \fi] [\ifcat \AA\A YES \else NOP \fi]
```

Close reading is needed here:

```
[NOP ] [ YES ] [ YES ] [ YES ] [NOP ] [YES ] [YES ] [YES ] [YES ] [AYES ]
```

This traditional  $\TeX$  condition as well as the one in the previous section are hardly used in  $\text{Con}\TeX$ t, if only because they expand what follows and we seldom need to compare characters.

### 340 \ifchkdim

A variant on the checker in the previous section is a dimension checker:

```

\ifchkdim oeps      \or okay\else error\fi\quad
\ifchkdim 12        \or okay\else error\fi\quad
\ifchkdim 12pt      \or okay\else error\fi\quad
\ifchkdim 12pt or more\or okay\else error\fi

```

We get:

```
error error okay okay
```

### 341 \ifchkdimension

ConTrary to \ifchkdim this test doesn't accept trailing crap:

```

\ifchkdimension oeps      \or okay\else error\fi\quad
\ifchkdimension 12        \or okay\else error\fi\quad
\ifchkdimension 12pt      \or okay\else error\fi\quad
\ifchkdimension 12pt or more\or okay\else error\fi

```

reports:

```
error error okay error
```

### 342 \ifchknum

In ConT<sub>E</sub>Xt there are quite some cases where a variable can have a number or a keyword indicating a symbolic name of a number or maybe even some special treatment. Checking if a valid number is given is possible to some extend, but a native checker makes much sense too. So here is one:

```

\ifchknum oeps      \or okay\else error\fi\quad
\ifchknum 12        \or okay\else error\fi\quad
\ifchknum 12pt      \or okay\else error\fi\quad
\ifchknum 12pt or more\or okay\else error\fi

```

The result is as expected:

```
error okay okay okay
```

### 343 \ifchknumber

This check is more restrictive than \ifchknum discussed in the previous section:

```

\ifchknumber oeps      \or okay\else error\fi\quad
\ifchknumber 12        \or okay\else error\fi\quad
\ifchknumber 12pt      \or okay\else error\fi\quad
\ifchknumber 12pt or more\or okay\else error\fi

```

Here we get:

```
error okay error error
```

### 344 \ifcmpdim

This conditional compares two dimensions and the resulting \ifcase reflects their relation:

```
[1pt 2pt : \ifcmpdim 1pt 2pt less\or equal\or more\fi]\quad
[1pt 1pt : \ifcmpdim 1pt 1pt less\or equal\or more\fi]\quad
[2pt 1pt : \ifcmpdim 2pt 1pt less\or equal\or more\fi]
```

This gives:

```
[1pt 2pt : less] [1pt 1pt : equal] [2pt 1pt : more]
```

### 345 \ifcmpnum

This conditional compares two numbers and the resulting \ifcase reflects their relation:

```
[1 2 : \ifcmpnum 1 2 less\or equal\or more\fi]\quad
[1 1 : \ifcmpnum 1 1 less\or equal\or more\fi]\quad
[2 1 : \ifcmpnum 2 1 less\or equal\or more\fi]
```

This gives:

```
[1 2 : less] [1 1 : equal] [2 1 : more]
```

### 346 \ifcondition

The conditionals in T<sub>E</sub>X are hard coded as primitives and although it might look like \newif creates one, it actually just defined three macros.

```
\newif\ifMyTest
\meaning\MyTesttrue \crlf
\meaning\MyTestfalse \crlf
\meaning\ifMyTest \crlf \MyTesttrue
\meaning\ifMyTest \par
```

```
protected macro:\always \let \ifMyTest \iftrue
protected macro:\always \let \ifMyTest \iffalse
\iffalse
\iftrue
```

This means that when you say:

```
\ifMytest ... \else ... \fi
```

You actually have one of:

```
\iftrue ... \else ... \fi
\iffalse ... \else ... \fi
```

and because these are proper conditions nesting them like:

```
\ifnum\scratchcounter > 0 \ifMyTest A\else B\fi \fi
```

will work out well too. This is not true for macros, so for instance:

```
\scratchcounter = 1
\unexpanded\def\ifMyTest{\iftrue}
```



```
\ifnum\scratchcounter > 0 \ifMyTest A\else B\fi \fi
```

will make a run fail with an error (or simply loop forever, depending on your code). This is where `\ifcondition` enters the picture:

```
\def\MyTest{\iftrue} \scratchcounter0
\ifnum\scratchcounter > 0
  \ifcondition\MyTest A\else B\fi
\else
  x
\fi
```

This primitive is seen as a proper condition when  $\text{\TeX}$  is in “fast skipping unused branches” mode but when it is expanding a branch, it checks if the next expanded token is a proper tests and if so, it deals with that test, otherwise it fails. The main condition here is that the `\MyTest` macro expands to a proper true or false test, so, a definition like:

```
\def\MyTest{\ifnum\scratchcounter<10 }
```

is also okay. Now, is that neat or not?

### 347 \ifcramped

Depending on the given math style this returns true of false:

```
\ifcramped\mathstyle      no \fi
\ifcramped\crampedtextstyle yes \fi
\ifcramped\textstyle       no \fi
\ifcramped\displaystyle    yes \fi
```

gives: yes.

### 348 \ifcsname

This is an  $\varepsilon\text{-TeX}$  conditional that complements the one on the previous section:

```
\expandafter\ifx\csname MyMacro\endcsname\relax ... \else ... \fi
\ifcsname MyMacro\endcsname ... \else ... \fi
```

Here the first one has the side effect of defining the macro and defaulting it to `\relax`, while the second one doesn't do that. Just think of checking a few million different names: the first one will deplete the hash table and probably string space too.

In  $\text{\LuaMetaTeX}$  the construction stops when there is no letter or other character seen ( $\text{\TeX}$  expands on the go so expandable macros are dealt with). Instead of an error message, the match is simply false and all tokens till the `\endcsname` are gobbled.

### 349 \ifcstok

A variant on the primitive mentioned in the previous section is one that operates on lists and macros:

```
\def\{a\} \def\{b\} \def\{c\}
```

This:

```
\ifcstok{a}\b Y\else N\fi\space
\ifcstok{a}\c Y\else N\fi\space
\ifcstok{\a}\c Y\else N\fi\space
\ifcstok{a}\c Y\else N\fi
```

will give us: N Y Y Y.

### 350 \ifdefined

In traditional T<sub>E</sub>X checking for a macro to exist was a bit tricky and therefore  $\epsilon$ -T<sub>E</sub>X introduced a convenient conditional. We can do this:

```
\ifx\MyMacro\undefined ... \else ... \fi
```

but that assumes that `\undefined` is indeed undefined. Another test often seen was this:

```
\expandafter\ifx\csname MyMacro\endcsname\relax ... \else ... \fi
```

Instead of comparing with `\undefined` we need to check with `\relax` because the control sequence is defined when not yet present and defaults to `\relax`. This is not pretty.

### 351 \ifdim

Dimensions can be compared with this traditional T<sub>E</sub>X primitive.

```
\scratchdimen=1pt \scratchcounter=65536
```

```
\ifdim\scratchdimen=\scratchcounter sp YES \else NOP\fi
\ifdim\scratchdimen=1 pt YES \else NOP\fi
```

The units are mandate:

YES YES

### 352 \ifdimexpression

The companion of the previous primitive is:

This matches when the result is non zero, and you can mix calculations and tests as with normal expressions. Contrary to the number variant units can be used and precision kicks in.

### 353 \ifdimval

This conditional is a variant on `\ifchkdim` and provides some more detailed information about the value:

```
[-12pt : \ifdimval -12pt\or negative\or zero\or positive\else error\fi]\quad
[0pt : \ifdimval 0pt\or negative\or zero\or positive\else error\fi]\quad
[12pt : \ifdimval 12pt\or negative\or zero\or positive\else error\fi]\quad
[oeps : \ifdimval oeps\or negative\or zero\or positive\else error\fi]
```

This gives:

[ -12pt : negative] [ 0pt : zero] [ 12pt : positive] [ oeps : error]

### 354 \ifempty

This conditional checks if a control sequence is empty:

is `\ifempty\MyMacro` `\else` not `\fi` empty

It is basically a shortcut of:

is `\ifx\MyMacro\empty` `\else` not `\fi` empty

with:

`\def\empty{}`

Of course this is not empty at all:

`\def\notempty#1{}`

### 355 \iffalse

Here we have a traditional T<sub>E</sub>X conditional that is always false (therefore the same is true for any macro that is `\let` to this primitive).

### 356 \ifflags

This test primitive relates to the various flags that one can set on a control sequence in the perspective of overload protection and classification.

`\protected\untraced\tolerant\def\foo[#1]{...#1...}`  
`\permanent\constant` `\def\oof{okay}`

flag	<code>\foo</code>	<code>\oof</code>	flag	<code>\foo</code>	<code>\oof</code>
frozen	N	N	permanent	N	Y
immutable	N	N	mutable	N	N
noaligned	N	N	instance	N	N
untraced	Y	N	global	N	N
tolerant	Y	N	constant	N	Y
protected	Y	N	semiprotected	N	N

Instead of checking against a prefix you can test against a bitset made from:

0x1	frozen	0x2	permanent	0x4	immutable	0x8	primitive
0x10	mutable	0x20	noaligned	0x40	instance	0x80	untraced
0x100	global	0x200	tolerant	0x400	protected	0x800	overloaded
0x1000	aliased	0x2000	immediate	0x4000	conditional	0x8000	value
0x10000	semiprotected	0x20000	inherited	0x40000	constant	0x80000	deferred

### 357 \iffloat

This test does for floats what \ifnum, \ifdim do for numbers and dimensions: comparing two of them.

### 358 \iffontchar

This is an  $\varepsilon$ -TeX conditional. It takes a font identifier and a character number. In modern fonts simply checking could not be enough because complex font features can swap in other ones and their index can be anything. Also, a font mechanism can provide fallback fonts and characters, so don't rely on this one too much. It just reports true when the font passed to the frontend has a slot filled.

### 359 \ifhaschar

This one is a simplified variant of the above:

```
\ifhaschar !{this ! works} yes \else no \fi
```

and indeed we get: yes! Of course the spaces in this this example code are normally not present in such a test.

### 360 \ifhastok

This conditional looks for occurrences in token lists where each argument has to be a proper list.

```
\def\scratchtoks{x}

\ifhastoks{yz}      {xyz} Y\else N\fi\quad
\ifhastoks\scratchtoks {xyz} Y\else N\fi
```

We get:

```
Y  Y
```

### 361 \ifhastoks

This test compares two token lists. When a macro is passed it's meaning gets used.

```
\def\x {x}
\def\xyz{xyz}

(\ifhastoks {x} {xyz}Y\else N\fi)\quad
(\ifhastoks {\x} {xyz}Y\else N\fi)\quad
(\ifhastoks \x {xyz}Y\else N\fi)\quad
(\ifhastoks {y} {xyz}Y\else N\fi)\quad
(\ifhastoks {yz} {xyz}Y\else N\fi)\quad
(\ifhastoks {yz} {\xyz}Y\else N\fi)
```

```
(Y) (N) (Y) (Y) (Y) (N)
```

### 362 \ifhasxtoks

This primitive is like the one in the previous section but this time the given lists are expanded.

```

\def\x {x}
\def\xyz{\x yz}

(\ifhasxtoks {x} {xyz}Y\else N\fi)\quad
(\ifhasxtoks {\x} {xyz}Y\else N\fi)\quad
(\ifhasxtoks \x {xyz}Y\else N\fi)\quad
(\ifhasxtoks {y} {xyz}Y\else N\fi)\quad
(\ifhasxtoks {yz} {xyz}Y\else N\fi)\quad
(\ifhasxtoks {yz} {\xyz}Y\else N\fi)

```

(Y) (Y) (Y) (Y) (Y) (Y)

This primitive has some special properties.

```

\edef\+{\expandtoken 9 `+}

\ifhasxtoks {xy} {xyz}Y\else N\fi\quad
\ifhasxtoks {x\+y} {xyz}Y\else N\fi

```

Here the first argument has a token that has category code ‘ignore’ which means that such a character will be skipped when seen. So the result is:

Y Y

This permits checks like these:

```

\edef\,{\expandtoken 9 `,}

\ifhasxtoks{\,x\,} {,x,y,z,}Y\else N\fi\quad
\ifhasxtoks{\,y\,} {,x,y,z,}Y\else N\fi\quad
\ifhasxtoks{\,z\,} {,x,y,z,}Y\else N\fi\quad
\ifhasxtoks{\,x\,} {,xy,z,}Y\else N\fi

```

I admit that it needs a bit of a twisted mind to come up with this, but it works ok:

Y Y Y N

### 363 \ifhbox

This traditional conditional checks if a given box register or internal box variable represents a horizontal box,

### 364 \ifhmode

This traditional conditional checks we are in (restricted) horizontal mode.

### 365 \iffinalignment

As the name indicates, this primitive tests for being in an alignment. Roughly spoken, the engine is either in a state of align, handling text or dealing with math.

### 366 `\ifincsname`

This conditional is sort of obsolete and can be used to check if we're inside a `\csname` or `\ifcsname` construction. It's not used in ConT<sub>E</sub>Xt.

### 367 `\ifinner`

This traditional one can be confusing. It is true when we are in restricted horizontal mode (a box), internal vertical mode (a box), or inline math mode.

```
test \ifhmode \ifinner INNER\fi HMODE\fi\crlf
\hbox{test \ifhmode \ifinner INNER \fi HMODE\fi} \par

\ifvmode \ifinner INNER\fi VMODE \fi\crlf
\ vbox{\ifvmode \ifinner INNER \fi VMODE\fi} \crlf
\ vbox{\ifinner INNER \ifvmode VMODE \fi \fi} \par
```

Watch the last line: because we typeset `INNER` we enter horizontal mode:

```
test HMODE
test INNER HMODE

VMODE
INNER VMODE
INNER
```

### 368 `\ifinsert`

This is the equivalent of `\ifvoid` for a given insert class.

### 369 `\ifintervalldim`

This conditional is true when the intervals around the values of two dimensions overlap. The first dimension determines the interval.

```
[\ifintervalldim1pt 20pt 21pt \else no \fi overlap]
[\ifintervalldim1pt 18pt 20pt \else no \fi overlap]
```

So here: [overlap] [no overlap]

### 370 `\ifintervalfloat`

This one does with floats what we described under `\ifintervalldim`.

### 371 `\ifintervalnum`

This one does with integers what we described under `\ifintervalldim`.

### 372 `\iflastnamedcs`

When a `\csname` is constructed and succeeds the last one is remembered and can be accessed with `\lastnamedcs`. It can however be an undefined one. That state can be checked with this primitive. Of course it also works with the `\ifcsname` and `\begincsname` variants.

### 373 `\ifmathparameter`

This is an `\ifcase` where the value depends on if the given math parameter is zero, (0), set (1), or unset (2).

```
\ifmathparameter\Umathpunctclosespacing\displaystyle
  zero      \or
  nonzero   \or
  unset     \fi
```

### 374 `\ifmathstyle`

This is a variant of `\ifcase` where the number is one of the seven possible styles: display, text, cramped text, script, cramped script, script script, cramped script script.

```
\ifmathstyle
  display
\or
  text
\or
  cramped text
\else
  normally smaller than text
\fi
```

### 375 `\ifmmode`

This traditional conditional checks we are in (inline or display) math mode mode.

### 376 `\ifnum`

This is a frequently used conditional: it compares two numbers where a number is anything that can be seen as such.

```
\scratchcounter=65 \chardef\A=65

\ifnum65=`A          YES \else NOP\fi
\ifnum\scratchcounter=65 YES \else NOP\fi
\ifnum\scratchcounter=\A YES \else NOP\fi
```

Unless a number is an unexpandable token it ends with a space or `\relax`, so when you end up in the true branch, you'd better check if  $\TeX$  could determine where the number ends.

YES YES YES

On top of these ascii combinations, the engine also accepts some Unicode characters. This brings the full repertoire to:

character	operation	
0x003C	<	less
0x003D	=	equal
0x003E	>	more
0x2208	∈	element of
0x2209	∉	not element of
0x2260	≠	not equal
0x2264	≤	less equal
0x2265	≥	greater equal
0x2270	≧	not less equal
0x2271	≨	not greater equal

This also applied to `\ifdim` although in the case of element we discard the fractional part (read: divide the numeric representation by 65536).

### 377 `\ifnumexpression`

Here is an example of a conditional using expressions:

This matches when the result is non zero, and you can mix calculations and tests as with normal expressions.

### 378 `\ifnumval`

This conditional is a variant on `\ifchknum`. This time we get some more detail about the value:

```
[ -12 : \ifnumval -12\or negative\or zero\or positive\else error\fi]\quad
[ 0   : \ifnumval  0\or negative\or zero\or positive\else error\fi]\quad
[ 12  : \ifnumval 12\or negative\or zero\or positive\else error\fi]\quad
[oeps : \ifnumval oeps\or negative\or zero\or positive\else error\fi]
```

This gives:

```
[-12 : negative] [0 : zero] [12 : positive] [oeps : error]
```

### 379 `\ifodd`

One reason for this condition to be around is that in a double sided layout we need test for being on an odd or even page. It scans for a number the same was as other primitives,

```
\ifodd65 YES \else NO\fi &
\ifodd`B YES \else NO\fi .
```

So: YES & NO.



### 380 \ifparameter

In a macro body `#1` is a reference to a parameter. You can check if one is set using a dedicated parameter condition:

```
\tolerant\def\foo[#1]*[#2]%
  {\ifparameter#1\or one\else no one\fi\enspace
   \ifparameter#2\or two\else no two\fi\emspace}

\foo
\foo[1]
\foo[1][2]
```

We get:

no one no two one no two one two

### 381 \ifparameters

This is equivalent to an `\ifcase` with as value the number of parameters passed to the current macro.

### 382 \ifrelax

This is a convenient shortcut for `\ifx\relax` and the motivation for adding this one is (as with some others) to get less tracing.

### 383 \iftok

When you want to compare two arguments, the usual way to do this is the following:

```
\edef\tempA{#1}
\edef\tempB{#2}
\ifx\tempA\tempB
  the same
\else
  different
\fi
```

This works quite well but the fact that we need to define two macros can be considered a bit of a nuisance. It also makes macros that use this method to be not so called ‘fully expandable’. The next one avoids both issues:

```
\iftok{#1}{#2}
  the same
\else
  different
\fi
```

Instead of direct list you can also pass registers, so given:

```
\scratchtoks{a}%
```

```
\toks0{a}%
```

This:

```
\iftok 0 \scratchtoks      Y\else N\fi\space
\iftok{a}\scratchtoks      Y\else N\fi\space
\iftok\scratchtoks\scratchtoks Y\else N\fi
```

gives: Y Y Y.

### 384 \iftrue

Here we have a traditional T<sub>E</sub>X conditional that is always true (therefore the same is true for any macro that is \let to this primitive).

### 385 \ifvbox

This traditional conditional checks if a given box register or internal box variable represents a vertical box,

### 386 \ifvmode

This traditional conditional checks we are in (internal) vertical mode.

### 387 \ifvoid

This traditional conditional checks if a given box register or internal box variable has any content.

### 388 \ifx

We use this traditional T<sub>E</sub>X conditional a lot in ConT<sub>E</sub>Xt. Contrary to \if the two tokens that are compared are not expanded. This makes it possible to compare the meaning of two macros. Depending on the need, these macros can have their content expanded or not. A different number of parameters results in false.

Control sequences are identical when they have the same command code and character code. Because a \let macro is just a reference, both let macros are the same and equal to \relax:

```
\let\one\relax \let\two\relax
```

The same is true for other definitions that result in the same (primitive) or meaning encoded in the character field (think of \chardefs and so).

### 389 \ifzerodim

This tests for a dimen (dimension) being zero so we have:

```
\ifdim<dimension>=0pt
\ifzerodim<dimension>
\ifcase<dimension register>
```

### 390 \ifzerofloat

As the name indicated, this tests for a zero float value.

```
[\scratchfloat\zerofloat \ifzerofloat\scratchfloat \else not \fi zero]
[\scratchfloat\plusone \ifzerofloat\scratchfloat \else not \fi zero]
[\scratchfloat 0.01 \ifzerofloat\scratchfloat \else not \fi zero]
[\scratchfloat 0.0e0 \ifzerofloat\scratchfloat \else not \fi zero]
[\scratchfloat \zeropoint\ifzerofloat\scratchfloat \else not \fi zero]
```

So: [zero] [not zero] [ not zero] [ zero] [zero]

### 391 \ifzeronum

This tests for a number (integer) being zero so we have these variants now:

```
\ifnum<integer or equivalent>=0
\ifzeronum<integer or equivalent>
\ifcase<integer or equivalent>
```

### 392 \ignorearguments

This primitive will quit argument scanning and start expansion of the body of a macro. The number of grabbed arguments can be tested as follows:

```
\def\MyMacro[#1][#2][#3]%
  {\ifarguments zero\or one\or two\or three \else hm\fi}

\MyMacro \ignorearguments \quad
\MyMacro [1]\ignorearguments \quad
\MyMacro [1][2]\ignorearguments \quad
\MyMacro [1][2][3]\ignorearguments \par
```

zero one two three

*Todo: explain optional delimiters.*

### 393 \ignoredepthcriterion

When setting the \prevdepth (either by T<sub>E</sub>X or by the current user) of the current vertical list the value 1000pt is a signal for special treatment of the skip between ‘lines’. There is an article on that in the distribution. It also demonstrates that \ignoredepthcriterion can be used to change this special signal, just in case it is needed.

### 394 \ignorenestedupto

This primitive gobbles following tokens and can deal with nested ‘environments’, for example:

```
\def\startfoo{\ignorenestedupto\startfoo\stopfoo}
```

(before

```

\startfoo
  test \startfoo test \stopfoo
  {test \startfoo test \stopfoo}
\stopfoo
after)

```

delivers:

(before after)

### 395 \ignorepars

This is a variant of `\ignorespaces`: following spaces *and* `\par` equivalent tokens are ignored, so for instance:

```

one + \ignorepars
two = \ignorepars \par
three

```

renders as: one + two = three. Traditionally  $\text{\TeX}$  has been sensitive to `\par` tokens in some of its building blocks. This has to do with the fact that it could indicate a runaway argument which in the times of slower machines and terminals was best to catch early. In  $\text{\LuaMetaTeX}$  we no longer have long macros and the mechanisms that are sensitive can be told to accept `\par` tokens (and  $\text{\ConTeXt}$  set them such that this is the case).

### 396 \ignorerest

An example shows what this primitive does:

```

\tolerant\def\foo[#1]*[#2]%
{1234
  \ifparameter#1\or\else
    \expandafter\ignorerest
  \fi
  /#1/
  \ifparameter#2\or\else
    \expandafter\ignorerest
  \fi
  /#2/ }

\foo test \foo[456] test \foo[456][789] test

```

As this likely makes most sense in conditionals you need to make sure the current state is properly finished. Because `\expandafter` bumps the input state, here we actually quit two levels; this is because so called ‘backed up text’ is intercepted by this primitive.

1234 test 1234 /456/ test 1234 /456/ /789/ test

### 397 \ignorespaces

This traditional  $\text{\TeX}$  primitive signals the scanner to ignore the following spaces, if any. We mention it because we show a companion in the next section.

### 398 `\ignoreupto`

This ignores everything upto the given token, so

`\ignoreupto \foo` not this but `\foo` only this

will give: only this.

### 399 `\immediate`

This one has no effect unless you intercept it at the Lua end and act upon it. In original  $\TeX$  `immediate` is used in combination with `read` from and `write` to file operations. So, this is an old primitive with a new meaning.

### 400 `\immutable`

This prefix flags what follows as being frozen and is usually applied to for instance `\integerdef`'d control sequences. In that respect is like `\permanent` but it makes it possible to distinguish quantities from macros.

### 401 `\indent`

In engines other than `LuaMetaTeX` a paragraph starts with an indentation box. The width of that (empty) box is determined by `\parindent`. In `LuaMetaTeX` we can use a dedicated indentation skip instead (as part of paragraph normalization). An indentation can be zero'd with `\undent`.

### 402 `\indexofcharacter`

This primitive is more versatile variant of the backward quote operator, so instead of:

```
\number`|
\number`~
\number`\a
\number`\q
```

you can say:

```
\the\indexofcharacter |
\the\indexofcharacter ~
\the\indexofcharacter \a
\the\indexofcharacter \q
```

In both cases active characters and unknown single character control sequences are valid. In addition this also works:

```
\chardef    \foo 128
\mathchardef\oof 130

\the\indexofcharacter \foo
\the\indexofcharacter \oof
```

An important difference is that `\indexofcharacter` returns an integer and not a serialized number. A negative value indicates no valid character.

### 403 `\indexofregister`

You can use this instead of `\number` for determining the index of a register but it also returns a number when a register value is seen. The result is an integer, not a serialized number.

### 404 `\inherited`

When this prefix is used in a definition using `\let` the target will inherit all the properties of the source.

### 405 `\initcatcodetable`

This initializes the catcode table with the given index.

### 406 `\initialpageskip`

When a page starts the value of this register are used to initialize `\pagetotal`, `\pagestretch` and `\pageshrink`. This make nicer code than using a `\topskip` with weird values.

### 407 `\initialtopskip`

When set this one will be used instead of `\topskip`. The rationale is that the `\topskip` is often also used for side effects and compensation.

### 408 `\input`

There are several ways to use this primitive:

```
\input test
\input {test}
\input "test"
\input 'test'
```

When no suffix is given,  $\text{\TeX}$  will assume the suffix is `.tex`. The second one is normally used.

### 409 `\inputlineno`

This integer holds the current linenumber but it is not always reliable.

### 410 `\insert`

This stores content in the insert container with the given index. In  $\text{\LuaMetaTeX}$  inserts bubble up to outer boxes so we don't have the ‘deeply buried insert issue’.

## 411 `\insertbox`

This is the accessor for the box (with results) of an insert with the given index. This is equivalent to the `\box` in the traditional method.

## 412 `\insertcopy`

This is the accessor for the box (with results) of an insert with the given index. It makes a copy so the original is kept. This is equivalent to a `\copy` in the traditional method.

## 413 `\insertdepth`

This is the (current) depth of the inserted material with the given index. It is comparable to the `\dp` in the traditional method.

## 414 `\insertdistance`

This is the space before the inserted material with the given index. This is equivalent to `\glue` in the traditional method.

## 415 `\insertheight`

This is the (current) depth of the inserted material with the given index. It is comparable to the `\ht` in the traditional method.

## 416 `\insertheights`

This is the combined height of the inserted material.

## 417 `\insertlimit`

This is the maximum height that the inserted material with the given index can get. This is equivalent to `\dimen` in the traditional method.

## 418 `\insertmaxdepth`

This is the maximum depth that the inserted material with the given index can get.

## 419 `\insertmode`

In traditional  $\text{\TeX}$  inserts are controlled by a `\box`, `\dimen`, `\glue` and `\count` register with the same index. The allocators have to take this into account. When this primitive is set to one a different model is followed with its own namespace. There are more abstract accessors to interface to this.<sup>3</sup>

---

<sup>3</sup> The old model might be removed at some point.

**420 \insertmultiplier**

This is the height (contribution) multiplier for the inserted material with the given index. This is equivalent to \count in the traditional method.

**421 \insertpenalties**

This dual purpose internal counter holds the sum of penalties for insertions that got split. When we're the output routine in reports the number of insertions that is kept in store.

**422 \insertpenalty**

This is the insert penalty associated with the inserted material with the given index.

**423 \insertprogress**

This returns the current accumulated insert height of the insert with the given index.

**424 \insertstorage**

The value passed will enable (one) or disable (zero) the insert with the given index.

**425 \insertstoring**

The value passed will enable (one) or disable (zero) inserts.

**426 \insertunbox**

This is the accessor for the box (with results) of an insert with the given index. It makes a copy so the original is kept. The content is unpacked and injected. This is equivalent to an \unvbox in the traditional method.

**427 \insertuncopy**

This is the accessor for the box (with results) of an insert with the given index. It makes a copy so the original is kept. The content is unpacked and injected. This is equivalent to the \unvcopy in the traditional method.

**428 \insertwidth**

This is the (current) width of the inserted material with the given index. It is comparable to the \wd in the traditional method.

**429 \instance**

This prefix flags a macro as an instance which is mostly relevant when a macro package want to categorize macros.



## 430 `\integerdef`

You can alias to a count (integer) register with `\countdef`:

```
\countdef\MyCount134
```

Afterwards the next two are equivalent:

```
\MyCount    = 99
\count1234 = 99
```

where `\MyCount` can be a bit more efficient because no index needs to be scanned. However, in terms of storage the value (here 99) is always in the register so `\MyCount` has to get there. This indirectness has the benefit that directly setting the value is reflected in the indirect accessor.

```
\integerdef\MyCount = 99
```

This primitive also defines a numeric equivalent but this time the number is stored with the equivalent. This means that:

```
\let\MyCopyOfCount = \MyCount
```

will store the *current* value of `\MyCount` in `\MyCopyOfCount` and changing either of them is not reflected in the other.

The usual `\advance`, `\multiply` and `\divide` can be used with these integers and they behave like any number. But compared to registers they are actually more a constant.

## 431 `\interactionmode`

This internal integer can be used to set or query the current interaction mode:

<code>\batchmode</code>	0	omits all stops and terminal output
<code>\nonstopmode</code>	1	omits all stops
<code>\scrollmode</code>	2	omits error stops
<code>\errorstopmode</code>	3	stops at every opportunity to interact

## 432 `\interlinepenalties`

This is a more granular variant of `\interlinepenalty`: an array of penalties to be put between successive line from the start of a paragraph. The list starts with the number of penalties that gets passed.

## 433 `\interlinepenalty`

This is the penalty that is put between lines.

## 434 `\jobname`

This gives the current job name without suffix: `primitives`.

### 435 `\kern`

A kern is injected with the given dimension. For variants that switch to a mode we have `\hkern` and `\vkern`.

### 436 `\language`

Sets (or returns) the current language, a number. In LuaT<sub>E</sub>X and LuaMetaT<sub>E</sub>X the current language is stored in the glyph nodes.

### 437 `\lastarguments`

```

\def\MyMacro    #1{\the\lastarguments (#1) }           \MyMacro{1}      \crlf
\def\MyMacro    #1#2{\the\lastarguments (#1) (#2)}      \MyMacro{1}{2}      \crlf
\def\MyMacro#1#2#3{\the\lastarguments (#1) (#2) (#3)} \MyMacro{1}{2}{3} \par

\def\MyMacro    #1{(#1)                \the\lastarguments} \MyMacro{1}      \crlf
\def\MyMacro    #1#2{(#1) (#2)          \the\lastarguments} \MyMacro{1}{2}      \crlf
\def\MyMacro#1#2#3{(#1) (#2) (#3) \the\lastarguments} \MyMacro{1}{2}{3} \par

```

The value of `\lastarguments` can only be trusted in the expansion until another macro is seen and expanded. For instance in these examples, as soon as a character (like the left parenthesis) is seen, horizontal mode is entered and `\everypar` is expanded which in turn can involve macros. You can see that in the second block (that is: unless we changed `\everypar` in the meantime).

```

1(1)
2(1) (2)
3(1) (2) (3)

```

```

(1) 0
(1) (2) 2
(1) (2) (3) 3

```

### 438 `\lastatomclass`

This returns the class number of the last atom seen in the math input parser.

### 439 `\lastboundary`

This primitive looks back in the list for a user boundary injected with `\boundary` and when seen it returns that value or otherwise zero.

### 440 `\lastbox`

When issued this primitive will, if possible, pull the last box from the current list.

### 441 `\lastchkdimension`

When the last check for a dimension with `\ifchkdimension` was successful this primitive returns the value.

## 442 `\lastchknumber`

When the last check for an integer with `\ifchknumber` was successful this primitive returns the value.

## 443 `\lastkern`

This returns the last kern seen in the list (if possible).

## 444 `\lastleftclass`

This variable registers the first applied math class in a formula.

## 445 `\lastlinefit`

The  $\varepsilon$ -T<sub>E</sub>X manuals explains this parameter in detail but in practice it is enough to know that when set to 1000 spaces in the last line might match those in the previous line. Basically it counters the strong push of a `\parfillskip`.

## 446 `\lastloopiterator`

In addition to `\currentloopiterator` we have a variant that stores the value in case an unexpanded loop is used:

```
\localcontrolledrepeat 8 { [\the\currentloopiterator\eq\the\lastloopiterator] }
\expandedrepeat        8 { [\the\currentloopiterator\eq\the\lastloopiterator] }
\unexpandedrepeat      8 { [\the\currentloopiterator\ne\the\lastloopiterator] }
```

```
[1=1] [2=2] [3=3] [4=4] [5=5] [6=6] [7=7] [8=8]
[1=1] [2=2] [3=3] [4=4] [5=5] [6=6] [7=7] [8=8]
[0≠1] [0≠2] [0≠3] [0≠4] [0≠5] [0≠6] [0≠7] [0≠8]
```

## 447 `\lastnamedcs`

The example code in the previous section has some redundancy, in the sense that there to be looked up control sequence name `mymacro` is assembled twice. This is no big deal in a traditional eight bit T<sub>E</sub>X but in a Unicode engine multi-byte sequences demand some more processing (although it is unlikely that control sequences have many multi-byte utf8 characters).

```
\ifcsname mymacro\endcsname
  \csname mymacro\endcsname
\fi
```

Instead we can say:

```
\ifcsname mymacro\endcsname
  \lastnamedcs
\fi
```

Although there can be some performance benefits another advantage is that it uses less tokens and parsing. It might even look nicer.

## 448 `\lastnodesubtype`

When possible this returns the subtype of the last node in the current node list. Possible values can be queried (for each node type) via Lua helpers.

## 449 `\lastnodetype`

When possible this returns the type of the last node in the current node list. Possible values can be queried via Lua helpers.

## 450 `\lastpageextra`

This reports the last applied (permitted) overshoot.

## 451 `\lastparcontext`

When a paragraph is wrapped up the reason is reported by this state variable. Possible values are:

0x00	normal	0x04	dbbox	0x08	output	0x0B	span
0x01	vmode	0x05	vcenter	0x09	align	0x0C	reset
0x02	vbox	0x06	vadjust	0x0A	noalign		
0x03	vtop	0x07	insert				

## 452 `\lastpartrigger`

There are several reasons for entering a paragraphs and some are automatic and triggered by other commands that force  $\TeX$  into horizontal mode.

0x00	normal	0x04	mathchar	0x08	math	0x0C	valign
0x01	force	0x05	char	0x09	kern	0x0D	vrule
0x02	indent	0x06	boundary	0x0A	hskip		
0x03	noindent	0x07	space	0x0B	unhbox		

## 453 `\lastpenalty`

This returns the last penalty seen in the list (if possible).

## 454 `\lastrightclass`

This variable registers the last applied math class in a formula.

## 455 `\lastskip`

This returns the last glue seen in the list (if possible).

**456 \lccode**

When the `\lowercase` operation is applied the lowercase code of a character is used for the replacement. This primitive is used to set that code, so it expects two character number. The code is also used to determine what characters make a word suitable for hyphenation, although in LuaTeX we introduced the `\hj` code for that.

**457 \leaders**

See `\gleaders` for an explanation.

**458 \left**

Inserts the given delimiter as left fence in a math formula.

**459 \lefthyphenmin**

This is the minimum number of characters after the last hyphen in a hyphenated word.

**460 \leftmarginkern**

The dimension returned is the protrusion kern that has been added (if at all) to the left of the content in the given box.

**461 \leftskip**

This skip will be inserted at the left of every line.

**462 \leqno**

This primitive stores the (typeset) content (presumably a number) and when the display formula is wrapped that number will end up left of the formula.

**463 \let**

Where a `\def` creates a new macro, either or not with argument, a `\let` creates an alias. You are not limited to aliasing macros, basically everything can be aliased.

**464 \letcharcode**

Assigning a meaning to an active character can sometimes be a bit cumbersome; think of using some documented uppercase magic that one tends to forget as it's used only a few times and then never looked at again. So we have this:

```
{\letcharcode 65 1 \catcode 65 13 A : \meaning A}\crlf
{\letcharcode 65 2 \catcode 65 13 A : \meaning A}\par
```

here we define A as an active charcter with meaning 1 in the first line and 2 in the second.

- 1 : the character U+0031 1
- 2 : the character U+0032 2

Normally one will assign a control sequence:

```
{\letcharcode 66 \bf \catcode 66 13 {B bold}: \meaning B}\crlf
{\letcharcode 73 \it \catcode 73 13 {I italic}: \meaning I}\par
```

Of course `\bf` and `\it` are ConT<sub>E</sub>Xt specific commands:

```
bold: protected macro:\ifmmode \expandafter \mathbf \else \expandafter \normalbf \fi
italic: protected macro:\ifmmode \expandafter \mathit \else \expandafter \normalit \fi
```

## 465 \letcsname

It is easy to see that we save two tokens when we use this primitive. As with the `..defcs..` variants it also saves a push back of the composed macro name.

```
\expandafter\let\csname MyMacro:1\endcsname\relax
\letcsname MyMacro:1\endcsname\relax
```

## 466 \letfrozen

You can explicitly freeze an unfrozen macro:

```
\def\MyMacro{...}
\letfrozen\MyMacro
```

A redefinition will now give:

! You can't redefine a frozen macro.

## 467 \letmathatomrule

You can change the class for a specific style. This probably only makes sense for user classes. It's one of those features that we used when experimenting with more control.

```
\letmathatomrule 4 = 4 4 0 0
\letmathatomrule 5 = 5 5 0 0
```

This changes the classes 4 and 5 into class 0 in the two script styles and keeps them the same in display and text. We leave it to the reader to ponder how useful this is.

## 468 \letmathparent

This primitive takes five arguments: the target class, and four classes that determine the pre penalty class, post penalty class, options class and a dummy class for future use.

## 469 \letmathspacing

By default inter-class spacing inherits from the ordinary class but you can remap specific combinations is you want:

```
\letmathspacing \mathfunctioncode
\mathordinarycode \mathordinarycode
\mathordinarycode \mathordinarycode
```

The first value is the target class, and the nest four tell how it behaves in display, text, script and script script style. Here `\mathfunctioncode` is a ConT<sub>E</sub>Xt specific class (26), one of the many.

## 470 \letprotected

Say that you have these definitions:

```
\def \MyMacroA{alpha}
\protected \def \MyMacroB{beta}
\edef \MyMacroC{\MyMacroA\MyMacroB}
\letprotected \MyMacroA
\edef \MyMacroD{\MyMacroA\MyMacroB}
\meaning \MyMacroC\crlf
\meaning \MyMacroD\par
```

The typeset meaning in this example is:

```
macro:alpha\MyMacroB
macro:\MyMacroA \MyMacroB
```

## 471 \lettolastnamedcs

The `\lastnamedcs` primitive is somewhat special as it is a (possible) reference to a control sequence which is why we have a dedicated variant of `\let`.

```
\csname relax\endcsname\let \foo\lastnamedcs \meaning\foo
\csname relax\endcsname\expandafter\let\expandafter \oof\lastnamedcs \meaning\oof
\csname relax\endcsname\lettolastnamedcs \ofo \meaning\ofo
```

These give the following where the first one obviously is not doing what we want and the second one is kind of cumbersome.

```
\lastnamedcs
\relax
\relax
```

## 472 \lettonothing

This one let's a control sequence to nothing. Assuming that `\empty` is indeed empty, these two lines are equivalent.

```
\let \foo\empty
\lettonothing\oof
```

## 473 \limits

This is a modifier: it flags the previous math atom to have its scripts above and below the (summation, product, integral etc.) symbol. In LuaMetaT<sub>E</sub>X this can be any atom (that is: any class). In display mode the location defaults to above and below.

Like any modifier it looks back for a math specific element. This means that the following will work well:

```
\sum \limits ^2 _3
\sum ^2 \limits _3
\sum ^2 _3 \limits
\sum ^2 _3 \limits \nolimits \limits
```

because scripts are bound to these elements so looking back just sees the element.

## 474 `\linebreakcriterion`

The par builder has a concept of ‘compatible’ lines, and has categories for that: loose, decent and tight. In LuaMetaTeX we also have semi-loose and semi-tight as intermediate categories. This parameter can set the ranges for those, using four double bytes:

0x7F000000	semi tight	12
0x007F0000	decent	12
0x00007F00	semi loose	12
0x0000007F	loose	99

The (decimal) default values are given in the last column. Don't expect a big influence from changing these and this option is mostly a side effect of experiments.

## 475 `\linebreakoptional`

This selects the optional text range that is to be used. Optional content is marked with optionalboundary nodes.

## 476 `\linebreakpasses`

When set to a positive value it will apply additional line break runs defined with `\parpasses` until the criteria set in there are met. When set to  $-1$  it will signal a final pass

## 477 `\linedirection`

This sets the text direction (1 for r2l) to the given value but keeps preceding glue into the range.

## 478 `\linepenalty`

Every line gets this penalty attached, so normally it is a small value, like here: 10.

## 479 `\lineskip`

This is the amount of glue that gets added when the distance between lines falls below `\lineskiplimit`.



## 480 \lineskiplimit

When the distance between two lines becomes less than `\lineskiplimit` a `\lineskip` glue item is added.

```
\ruledvbox{
  \lineskiplimit 0pt \lineskip3pt \baselineskip0pt
  \ruledhbox{line 1}
  \ruledhbox{line 2}
  \ruledhbox{\textcolor{red}{line 3}}
}
```

Normally the `\baselineskip` kicks in first but here we've set that to zero, so we get two times a 3pt glue injected.



## 481 \localcontrol

This primitive takes a single token:

```
\edef\testa{\scratchcounter123 \the\scratchcounter}
\edef\testc{\textcolor{red}{\testa} \the\scratchcounter}
\edef\testd{\localcontrol\testa \the\scratchcounter}
```

The three meanings are:

123

```
\testa macro:\scratchcounter 123 123
\testc macro:\scratchcounter 123 123123
\testd macro:123
```

The `\localcontrol` makes that the following token gets expanded so we don't see the yet to be expanded assignment show up in the macro body.

## 482 \localcontrolled

The previously described local control feature comes with two extra helpers. The `\localcontrolled` primitive takes a token list and wraps this into a local control sidetrack. For example:

```
\edef\testa{\scratchcounter123 \the\scratchcounter}
\edef\testb{\localcontrolled{\scratchcounter123}\the\scratchcounter}
```

The two meanings are:

```
\testa macro:\scratchcounter 123 123
\testb macro:123
```

The assignment is applied immediately in the expanded definition.

## 483 `\localcontrolledendless`

As the name indicates this will loop forever. You need to explicitly quit the loop with `\quitloop` or `\quitloopnow`. The first quitter aborts the loop at the start of a next iteration, the second one tries to exit immediately, but is sensitive for interference with for instance nested conditionals.

## 484 `\localcontrolledloop`

As with more of the primitives discussed here, there is a manual in the ‘lowlevel’ subset that goes into more detail. So, here a simple example has to do:

```
\localcontrolledloop 1 100 1 {%
  \ifnum\currentloopiterator>6\relax
    \quitloop
  \else
    [\number\currentloopnesting:\number\currentloopiterator]
    \localcontrolledloop 1 8 1 {%
      (\number\currentloopnesting:\number\currentloopiterator)
    }\par
  \fi
}
```

Here we see the main loop primitive being used nested. The code shows how we can `\quitloop` and have access to the `\currentloopiterator` as well as the nesting depth `\currentloopnesting`.

```
[1:1] (2:1) (2:2) (2:3) (2:4) (2:5) (2:6) (2:7) (2:8)
[1:2] (2:1) (2:2) (2:3) (2:4) (2:5) (2:6) (2:7) (2:8)
[1:3] (2:1) (2:2) (2:3) (2:4) (2:5) (2:6) (2:7) (2:8)
[1:4] (2:1) (2:2) (2:3) (2:4) (2:5) (2:6) (2:7) (2:8)
[1:5] (2:1) (2:2) (2:3) (2:4) (2:5) (2:6) (2:7) (2:8)
[1:6] (2:1) (2:2) (2:3) (2:4) (2:5) (2:6) (2:7) (2:8)
```

Be aware of the fact that `\quitloop` will end the loop at the *next* iteration so any content after it will show up. Normally this one will be issued in a condition and we want to end that properly. Also keep in mind that because we use local control (a nested T<sub>E</sub>X expansion loop) anything you feed back can be injected out of order.

The three numbers can be separated by an equal sign which is a trick to avoid look ahead issues that can result from multiple serialized numbers without spaces that indicate the end of sequence of digits.

## 485 `\localcontrolledrepeat`

This one takes one instead three arguments which looks a bit better in simple looping.

## 486 `\localleftbox`

This sets the box that gets injected at the left of every line.

## 487 `\localleftboxbox`

This returns the box set with `\localleftbox`.

**488 \localmiddlebox**

This sets the box that gets injected at the left of every line but its width is ignored.

**489 \localmiddleboxbox**

This returns the box set with \localmiddlebox.

**490 \localrightbox**

This sets the box that gets injected at the right of every line.

**491 \localrightboxbox**

This returns the box set with \localrightbox.

**492 \long**

This original prefix gave the macro being defined the property that it could not have \par (or the often equivalent empty lines) in its arguments. It was mostly a protection against a forgotten right curly brace, resulting in a so called run-away argument. That mattered on a paper terminal or slow system where such a situation should be caught early. In LuaT<sub>E</sub>X it was already optional, and in LuaMetaT<sub>E</sub>X we dropped this feature completely (so that we could introduce others).

**493 \looseness**

The number of lines in the current paragraph will be increased by given number of lines. For this to succeed there need to be enough stretch in the spacing to make that happen. There is some wishful thinking involved.

**494 \lower**

This primitive takes two arguments, a dimension and a box. The box is moved down. The operation only succeeds in horizontal mode.

**495 \lowercase**

This token processor converts character tokens to their lowercase counterparts as defined per \lc-code. In order to permit dirty tricks active characters are also processed. We don't really use this primitive in ConT<sub>E</sub>Xt, but for consistency we let it respond to \expand:<sup>4</sup>

```
\edef          \foo          {\lowercase{tex TeX \TEX}} \meaningless\foo
\lowercase{\edef\foo          {tex TeX \TEX}} \meaningless\foo
\edef          \foo{\expand\lowercase{tex TeX \TEX}} \meaningless\foo
```

Watch how \lowercase is not expandable but can be forced to. Of course, as the logo macro is protected the T<sub>E</sub>X logo remains mixed case.

<sup>4</sup> Instead of providing \lowercased and \uppercased primitives that would clash with macros anyway.

```
\lowercase {tex TeX \TeX }
tex tex \TeX
tex tex \TeX
```

## 496 \lpcode

This one can be used to set the left protrusion factor of a glyph in a font and takes three arguments: font, character code and factor. It is kind of obsolete because we can set up vectors at definition time and tweaking from T<sub>E</sub>X can have side effects because it globally adapts the font.

## 497 \lua bytecode

This behaves like `\luafunction` but here the number is a byte code register. These bytecodes are in the `lua.bytecode` array.

## 498 \lua bytecodecall

This behaves like `\luafunctioncall` but here the number is a byte code register. These bytecodes are in the `lua.bytecode` array.

## 499 \luacopyinputnodes

When set to a positive value this will ensure that when nodes are printed from Lua to T<sub>E</sub>X copies are used.

## 500 \lua def

This command relates a (user) command to a Lua function registered in the `lua.lualib_get_functions_table()`, so after:

```
\lua def\foo123
```

the `\foo` command will trigger the function at index 123. Of course a macro package has to make sure that these definitions are unique.<sup>5</sup>

This command is accompanied by `\luafunctioncall` and `\luafunction`. When we have function 123 defined as

```
function() tex.sprint("!") end
```

the following:

```
(\luafunctioncall \foocode ?)
(\normalluafunction\foocode ?)
(\foo ?)
```

gives three times (!?). But this:

---

<sup>5</sup> Plain T<sub>E</sub>X established a norm for allocating registers, like `\newdimen` but there is no such convention for Lua functions.

```

\edef\oof{\foo } \meaning\oof % protected
\edef\oof{\luafunctioncall \foocode} \meaning\oof % protected
\edef\oof{\normalluafunction\foocode} \meaning\oof % expands

```

returns:

```

macro:!
macro:\luafunctioncall 1740
macro:!

```

Because the definition command is like any other

```
\permanent\protected\luaodef\foo123
```

boils down to:

```
permanent protected luacall 123
```

## 501 \luaescapestring

This command converts the given (token) list into something that is acceptable for Lua. It is inherited from Lua<sub>TEX</sub> and not used in Con<sub>TEX</sub>t.

```
\directlua { tex.print ("luaescapestring {\tt This is a "test".}) }
```

Results in: This is a "test". (Watch the grouping.)

## 502 \luafunction

The integer passed to this primitive is the index in the table returned by `lua.luaolib_get_functions_table()`. Of course a macro package has to provide reliable management for this. This is a so called convert command so it expands in an expansion context (like an `\edef`).

## 503 \luafunctioncall

The integer passed to this primitive is the index in the table returned by `lua.luaolib_get_functions_table()`. Of course a macro package has to provide reliable management for this. This primitive doesn't expand in an expansion context (like an `\edef`).

## 504 \luatexbanner

This gives: This is LuaMetaTeX, Version 2.10.11.

## 505 \luatexrevision

This is an integer. The current value is: 10.

## 506 \luatexversion

This is an integer. The current value is: 210.

## 507 `\mark`

The given token list is stored in a node in the current list and might become content of `\topmark`, `\botmark` or `\firstmark` when a page split off, or in the case of a box split in `\splitbotmark` or `\splitfirstmark`. In LuaMetaTeX deeply burried marks bubbly up to an outer box level.

## 508 `\marks`

This command is similar to `\mark` but first expects a number of a mark register. Multiple marks were introduced in  $\varepsilon$ -TeX.

## 509 `\mathaccent`

This takes a number and a math object to put the accent on. The four byte number has a dummy class byte, a family byte and two index bytes. It is replaced by `\Umathaccent` that handles wide fonts.

## 510 `\mathatom`

This operation wraps following content in a atom with the given class. It is part of LuaMetaTeX's extended math support. There are three class related key/values: `class`, `leftclass` and `rightclass` (or all for all of them). When none is given this command expects a class number before scanning the content. The options key expects a bitset but there are also direct option keys, like `limits`, `nolimits`, `unpack`, `unroll`, `single`, `nooverflow`, `void` and `phantom`. A source id can be set, one or more attr assigned, and for specific purposes `textfont` and `mathfont` directives are accepted. Features like this are discussed in dedicated manuals.

## 511 `\mathatomglue`

This returns the glue that will be inserted between two atoms of a given class for a specific style.

```
\the\mathatomglue \textstyle 1 1
\the\mathatomglue \textstyle 0 2
\the\mathatomglue \scriptstyle 1 1
\the\mathatomglue \scriptstyle 0 2

1.66667mu
2.22223mu plus 1.11111mu minus 1.11111mu
1.66667mu
0.55556mu minus 0.27777mu
```

## 512 `\mathatomskip`

This injects a glue with the given style and class pair specification: `xx xx xx xx xx xx`.

```
$x x$
$x \mathatomskip \textstyle 1 1 x$
$x \mathatomskip \textstyle 0 2 x$
$x \mathatomskip \scriptstyle 1 1 x$
$x \mathatomskip \scriptstyle 0 2 x$
```

### 513 `\mathbackwardpenalties`

See `\mathforwardpenalties` for an explanation.

### 514 `\mathbeginclass`

This variable can be set to signal the class that starts the formula (think of an imaginary leading atom).

### 515 `\mathbin`

This operation wraps following content in a atom with class ‘binary’.

### 516 `\mathboundary`

This primitive is part of an experiment with granular penalties in math. When set nested fences will use the `\mathdisplaypenaltyfactor` or `\mathinlinepenaltyfactor` to increase nested penalties. A bit more control is possible with `\mathboundary`:

```
0 begin factor 1000
1 end factor 1000
2 begin given factor
3 end given factor
```

These will be used when the mentioned factors are zero.

### 517 `\mathchar`

Replaced by `\Umathchar` this old one takes a four byte number: one byte for the class, one for the family and two for the index. The specified character is appended to the list.

### 518 `\mathcharclass`

Returns the slot (in the font) of the given math character.

```
\the\mathcharclass\Umathchar 4 2 123
```

The first passed number is the class, so we get: 4.

### 519 `\mathchardef`

Replaced by `\Umathchardef` this primitive relates a control sequence with a four byte number: one byte for the class, one for the family and two for the index. The defined command will insert that character.

### 520 `\mathcharfam`

Returns the family number of the given math character.

```
\the\mathcharfam\Umathchar 4 2 123
```

The second passed number is the family, so we get: 2.

## 521 `\mathcharslot`

Returns the slot (or index in the font) of the given math character.

`\the\mathcharslot\Umathchar 4 2 123`

The third passed number is the slot, so we get: 123.

## 522 `\mathcheckfencesmode`

When set to a positive value there will be no warning if a right fence (`\right` or `\Uright`) is missing.

## 523 `\mathchoice`

This command expects four subformulas, for display, text, script and scriptscript and it will eventually use one of them depending on circumstances later on. Keep in mind that a formula is first scanned and when that is finished the analysis and typesetting happens.

## 524 `\mathclass`

There are build in classes and user classes. The first possible user class is 20 and the last one is 60. You can better not touch the special classes ‘all’ (61), ‘begin’ (62) and ‘end’ (63). The basic 8 classes that original  $\TeX$  provides are of course also present in LuaMeta $\TeX$ . In addition we have some that relate to constructs that the engine builds.

---

ordinary	ord	0	the default
operator	op	1	small and large operators
binary	bin	2	
relation	rel	3	
open		4	
close		5	
punctuation	punct	6	
variable		7	adapts to the current family
active		8	character marked as such becomes active
inner		9	this class is not possible for characters

---

under	10
over	11
fraction	12
radical	13
middle	14
accent	16
fenced	17
ghost	18
vcenter	19

---



There is no standard for user classes but ConT<sub>E</sub>Xt users should be aware of quite some additional ones that are set up. The engine initialized the default properties of classes (spacing, penalties, etc.) the same as original T<sub>E</sub>X.

Normally characters have class bound to them but you can (temporarily) overload that one. The `\mathclass` primitive expects a class number and a valid character number or math character and inserts the symbol as if it were of the given class; so the original class is replaced.

`\ruledhbox{$(x)$}` and `\ruledhbox{$\mathclass 1 `(x\mathclass 1 `)$}`

Changing the class is likely to change the spacing, compare  $(x)$  and  $(\mathclass 1 `(x\mathclass 1 `))$ .

## 525 `\mathclose`

This operation wraps following content in a atom with class ‘close’.

## 526 `\mathcode`

This maps a character to one in a family: the assigned value has one byte for the class, one for the family and two for the index. It has little use in an OpenType math setup.

## 527 `\mathdictgroup`

This is an experimental feature that in due time will be explored in ConT<sub>E</sub>Xt. It currently has no consequences for rendering.

## 528 `\mathdictionary`

This is an experimental feature that in due time will be explored in ConT<sub>E</sub>Xt. It currently has no consequences for rendering.

## 529 `\mathdictproperties`

This is an experimental feature that in due time will be explored in ConT<sub>E</sub>Xt. It currently has no consequences for rendering.

## 530 `\mathdirection`

When set to 1 this will result in r2l typeset math formulas but of course you then also need to set up math accordingly (which is the case in ConT<sub>E</sub>Xt).

## 531 `\mathdisplaymode`

Display mode is entered with two dollars (other characters can be used but the dollars are a convention). Mid paragraph display formulas get a different treatment with respect to the width and indentation than stand alone. When `\mathdisplaymode` is larger than zero the double dollars (or equivalents) will behave as inline formulas starting out in `\displaystyle` and with `\everydisplay` expanded.

## 532 `\mathdisplaypenaltyfactor`

This one is similar to `\mathinlinepenaltyfactor` but is used when we're in display style.

## 533 `\mathdisplayskipmode`

A display formula is preceded and followed by vertical glue specified by `\abovedisplayskip` and `\belowdisplayskip` or `\abovedisplayshortskip` and `\belowdisplayshortskip`. Spacing 'above' is always inserted, even when zero, but the spacing 'below' is only inserted when it is non-zero. There's also `\baselineskip` involved. The way spacing is handled can be influenced with `\mathdisplayskipmode`, which takes the following values:

- 0 does the same as any  $\TeX$  engine
- 1 idem
- 2 only insert spacing when it is not zero
- 3 never insert spacing

## 534 `\mathdoublescriptmode`

When this parameter has a negative value double scripts trigger an error, so with `\superscript`, `\nosuperscript`, `\shiftedsuperscript`, `\superprescript`, `\nosuperprescript`, `\shiftedsuperprescript`, `\subscript`, `\nosubscript`, `\shiftedsubscript`, `\subprescript`, `\nosubprescript`, `\shiftedsubprescript` and `\primescript`, as well as their (multiple) `_` and `^` aliases.

A value of zero does the normal and inserts a dummy atom (basically a `{}`) but a positive value is more interesting. Compare these:

```
{\mathdoublescriptmode 0      $x_x_x$}
{\mathdoublescriptmode"000000 $x_x_x$}
{\mathdoublescriptmode"030303 $x_x_x$}
{$x_x_x$}
```

The three pairs of bytes indicate the main class, left side class and right side class of the inserted atom, so we get this:  $x_{xx} x_{xx} x_{xx}$ . The last line gives what Con $\TeX$ t is configured for.

## 535 `\mathendclass`

This variable can be set to signal the class that ends the formula (think of an imaginary trailing atom).

## 536 `\matheqnogapstep`

The display formula number placement heuristic puts the number on the same line when there is place and then separates it by a quad. In Lua $\TeX$  we decided to keep that quantity as it can be tight into the math font metrics but introduce a multiplier `\matheqnogapstep` that defaults to 1000.

## 537 `\mathfontcontrol`

This bitset controls how the math engine deals with fonts, and provides a way around dealing with inconsistencies in the way they are set up. The `\fontmathcontrol` makes it possible to bind options

ot a specific math font. In practice, we just set up the general approach which ii possible because we normalize the math fonts and ‘fix’ issues at runtime.

```

0x00000001 usefontcontrol
0x00000002 overrule
0x00000004 underrule
0x00000008 radicalrule
0x00000010 fractionrule
0x00000020 accentskewhalf
0x00000040 accentskewapply
0x00000080 applyordinarykernpair
0x00000100 applyverticalitalickern
0x00000200 applyordinaryitalickern
0x00000400 applycharitalickern
0x00000800 reboxcharitalickern
0x00001000 applyboxeditalickern
0x00002000 staircasekern
0x00004000 applytextitalickern
0x00008000 checktextitalickern
0x00010000 checkspaceitalickern
0x00020000 applyscriptitalickern
0x00040000 analyzescrptnucleuschar
0x00080000 analyzescrptnucleuslist
0x00100000 analyzescrptnucleusbox
0x00200000 accenttopskewwithoffset
0x00400000 ignorekerndimensions
0x00800000 ignoreflataccents
0x01000000 extendaccents
0x02000000 extenddelimiters

```

## 538 \mathforwardpenalties

Inline math can have multiple atoms and constructs and one can configure the penalties between then bases on classes. In addition it is possible to configure additional penalties starting from the beginning or end using `\mathforwardpenalties` and `\mathbackwardpenalties`. This is one the features that we added in the perspective of breaking paragraphs heavy on math into lines. It not that easy to come up with useable values.

## 539 \mathgluemode

We can influence the way math glue is handled. By default stretch and shrink is applied but this variable can be used to change that. The limit option ensures that the stretch and shrink doesn't go beyond their natural values.

```

0x01 stretch
0x02 shrink
0x04 limit

```

## 540 `\mathgroupingmode`

Normally a `{ }` or `\bgroup-\egroup` pair in math create a math list. However, users are accustomed to using it also for grouping and then a list being created might not be what a user wants. As an alternative to the more verbose `\begingroup-\endgroup` or even less sensitive `\beginmathgroup-\endmathgroup` you can set the math grouping mode to a non zero value which makes curly braces (and the aliases) behave as expected.

## 541 `\mathinlinepenaltyfactor`

A math formula can have nested (sub)formulas and one might want to discourage a line break inside those. If this value is non zero it becomes a multiplier, so a value of 1000 will make an inter class penalty of 100 into 200 when at nesting level 2 and 500 when at level 5.

## 542 `\mathinner`

This operation wraps following content in a atom with class ‘inner’. In LuaMetaTeX we have more classes and this general wrapper one is therefore kind of redundant.

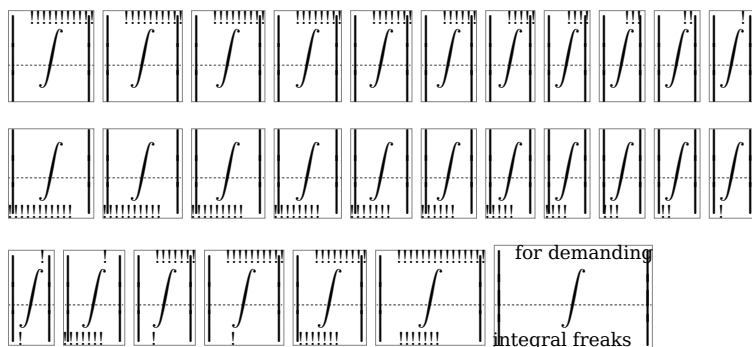
## 543 `\mathleftclass`

When set this class will be used when a formula starts.

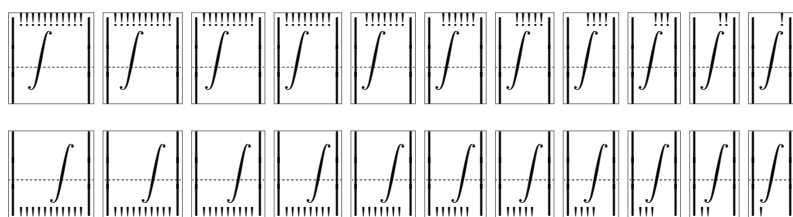
## 544 `\mathlimitsmode`

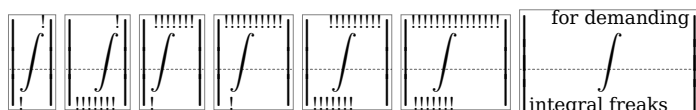
When this parameter is set to a value larger than zero real dimensions are used and longer limits will not stick out, which is a traditional T<sub>E</sub>X feature. We could have more advanced control but this will do.

Compare the zero setting:



with the positive variant:





Here we switched to Latin Modern because it's font dependent how serious this issue is. In Pagella all is fine in both modes.

## 545 `\mathmainstyle`

This inspector returns the outermost math style (contrary to `\mathstyle`), as we can see in the next examples where use these snippets:

```
\def\foo{(\the\mathmainstyle,\the\mathstyle)}
\def\oof{\sqrt{\foo}{\foo}}
\def\ofo{\frac{\foo}{\foo}}
\def\fof{\mathchoice{\foo}{\foo}{\foo}{\foo}}
```

When we use the regular math triggers we get this:

```
$\displaystyle \foo + \oof + \ofo$
$\textstyle \foo + \oof + \ofo$
$\displaystyle \foo + \fof$
$\textstyle \foo + \fof$
$\scriptstyle \foo + \fof$
$\scriptscriptstyle\foo + \fof$
```

```
(2, 0) + (2, 0)√(2, 1) + (2, 5)
(2, 2) + (2, 2)√(2, 3) + (2, 5)
(2, 0) + (2, 0)
(2, 2) + (2, 2)
(2, 4) + (2, 4)
(2, 6) + (2, 6)
```

But we can also do this:

```
\Ustartmathmode \displaystyle \foo + \oof + \ofo \Ustopmathmode
\Ustartmathmode \textstyle \foo + \oof + \ofo \Ustopmathmode
\Ustartmathmode \displaystyle \foo + \fof \Ustopmathmode
\Ustartmathmode \textstyle \foo + \fof \Ustopmathmode
\Ustartmathmode \scriptstyle \foo + \fof \Ustopmathmode
\Ustartmathmode \scriptscriptstyle\foo + \fof \Ustopmathmode
```

```
(0, 0) + (0, 0)√(0, 1) + (0, 5)
(2, 2) + (2, 2)√(2, 3) + (2, 5)
(0, 0) + (0, 0)
(2, 2) + (2, 2)
(4, 4) + (4, 4)
(6, 6) + (6, 6)
```

## 546 `\mathnolimitsmode`

This parameter influences the placement of scripts after an operator. The reason we have this lays in the fact that traditional T<sub>E</sub>X uses italic correction and OpenType math does the same but fonts are not

consistent in how they set this up. Actually, in OpenType math it's the only reason that there is italic correction. Say that we have a shift  $\delta$  determined by the italic correction:

mode	top	bottom
0	0	$-\delta$
1	$\delta \times f_t$	$\delta \times f_b$
2	0	0
3	0	$-\delta/2$
4	$\delta/2$	$-\delta/2$
> 15	0	$-n \times \delta/1000$

Mode 1 uses two font parameters:  $f_b$ : `\Umathnolimitsubfactor` and  $f_t$ : `\Umathnolimitsupfactor`.

## 547 `\mathop`

This operation wraps following content in a atom with class ‘operator’.

## 548 `\mathopen`

This operation wraps following content in a atom with class ‘open’.

## 549 `\mathord`

This operation wraps following content in a atom with class ‘ordinary’.

## 550 `\mathparentstyle`

This inspector returns the math style used in a construct, so is either equivalent to `\mathmainstyle` or a nested `\mathstyle`. For instance in a nested fraction we get this (in ConT<sub>E</sub>Xt) in display formulas:

$$\frac{\frac{\frac{(0,1,1)}{(0,1,1)}}{(0,1,1)}}{(0,1,1)} + (0,0,0)$$

but this in inline formulas:

$$\frac{\frac{\frac{(2,5,7)}{(2,5,7)}}{(2,5,7)}}{(2,5,7)} + (2,2,2)$$

where the first element in a nested fraction.

## 551 `\mathpenaltiesmode`

Normally the T<sub>E</sub>X math engine only inserts penalties when in textstyle. You can force penalties in displaystyle with this parameter. In inline math we always honor penalties, with mode 0 and mode 1 we get this:

$$\begin{array}{l} x + 2x = 0 \\ x + 2x = 1 \end{array}$$

However in ConT<sub>E</sub>Xt, where all is done in inline math mode, we set this parameter to 1, otherwise we wouldn't get these penalties, as shown next:

$$x + 2x = 0$$

$$x + 2x = 1$$

If one uses a callback it is possible to force penalties from there too.

## 552 `\mathpretolerance`

This is used instead of `\pretolerance` when a breakpoint is calculated when a math formula starts.

## 553 `\mathpunct`

This operation wraps following content in a atom with class 'punctuation'.

## 554 `\mathrel`

This operation wraps following content in a atom with class 'relation'.

## 555 `\mathrightclass`

When set this class will be used when a formula ends.

## 556 `\mathrulesfam`

When set, this family will be used for setting rule properties in fractions, under and over.

## 557 `\mathrulesmode`

When set to a non zero value rules (as in fractions and radicals) will be based on the font parameters in the current family.

## 558 `\mathscale`

In LuaMetaT<sub>E</sub>X we can either have a family of three (text, script and scriptscript) fonts or we can use one font that we scale and where we also pass information about alternative shapes for the smaller sizes. When we use this more compact mode this primitive reflects the scale factor used.

What gets reported depends on how math is implemented, where in ConT<sub>E</sub>Xt we can have either normal or compact mode: 1000 700 550 1000 700 550. In compact mode we have the same font three times so then it doesn't matter which of the three is passed.

## 559 `\mathscriptsmode`

There are situations where you don't want T<sub>E</sub>X to be clever and optimize the position of super- and subscripts by shifting. This parameter can be used to influence this.

$0: x_2^2 + y_x^x + z_2 + w^2$	$0: x_2^2 + y_x^x + z_2 + w^2$	$2: x_2^2 + y_x^x + z_2 + w^2$
$0: x_f^f + y_x^x + z_f + w^f$	$0: x_f^f + y_x^x + z_f + w^f$	$2: x_f^f + y_x^x + z_f + w^f$
1 over 0	2 over 0	2 over 1

The next table shows what parameters kick in when:

	or (1)	and (2)	otherwise
<b>super</b>	sup shift up	sup shift up	sup shift up, sup bot min
<b>sub</b>	sub shift down	sub sup shift down	sub shift down, sub top max
<b>both</b>	sub shift down	sub sup shift down	sub sup shift down, sub sup vgap, sup sub bot max

## 560 `\mathslackmode`

When positive this parameter will make sure that script spacing is discarded when there is no reason to add it.

$x^2 + x^2 x^2$	$x^2 + x^2 x^2$	$x^2 + x^2 x^2$
disabled (0)	enabled (1)	enabled over disabled

## 561 `\mathspacingmode`

Zero inter-class glue is not injected but setting this parameter to a positive value bypasses that check. This can be handy when checking (tracing) how (and what) spacing is applied. Keep in mind that glue in math is special in the sense that it is not a valid breakpoint. Line breaks in (inline) math are driven by penalties.

## 562 `\mathstack`

There are a few commands in  $\text{T}_{\text{E}}\text{X}$  that can behave confusing due to the way they are scanned. Compare these:

```
$ 1 \over 2 $
$ 1 + x \over 2 + x $
$ {1 + x} \over {2 + x} $
$ {{1 + x} \over {2 + x}} $
```

A single 1 is an atom as is the curly braced 1 + x. The two arguments to `\over` eventually will get typeset in the style that this fraction constructor uses for the numerator and denominator but one might actually also like to relate that to the circumstances. It is comparable to using a `\mathchoice`. In order not to waste runtime on four variants, which itself can have side effects, for instance when counters are involved,  $\text{LuaT}_{\text{E}}\text{X}$  introduced `\mathstack`, used like:

```
$\mathstack {1 \over 2}$
```

This `\mathstack` command will scan the next brace and opens a new math group with the correct (in this case numerator) math style. The `\mathstackstyle` primitive relates to this feature that defaults to ‘smaller unless already scriptscript’.



### 563 `\mathstackstyle`

This returns the (normally) numerator style but the engine can be configured to default to another style. Although all these in the original T<sub>E</sub>X engines hard coded style values can be changed in Lua-MetaT<sub>E</sub>X it is unlikely to happen. So this primitive will normally return the (current) style ‘smaller unless already scriptscript’.

### 564 `\mathstyle`

This returns the current math style, so `$\the\mathstyle$` gives 2.

### 565 `\mathstylefontid`

This returns the font id (a number) of a style/family combination. What you get back depends on how a macro package implements math fonts.

```
(\the\mathstylefontid\textstyle      \fam)
(\the\mathstylefontid\scriptstyle    \fam)
(\the\mathstylefontid\scriptscriptstyle\fam)
```

In ConT<sub>E</sub>Xt gives: (2) (2) (2).

### 566 `\mathsurround`

The kern injected before and after an inline math formula. In practice it will be set to zero, if only because otherwise nested math will also get that space added. We also have `\mathsurroundskip` which, when set, takes precedence. Spacing is controlled by `\mathsurroundmode`.

### 567 `\mathsurroundmode`

The possible ways to control spacing around inline math formulas in other manuals and mostly serve as playground.

### 568 `\mathsurroundskip`

When set this one wins over `\mathsurround`.

### 569 `\maththreshold`

This is a glue parameter. The amount determines what happens: when it is non zero and the inline formula is less than that value it will become a special kind of box that can stretch and/ or shrink within the given specification. The par builder will use these stretch and/ or shrink components but it is up to one of the Lua callbacks to deal with the content eventually (if at all). As this is somewhat specialized, more details can be found on ConT<sub>E</sub>Xt documentation.

### 570 `\mathtolerance`

This is used instead of `\tolerance` when a breakpoint is calculated when a math formula starts.

## 571 `\maxdeadcycles`

When the output routine is called this many times and no page is shipped out an error will be triggered. You therefore need to reset its companion counter `\deadcycles` if needed. Keep in mind that LuaMeta-TeX has no real `\shipout` because providing a backend is up to the macro package.

## 572 `\maxdepth`

The depth of the page is limited to this value.

## 573 `\meaning`

We start with a primitive that will be used in the following sections. The reported meaning can look a bit different than the one reported by other engines which is a side effect of additional properties and more extensive argument parsing.

```
\tolerant\permanent\protected\gdef\foo[#1]#*[#2]{(#1)(#2)} \meaning\foo
```

```
tolerant protected macro:[#1]#*[#2]->(#1)(#2)
```

## 574 `\meaningasis`

Although it is not really round trip with the original due to information being lost this primitive tries to return an equivalent definition.

```
\tolerant\permanent\protected\gdef\foo[#1]#*[#2]{(#1)(#2)} \meaningasis\foo
```

```
\permanent \tolerant \protected \def \foo [#1]#*[#2]{(#1)(#2)}
```

## 575 `\meaningful`

This one reports a bit less than `\meaningful`.

```
\tolerant\permanent\protected\gdef\foo[#1]#*[#2]{(#1)(#2)} \meaningful\foo
```

```
permanent tolerant protected macro
```

## 576 `\meaningfull`

This one reports a bit more than `\meaning`.

```
\tolerant\permanent\protected\gdef\foo[#1]#*[#2]{(#1)(#2)} \meaningfull\foo
```

```
permanent tolerant protected macro:[#1]#*[#2]->(#1)(#2)
```

## 577 `\meaningless`

This one reports a bit less than `\meaningless`.

```
\tolerant\permanent\protected\gdef\foo[#1]#*[#2]{(#1)(#2)} \meaningless\foo
```

`[#1]#*[#2]`

## 578 `\meaningless`

This one reports a bit less than `\meaning`.

`\tolerant\permanent\protected\gdef\foo[#1]#*[#2]{(#1)(#2)} \meaningless\foo`

`[#1]#*[#2]->(#1)(#2)`

## 579 `\medmuskip`

A predefined mu skip register that can be used in math (inter atom) spacing. The current value is 4.0mu plus 2.0mu minus 2.0mu. In traditional  $\TeX$  most inter atom spacing is hard coded using the predefined registers.

## 580 `\message`

Prints the serialization of the (tokenized) argument to the log file and/or console.

## 581 `\middle`

Inserts the given delimiter as middle fence in a math formula. In LuaMeta $\TeX$  it is a full blown fence and not (as in  $\varepsilon\text{-}\TeX$ ) variation of `\open`.

## 582 `\mkern`

This one injects a kern node in the current (math) list and expects a value in so called mu units.

## 583 `\month`

This internal number starts out with the month that the job started.

## 584 `\moveleft`

This primitive takes two arguments, a dimension and a box. The box is moved to the left. The operation only succeeds in vertical mode.

## 585 `\moveright`

This primitive takes two arguments, a dimension and a box. The box is moved to the right. The operation only succeeds in vertical mode.

## 586 `\mskip`

The given math glue (in mu units) is injected in the horizontal list. For this to succeed we need to be in math mode.

## 587 `\muexpr`

This is a companion of `\glueexpr` so it handles the optional stretch and shrink components. Here math units (mu) are expected.

## 588 `\mugluespecdef`

A variant of `\gluespecdef` that expects mu units is:

```
\mugluespecdef\MyGlue = 3mu plus 2mu minus 1mu
```

The properties are comparable to the ones described in the previous sections.

## 589 `\multiply`

The given quantity is multiplied by the given integer (that can be preceded by the keyword ‘by’, like:

```
\scratchdimen=10pt \multiply\scratchdimen by 3
```

## 590 `\multiplyby`

This is slightly more efficient variant of `\multiply` that doesn't look for by. See previous section.

## 591 `\muskip`

This is the accessor for an indexed muskip (muglue) register.

## 592 `\muskipdef`

This command associates a control sequence with a muskip (math skip) register (accessed by number).

## 593 `\mutable`

This prefix flags what follows can be adapted and is not subjected to overload protection.

## 594 `\mutogluue`

The sequence `\the\mutogluue 20mu plus 10mu minus 5mu` gives 20.0pt plus 10.0pt minus 5.0pt.

## 595 `\nestedloopiterator`

This is one of the accessors of loop iterators:

```
\expandedrepeat 2 {%
  \expandedrepeat 3 {%
    (n=\the\nestedloopiterator 1,
     p=\the\previousloopiterator1,
     c=\the\currentloopiterator)
```

```
}%
}%
```

Gives:

```
(n=1, p=1, c=1) (n=2, p=1, c=2) (n=3, p=1, c=3) (n=1, p=2, c=1) (n=2, p=2, c=2) (n=3, p=2, c=3)
```

Where a nested iterator starts relative to innermost loop, the previous one is relative to the outer loop (which is less predictable because we can already be in a loop).

## 596 `\newlinechar`

When something is printed to one of the log channels the character with this code will trigger a linebreak. That also resets some counters that deal with suppressing redundant ones and possible indentation. Contrary to other engines LuaMetaTeX doesn't bother about the length of lines.

## 597 `\noalign`

The token list passed to this primitive signals that we don't enter a table row yet but for instance in a `\halign` do something between the lines: some calculation or injecting inter-row material. In LuaMetaTeX this primitive can be used nested.

## 598 `\noaligned`

The alignment mechanism is kind of special when it comes to expansion because it has to look ahead for a `\noalign`. This interferes with for instance protected macros, but using this prefix we get around that. Among the reasons to use protected macros inside an alignment is that they behave better inside for instance `\expanded`.

## 599 `\noatomruling`

Spacing in math is based on classes and this primitive inserts a signal that there is no ruling in place here. Basically we have a zero skip glue tagged as non breakable because in math mode glue is not a valid breakpoint unless we have configured inter-class penalties.

## 600 `\noboundary`

This inserts a boundary node with no specific property. It can still serve as boundary but is not interpreted in special ways, like the others.

## 601 `\noexpand`

This prefix prevents expansion in a context where expansion happens. Another way to prevent expansion is to define a macro as `\protected`.

```
\def\foo{foo} \edef\oof{we expanded \foo} \meaning\oof
\def\foo{foo} \edef\oof{we keep \noexpand\foo} \meaning\oof
\protected\def\foo{foo} \edef\oof{we keep \foo} \meaning\oof
```

```
macro:we expanded foo
macro:we keep \foo
macro:we keep \foo
```

## 602 \nohrule

This is a rule but flagged as empty which means that the dimensions kick in as for a normal rule but the backend can decide not to show it.

## 603 \noindent

This starts a paragraph. In Lua $\TeX$  (and LuaMeta $\TeX$ ) a paragraph starts with a so called par node (see `\indent` on how control that. After that comes either `\parindent` glue or a horizontal box. The `\indent` makes gives them some width, while `\noindent` keeps that zero.

## 604 \nolimits

This is a modifier: it flags the previous math atom to have its scripts after the the atom (contrary to `\limits`. In LuaMeta $\TeX$  this can be any atom (that is: any class). In display mode the location defaults to above and below.

## 605 \nonscript

This prevents  $\TeX$  from adding inter-atom glue at this spot in script or scriptscript mode. It actually is a special glue itself that serves as signal.

## 606 \nonstopmode

This directive omits all stops.

## 607 \norelax

The rationale for this command can be shown by a few examples:

```
\dimen0 1pt \dimen2 1pt \dimen4 2pt
\edef\testa{\ifdim\dimen0=\dimen2\norelax N\else Y\fi}
\edef\testb{\ifdim\dimen0=\dimen2\relax N\else Y\fi}
\edef\testc{\ifdim\dimen0=\dimen4\norelax N\else Y\fi}
\edef\testd{\ifdim\dimen0=\dimen4\relax N\else Y\fi}
\edef\teste{\norelax}
```

The five meanings are:

```
\testa macro:N
\testb macro:\relax N
\testc macro:Y
\testd macro:Y
\teste macro:
```

So, the `\norelax` acts like `\relax` but is not pushed back as usual (in some cases).

## 608 `\normalizelinemode`

The  $\text{\TeX}$  engine was not designed to be opened up, and therefore the result of the linebreak effort can differ depending on the conditions. For instance not every line gets the left- or rightskip. The first and last lines have some unique components too. When  $\text{\LuaTeX}$  made it possible too get the (intermediate) result manipulating the result also involved checking what one encountered, for instance glue and its origin. In  $\text{\LuaMetaTeX}$  we can normalize lines so that they have for instance balanced skips.

0x0001	normalizeline	0x0020	removemarginkerns
0x0002	parindent	0x0040	clipwidth
0x0004	swaphangindent	0x0080	flattendiscretionaries
0x0008	swapparshape	0x0100	discardzerotabskips
0x0010	breakafterdir	0x0200	flattenhleaders

The order in which the skips get inserted when we normalize is as follows:

<code>\lefthangskip</code>	the hanging indentation (or zero)
<code>\leftskip</code>	the value even when zero
<code>\parfillleftskip</code>	only on the last line
<code>\parinitleftskip</code>	only on the first line
<code>\indent</code>	the amount of indentation
...	the (optional) content
<code>\parinitrightskip</code>	only on the first line
<code>\parfillrightskip</code>	only on the last line
<code>\correction</code>	the correction needed to stay within the <code>\hsize</code>
<code>\rightskip</code>	the value even when zero
<code>\righthangskip</code>	the hanging indentation (or zero)

The init and fill skips can both show up when we have a single line. The correction skip replaces the traditional juggling with the right skip and shift of the boxed line.

For now we leave the other options to your imagination. Some of these can be achieved by callbacks (as we did in older versions of  $\text{\ConTeXt}$ ) but having the engine do the work we get a better performance.

## 609 `\normalizeparmode`

For now we just mention the few options available. It is also worth mentioning that  $\text{\LuaMetaTeX}$  tries to balance the direction nodes.

0x01	normalizepar	0x04	limitprevgraf
0x02	flattenvleaders		

## 610 `\nospaces`

When `\nospaces` is set to 1 no spaces are inserted, when its value is 2 a zero space is inserted. The default value is 0 which means that spaces become glue with properties depending on the font, specific

parameters and/or space factors determined preceding characters.

### 611 `\nosubprescript`

This processes the given script in the current style, so:

comes out as:  ${}_2x + {}_2x + {}_2x$ .

### 612 `\nosubscript`

This processes the given script in the current style, so:

comes out as:  ${}_2x + x_2 + x_2$ .

### 613 `\nosuperprescript`

This processes the given script in the current style, so:

comes out as:  ${}_2x + {}^2x + {}^2x$ .

### 614 `\nosuperscript`

This processes the given script in the current style, so:

comes out as:  ${}_2x + {}^2x + {}^2x$ .

### 615 `\novrule`

This is a rule but flagged as empty which means that the dimensions kick in as for a normal rule but the backend can decide not to show it.

### 616 `\nulldelimiterspace`

In fenced math delimiters can be invisible in which case this parameter determines the amount of space (width) that ghost delimiter takes.

### 617 `\nullfont`

This a symbolic reference to a font with no glyphs and a minimal set of font dimensions.

### 618 `\number`

This  $\text{\TeX}$  primitive serializes the next token into a number, assuming that it is indeed a number, like

`\number`A`

`\number65`

`\number\scratchcounter`



For counters and such the `\the` primitive does the same, but when you're not sure if what follows is a verbose number or (for instance) a counter the `\number` primitive is a safer bet, because `\the 65` will not work.

## 619 `\numericsscale`

This primitive can best be explained by a few examples:

```
\the\numericsscale 1323
\the\numericsscale 1323.0
\the\numericsscale 1.323
\the\numericsscale 13.23
```

In several places  $\text{\TeX}$  uses a scale but due to the lack of floats it then uses 1000 as 1.0 replacement. This primitive can be used for ‘real’ scales:

```
1323000
1323000
1323
13230
```

## 620 `\numericsscaled`

This is a variant if `\numericsscale`:

```
\scratchcounter 1000
\the\numericsscaled 1323 \scratchcounter
\the\numericsscaled 1323.0 \scratchcounter
\the\numericsscaled 1.323 \scratchcounter
\the\numericsscaled 13.23 \scratchcounter
```

The second number gets multiplied by the first fraction:

```
1323000
1323000
1323
13230
```

## 621 `\numexpr`

This primitive was introduced by  $\varepsilon\text{-TeX}$  and supports a simple expression syntax:

```
\the\numexpr 10 * (1 + 2 - 5) / 2 \relax
```

gives: -10. You can mix in symbolic integers and dimensions.

## 622 `\numexpression`

The normal `\numexpr` primitive understands the `+`, `-`, `*` and `/` operators but in  $\text{\LuaMetaTeX}$  we also can use `:` for a non rounded integer division (think of Lua's `//`). if you want more than that, you can use the new expression primitive where you can use the following operators.

<b>add</b>	+	
<b>subtract</b>	-	
<b>multiply</b>	*	
<b>divide</b>	/ :	
<b>mod</b>	%	mod
<b>band</b>	&	band
<b>bxor</b>	^	bxor
<b>bor</b>	v	bor
<b>and</b>	&&	and
<b>or</b>		or
<b>setbit</b>	<undecided>	bset
<b>resetbit</b>	<undecided>	breset
<b>left</b>	<<	
<b>right</b>	>>	
<b>less</b>	<	
<b>lessequal</b>	<=	
<b>equal</b>	= ==	
<b>moreequal</b>	>=	
<b>more</b>	>	
<b>unequal</b>	<> != ~=	
<b>not</b>	! ~	not

An example of the verbose bitwise operators is:

```
\scratchcounter = \numexpression
"00000 bor "00001 bor "00020 bor "00400 bor "08000 bor "F0000
\relax
```

In the table you might have notices that some operators have equivalents. This makes the scanner a bit less sensitive for catcode regimes.

When `\tracingexpressions` is set to one or higher the intermediate ‘reverse polish notation’ stack that is used for the calculation is shown, for instance:

```
4:8: {numexpression rpn: 2 5 > 4 5 > and}
```

When you want the output on your console, you need to say:

```
\tracingexpressions 1
\tracingonline      1
```

## 623 \omit

This primitive cancels the template set for the upcoming cell. Often it is used in combination with `\span`.

## 624 \optionalboundary

This boundary is used to mark optional content. An positive `\optionalboundary` starts a range and a zero one ends it. Nesting is not supported. Optional content is considered when an additional paragraph pass enables it as part of its recipe.

## 625 \or

This traditional primitive is part of the condition testing mechanism and relates to an `\ifcase` test (or a similar test to be introduced in later sections). Depending on the value,  $\text{\TeX}$  will do a fast scanning till the right `\or` is seen, then it will continue expanding till it sees a `\or` or `\else` or `\orelse` (to be discussed later). It will then do a fast skipping pass till it sees an `\fi`.

## 626 \orelse

This primitive provides a convenient way to flatten your conditional tests. So instead of

```
\ifnum\scratchcounter<-10
  too small
\else\ifnum\scratchcounter>10
  too large
\else
  just right
\fi\fi
```

You can say this:

```
\ifnum\scratchcounter<-10
  too small
\orelse\ifnum\scratchcounter>10
  too large
\else
  just right
\fi
```

You can mix tests and even the case variants will work in most cases<sup>6</sup>

```
\ifcase\scratchcounter      zero
\or                          one
\or                          two
\orelse\ifnum\scratchcounter<10 less than ten
\else                        ten or more
\fi
```

Performance wise there are no real benefits although in principle there is a bit less housekeeping involved than with nested checks. However you might like this:

```
\ifnum\scratchcounter<-10
  \expandafter\toosmall
\orelse\ifnum\scratchcounter>10
  \expandafter\toolarge
\else
  \expandafter\justright
\fi
```

---

<sup>6</sup> I just play safe because there are corner cases that might not work yet.

over:

```
\ifnum\scratchcounter<-10
  \expandafter\toosmall
\else\ifnum\scratchcounter>10
  \expandafter\expandafter\expandafter\toolarge
\else
  \expandafter\expandafter\expandafter\justright
\fi\fi
```

or the more ConT<sub>E</sub>Xt specific:

```
\ifnum\scratchcounter<-10
  \expandafter\toosmall
\else\ifnum\scratchcounter>10
  \doubleexpandafter\toolarge
\else
  \doubleexpandafter\justright
\fi\fi
```

But then, some T<sub>E</sub>Xies like complex and obscure code and throwing away working old code that took ages to perfect and get working and also showed that one masters T<sub>E</sub>X might hurt.

## 627 \orphanpenalties

This an (single entry) array parameter: first the size is given followed by that amount of penalties. These penalties are injected before spaces, going backward from the end of a paragraph. When we see a math node with a penalty set then we take the max and jump over a (preceding) skip.

## 628 \orphanpenalty

This penalty is inserted before the last space in a paragraph, unless \orphanpenalties mandates otherwise.

## 629 \orunless

This is the negated variant of \orelse (prefixing that one with \unless doesn't work well.

## 630 \outer

An outer macro is one that can only be used at the outer level. This property is no longer supported. Like \long, the \outer prefix is now an no-op (and we don't expect this to have unfortunate side effects).

## 631 \output

This token list register holds the code that will be expanded when T<sub>E</sub>X enters the output routine. That code is supposed to do something with the content in the box with number \outputbox. By default this is box 255 but that can be changed with \outputbox.

## 632 `\outputbox`

This is where the split off page content ends up when the output routine is triggered.

## 633 `\outputpenalty`

This is the penalty that triggered the output routine.

## 634 `\over`

This math primitive is actually a bit of a spoiler for the parser as it is one of the few that looks back. The `\Uover` variant is different and takes two arguments. We leave it to the user to predict the results of:

```
$ {1} \over {x} $
$ 1 \over x $
$ 12 \over x / y $
$ a + 1 \over {x} $
```

and:

```
$ \textstyle 1 \over x $
$ {\textstyle 1} \over x $
$ \textstyle {1 \over x} $
```

It's one of the reasons why macro packages provide `\frac`.

## 635 `\overfullrule`

When an overfull box is encountered a rule can be shown in the margin and this parameter sets its width. For the record: `ConTEXt` does it differently.

## 636 `\overline`

This is a math specific primitive that draws a line over the given content. It is a poor man's replacement for a delimiter. The thickness is set with `\Umathoverbarrule`, the distance between content and rule is set by `\Umathoverbarvgap` and `\Umathoverbarkern` is added above the rule. The style used for the content under the rule can be set with `\Umathoverlinevariant`.


Because `ConTEXt` set up math in a special way, the following example:

```
\normaloverline {
  \blackrule[color=red, height=1ex,depth=0ex,width=2cm]%
  \kern-2cm
  \blackrule[color=blue,height=0ex,depth=.5ex,width=2cm]
  x + x
}
```

gives:   $x + x$ , while:

```
\mathfontcontrol\zerocount
```

```
\Umathoverbarkern\allmathstyles10pt
\Umathoverbarvgap\allmathstyles5pt
\Umathoverbarrule\allmathstyles2.5pt
\Umathoverlinevariant\textstyle\scriptstyle
```

gives this: . We have to disable the related `\mathfontcontrol` bits because otherwise the thickness is taken from the font. The variant is just there to overload the (in traditional  $\text{\TeX}$  engines) default.

## 637 `\overloaded`

This prefix can be used to overload a frozen macro.

## 638 `\overloadmode`

The overload protection mechanism can be used to prevent users from redefining a control sequence. The mode can have several values, the higher the more strict we are:

		immutable	permanent	primitive	frozen	instance
1	warning	+	+	+		
2	error	+	+	+		
3	warning	+	+	+	+	
4	error	+	+	+	+	
5	warning	+	+	+	+	+
6	error	+	+	+	+	+

When you set a high error value, you can of course temporary lower or even zero the mode. In Con- $\text{\TeX}$ t all macros and quantities are tagged so there setting the mode to 6 gives a proper protection against overloading. We need to zero the mode when we load for instance `tikz`, so when you use that generic package, you loose some.

## 639 `\overshoot`

This primitive is a companion to `\badness` and reports how much a box overflows.

```
\setbox0\hbox to 1em {mmm} \the\badness\quad\the\overshoot
\setbox0\hbox          {mm} \the\badness\quad\the\overshoot
\setbox0\hbox to 3em   {m}  \the\badness\quad\the\overshoot
```

This reports:

```
1000000 18.44727pt
0 0.0pt
10000 0.0pt
```

When traditional  $\text{\TeX}$  wraps up the lines in a paragraph it uses a mix of shift (a box property) to position the content suiting the hanging indentation and/or paragraph shape, and fills up the line using right skip glue, also in order to silence complaints in packaging. In LuaMeta $\text{\TeX}$  the lines can be normalized so that they all have all possible skips to the left and right (even if they're zero). The

`\overshoot` primitive fits into this picture and is present as a compensation glue. This all fits better in a situation where the internals are opened up via Lua.

## 640 `\overwithdelims`

This is a variant of `\over` but with delimiters. It has a more advanced upgrade in `\Uoverwithdelims`.

## 641 `\pageboundary`

In order to avoid side effects of triggering the page builder with a specific penalty we can use this primitive which expects a value that actually gets inserted as zero penalty before triggering the page builder callback. Think of adding a no-op to the contribution list. We fake a zero penalty so that all gets processed. The main rationale is that we get a better indication of what we do. Of course a callback can remove this node so that it is never seen. Triggering from the callback is not doable. Consider this experimental code (which is actually used in ConT<sub>E</sub>Xt anyway).

## 642 `\pagedepth`

This page property holds the depth of the page.

## 643 `\pagediscards`

The left-overs after a page is split of the main vertical list when glue and penalties are normally discarded. The discards can be pushed back in (for instance) trial runs.

## 644 `\pageexcess`

This page property hold the amount of overflow when a page break occurs.

## 645 `\pageextragoal`

This (experimental) dimension will be used when the page overflows but a bit of overshoot is considered okay.

## 646 `\pagefilllstretch`

The accumulated amount of third order stretch on the current page.

## 647 `\pagefillstretch`

The accumulated amount of second order stretch on the current page.

## 648 `\pagefilstretch`

The accumulated amount of first order stretch on the current page.

**649 \pagefistretch**

The accumulated amount of zero order stretch on the current page.

**650 \pagegoal**

The target height of a page (the running text). This value will be decreased by the height of inserts something to keep into mind when messing around with this and other (pseudo) page related parameters like \pagetotal.

**651 \pagelastdepth**

The accumulated depth of the current page.

**652 \pagelastfillllstretch**

The accumulated amount of third order stretch on the current page. Contrary to \pagefillllstretch this is the really contributed amount, not the upcoming.

**653 \pagelastfillstretch**

The accumulated amount of second order stretch on the current page. Contrary to \pagefillstretch this is the really contributed amount, not the upcoming.

**654 \pagelastfilstretch**

The accumulated amount of first order stretch on the current page. Contrary to \pagefilstretch this is the really contributed amount, not the upcoming.

**655 \pagelastfistretch**

The accumulated amount of zero order stretch on the current page. Contrary to \pagefistretch this is the really contributed amount, not the upcoming.

**656 \pagelastheight**

The accumulated height of the current page.

**657 \pagelastshrink**

The accumulated amount of shrink on the current page. Contrary to \pageshrink this is the really contributed amount, not the upcoming.

**658 \pagelaststretch**

The accumulated amount of stretch on the current page. Contrary to \pagestretch this is the really contributed amount, not the upcoming.



**659 \pageshrink**

The accumulated amount of shrink on the current page.

**660 \pagestretch**

The accumulated amount of stretch on the current page.

**661 \pagetotal**

The accumulated page total (height) of the current page.

**662 \pagevsize**

This parameter, when set, is used as the target page height. This lessens the change of \vsize interfering.

**663 \par**

This is the explicit ‘finish paragraph’ command. Internally we distinguish a par triggered by a new line, as side effect of another primitive or this \par command.

**664 \parametercount**

The number of parameters passed to the current macro.

**665 \parameterdef**

Here is an example of binding a variable to a parameter. The alternative is of course to use an \edef.

```
\def\foo#1#2%
  {\parameterdef\MyIndexOne\plusone % 1
   \parameterdef\MyIndexTwo\plustwo % 2
   \oof{P}\oof{Q}\oof{R}\norelax}

\def\oof#1%
  {<1:\MyIndexOne><1:\MyIndexOne>%
   #1%
   <2:\MyIndexTwo><2:\MyIndexTwo>}

\foo{A}{B}
```

The outcome is:

```
<1:A><1:A>P<2:B><2:B><1:A><1:A>Q<2:B><2:B><1:A><1:A>R<2:B><2:B>
```

**666 \parameterindex**

This gives the zero based position on the parameter stack. One reason for introducing \parameterdef is that the position remains abstract so there we don't need to use \parameterindex.

**667 `\parametermark`**

This is an equivalent for `#`.

**668 `\parametermode`**

Setting this internal integer to a positive value (best use 1 because future versions might use bit set) will enable the usage of `#` for escaped in the main text and body of macros.

**669 `\parattribute`**

This primitive takes an attribute index and value and sets that attribute on the current paragraph.

**670 `\pardirection`**

This set the text direction for the whole paragraph which in the case of `r2l` (1) makes the right edge the starting point.

**671 `\parfillleftskip`**

The glue inserted at the start of the last line.

**672 `\parfillrightskip`**

The glue inserted at the end of the last line (aka `\parfillskip`).

**673 `\parfillskip`**

The glue inserted at the end of the last line.

**674 `\parindent`**

The amount of space inserted at the start of the first line. When bit 2 is set in `\normalizelinemode` a glue is inserted, otherwise an empty `\hbox` with the given width is inserted.

**675 `\parinitleftskip`**

The glue inserted at the start of the first line.

**676 `\parinitrightskip`**

The glue inserted at the end of the first line.

**677 `\parpasses`**

Specifies one or more recipes for additional second linebreak passes. Examples can be found in the Con<sub>T</sub>E<sub>X</sub>t distribution.

**678 `\parshape`**

Stores a shape specification. The first argument is the length of the list, followed by that amount of indentation-width pairs (two dimensions).

**679 `\parshapedimen`**

This oddly named ( $\varepsilon$ -TeX) primitive returns the width component (dimension) of the given entry (an integer). Obsoleted by `\parshapewidth`.

**680 `\parshapeindent`**

Returns the indentation component (dimension) of the given entry (an integer).

**681 `\parshapelength`**

Returns the number of entries (an integer).

**682 `\parshapewidth`**

Returns the width component (dimension) of the given entry (an integer).

**683 `\parskip`**

This is the amount of glue inserted before a new paragraph starts.

**684 `\patterns`**

The argument to this primitive contains hyphenation patterns that are bound to the current language. In LuaTeX and LuaMetaTeX we can also manage this at the Lua end. In LuaMetaTeX we don't store patterns in the format file.

**685 `\pausing`**

In LuaMetaTeX this variable is ignored but in other engines it can be used to single step through the input file by setting it to a positive value.

**686 `\penalty`**

The given penalty (a number) is inserted at the current spot in the horizontal or vertical list. We also have `\vpenalty` and `\hpenalty` that first change modes.

**687 `\permanent`**

This is one of the prefixes that is part of the overload protection mechanism. It is normally used to flag a macro as being at the same level as a primitive: don't touch it. Primitives are flagged as such

but that property cannot be set on regular macros. The similar `\immutable` flag is normally used for variables.

## 688 `\pettymuskip`

A predefined mu skip register that can be used in math (inter atom) spacing. The current value is  $1.0\mu$  minus  $0.5\mu$ . This one complements `\thinmuskip`, `\medmuskip`, `\thickmuskip` and the new `\tinymuskip`.

## 689 `\positdef`

The engine uses 32 bit integers for various purposes and has no (real) concept of a floating point quantity. We get around this by providing a floating point data type based on 32 bit unums (posits). These have the advantage over native floats of more precision in the lower ranges but at the cost of a software implementation.

The `\positdef` primitive is the floating point variant of `\integerdef` and `\dimensiondef`: an efficient way to implement named quantities other than registers.

```
\positdef      \MyFloatA 5.678
\positdef      \MyFloatB 567.8
[\the\MyFloatA] [\todimension\MyFloatA] [\tointeger\MyFloatA]
[\the\MyFloatB] [\todimension\MyFloatB] [\tointeger\MyFloatB]
```

For practical reasons we can map posit (or float) onto an integer or dimension:

```
[5.6780000030994415283] [5.678pt] [6]
[567.8000030517578125] [567.80005pt] [568]
```

## 690 `\postdisplaypenalty`

This is the penalty injected after a display formula.

## 691 `\postexhyphenchar`

This primitive expects a language number and a character code. A negative character code is equivalent to ignore. In case of an explicit discretionary the character is injected at the beginning of a new line.

## 692 `\posthyphenchar`

This primitive expects a language number and a character code. A negative character code is equivalent to ignore. In case of an automatic discretionary the character is injected at the beginning of a new line.

## 693 `\postinlinepenalty`

When set this penalty is inserted after an inline formula unless we have a short formula and `\post-shortinlinepenalty` is set.

**694 `\postshortinlinepenalty`**

When set this penalty is inserted after a short inline formula. The criterium is set by `\shortinline-maththreshold` but only applied when it is enabled for the class involved.

**695 `\prebinoppenalty`**

This internal quantity is a compatibility feature because normally we will use the inter atom spacing variables.

**696 `\predisplaydirection`**

This is the direction that the math sub engine will take into account when dealing with right to left typesetting.

**697 `\predisplaygapfactor`**

The heuristics related to determine if the previous line in a formula overlaps with a (display) formula are hard coded but in Lua $\TeX$  to be two times the quad of the current font. This parameter is a multiplier set to 2000 and permits you to change the overshoot in this heuristic.

**698 `\predisplaypenalty`**

This is the penalty injected before a display formula.

**699 `\preplaysize`**

This parameter holds the length of the last line in a paragraph when a display formula is part of it.

**700 `\preexhyphenchar`**

This primitive expects a language number and a character code. A negative character code is equivalent to ignore. In case of an explicit discretionary the character is injected at the end of the line.

**701 `\prehyphenchar`**

This primitive expects a language number and a character code. A negative character code is equivalent to ignore. In case of an automatic discretionary the character is injected at the end of the line.

**702 `\preinlinepenalty`**

When set this penalty is inserted before an inline formula unless we have a short formula and `\preshort-inlinepenalty` is set.

**703 `\prerelpenalty`**

This internal quantity is a compatibility feature because normally we will use the inter atom spacing variables.

## 704 `\preshortinlinepenalty`

When set this penalty is inserted before a short inline formula. The criterium is set by `\shortinline-maththreshold` but only applied when it is enabled for the class involved.

## 705 `\pretolerance`

When the badness of a line in a paragraph exceeds this value a second linebreak pass will be enabled.

## 706 `\prevdepth`

The depth of current list. It can also be set to special (signal) values in order to inhibit line corrections. It is not an internal dimension but a (current) list property.

## 707 `\prevgraf`

The number of lines in a previous paragraph.

## 708 `\previousloopiterator`

```
\edef\testA{
  \expandedrepeat 2 {%
    \expandedrepeat 3 {%
      (\the\previousloopiterator1:\the\currentloopiterator)
    }%
  }%
}
\edef\testB{
  \expandedrepeat 2 {%
    \expandedrepeat 3 {%
      (#P:#I) % #G is two levels up
    }%
  }%
}
```

These give the same result:

```
\def \testA { (1:1) (1:2) (1:3) (2:1) (2:2) (2:3) }
\def \testB { (1:1) (1:2) (1:3) (2:1) (2:2) (2:3) }
```

The number indicates the number of levels we go up the loop chain.

## 709 `\primescript`

This is a math script primitive dedicated to primes (which are somewhat troublesome on math). It complements the six script primitives (like `\subscript` and `\presuperscript`).

## 710 `\protected`

A protected macro is one that doesn't get expanded unless it is time to do so. For instance, inside an `\edef` it just stays what it is. It often makes sense to pass macros as-is to (multi-pass) file (for tables of contents).

In ConT<sub>E</sub>Xt we use either `\protected` or `\unexpanded` because the later was the command we used to achieve the same results before  $\varepsilon$ -T<sub>E</sub>X introduced this protection primitive. Originally the `\protected` macro was also defined but it has been dropped.

## 711 `\protecteddetokenize`

This is a variant of `\protecteddetokenize` that uses some escapes encoded as body parameters, like `#H` for a hash.

## 712 `\protectedexpandeddetokenize`

This is a variant of `\expandeddetokenize` that uses some escapes encoded as body parameters, like `#H` for a hash.

## 713 `\protrudechars`

This variable controls protrusion (into the margin). A value 2 is comparable with other engines, while a value of 3 does a bit more checking when we're doing right-to-left typesetting.

## 714 `\protrusionboundary`

This injects a boundary with the given value:

```
0x00 skipnone
0x01 skipnext
0x02 skipprevious
0x03 skipboth
```

This signal makes the protrusion checker skip over a node.

## 715 `\pxdimen`

The current numeric value of this dimension is 65781, 1.00374pt: one bp. We kept it around because it was introduced in pdfT<sub>E</sub>X and made it into LuaT<sub>E</sub>X, where it relates to the resolution of included images. In ConT<sub>E</sub>Xt it is not used.

## 716 `\quitloop`

There are several loop primitives and they can be quit with `\quitloop` at the next the *next* iteration. An immediate quit is possible with `\quitloopnow`. An example is given with `\localcontrolledloop`.

## 717 `\quitloopnow`

There are several loop primitives and they can be quit with `\quitloopnow` at the spot.

## 718 `\quitvmode`

This primitive forces horizontal mode but has no side effects when we're already in that mode.

## 719 `\radical`

This old school radical constructor is replaced by `\Uradical`. It takes a number where the first byte is the small family, the next two index of this symbol from that family, and the next three the family and index of the first larger variant.

## 720 `\raise`

This primitive takes two arguments, a dimension and a box. The box is moved up. The operation only succeeds in horizontal mode.

## 721 `\rdivide`

This is variant of `\divide` that rounds the result. For integers the result is the same as `\edivide`.

```
\the\dimexpr .4999pt : 2 \relax =.24994pt
\the\dimexpr .4999pt / 2 \relax =.24995pt
\scratchdimen.4999pt \divide \scratchdimen 2 \the\scratchdimen =.24994pt
\scratchdimen.4999pt \edivide\scratchdimen 2 \the\scratchdimen =.24995pt
\scratchdimen 4999pt \rdivide\scratchdimen 2 \the\scratchdimen =2500.0pt
\scratchdimen 5000pt \rdivide\scratchdimen 2 \the\scratchdimen =2500.0pt
```

```
\the\numexpr 1001 : 2 \relax =500
\the\numexpr 1001 / 2 \relax =501
\scratchcounter1001 \divide \scratchcounter 2 \the\scratchcounter=500
\scratchcounter1001 \edivide\scratchcounter 2 \the\scratchcounter=501
\scratchcounter1001 \rdivide\scratchcounter 2 \the\scratchcounter=501
```

```
0.24994pt=.24994pt
0.24995pt=.24995pt
0.24994pt=.24994pt
0.24995pt=.24995pt
2500.0pt=2500.0pt
2500.0pt=2500.0pt
```

```
500=500
501=501
500=500
501=501
501=501
```

## 722 `\rdivideby`

This is the by-less companion to `\rdivide`.



## 723 `\realign`

Where `\omit` suspends a preamble template, this one overloads is for the current table cell. It expects two token lists as arguments.

## 724 `\relax`

This primitive does nothing and is often used to end a verbose number or dimension in a comparison, for example:

```
\ifnum \scratchcounter = 123\relax
```

which prevents a lookahead. A variant would be:

```
\ifnum \scratchcounter = 123 %
```

assuming that spaces are not ignored. Another application is finishing an expression like `\numexpr` or `\dimexpr`. It is also used to prevent lookahead in cases like:

```
\vrule height 3pt depth 2pt width 5pt\relax
\hskip 5pt plus 3pt minus 2pt\relax
```

Because `\relax` is not expandable the following:

```
\edef\foo{\relax} \meaningfull\foo
\edef\oof{\norelax} \meaningfull\oof
```

gives this:

```
macro:\relax
macro:
```

A `\norelax` disappears here but in the previously mentioned scenarios it has the same function as `\relax`. It will not be pushed back either in cases where a lookahead demands that.

## 725 `\relpenalty`

This internal quantity is a compatibility feature because normally we will use the inter atom spacing variables.

## 726 `\resetmathspacing`

This initializes all parameters to their initial values.

## 727 `\restorecatcodetable`

This is an experimental feature that should be used with care. The next example shows usage. It was added when debugging and exploring a side effect.

```
\tracingonline1
```

```
\bgroup
```

```
\catcode`6 = 11 \catcode`7 = 11
```

```
\bgroup
```

```
\tracingonline1
```

```
current: \the\catcodetable
```

```
original: \the\catcode`6\quad \the\catcode`7
```

```
\catcode`6 = 11 \catcode`7 = 11
```

```
\showcodestack\catcode
```

```
assigned: \the\catcode`6\quad \the\catcode`7
```

```
\showcodestack\catcode
```

```
\catcodetable\ctxcatcodes switched: \the\catcodetable
```

```
stored: \the\catcode`6\quad \the\catcode`7
```

```
\showcodestack\catcode
```

```
\restorecatcodetable\ctxcatcodes
```

```
\showcodestack\catcode
```

```
restored: \the\catcode`6\quad \the\catcode`7
```

```
\showcodestack\catcode
```

```
\egroup
```

```
\catcodetable\ctxcatcodes
```

```
inner: \the\catcode`6\quad\the\catcode`7
```

```
\egroup
```

```
outer: \the\catcode`6\quad\the\catcode`7
```

In ConT<sub>E</sub>Xt this typesets:

```
current: 9
```

```
original: 11 11
```

```
assigned: 11 11
```

```
switched: 9
```

```
stored: 11 11
```

```
restored: 12 12
```

```
inner: 11 11
```

```
outer; 12 12
```

and on the console we see:

```
3:3: [codestack 1, size 3]
```

```

3:3: [1: level 2, code 54, value 12]
3:3: [2: level 2, code 55, value 12]
3:3: [3: level 3, code 54, value 11]
3:3: [4: level 3, code 55, value 11]
3:3: [codestack 1 bottom]
3:3: [codestack 1, size 3]
3:3: [1: level 2, code 54, value 12]
3:3: [2: level 2, code 55, value 12]
3:3: [3: level 3, code 54, value 11]
3:3: [4: level 3, code 55, value 11]
3:3: [codestack 1 bottom]
3:3: [codestack 1, size 3]
3:3: [1: level 2, code 54, value 12]
3:3: [2: level 2, code 55, value 12]
3:3: [3: level 3, code 54, value 11]
3:3: [4: level 3, code 55, value 11]
3:3: [codestack 1 bottom]
3:3: [codestack 1, size 7]
3:3: [1: level 2, code 54, value 12]
3:3: [2: level 2, code 55, value 12]
3:3: [3: level 3, code 54, value 11]
3:3: [4: level 3, code 55, value 11]
3:3: [5: level 3, code 55, value 11]
3:3: [6: level 3, code 54, value 11]
3:3: [7: level 3, code 55, value 11]
3:3: [8: level 3, code 54, value 11]
3:3: [codestack 1 bottom]
3:3: [codestack 1, size 7]
3:3: [1: level 2, code 54, value 12]
3:3: [2: level 2, code 55, value 12]
3:3: [3: level 3, code 54, value 11]
3:3: [4: level 3, code 55, value 11]
3:3: [5: level 3, code 55, value 11]
3:3: [6: level 3, code 54, value 11]
3:3: [7: level 3, code 55, value 11]
3:3: [8: level 3, code 54, value 11]
3:3: [codestack 1 bottom]

```

So basically `\restorecatcodetable` brings us (temporarily) back to the global settings.

## 728 `\retained`

When a value is assigned inside a group `TEX` pushes the current value on the save stack in order to be able to restore the original value after the group has ended. You can reach over a group by using the `\global` prefix. A mix between local and global assignments can be achieved with the `\retained` primitive.

```

\MyDim 15pt \bgroup \the\MyDim \space
\bgroup
  \bgroup

```

```

    \bgroup \advance\MyDim10pt \the\MyDim \egroup\space
    \bgroup \advance\MyDim10pt \the\MyDim \egroup\space
\egroup
\bgroup
    \bgroup \advance\MyDim10pt \the\MyDim \egroup\space
    \bgroup \advance\MyDim10pt \the\MyDim \egroup\space
\egroup
\egroup
\egroup \the\MyDim

\MyDim 15pt \bgroup \the\MyDim \space
\bgroup
    \bgroup
        \bgroup \global\advance\MyDim10pt \the\MyDim \egroup\space
        \bgroup \global\advance\MyDim10pt \the\MyDim \egroup\space
    \egroup
    \bgroup
        \bgroup \global\advance\MyDim10pt \the\MyDim \egroup\space
        \bgroup \global\advance\MyDim10pt \the\MyDim \egroup\space
    \egroup
\egroup
\egroup \the\MyDim

\MyDim 15pt \bgroup \the\MyDim \space
    \constrained\MyDim\zeropoint
    \bgroup
        \bgroup \retained\advance\MyDim10pt \the\MyDim \egroup\space
        \bgroup \retained\advance\MyDim10pt \the\MyDim \egroup\space
    \egroup
    \bgroup
        \bgroup \retained\advance\MyDim10pt \the\MyDim \egroup\space
        \bgroup \retained\advance\MyDim10pt \the\MyDim \egroup\space
    \egroup
\egroup \the\MyDim

```

These lines result in:

```

15.0pt 25.0pt 25.0pt 25.0pt 25.0pt 15.0pt
15.0pt 25.0pt 35.0pt 45.0pt 55.0pt 55.0pt
15.0pt 10.0pt 20.0pt 30.0pt 40.0pt 15.0pt

```

Because LuaMetaTeX avoids redundant stack entries and reassignments this mechanism is a bit fragile but the `\constrained` prefix makes sure that we do have a stack entry. If it is needed depends on the usage pattern.

## 729 \retokenized

This is a companion of `\tokenized` that accepts a catcode table, so the whole repertoire is:

```

\tokenized                                {test $x$ test: current}
\tokenized catcodetable \ctxcatcodes {test $x$ test: context}

```

```

\tokenized   catcodetable \vrbcatcodes {test $x$ test: verbatim}
\retokenized          \ctxcatcodes {test $x$ test: context}
\retokenized          \vrbcatcodes {test $x$ test: verbatim}

```

Here we pass the numbers known to ConT<sub>E</sub>Xt and get:

```

test x test: current
test x test: context
test $x$ test: verbatim
test x test: context
test $x$ test: verbatim

```

## 730 \right

Inserts the given delimiter as right fence in a math formula.

## 731 \righthyphenmin

This is the minimum number of characters before the first hyphen in a hyphenated word.

## 732 \rightmarginkern

The dimension returned is the protrusion kern that has been added (if at all) to the left of the content in the given box.

## 733 \rightskip

This skip will be inserted at the right of every line.

## 734 \romannumeral

This converts a number into a sequence of characters representing a roman numeral. Because the Romans had no zero, a zero will give no output, a fact that is sometimes used for hacks and showing off ones macro coding capabilities. A large number will for sure result in a long string because after thousand we start duplicating.

## 735 \rpcode

This is the companion of \lpcode (see there) and also takes three arguments: font, character code and factor.

## 736 \savecatcodetable

This primitive stores the currently set catcodes in the current table.

## 737 \savingshyphcodes

When set to non-zero, this will trigger the setting of \hjcodes from \lccodes for the current font. These codes determine what characters are taken into account when hyphenating words.

**738 \savingvdiscards**

When set to a positive value the page builder will store the discarded items (like glues) so that they can later be retrieved and pushed back if needed with \pagediscards or \splitdiscards.

**739 \scaledemwidth**

Returns the current (font specific) emwidth scaled according to \glyphscale and \glyphxscale.

**740 \scaledexheight**

Returns the current (font specific) exheight scaled according to \glyphscale and \glyphyscale.

**741 \scaledextraspacer**

Returns the current (font specific) extra space value scaled according to \glyphscale and \glyphxscale.

**742 \scaledfontcharba**

Returns the bottom accent position of the given font-character pair scaled according to \glyphscale and \glyphyscale.

**743 \scaledfontchardp**

Returns the depth of the given font-character pair scaled according to \glyphscale and \glyphyscale.

**744 \scaledfontcharht**

Returns the height of the given font-character pair scaled according to \glyphscale and \glyphyscale.

**745 \scaledfontcharic**

Returns the italic correction of the given font-character pair scaled according to \glyphscale and \glyphxscale. This property is only real for traditional fonts.

**746 \scaledfontcharta**

Returns the top accent position of the given font-character pair scaled according to \glyphscale and \glyphxscale.

**747 \scaledfontcharwd**

Returns width of the given font-character pair scaled according to \glyphscale and \glyphxscale.

**748 `\scaledfontdimen`**

Returns value of a (numeric) font dimension of the given font-character pair scaled according to `\glyphscale` and `\glyphxscale` and/or `\glyphyscale`.

**749 `\scaledinterwordshrink`**

Returns the current (font specific) shrink of a space value scaled according to `\glyphscale` and `\glyphxscale`.

**750 `\scaledinterwordspace`**

Returns the current (font specific) space value scaled according to `\glyphscale` and `\glyphxscale`.

**751 `\scaledinterwordstretch`**

Returns the current (font specific) stretch of a space value scaled according to `\glyphscale` and `\glyphxscale`.

**752 `\scaledmathaxis`**

This primitive returns the math axis of the given math style. It's a dimension.

**753 `\scaledmathemwidth`**

Returns the emwidth of the given style scaled according to `\glyphscale` and `\glyphxscale`.

**754 `\scaledmathexheight`**

Returns the exheight of the given style scaled according to `\glyphscale` and `\glyphyscale`.

**755 `\scaledmathstyle`**

This command inserts a signal in the math list that tells how to scale the (upcoming) part of the formula.

```
$ x + {\scaledmathstyle900 x} + x$
```

We get:  $x + x + x$ . Of course using this properly demands integration in the macro packages font system.

**756 `\scaledslantperpoint`**

This primitive is equivalent to `\scaledfontdimen1\font` where ‘scaled’ means that we multiply by the glyph scales.

**757 `\scantextokens`**

This primitive scans the input as if it comes from a file. In the next examples the `\detokenize` primitive turns tokenized code into verbatim code that is similar to what is read from a file.

```

\edef\whatever{\detokenize{This is {\bf bold} and this is not.}}
\detokenize {This is {\bf bold} and this is not.}\crlf
\scantextokens{This is {\bf bold} and this is not.}\crlf
\scantextokens{\whatever}\crlf
\scantextokens\expandafter{\whatever}\par

```

This primitive does not have the end-of-file side effects of its precursor `\scantokens`.

This is `{\bf bold}` and this is not.

This is **bold** and this is not.

This is `{\bf bold}` and this is not.

This is **bold** and this is not.

## 758 `\scantokens`

Just forget about this  $\varepsilon$ -TeX primitive, just take the one in the next section.

## 759 `\scriptfont`

This primitive is like `\font` but with a family number as (first) argument so it is specific for math. It is the middle one of the three family members; its relatives are `\textfont` and `\scriptscriptfont`.

## 760 `\scriptscriptfont`

This primitive is like `\font` but with a family number as (first) argument so it is specific for math. It is the smallest of the three family members; its relatives are `\textfont` and `\scriptfont`.

## 761 `\scriptscriptstyle`

One of the main math styles, normally one size smaller than `\scriptstyle`: integer representation: 6.

## 762 `\scriptspace`

The math engine will add this amount of space after subscripts and superscripts. It can be seen as compensation for the often too small widths of characters (in the traditional engine italic correction is used too). It prevents scripts from running into what follows.

## 763 `\scriptstyle`

One of the main math styles, normally one size smaller than `\displaystyle` and `\textstyle`; integer representation: 4.

## 764 `\scrollmode`

This directive omits error stops.



## 765 `\semiexpand`

This command expands the next macro when it is protected with `\semprotected`. See that primitive there for an example.

## 766 `\semiexpanded`

This command expands the tokens in the given list including the macros protected by with `\semprotected`. See that primitive there for an example.

## 767 `\semiprotected`

The working of this prefix can best be explained with an example. We define a few macros first:

```

\def\TestA{A}
\semiprotected\def\TestB{B}
\protected\def\TestC{C}

\edef\TestD{\TestA      \TestB      \TestC}
\edef\TestE{\TestA\semiexpand\TestB\semiexpand\TestC}
\edef\TestF{\TestA\expand  \TestB\expand  \TestC}

\edef\TestG{\normalexpanded  {\TestA\TestB\TestC}}
\edef\TestH{\normalsemiexpanded{\TestA\TestB\TestC}}
```

The meaning of the macros that are made from the other three are:

Here we use the `\normal`... variants because (currently) we still have the macro with the `\expanded` in the ConT<sub>E</sub>Xt core.

```

A\TestB \TestC
AB\TestC
ABC
A\TestB \TestC
AB\TestC
```

## 768 `\setbox`

This important primitive is used to set a box register. It expects a number and a box, like `\hbox` or `\box`. There is no `\boxdef` primitive (analogue to other registers) because it makes no sense but numeric registers or equivalents are okay as register value.

## 769 `\setdefaultmathcodes`

This sets the math codes of upper- and lowercase alphabet and digits and the delimiter code of the period. It's not so much a useful feature but more just an accessor to the internal initializer.

## 770 `\setfontid`

Internally a font instance has a number and this number is what gets assigned to a glyph node. You can get the number with `\fontid` and set it with `\setfontid`.

## `\setfontid\fontid\font`

The code above shows both primitives and effectively does nothing useful but shows the idea.

## 771 `\setlanguage`

In Lua<sub>T</sub><sub>E</sub>X and LuaMeta<sub>T</sub><sub>E</sub>X this is equivalent to `\language` because we carry the language in glyph nodes instead of putting triggers in the list.

## 772 `\setmathatomrule`

The math engine has some built in logic with respect to neighboring atoms that change the class. The following combinations are intercepted and remapped:

<b>old first</b>	<b>old second</b>	new first	new second
begin	binary	ordinary	ordinary
operator	binary	operator	ordinary
open	binary	open	ordinary
punctuation	binary	punctuation	ordinary
binary	end	ordinary	ordinary
binary	binary	binary	ordinary
binary	close	ordinary	close
binary	punctuation	ordinary	punctuation
binary	relation	ordinary	relation
relation	binary	relation	ordinary
relation	close	ordinary	close
relation	punctuation	ordinary	punctuation

You can change this logic if needed, for instance:

```
\setmathatomrule 1 2 \allmathstyles 1 1
```

Keep in mind that the defaults are what users expect. You might set them up for additional classes that you define but even then you probably clone an existing class and patch its properties. Most extra classes behave like ordinary anyway.

## 773 `\setmathdisplaypostpenalty`

This penalty is inserted after an item of a given class but only in inline math when display style is used, for instance:

```
\setmathdisplayprepenalty 2 750
```

## 774 `\setmathdisplayprepenalty`

This penalty is inserted before an item of a given class but only in inline math when display style is used, for instance:

**`\setmathdisplayprepenalty`** 2 750

## 775 `\setmathignore`

You can flag a math parameter to be ignored, like:

```
\setmathignore \Umathxscale      2
\setmathignore \Umathyscale      2
\setmathignore \Umathspacebeforescript 1
\setmathignore \Umathspaceafterscript 1
```

A value of two will not initialize the variable, so its old value (when set) is kept. This is somewhat experimental and more options might show up.

## 776 `\setmathoptions`

This primitive expects a class (number) and a bitset.

0x00000001	nopreslack	0x00004000	raiseprime
0x00000002	nopostslack	0x00008000	carryoverlefttopkern
0x00000004	lefttopkern	0x00010000	carryoverrighttopkern
0x00000008	righttopkern	0x00020000	carryoverleftbottomkern
0x00000010	leftbottomkern	0x00040000	carryoverrightbottomkern
0x00000020	rightbottomkern	0x00080000	preferdelimeterdimensions
0x00000040	lookaheadforend	0x00100000	autoinject
0x00000080	noitaliccorrection	0x00200000	removeitaliccorrection
0x00000100	checkligature	0x00400000	operatoritaliccorrection
0x00000200	checkitaliccorrection	0x00800000	shortinline
0x00000400	checkkernpair	0x01000000	pushnesting
0x00000800	flatten	0x02000000	popnesting
0x00001000	omitpenalty	0x04000000	obeynesting
0x00002000	unpack		

## 777 `\setmathpostpenalty`

This penalty is inserted after an item of a given class but only in inline math when text, script or scriptscript style is used, for instance:

**`\setmathpostpenalty`** 2 250

## 778 `\setmathprepenalty`

This penalty is inserted before an item of a given class but only in inline math when text, script or scriptscript style is used, for instance:

**`\setmathprepenalty`** 2 250

## 779 `\setmathspacing`

More details about this feature can be found in ConT<sub>E</sub>Xt but it boils down to registering what spacing gets inserted between a pair of classes. It can be defined per style or for a set of styles, like:

```
\inherited\setmathspacing
  \mathimplicationcode \mathbinarycode
  \alldisplaystyles \thickermuskip
\inherited\setmathspacing
  \mathradicalcode \mathmiddlecode
  \allunsplitstyles \pettymuskip
```

Here the `\inherited` prefix signals that a change in for instance `\pettymuskip` is reflected in this spacing pair. In ConT<sub>E</sub>Xt there is a lot of granularity with respect to spacing and it took years of experimenting (and playing with examples) to get at the current stage. In general users are not invited to mess around too much with these values, although changing the bound registers (here `\pettymuskip` and `\thickermuskip`) is no problem as it consistently makes related spacing pairs follow.

## 780 `\sfcode`

You can set a space factor on a character. That factor is used when a space factor is applied (as part of spacing). It is (mostly) used for adding a different space (glue) after punctuation. In some languages different punctuation has different factors.

## 781 `\shapingpenaltiesmode`

Shaping penalties are inserted after the lines of a `\parshape` and accumulate according to this mode, a bitset of:

```
0x01 interlinepenalty
0x02 widowpenalty
0x04 clubpenalty
0x08 brokenpenalty
```

## 782 `\shapingpenalty`

In order to prevent a `\parshape` to break in unexpected ways we can add a dedicated penalty, specified by this parameter.

## 783 `\shiftedsuperscript`

This primitive (or `_____`) influences the alignment of scripts:

```
$
  x \shiftedsuperscript{2} \subscript      {2} +
  x \superscript          {2} \shiftedsuperscript{2} +
  x \superscript          {2} _____    {2} =
  x \superscript          {2} \subscript      {2}
$
```

Gives:  $\frac{2}{2}x + \frac{2}{2}x + \frac{2}{2}x = \frac{2}{2}x$ .

## 784 `\shiftedsuperscript`

This primitive (or `_____`) influences the alignment of scripts:

```
$
x \shiftedsuperscript{2} \subscript      {2} +
x \superscript          {2} \shiftedsuperscript{2} +
x \superscript          {2} _____ {2} =
x \superscript          {2} \subscript      {2}
```

Gives:  $x^2_2 + x_2^2 + x_2^2 = x_2^2$ .

## 785 `\shiftedsuperprescript`

This primitive (or `^^^^`) influences the alignment of scripts:

```
$
x \shiftedsuperprescript{2} \subprescript      {2} +
x ^^^^^                  {2} \subprescript      {2} +
x \superprescript        {2} \shiftedsubprescript{2} =
x \superprescript        {2} \subprescript      {2}
```

Gives:  $\frac{2}{2}x + \frac{2}{2}x + \frac{2}{2}x = \frac{2}{2}x$ .

## 786 `\shiftedsuperscript`

This primitive (or `^^^`) influences the alignment of scripts:

```
$
x \shiftedsuperscript{2} \subscript      {2} +
x ^^^                  {2} \subscript      {2} +
x \superscript          {2} \shiftedsuperscript{2} =
x \superscript          {2} \subscript      {2}
```

Gives:  $x^2_2 + x^2_2 + x_2^2 = x_2^2$ .

## 787 `\shipout`

Because there is no backend, this is not supposed to be used. As in traditional T<sub>E</sub>X a box is grabbed but instead of it being processed it gets shown and then wiped. There is no real benefit of turning it into a callback.

## 788 `\shortinlinemaththreshold`

This parameter determines when an inline formula is considered to be short. This criterium is used for `\preshortinlinepenalty` and `\postshortinlinepenalty`.

## 789 `\shortinlineorphanpenalty`

Short formulas at the end of a line are normally not followed by something other than punctuation. This penalty will discourage a break before a short inline formula. In practice one can set this penalty to e.g. a relatively low 200 to get the desired effect.

## 790 `\show`

Prints to the console (and/or log) what the token after it represents.

## 791 `\showbox`

The given box register is shown in the log and on the console (depending on `\tracingonline`). How much is shown depends on `\showboxdepth` and `\showboxbreadth`. In LuaMetaTeX we show more detailed information than in the other engines; some specific information is provided via callbacks.

## 792 `\showboxbreadth`

This primitive determines how much of a box is shown when asked for or when tracing demands it.

## 793 `\showboxdepth`

This primitive determines how deep tracing a box goes into the box. Some boxes, like the ones that has the assembled page.

## 794 `\showcodestack`

This inspector is only useful for low level debugging and reports the current state of for instance the current catcode table: `\showcodestack\catcode`. See `\restorecatcodes` for an example.

## 795 `\showgroups`

This primitive reports the group nesting. At this spot we have a not so impressive nesting:

```
2:3: simple group entered at line 9375:
1:3: semisimple group: \beginngroup
0:3: bottomlevel
```

## 796 `\showifs`

This primitive will show the conditional stack in the log file or on the console (assuming `\tracingonline` being non-zero). The shown data is different from other engines because we have more conditionals and also support a more flat nesting model

## 797 `\showlists`

This shows the currently built list.

## 798 \shownodedetails

When set to a positive value more details will be shown of nodes when applicable. Values larger than one will also report attributes. What gets shown depends on related callbacks being set.

## 799 \showstack

This tracer is only useful for low level debugging of macros, for instance when you run out of save space or when you encounter a performance hit.

```
test\scratchcounter0 \showstack
{test\scratchcounter1 \showstack}
{{test\scratchcounter1 \showstack}}
```

reports

```
1:3: [savestack size 0]
1:3: [savestack bottom]

2:3: [savestack size 2]
2:3: [1: restore, level 1, cs \scratchcounter=integer 1]
2:3: [0: boundary, group 'bottomlevel', boundary 0, attrlist 3600, line 0]
2:3: [savestack bottom]

3:3: [savestack size 3]
3:3: [2: restore, level 1, cs \scratchcounter=integer 1]
3:3: [1: boundary, group 'simple', boundary 0, attrlist 3600, line 12]
3:3: [0: boundary, group 'bottomlevel', boundary 0, attrlist 3600, line 0]
3:3: [savestack bottom]
```

while

```
test\scratchcounter1 \showstack
{test\scratchcounter1 \showstack}
{{test\scratchcounter1 \showstack}}
```

shows this:

```
1:3: [savestack size 0]
1:3: [savestack bottom]

2:3: [savestack size 1]
2:3: [0: boundary, group 'bottomlevel', boundary 0, attrlist 3600, line 0]
2:3: [savestack bottom]

3:3: [savestack size 2]
3:3: [1: boundary, group 'simple', boundary 0, attrlist 3600, line 16]
3:3: [0: boundary, group 'bottomlevel', boundary 0, attrlist 3600, line 0]
3:3: [savestack bottom]
```

Because in the second example the value of `\scratchcounter` doesn't really change inside the group there is no need for a restore entry on the stack. In LuaMetaTeX there are checks for that so that we

consume less stack space. We also store some states (like the line number and current attribute list pointer) in a stack boundary.

## 800 `\showthe`

Prints to the console (and/or log) the value of token after it.

## 801 `\showtokens`

This command expects a (balanced) token list, like

```
\showtokens{a few tokens}
```

Depending on what you want to see you need to expand:

```
\showtokens\expandafter{\the\everypar}
```

which is equivalent to `\showthe\everypar`. It is an  $\varepsilon$ -T<sub>E</sub>X extension.

## 802 `\singlelinepenalty`

This is a penalty that gets injected before a paragraph that has only one line. It is a one-shot parameter, so like `\looseness` it only applies to the upcoming (or current) paragraph.

## 803 `\skewchar`

This is an (imaginary) character that is used in math fonts. The kerning pair between this character and the current one determines the top anchor of a possible accent. In OpenType there is a dedicated character property for this (but for some reason not for the bottom anchor).

## 804 `\skip`

This is the accessor for an indexed skip (glue) register.

## 805 `\skipdef`

This command associates a control sequence with a skip register (accessed by number).

## 806 `\snapshotpar`

There are many parameters involved in typesetting a paragraph. One complication is that parameters set in the middle might have unpredictable consequences due to grouping, think of:

```
text text <some setting> text text \par
text {text <some setting> text } text \par
```

This makes in traditional T<sub>E</sub>X because there is no state related to the current paragraph. But in Lua-T<sub>E</sub>X we have the initial so called par node that remembers the direction as well as local boxes. In



LuaMetaT<sub>E</sub>X we store way more when this node is created. That means that later settings no longer replace the stored ones.

The `\snapshotpar` takes a bitset that determine what stored parameters get updated to the current values.

0x00000001	hsize	0x00000200	looseness	0x00040000	line
0x00000002	skip	0x00000400	lastline	0x00080000	hyphenation
0x00000004	hang	0x00000800	linepenalty	0x00100000	shapingpenalty
0x00000008	indent	0x00001000	clubpenalty	0x00200000	orphanpenalty
0x00000010	parfill	0x00002000	widowpenalty	0x00400000	emergency
0x00000020	adjust	0x00004000	displaypenalty	0x00800000	parpasses
0x00000040	protrude	0x00008000	brokenpenalty	0x01000000	singlelinepenalty
0x00000080	tolerance	0x00010000	demerits		
0x00000100	stretch	0x00020000	shape		

One such value covers multiple values, so for instance `skip` is good for storing the current `\leftskip` and `\rightskip` values. More about this feature can be found in the ConT<sub>E</sub>Xt documentation.

The list of parameters that gets reset after a paragraph is longer than for pdfT<sub>E</sub>X and LuaMetaT<sub>E</sub>X: `\emergencyleftskip`, `\emergencyrightskip`, `\hangafter`, `\hangindent`, `\interlinepenalties`, `\localbrokenpenalty`, `\localinterlinepenalty`, `\localpretolerance`, `\localtolerance`, `\looseness`, `\parshape` and `\singlelinepenalty`.

## 807 `\spacefactor`

The space factor is a somewhat complex feature. When during scanning a character is appended that has a `\sfcode` other than 1000, that value is saved. When the time comes to insert a space triggered glue, and that factor is 2000 or more, and when `\xspaceskip` is nonzero, that value is used and we're done.

If these criteria are not met, and `\spaceskip` is nonzero, that value is used, otherwise the space value from the font is used. Now, if the space factor is larger than 2000 the extra space value from the font is added to the set value. Next the engine is going to tweak the stretch and shrink if that value and in LuaMetaT<sub>E</sub>X that can be done in different ways, depending on `\spacefactormode`, `\spacefactorstretchlimit` and `\spacefactorshrinklimit`.

First the stretch. When the set limit is 1000 or more and the saved space factor is also 1000 or more, we multiply the stretch by the limit, otherwise the saved space factor is used.

Shrink is done differently. When the shrink limit and space factor are both 1000 or more, we will scale the shrink component by the limit, otherwise we multiply by the saved space factor but here we have three variants, determined by the value of `\spacefactormode`.

In the first case, when the limit kicks in, a mode value 1 will multiply by limit and divides by 1000. A value of 2 multiplies by 2000 and divides by the limit. Other mode values multiply by 1000 and divide by the limit. When the limit is not used, the same happens but with the saved space factor.

If this sounds complicated, here is what regular T<sub>E</sub>X does: stretch is multiplied by the factor and divided by 1000 while shrink is multiplied by 1000 and divided by the saved factor. The (new) mode

driven alternatives are the result of extensive experiments done in the perspective of enhancing the rendering of inline math as well as additional par builder passes. For sure alternative strategies are possible and we can always add more modes.

A better explanation of the default strategy around spaces can be found in (of course) The  $\text{\TeX}$ book and  $\text{\TeX}$  by Topic.

## 808 $\backslash\text{spacefactormode}$

Its setting determines the way the glue components (currently only shrink) adapts itself to the current space factor (determined by the character preceding a space).

## 809 $\backslash\text{spacefactorshrinklimit}$

This limit is used when  $\backslash\text{spacefactormode}$  is set. See  $\backslash\text{spacefactor}$  for a bit more explanation.

## 810 $\backslash\text{spacefactorstretchlimit}$

This limit is used when  $\backslash\text{spacefactormode}$  is set. See  $\backslash\text{spacefactor}$  for a bit more explanation.

## 811 $\backslash\text{spaceskip}$

Normally the glue inserted when a space is encountered is taken from the font but this parameter can overrule that.

## 812 $\backslash\text{span}$

This primitive combined two upcoming cells into one. Often it is used in combination with  $\backslash\text{omit}$ . However, in the preamble it forces the next token to be expanded, which means that nested  $\backslash\text{tabstops}$  and align content markers are seen.

## 813 $\backslash\text{splitbotmark}$

This is a reference to the last mark on the currently split off box, it gives back tokens.

## 814 $\backslash\text{splitbotmarks}$

This is a reference to the last mark with the given id (a number) on the currently split off box, it gives back tokens.

## 815 $\backslash\text{splitdiscards}$

When a box is split off, items like glue are discarded. This internal register keeps the that list so that it can be pushed back if needed.

## 816 $\backslash\text{splitfirstmark}$

This is a reference to the first mark on the currently split off box, it gives back tokens.

**817 `\splitfirstmarks`**

This is a reference to the first mark with the given id (a number) on the currently split off box, it gives back tokens.

**818 `\splitmaxdepth`**

The depth of the box that results from a `\vsplit`.

**819 `\splittopskip`**

This is the amount of glue that is added to the top of a (new) split of part of a box when `\vsplit` is applied.

**820 `\srule`**

This inserts a rule with no width. When a font and a char are given the height and depth of that character are taken. Instead of a font `fam` is also accepted so that we can use it in math mode.

**821 `\string`**

We mention this original primitive because of the one in the next section. It expands the next token or control sequence as if it was just entered, so normally a control sequence becomes a backslash followed by characters and a space.

**822 `\subscript`**

Instead of two characters with catcode 8 (like `__`) this primitive can be used. It will add the following argument as lower left script to the nucleus.

**823 `\subscript`**

Instead of a character with catcode 8 (normally `_`) this primitive can be used. It will add the following argument as lower right script to the nucleus.

**824 `\superprescript`**

Instead of two characters with catcode 7 (like `^^`) this primitive can be used. It will add the following argument as upper left script to the nucleus.

**825 `\superscript`**

Instead of a character with catcode 7 (normally `^`) this primitive can be used. It will add the following argument as upper right script to the nucleus.

**826 `\supmarkmode`**

As in other languages,  $\TeX$  has ways to escape characters and get whatever character needed into the input. By default multiple `^` are used for this. The dual `^^` variant is a bit weird as it is not continuous

but `^^^^` and `^^^^^^` provide four or six byte hexadecimal references of characters. The single `^` is also used for superscripts but because we support prescripts and indices we get into conflicts with the escapes.

When this internal quantity is set to zero, multiple `^`'s are interpreted in the input and produce characters. Other values disable the multiple parsing in text and/or math mode:

```
\normalsupmarkmode0 $ X^58 \quad X^^58 $ ^^58
\normalsupmarkmode1 $ X^58 \quad X^^58 $ ^^58
\normalsupmarkmode2 $ X^58 \quad X^^58 $ % ^^58 : error
```

In ConT<sub>E</sub>Xt we default to one but also have the `\catcode` set to 12 and the `\amcode` to 7.

```
X58   X X X
X58   5X8 X
X58   5X8
```

## 827 `\swapcsvalues`

Because we mention some `def` and `let` primitives here, it makes sense to also mention a primitive that will swap two values (meanings). This one has to be used with care. Of course that what gets swapped has to be of the same type (or at least similar enough not to cause issues). Registers for instance store their values in the token, but as soon as we are dealing with token lists we also need to keep an eye on reference counting. So, to some extent this is an experimental feature.

## 828 `\tabsize`

This primitive can be used in the preamble of an alignment and sets the size of a column, as in:

```
\halign{%
  \aligncontent          \aligntab
  \aligncontent\tabsize 3cm \aligntab
  \aligncontent          \aligntab
  \aligncontent\tabsize 0cm \cr
  1  \aligntab 111\aligntab 1111\aligntab 11\cr
  222\aligntab 2  \aligntab 2222\aligntab 22\cr
}
```

As with `\tabskip` you need to reset the value explicitly, so that is why we get two wide columns:

```
1 111 1111 11
2222 2222 22
```

## 829 `\tabskip`

This traditional primitive can be used in the preamble of an alignment and sets the space added between columns, for example:

```
\halign{%
  \aligncontent          \aligntab
  \aligncontent\tabskip 3cm \aligntab
```

```

\aligncontent          \aligntab
\aligncontent\tabskip 0cm \cr
1 \aligntab 111\aligntab 1111\aligntab 11\cr
222\aligntab 2 \aligntab 2222\aligntab 22\cr
}

```

You need to reset the skip explicitly, which is why we get it applied twice here:

## 830 \textdirection

This set the text direction to l2r (0) or r2l (1). It also triggers additional checking for balanced flipping in node lists.

## 831 \textfont

This primitive is like \font but with a family number as (first) argument so it is specific for math. It is the largest one of the three family members; its relatives are \scriptfont and \scriptscriptfont.

## 832 \textstyle

One of the main math styles; integer representation: 2.

## 833 \the

The \the primitive serializes the following token, when applicable: integers, dimensions, token registers, special quantities, etc. The catcodes of the result will be according to the current settings, so in \the\dimen0, the pt will have catcode ‘letter’ and the number and period will become ‘other’.

## 834 \thewithoutunit

The \the primitive, when applied to a dimension variable, adds a pt unit. because dimensions are the only traditional unit with a fractional part they are sometimes used as pseudo floats in which case \thewithoutunit can be used to avoid the unit. This is more convenient than stripping it off afterwards (via an expandable macro).

## 835 \thickmuskip

A predefined mu skip register that can be used in math (inter atom) spacing. The current value is 5.0mu plus 3.0mu minus 1.0mu. In traditional T<sub>E</sub>X most inter atom spacing is hard coded using the predefined registers.

## 836 \thinmuskip

A predefined mu skip register that can be used in math (inter atom) spacing. The current value is 3.0mu. In traditional T<sub>E</sub>X most inter atom spacing is hard coded using the predefined registers.

**837 \time**

This internal number starts out with minute (starting at midnight) that the job started.

**838 \tinymuskip**

A predefined mu skip register that can be used in math (inter atom) spacing. The current value is 2.0mu minus 1.0mu. This one complements \thinmuskip, \medmuskip, \thickmuskip and the new \pettymuskip

**839 \tocharacter**

The given number is converted into an utf-8 sequence. In LuaTeX this one is named \Uchar.

**840 \todimension**

The following code gives this: 1234.0pt and like its numeric counterparts accepts anything that resembles a number this one goes beyond (user, internal or pseudo) registers values too.

```
\scratchdimen = 1234pt \todimension\scratchdimen
```

**841 \tohexadecimal**

The following code gives this: 4D2 with uppercase letters.

```
\scratchcounter = 1234 \tohexadecimal\scratchcounter
```

**842 \tointeger**

The following code gives this: 1234 and is equivalent to \number.

```
\scratchcounter = 1234 \tointeger\scratchcounter
```

**843 \tokenized**

Just as \expanded has a counterpart \unexpanded, it makes sense to give \detokenize a companion:

```
\edef\foo{\detokenize{\inframed{foo}}}  
\edef\oof{\detokenize{\inframed{oof}}}
```

```
\meaning\foo \crlf \dontleavehmode\foo
```

```
\edef\foo{\tokenized{\foo\foo}}
```

```
\meaning\foo \crlf \dontleavehmode\foo
```

```
\dontleavehmode\tokenized{\foo\oof}
```

```
macro:\inframed {foo}  
\inframed {foo}
```

```
macro:\inframed {foo}\inframed {foo}
```

foo	foo
-----	-----

foo	foo	oof
-----	-----	-----

This primitive is similar to:

```
\def\tokenized#1{\scantextokens\expandafter{\normalexpanded{#1}}}
```

and should be more efficient, not that it matters much as we don't use it that much (if at all).

## 844 \toks

This is the accessor of a token register so it expects a number or \toksdef'd macro.

## 845 \toksapp

One way to append something to a token list is the following:

```
\scratchtoks\expandafter{\the\scratchtoks more stuff}
```

This works all right, but it involves a copy of what is already in \scratchtoks. This is seldom a real issue unless we have large token lists and many appends. This is why LuaTeX introduced:

```
\toksapp\scratchtoks{more stuff}  
\toksapp\scratchtoksone\scratchtokstwo
```

At some point, when working on LuaMetaTeX, I realized that primitives like this one and the next appenders and prependers to be discussed were always on the radar of Taco and me. Some were even implemented in what we called eetex: extended  $\varepsilon$ -TeX, and we even found back the prototypes, dating from pre-pdfTeX times.

## 846 \toksdef

The given name (control sequence) will be bound to the given token register (a number). Often this primitive is hidden in a high level macro that manages allocation.

## 847 \tokspre

Where appending something is easy because of the possible \expandafter trickery a prepend would involve more work, either using temporary token registers and/or using a mixture of the (no)expansion added by  $\varepsilon$ -TeX, but all are kind of inefficient and cumbersome.

```
\tokspre\scratchtoks{less stuff}  
\tokspre\scratchtoksone\scratchtokstwo
```

This prepends the token list that is provided.

## 848 \tolerance

When the par builder runs into a line with a badness larger than this value and when \emergencys-tretch is set a third pass is enabled. In LuaMetaTeX we can have more than one second pass and there are more parameters that influence the process.

## 849 `\tolerant`

This prefix tags the following macro as being tolerant with respect to the expected arguments. It only makes sense when delimited arguments are used or when braces are mandate.

```
\tolerant\def\foo[#1]#*[#2]{(#1)(#2)}
```

This definition makes `\foo` tolerant for various calls:

```
\foo \foo[1] \foo [1] \foo[1] [2] \foo [1] [2]
```

these give:  $()(1)() (1)() (1)(2) (1)(2)$ . The spaces after the first call disappear because the macro name parser gobbles it, while in the second case the `#*` gobbles them. Here is a variant:

```
\tolerant\def\foo[#1]#,[#2]{!#1!#2!}
```

```
\foo[?] x
```

```
\foo[?] [?] x
```

```
\tolerant\def\foo[#1]#*[#2]{!#1!#2!}
```

```
\foo[?] x
```

```
\foo[?] [?] x
```

We now get the following:

```
!?! x !?! x
```

```
!?!x !?! x
```

Here the `#`, remembers that spaces were gobbles and they will be put back when there is no further match. These are just a few examples of this tolerant feature. More details can be found in the lowlevel manuals.

## 850 `\tomathstyle`

Internally math styles are numbers, where `\displaystyle` is 0 and `\crampedscriptscriptstyle` is 7. You can convert the verbose style to a number with `\tomathstyle`.

## 851 `\topmark`

This is a reference to the last mark on the previous (split off) page, it gives back tokens.

## 852 `\topmarks`

This is a reference to the last mark with the given id (a number) on the previous page, it gives back tokens.

## 853 `\topskip`

This is the amount of glue that is added to the top of a (new) page.



## 854 `\toscaled`

The following code gives this: 1234.0 is similar to `\todimension` but omits the pt so that we don't need to revert to some nasty stripping code.

```
\scratchdimen = 1234pt \toscaled\scratchdimen
```

## 855 `\tosparsedimension`

The following code gives this: 1234pt where 'sparse' indicates that redundant trailing zeros are not shown.

```
\scratchdimen = 1234pt \tosparsedimension\scratchdimen
```

## 856 `\tosparsescaled`

The following code gives this: 1234 where 'sparse' means that redundant trailing zeros are omitted.

```
\scratchdimen = 1234pt \tosparsescaled\scratchdimen
```

## 857 `\tpack`

This primitive is like `\vtop` but without the callback overhead.

## 858 `\tracingadjusts`

In LuaMetaTeX the adjust feature has more functionality and also is carried over. When set to a positive values `\vadjust` processing reports details. The higher the number, the more you'll get.

## 859 `\tracingalignments`

When set to a positive value the alignment mechanism will keep you informed about what is done in various stages. Higher values unleash more information, including what callbacks kick in.

## 860 `\tracingassigns`

When set to a positive values assignments to parameters and variables are reported on the console and/or in the log file. Because LuaMetaTeX avoids redundant assignments these don't get reported.

## 861 `\tracingcommands`

When set to a positive values the commands (primitives) are reported on the console and/or in the log file.

## 862 `\tracingexpressions`

The extended expression commands like `\numexpression` and `\dimexpression` can be traced by setting this parameter to a positive value.

### 863 \tracingfullboxes

When set to a positive value the box will be shown in case of an overfull box. When a quality callback is set this will not happen as all reporting is then delegated.

### 864 \tracinggroups

When set to a positive values grouping is reported on the console and/or in the log file.

### 865 \tracinghyphenation

When set to a positive values the hyphenation process is reported on the console and/or in the log file.

### 866 \tracingifs

When set some details of what gets tested and what results are seen is reported.

### 867 \tracinginserts

A positive value enables tracing where values larger than 1 will report more details.

### 868 \tracinglevels

The lines in a log file can be prefixed with some details, depending on the bits set:

0x1 current group  
0x2 current input  
0x4 catcode table

### 869 \tracinglists

At various stages the lists being processed can be shown. This is mostly an option for developers.

### 870 \tracinglostchars

When set to one characters not present in a font will be reported in the log file, a value of two will also report this on the console.

### 871 \tracingmacros

This parameter controls reporting of what macros are seen and expanded.

### 872 \tracingmarks

Marks are information blobs that track states that can be queried when a page is handled over to the shipout routine. They travel through the system in a bit different than traditionally: like like adjusts

and inserts deeply buried ones bubble up to outer level boxes. This parameters controls what progress gets reported.

### 873 `\tracingmath`

The higher the value, the more information you will get about the various stages in rendering math. Because tracing of nodes is rather verbose you need to know a bit what this engine does. Conceptually there are differences between the LuaMetaTeX and traditional engine, like more passes, inter-atom spacing, different low level mechanisms. This feature is mostly meant for developers who tweak the many available parameters.

### 874 `\tracingnesting`

A positive value triggers log messages about the current level.

### 875 `\tracingnodes`

When set to a positive value more details about nodes (in boxes) will be reported. Because this is also controlled by callbacks what gets reported is macro package dependent.

### 876 `\tracingonline`

The engine has two output channels: the log file and the console and by default most tracing (when enabled) goes to the log file. When this parameter is set to a positive value tracing will also happen in the console. Messages from the Lua end can be channeled independently.

### 877 `\tracingoutput`

Values larger than one result in some information about what gets passed to the output routine.

### 878 `\tracingpages`

Values larger than one result in some information about the page building process. In LuaMetaTeX there is more info for higher values.

### 879 `\tracingparagraphs`

Values larger than one result in some information about the par building process. In LuaMetaTeX there is more info for higher values.

### 880 `\tracingpasses`

In LuaMetaTeX you can configure additional second stage par builder passes and this parameter controls what gets reported on the console and/or in the log file.

### 881 `\tracingpenalties`

This setting triggers reporting of actions due to special penalties in the page builder.

## 882 \tracingrestores

When set to a positive values (re)assignments after grouping to parameters and variables are reported on the console and/or in the log file. Because LuaMetaT<sub>E</sub>X avoids redundant assignments these don't get reported.

```
883 \tracingstats
```

This parameter is a dummy in LuaMetaTeX. There are anyway some statistic reported when the format is made but for a regular run it is up to the macro package to come up with useful information.

## 884 \tsplit

This splits like `\vsplit` but it returns a `\vtop` box instead.

## 885 \uccode

When the `\uppercase` operation is applied the uppercase code of a character is used for the replacement. This primitive is used to set that code, so it expects two character number.

**886 \uchyph**

When set to a positive number words that start with a capital will be hyphenated.

## 887 \uLeaders



















This leader adapts itself after a paragraph has been typeset. Here are a few examples:

test	<b>\leaders</b>	<b>\hbox</b>	<b>{x}\hfill\</b>	test
test	<b>\uleaders</b>	<b>\hbox</b>	<b>{x x x x}\hfill\</b>	test
test		<b>\hbox</b>	<b>{x x x x}\hskip 3cm plus 1cm\</b>	test
test	<b>\uleaders</b>	<b>\hbox</b>	<b>{x x x x}\hskip 3cm plus 1cm\</b>	test

When an \uleaders is used the glue in the given box will be adapted to the available space.

[illegible]

Optionally the callback followed by a number can be given, in which case a callback kicks in that gets that the node, a group identifier, and the number passed. It permits (for instance) adaptive graphics:

1=i  6=vi  11=xi  16=xvi  21=xxi  26=xxvi  31=xxxi  36=xxxvi  41=xli  
46=xlvi  51=li  56=lvi  61=lxvi  66=lxvi  71=lxxi  76=lxxvi  81=lxxxi  
86=lxxxvi  91=xci  96=xcvi  .

**888 \unboundary**

When possible a preceding boundary node will be removed.

## 889 `\indent`

When possible the already added indentation will be removed.

## 890 `\underline`

This is a math specific primitive that draws a line under the given content. It is a poor mans replacement for a delimiter. The thickness is set with `\Umathunderbarrule`, the distance between content and rule is set by `\Umathunderbarvgap` and `\Umathunderbarkern` is added above the rule. The style used for the content under the rule can be set with `\Umathunderlinevariant`. See `\overline` for what these parameters do.

## 891 `\unexpanded`

This is an  $\varepsilon$ -T<sub>E</sub>X enhancement. The content will not be expanded in a context where expansion is happening, like in an `\edef`. In ConT<sub>E</sub>Xt you need to use `\normalunexpanded` because we already had a macro with that name.

```
\def \A{!}           \meaning\A
\def \B{?}           \meaning\B
\edef\C{\A\B}        \meaning\C
\edef\C{\normalunexpanded{\A}\B} \meaning\C
```

```
macro:!  
macro:?  
macro:!?  
macro:\A ?
```

## 892 `\unexpandedendless`

This one loops forever so you need to quit explicitly.

## 893 `\unexpandedloop`

As follow up on `\expandedloop` we now show its counterpart:

```
\edef\whatever  
  {\unexpandedloop 1 10 1  
   {\scratchcounter=\the\currentloopiterator\relax}}
```

```
\meaningasis\whatever
```

```
\def \whatever {\scratchcounter =0\relax \scratchcounter =0\relax \scratchcounter =0\relax \scratchcounter  
=0\relax \scratchcounter =0\relax \scratchcounter =0\relax \scratchcounter =0\relax \scratchcounter =0\relax  
\scratchcounter =0\relax \scratchcounter =0\relax }
```

The difference between the (un)expanded loops and a local controlled one is shown here. Watch the out of order injection of A's.

```
\edef\TestA{\localcontrolledloop 1 5 1 {A}} % out of order
```

```
\edef\TestB{\expandedloop      1 5 1 {B}}
\edef\TestC{\unexpandedloop    1 5 1 {C\relax}}
```

AAAAA

We show the effective definition as well as the outcome of using them

```
\meaningasis\TestA
\meaningasis\TestB
\meaningasis\TestC
```

```
A: \TestA
B: \TestB
C: \TestC
```

```
\def \TestA {}
\def \TestB {BBBBB}
\def \TestC {C\relax C\relax C\relax C\relax C\relax }
```

```
A:
B: BBBBB
C: CCCCC
```

Watch how because it is empty `\TestA` has become a constant macro because that's what deep down empty boils down to.

## 894 `\unexpandedrepeat`

This one takes one instead of three arguments which looks better in simple loops.

## 895 `\unhbox`

A box is a packaged list and once packed travels through the system as a single object with properties, like dimensions. This primitive injects the original list and discards the wrapper.

## 896 `\unhcopy`

This is like `\unhbox` but keeps the original. It is one of the more costly operations.

## 897 `\unhpack`

This primitive is like `\unhbox` but without the callback overhead.

## 898 `\unkern`

This removes the last kern, if possible.

## 899 `\unless`

This  $\varepsilon$ -TeX prefix will negate the test (when applicable).

```

\ifx\one\two YES\else NO\fi
\unless\ifx\one\two NO\else YES\fi

```

This primitive is hardly used in ConT<sub>E</sub>Xt and we probably could get rid of these few cases.

## 900 \unletfrozen

A frozen macro cannot be redefined: you get an error. But as nothing in T<sub>E</sub>X is set in stone, you can do this:

```

\frozen\def\MyMacro{...}
\unletfrozen\MyMacro

```

and `\MyMacro` is no longer protected from overloading. It is still undecided to what extend ConT<sub>E</sub>Xt will use this feature.

## 901 \unletprotected

The complementary operation of `\letprotected` can be used to unprotect a macro, so that it gets expandable.

```

\def \MyMacroA{alpha}
\protected \def \MyMacroB{beta}
\edef \MyMacroC{\MyMacroA\MyMacroB}
\unletprotected \MyMacroB
\edef \MyMacroD{\MyMacroA\MyMacroB}
\meaning \MyMacroC\crlf
\meaning \MyMacroD\par

```

Compare this with the example in the previous section:

```

macro:alpha\MyMacroB
macro:alphabeta

```

## 902 \unpenalty

This removes the last penalty, if possible.

## 903 \unskip

This removes the last glue, if possible.

## 904 \untraced

Related to the meaning providers is the `\untraced` prefix. It marks a macro as to be reported by name only. It makes the macro look like a primitive.

```

\def\foo{}
\untraced\def\oof{}

```

```
\scratchtoks{\foo\foo\oof\oof}
\tracingall \the\scratchtoks \tracingnone
```

This will show up in the log as follows:

```
1:4: {\the}
1:5: \foo ->
1:5: \foo ->
1:5: \oof
1:5: \oof
```

This is again a trick to avoid too much clutter in a log. Often it doesn't matter to users what the meaning of a macro is (if they trace at all).<sup>7</sup>

## 905 \unvbox

A box is a packaged list and once packed travels through the system as a single object with properties, like dimensions. This primitive injects the original list and discards the wrapper.

## 906 \unvcopy

This is like \unvbox but keeps the original. It is one of the more costly operations.

## 907 \unvpack

This primitive is like \unvbox but without the callback overhead.

## 908 \uppercase

See its counterpart \lowercase for an explanation.

## 909 \vadjust

This injects a node that stores material that will be injected before or after the line where it has become part of. In LuaMetaTeX there are more features, driven by keywords.

## 910 \valign

This command starts vertically aligned material. Its counterpart \halign is used more frequently. Most macro packages provide wrappers around these commands. First one specifies a preamble which is then followed by entries (rows and columns).

## 911 \variablefam

In traditional TeX sets the family of what are considered variables (class 7) to the current family (which often means that they adapt to the current alphabet) and then injects a math character of class

---

<sup>7</sup> An earlier variant could also hide the expansion completely but that was just confusing.



ordinary. This parameter can be used to obey the given class when the family set for a character is the same as this parameter. So we then use the given class with the current family. It is mostly there for compatibility with LuaT<sub>E</sub>X and experimenting (outside ConT<sub>E</sub>Xt).

## 912 `\vbadness`

This sets the threshold for reporting a (vertical) badness value, its current value is 0.

## 913 `\vbox`

This creates a vertical box. In the process callbacks can be triggered that can preprocess the content, influence line breaking as well as assembling the resulting paragraph. More can be found in dedicated manuals. The baseline is at the bottom.

## 914 `\vcenter`

In traditional T<sub>E</sub>X this box packer is only permitted in math mode but in LuaMetaT<sub>E</sub>X it also works in text mode. The content is centered in the vertical box.

## 915 `\vfil`

This is a shortcut for `\vskip` plus 1 fil (first order filler).

## 916 `\vfill`

This is a shortcut for `\vskip` plus 1 fill (second order filler).

## 917 `\vfilneg`

This is a shortcut for `\vskip` plus - 1 fil so it can compensate `\vfil`.

## 918 `\vfuzz`

This dimension sets the threshold for reporting vertical boxes that are under- or overfull. The current value is 0.1pt.

## 919 `\virtualhrule`

This is a horizontal rule with zero dimensions from the perspective of the frontend but the backend can access them as set.

## 920 `\virtualvrule`

This is a vertical rule with zero dimensions from the perspective of the frontend but the backend can access them as set.

**921 \vkern**

This primitive is like `\kern` but will force the engine into vertical mode if it isn't yet.

**922 \vpack**

This primitive is like `\vbox` but without the callback overhead.

**923 \vpenalty**

This primitive is like `\penalty` but will force the engine into vertical mode if it isn't yet.

**924 \vrule**

This creates a vertical rule. Unless the height and depth are set they will stretch to fix the available space. In addition to the traditional width, height and depth specifiers some more are accepted. These are discussed in other manuals. See `\hrule` for a simple example.

**925 \vsize**

This sets (or gets) the current vertical size. While setting the `\hsize` inside a `\vbox` has consequences, setting the `\vsize` mostly makes sense at the outer level (the page).

**926 \vskip**

The given glue is injected in the vertical list. If possible vertical mode is entered.

**927 \vsplit**

This operator splits a given amount from a vertical box. In LuaMetaTeX we can split to but also upto, so that we don't have to repack the result in order to see how much is actually in there.

**928 \vss**

This is the vertical variant of `\hss`. See there for what it means.

**929 \vtop**

This creates a vertical box. In the process callbacks can be triggered that can preprocess the content, influence line breaking as well as assembling the resulting paragraph. More can be found in dedicated manuals. The baseline is at the top.

**930 \wd**

Returns the width of the given box.

### 931 `\widowpenalties`

This is an array of penalty put before the last lines in a paragraph. High values discourage (or even prevent) a lone line at the beginning of a next page. This command expects a count value indicating the number of entries that will follow. The first entry is ends up before the last line.

### 932 `\widowpenalty`

This is the penalty put before a widow line in a paragraph. High values discourage (or even prevent) a lone line at the beginning of a next page.

### 933 `\wordboundary`

The hyphenation routine has to decide where a word begins and ends. If you want to make sure that there is a proper begin or end of a word you can inject this boundary.

### 934 `\wrapuppar`

What this primitive does can best be shown with an example:

some text`\wrapuppar{one}` and some`\wrapuppar{two}` more

We get:

some text and some more twoone

So, it is a complementary command to `\everypar`. It can only be issued inside a paragraph.

### 935 `\xdef`

This is an alternative for `\global\edef`:

```
\xdef\MyMacro{...}
```

### 936 `\xdefcsname`

This is the companion of `\xdef`:

```
\expandafter\xdef\csname MyMacro:1\endcsname{...}
\xdefcsname MyMacro:1\endcsname{...}
```

### 937 `\xleaders`

See `\gleaders` for an explanation.

### 938 `\xspaceskip`

Normally the glue inserted when a space is encountered after a character with a space factor other than 1000 is taken from the font (fontdimen 7) unless this parameter is set in which case its value is added.

**939 \xtoks**

This is the global variant of \etoks.

**940 \xtoksapp**

This is the global variant of \etoksapp.

**941 \xtokspre**

This is the global variant of \etokspre.

**942 \year**

This internal number starts out with the year that the job started.

**Obsolete**

The LuaMetaTeX engine has more than its LuaTeX ancestor but it also has less. Because in the end the local control mechanism performed quite okay I decided to drop the \immediateassignment and \immediateassigned variants. They sort of used the same trick so there isn't much to gain and it was less generic (read: error prone).

## Syntax

### 1 accent

**t** `\accent`  
 $[xoffset\ dimension][yoffset\ dimension] integer character$

### 2 aftersomething

**l** `\afterassigned`  
 $\{tokens\}$   
**t** `\afterassignment`  
 $token$   
**t** `\aftergroup`  
 $token$   
**l** `\aftergrouped`  
 $\{tokens\}$   
**l** `\atendoffile`  
 $token$   
**l** `\atendoffiled`  
 $[reverse]\{tokens\}$   
**l** `\atendofgroup`  
 $token$   
**l** `\atendofgrouped`  
 $\{tokens\}$

### 3 alignmenttab

**l** `\aligntab`

### 4 arithmetic

**t** `\advance`  
 $quantity [by] quantity$   
**l** `\advanceby`  
 $quantity quantity$   
**t** `\divide`  
 $quantity [by] quantity$   
**l** `\divideby`  
 $quantity quantity$   
**l** `\edivide`  
 $quantity quantity$   
**l** `\edivideby`  
 $quantity quantity$   
**t** `\multiply`  
 $quantity [by] quantity$

**l** `\multiplyby`  
 $quantity quantity$   
**l** `\rdivide`  
 $quantity quantity$   
**l** `\rdivideby`  
 $quantity quantity$

### 5 association

**l** `\associateunit`  
 $\cs [=] integer$   
 $> \cs : integer$

### 6 auxiliary

**l** `\insertmode`  
 $integer$   
 $: integer$   
**e** `\interactionmode`  
 $integer$   
 $: integer$   
**t** `\prevdepth`  
 $dimension$   
 $: dimension$   
**t** `\prevgraf`  
 $integer$   
 $: integer$   
**t** `\spacefactor`  
 $integer$   
 $: integer$

### 7 begingroup

**t** `\begingroup`  
**l** `\beginmathgroup`  
**l** `\beginsimplegroup`

### 8 beginlocal

**l** `\beginlocalcontrol`  
**l** `\expandedendless`  
 $\{tokens\}$   
**l** `\expandedloop`  
 $integer integer integer \{tokens\}$   
**l** `\expandedrepeat`  
 $integer \{tokens\}$

```

l \localcontrol
    tokens\endlocalcontrol
l \localcontrolled
    { tokens }
l \localcontrolledendless
    { tokens }
l \localcontrolledloop
    see \expandedloop
l \localcontrolledrepeat
    integer { tokens }
l \unexpandedendless
    { tokens }
l \unexpandedloop
    see \expandedloop
l \unexpandedrepeat
    integer { tokens }

```

## 9 beginparagraph

```

t \indent
t \noindent
l \parattribute
    integer [=] integer
l \quitvmode
l \snapshotpar
    cardinal
    : integer
l \undent
l \wrapuppar
    [ reverse ] { tokens }

```

## 10 boundary

```

l \boundary
    [=] integer
l \mathboundary
    [=] integer
l \noboundary
l \optionalboundary
    [=] integer
l \pageboundary
    [=] integer
l \protrusionboundary
    [=] integer
l \wordboundary

```

## 11 boxproperty

```

l \boxadapt
    ( index | box ) [=] integer
    > ( index | box ) : dimension
l \boxanchor
    see \boxadapt
l \boxanchors
    ( index | box ) [=] integer integer
    > ( index | box ) : integer
l \boxattribute
    ( index | box ) integer [=] integer
    > ( index | box ) integer : integer
l \boxdirection
    see \boxadapt
l \boxfinalize
    see \boxadapt
l \boxfreeze
    see \boxadapt
l \boxgeometry
    see \boxadapt
l \boxlimit
    TODO
l \boxlimitate
    see \boxadapt
l \boxorientation
    see \boxadapt
l \boxrepack
    ( index | box )
    > ( index | box ) : dimension
l \boxshift
    ( index | box ) [=] dimension
    > ( index | box ) : dimension
l \boxshrink
    see \boxrepack
l \boxsource
    see \boxadapt
l \boxstretch
    see \boxrepack
l \boxtarget
    see \boxadapt
l \boxtotal
    see \boxrepack
l \boxvadjust
    ( index | box ) { tokens }
    > ( index | box ) : cardinal
l \boxxmove
    see \boxshift

```

**l \boxxoffset**  
     see \boxshift  
**l \boxymove**  
     see \boxshift  
**l \boxyoffset**  
     see \boxshift  
**t \dp**  
     see \boxshift  
**t \ht**  
     see \boxshift  
**t \wd**  
     see \boxshift

## 12 caseshift

**t \lowercase**  
     { *tokens* }  
**t \uppercase**  
     { *tokens* }

## 13 catcodetable

**l \initcatcodetable**  
     *integer*  
**l \restorecatcodetable**  
     TODO  
**l \savecatcodetable**  
     *integer*

## 14 charnumber

**t \char**  
     *integer*  
**l \glyph**  
     [ *xoffset dimension* ] [ *yoffset dimension* ] [ *scale integer* ] [ *xscale integer* ] [ *yscale integer* ] [ *left dimension* ] [ *right dimension* ] [ *raise dimension* ] [ *options integer* ] [ *font integer* ] [ *id integer* ] *integer*

## 15 combinetoks

**l \etoks**  
     *toks* { *tokens* }  
**l \etoksapp**  
     *toks* { *tokens* }

**l \etokspre**  
     *toks* { *tokens* }  
**l \gtoksapp**  
     *toks* { *tokens* }  
**l \gtokspre**  
     *toks* { *tokens* }  
**l \toksapp**  
     *toks* { *tokens* }  
**l \tokspre**  
     *toks* { *tokens* }  
**l \xtoks**  
     *toks* { *tokens* }  
**l \xtoksapp**  
     *toks* { *tokens* }  
**l \xtokspre**  
     *toks* { *tokens* }

## 16 convert

**l \csactive**  
     > *token* : *tokens*  
**l \csstring**  
     > *token* : *tokens*  
**l \detokened**  
     > ( *\cs* | { *tokens* } | *toks* ) : *tokens*  
**l \detokenized**  
     > { *tokens* } : *tokens*  
**l \directlua**  
     > { *tokens* } : *tokens*  
**l \expanded**  
     > { *tokens* } : *tokens*  
**t \fontname**  
     > ( *font* | *integer* ) : *tokens*  
**l \fontspecifiedname**  
     > ( *font* | *integer* ) : *tokens*  
**l \formatname**  
     : *tokens*  
**t \jobname**  
     : *tokens*  
**l \luabytecode**  
     > *integer* : *tokens*  
**l \luaescapestring**  
     > { *tokens* } : *tokens*  
**l \luafunction**  
     > *integer* : *tokens*  
**l \luatexbanner**  
     : *tokens*  
**t \meaning**  
     > *token* : *tokens*

```

l \meaningasis
    > token : tokens
l \meaningful
    > token : tokens
l \meaningfull
    > token : tokens
l \meaningles
    > token : tokens
l \meaningless
    > token : tokens
t \number
    > integer : tokens
t \romannumeral
    > integer : tokens
l \semiexpanded
    > {tokens} : tokens
t \string
    > token : tokens
l \tocharacter
    > integer : tokens
l \todimension
    > dimension : tokens
l \tohexadecimal
    > integer : tokens
l \tointeger
    > integer : tokens
l \tomathstyle
    > mathstyle : tokens
l \toscaled
    > dimension : tokens
l \tosparsedimension
    > dimension : tokens
l \tosparsecaled
    > dimension : tokens

```

## 17 csname

```

l \begincsname
    tokens\endcsname
t \csname
    tokens\endcsname
l \futurecsname
    tokens\endcsname
l \lastnamedcs

```

## 18 def

```

l \cdef
    \cs [preamble] {tokens}

```

```

l \cdefcsname
    tokens\endcsname [preamble] {tokens}
t \def
    \cs [preamble] {tokens}
l \defcsname
    tokens\endcsname [preamble] {tokens}
t \edef
    \cs [preamble] {tokens}
l \edefcsname
    tokens\endcsname [preamble] {tokens}
t \gdef
    \cs [preamble] {tokens}
l \gdefcsname
    tokens\endcsname [preamble] {tokens}
t \xdef
    \cs [preamble] {tokens}
l \xdefcsname
    tokens\endcsname [preamble] {tokens}

```

## 19 definecharcode

```

l \Udelcode
    integer [=] integer
    > integer : integer
l \Umathcode
    integer [=] integer
    > integer : integer
l \amcode
    integer [=] integer
    > integer : integer
t \catcode
    integer [=] integer
    > integer : integer
t \delcode
    integer [=] integer
    > integer : integer
l \hccode
    integer [=] integer
    > integer : integer
l \hmcode
    integer [=] integer
    > integer : integer
t \lccode
    integer [=] integer
    > integer : integer
t \mathcode
    integer [=] integer
    > integer : integer

```



**t \sfcode**  
*integer [=] integer*  
*> integer : integer*

**t \uccode**  
*integer [=] integer*  
*> integer : integer*

## 20 definefamily

**t \scriptfont**  
*family ( font | integer )*  
*> family : integer*

**t \scriptscriptfont**  
 see \scriptfont

**t \textfont**  
 see \scriptfont

## 21 definefont

**t \font**  
*\cs ( { filename } | filename ) [ ( at*  
*dimension | scaled integer ) ]*  
*: tokens*

## 22 delimiternumber

**l \Udelimiter**  
*integer integer integer*

**t \delimiter**  
*integer*

## 23 discretionary

**t \-**  
**l \automaticdiscretionary**  
**t \discretionary**  
*[ penalty ] [ postword ] [ preword ]*  
*[ break ] [ nobreak ] [ options ] [ class ]*  
*{ tokens } { tokens } { tokens }*

**l \explicitdiscretionary**

## 24 endcsname

**t \endcsname**

## 25 endgroup

**t \endgroup**

**l \endmathgroup**  
**l \endsimplegroup**

## 26 endjob

**t \dump**  
**t \end**

## 27 endlcal

**l \endlocalcontrol**

## 28 endparagraph

**t \par**

## 29 endtemplate

**l \aligncontent**

**t \cr**

**t \crrc**

**t \noalign**  
*{ tokens }*

**t \omit**

**l \realign**  
 TODO

**t \span**

## 30 equationnumber

**t \eqno**  
*{ tokens }*

**t \legno**  
*{ tokens }*

## 31 expandafter

**l \expand**  
*token*

**l \expandactive**  
*token*

**t \expandafter**  
*token token*

**l \expandafterpars**  
*token*

**l** `\expandafterspaces`

*token*

**l** `\expandcstoken`

*token*

**l** `\expandedafter`

*token {tokens}*

**l** `\expandparameter`

*integer*

**l** `\expandtoken`

*token*

**l** `\expandtoks`

*{tokens}*

**l** `\futureexpand`

*token token token*

**l** `\futureexpandis`

TODO

**l** `\futureexpandisap`

TODO

**l** `\semiexpand`

*token*

**e** `\unless`

## 32 explicit space

**t** `\`

**l** `\explicit space`

TODO

## 33 fontproperty

**l** `\cfcode`

*(font | integer) integer [=] integer*

*> (font | integer) integer : integer*

**l** `\efcode`

see `\cfcode`

**t** `\fontdimen`

*(font | integer) integer [=] dimension*

*> (font | integer) integer : dimension*

**t** `\hyphenchar`

*(font | integer) [=] integer*

*> (font | integer) : integer*

**l** `\lpcode`

see `\fontdimen`

**l** `\rpcode`

see `\fontdimen`

**l** `\scaledfontdimen`

see `\hyphenchar`

**t** `\skewchar`

see `\hyphenchar`

## 34 getmark

**t** `\botmark`

**e** `\botmarks`

*integer*

**l** `\currentmarks`

*integer*

**t** `\firstmark`

**e** `\firstmarks`

*integer*

**t** `\splitbotmark`

**e** `\splitbotmarks`

*integer*

**t** `\splitfirstmark`

**e** `\splitfirstmarks`

*integer*

**t** `\topmark`

**e** `\topmarks`

*integer*

## 35 halign

**t** `\halign`

*[ attr integer integer ] [ callback integer ] [ discard ] [ noskips ] [ reverse ] [ to dimension ] [ spread dimension ] {tokens}*

## 36 hmove

**t** `\moveleft`

*dimension box*

**t** `\moveright`

*dimension box*

## 37 hrule

**t** `\hrule`

*[ attr integer [=] integer ] [ width dimension ] [ height dimension ] [ depth dimension ] [ left dimension ] [ right dimension ] [ top dimension ] [ bottom dimension ] [ xoffset dimension ] [ yoffset dimension ] [ font integer ] [ fam integer ] [ char integer ]*

**l \nohrule**

see \hrule

**l \virtualhrule**

[*attr integer* [=] *integer*] [*width dimension*] [*height dimension*] [*depth dimension*] [*left dimension*] [*right dimension*] [*top dimension*] [*bottom dimension*] [*xoffset dimension*] [*yoffset dimension*]

**38 hskip****t \hfil****t \hfill****t \hfilneg****t \hskip**

*dimension* [plus  
(*dimension* | fi[n\*l]) ] [minus  
(*dimension* | fi[n\*l]) ]

**t \hss****39 hyphenation****l \hjcode***integer* [=] *integer***t \hyphenation**{*tokens*}**l \hyphenationmin**[=] *integer***t \patterns**{*tokens*}**l \postexhyphenchar**[=] *integer***l \posthyphenchar**[=] *integer***l \preexhyphenchar**[=] *integer***l \prehyphenchar**[=] *integer***40 iftest****t \else****t \fi****t \if****l \ifabsdim***dimension*

( ! | &lt; | = | &gt; | ∈ | ∉ | ≠ | ≤ | ≥ | ≠ | ≠ )

*dimension***l \ifabsfloat**

*float* ( ! | < | = | > | ∈ | ∉ | ≠ | ≤ | ≥ | ≠ | ≠ )  
*float*

**l \ifabsnum**

*integer*  
( ! | < | = | > | ∈ | ∉ | ≠ | ≤ | ≥ | ≠ | ≠ )  
*integer*

**l \ifarguments****l \ifboolean***integer***t \ifcase***integer***t \ifcat***token***l \ifchkdim***tokens*\or**l \ifchkdimension***tokens*\or**l \ifchknum***tokens*\or**l \ifchknumber***tokens*\or**l \ifcmpdim***dimension dimension***l \ifcmpnum***integer integer***l \ifcondition**

\if...

**l \ifcramped**

TODO

**e \ifcsname***tokens*\endcsname**l \ifcstok***tokens*\relax**e \ifdefined***token***t \ifdim**

see \ifabsdim

**l \ifdimexpression***tokens*\relax**l \ifdimval***tokens*\or**l \ifempty**( *token* | { *tokens* } )**t \iffalse****l \ifflags**

\cs

**l \iffloat**  
     see \ifabsfloat  
**e \iffontchar**  
     *integer integer*  
**l \ifhaschar**  
     *token {tokens}*  
**l \ifhastok**  
     *token {tokens}*  
**l \ifhastoks**  
     *tokens\relax*  
**l \ifhasxtoks**  
     *tokens\relax*  
**t \ifhbox**  
     (index | box)  
**t \ifhmode**  
**l \iffinalignment**  
**l \ifincsname**  
     *tokens\endcsname*  
**t \ifinner**  
**l \ifinsert**  
     *integer*  
**l \ifintervaldim**  
     *dimension dimension dimension*  
**l \ifintervalfloat**  
     *integer integer integer*  
**l \ifintervalnum**  
     *float float float*  
**l \iflastnamedcs**  
**l \ifmathparameter**  
     *integer*  
**l \ifmathstyle**  
     *mathstyle*  
**t \ifmmode**  
**t \ifnum**  
     see \ifabsnum  
**l \ifnumexpression**  
     *tokens\relax*  
**l \ifnumval**  
     *tokens\or*  
**t \ifodd**  
     *integer*  
**l \ifparameter**  
     parameter\or  
**l \ifparameters**  
**l \ifrelax**  
     *token*  
**l \iftok**  
     *tokens\relax*  
**t \iftrue**

**t \ifvbox**  
     see \ifhbox  
**t \ifvmode**  
**t \ifvoid**  
     see \ifhbox  
**t \ifx**  
     *token*  
**l \ifzerodim**  
     *dimension*  
**l \ifzerofloat**  
     *float*  
**l \ifzeronum**  
     *integer*  
**t \or**  
**l \orelse**  
**l \orunless**

## 41 ignoresomething

**l \ignorearguments**  
**l \ignorenestedupto**  
     *token*  
**l \ignorepars**  
**l \ignorereset**  
**t \ignorespaces**  
**l \ignoreupto**  
     *token*

## 42 input

**t \endinput**  
**t \eofinput**  
     {tokens} ( {filename} | filename )  
**t \input**  
     ( {filename} | filename )  
**l \quitloop**  
**l \quitloopnow**  
**l \retokenized**  
     [ catcodetable ] {tokens}  
**l \scantextokens**  
     {tokens}  
**e \scantokens**  
     {tokens}  
**l \tokenized**  
     {tokens}

**43 insert**

**t \insert**  
*integer*

**44 interaction**

**t \batchmode**  
**t \errorstopmode**  
**t \nonstopmode**  
**t \scrollmode**

**45 internaldimension**

**t \boxmaxdepth**  
 [=] *dimension*  
 : *dimension*  
**t \delimitershortfall**  
 [=] *dimension*  
 : *dimension*  
**t \displayindent**  
 [=] *dimension*  
 : *dimension*  
**t \displaywidth**  
 [=] *dimension*  
 : *dimension*  
**t \emergencyextrastretch**  
 [=] *dimension*  
 : *dimension*  
**t \emergencystretch**  
 [=] *dimension*  
 : *dimension*  
**l \glyphxoffset**  
 [=] *dimension*  
 : *dimension*  
**l \glyphyoffset**  
 [=] *dimension*  
 : *dimension*  
**t \hangindent**  
 [=] *dimension*  
 : *dimension*  
**t \hfuzz**  
 [=] *dimension*  
 : *dimension*  
**t \hsize**  
 [=] *dimension*  
 : *dimension*  
**l \ignoredepthcriterion**  
 [=] *dimension*

: *dimension*  
**t \lineskiplimit**  
 [=] *dimension*  
 : *dimension*  
**t \mathsurround**  
 [=] *dimension*  
 : *dimension*  
**t \maxdepth**  
 [=] *dimension*  
 : *dimension*  
**t \nulldelimiterspace**  
 [=] *dimension*  
 : *dimension*  
**t \overfullrule**  
 [=] *dimension*  
 : *dimension*  
**l \pageextragoal**  
 [=] *dimension*  
 : *dimension*  
**t \parindent**  
 [=] *dimension*  
 : *dimension*  
**t \predisplaysize**  
 [=] *dimension*  
 : *dimension*  
**l \pxdimen**  
 [=] *dimension*  
 : *dimension*  
**t \scriptspace**  
 [=] *dimension*  
 : *dimension*  
**l \shortinlinemaththreshold**  
 [=] *dimension*  
 : *dimension*  
**t \splitmaxdepth**  
 [=] *dimension*  
 : *dimension*  
**l \tabsize**  
 [=] *dimension*  
 : *dimension*  
**t \vfuzz**  
 [=] *dimension*  
 : *dimension*  
**t \vsize**  
 [=] *dimension*  
 : *dimension*

**46 internalglue**

```

t \abovedisplayshortskip
    [=] glue
    : glue
t \abovedisplayskip
    [=] glue
    : glue
l \additionalpageskip
    [=] glue
    : glue
t \baselineskip
    [=] glue
    : glue
t \belowdisplayshortskip
    [=] glue
    : glue
t \belowdisplayskip
    [=] glue
    : glue
l \emergencyleftskip
    [=] glue
    : glue
l \emergencyrightskip
    [=] glue
    : glue
l \initialpageskip
    [=] glue
    : glue
l \initialtopskip
    [=] glue
    : glue
t \leftskip
    [=] glue
    : glue
t \lineskip
    [=] glue
    : glue
l \mathsurroundskip
    [=] glue
    : glue
l \maththreshold
    [=] glue
    : glue
l \parfillleftskip
    [=] glue
    : glue
l \parfillrightskip
    [=] glue

```

```

    : glue
t \parfillskip
    [=] glue
    : glue
l \parinitleftskip
    [=] glue
    : glue
l \parinitrightskip
    [=] glue
    : glue
t \parskip
    [=] glue
    : glue
t \rightskip
    [=] glue
    : glue
t \spaceskip
    [=] glue
    : glue
t \splittopskip
    [=] glue
    : glue
t \tabskip
    [=] glue
    : glue
t \topskip
    [=] glue
    : glue
t \xspaceskip
    [=] glue
    : glue

```

**47 internalinteger**

```

t \adjdemerits
    [=] integer
    : integer
l \adjustspacing
    [=] integer
    : integer
l \adjustspacingshrink
    [=] integer
    : integer
l \adjustspacingstep
    [=] integer
    : integer
l \adjustspacingstretch
    [=] integer
    : integer

```

```

l \alignmentcellsource
    [=] integer
    : integer
l \alignmentwrapsource
    [=] integer
    : integer
l \automatichyphenpenalty
    [=] integer
    : integer
l \automigrationmode
    [=] integer
    : integer
l \autoparagraphmode
    [=] integer
    : integer
t \binoppenalty
    [=] integer
    : integer
l \boxlimitmode
    TODO
t \brokenpenalty
    [=] integer
    : integer
l \catcodetable
    [=] integer
    : integer
t \clubpenalty
    [=] integer
    : integer
t \day
    [=] integer
    : integer
t \defaultthyphenchar
    [=] integer
    : integer
t \defaultskewchar
    [=] integer
    : integer
t \delimiterfactor
    [=] integer
    : integer
l \discretionaryoptions
    [=] integer
    : integer
t \displaywidowpenalty
    [=] integer
    : integer
l \doubleadjdemerits
    [=] integer
    : integer
t \doublehyphendemerits
    [=] integer
    : integer
t \endlinechar
    [=] integer
    : integer
t \errorcontextlines
    [=] integer
    : integer
t \escapechar
    [=] integer
    : integer
l \eufactor
    [=] integer
    : integer
l \exceptionpenalty
    [=] integer
    : integer
t \exhyphenchar
    [=] integer
    : integer
t \exhyphenpenalty
    [=] integer
    : integer
l \explicithyphenpenalty
    [=] integer
    : integer
t \fam
    [=] integer
    : integer
t \finalhyphendemerits
    [=] integer
    : integer
l \firstvalidlanguage
    [=] integer
    : integer
t \floatingpenalty
    [=] integer
    : integer
t \globaldefs
    [=] integer
    : integer
l \glyphdatafield
    [=] integer
    : integer
l \glyphoptions
    [=] integer
    : integer

```

<b>l</b> <b>\glyphscale</b>	<b>t</b> <b>\interlinepenalty</b>
[=] <i>integer</i>	[=] <i>integer</i>
: <i>integer</i>	: <i>integer</i>
<b>l</b> <b>\glyphscriptfield</b>	<b>t</b> <b>\language</b>
[=] <i>integer</i>	[=] <i>integer</i>
: <i>integer</i>	: <i>integer</i>
<b>l</b> <b>\glyphscriptscale</b>	<b>e</b> <b>\lastlinefit</b>
[=] <i>integer</i>	[=] <i>integer</i>
: <i>integer</i>	: <i>integer</i>
<b>l</b> <b>\glyphscriptscriptscale</b>	<b>t</b> <b>\lefthyphenmin</b>
[=] <i>integer</i>	[=] <i>integer</i>
: <i>integer</i>	: <i>integer</i>
<b>l</b> <b>\glyphslant</b>	<b>l</b> <b>\linebreakcriterion</b>
[=] <i>integer</i>	[=] <i>integer</i>
: <i>integer</i>	: <i>integer</i>
<b>l</b> <b>\glyphstatefield</b>	<b>l</b> <b>\linebreakoptional</b>
[=] <i>integer</i>	[=] <i>integer</i>
: <i>integer</i>	: <i>integer</i>
<b>l</b> <b>\glyphtextscale</b>	<b>l</b> <b>\linebreakpasses</b>
[=] <i>integer</i>	[=] <i>integer</i>
: <i>integer</i>	: <i>integer</i>
<b>l</b> <b>\glyphweight</b>	<b>l</b> <b>\linedirection</b>
[=] <i>integer</i>	[=] <i>integer</i>
: <i>integer</i>	: <i>integer</i>
<b>l</b> <b>\glyphxscale</b>	<b>t</b> <b>\linepenalty</b>
[=] <i>integer</i>	[=] <i>integer</i>
: <i>integer</i>	: <i>integer</i>
<b>l</b> <b>\glyphyscale</b>	<b>l</b> <b>\localbrokenpenalty</b>
[=] <i>integer</i>	[=] <i>integer</i>
: <i>integer</i>	: <i>integer</i>
<b>t</b> <b>\hangafter</b>	<b>l</b> <b>\localinterlinepenalty</b>
[=] <i>integer</i>	[=] <i>integer</i>
: <i>integer</i>	: <i>integer</i>
<b>t</b> <b>\hbadness</b>	<b>l</b> <b>\localpretolerance</b>
[=] <i>integer</i>	[=] <i>integer</i>
: <i>integer</i>	: <i>integer</i>
<b>t</b> <b>\holdinginserts</b>	<b>l</b> <b>\localtolerance</b>
[=] <i>integer</i>	[=] <i>integer</i>
: <i>integer</i>	: <i>integer</i>
<b>l</b> <b>\holdingmigrations</b>	<b>t</b> <b>\looseness</b>
[=] <i>integer</i>	[=] <i>integer</i>
: <i>integer</i>	: <i>integer</i>
<b>l</b> <b>\hyphenationmode</b>	<b>l</b> <b>\luacopyinputnodes</b>
[=] <i>integer</i>	[=] <i>integer</i>
: <i>integer</i>	: <i>integer</i>
<b>t</b> <b>\hyphenpenalty</b>	<b>l</b> <b>\mathbeginclass</b>
[=] <i>integer</i>	[=] <i>integer</i>
: <i>integer</i>	: <i>integer</i>



<b>\mathcheckfencesmode</b>	<b>\mathnolimitsmode</b>
[=] integer	[=] integer
: integer	: integer
<b>\mathdictgroup</b>	<b>\mathpenaltiesmode</b>
[=] integer	[=] integer
: integer	: integer
<b>\mathdictproperties</b>	<b>\mathpretolerance</b>
[=] integer	[=] integer
: integer	: integer
<b>\mathdirection</b>	<b>\mathrightclass</b>
[=] integer	[=] integer
: integer	: integer
<b>\mathdisplaymode</b>	<b>\mathrulesfam</b>
[=] integer	[=] integer
: integer	: integer
<b>\mathdisplaypenaltyfactor</b>	<b>\mathrulesmode</b>
[=] integer	[=] integer
: integer	: integer
<b>\mathdisplayskipmode</b>	<b>\mathscriptsmode</b>
[=] integer	[=] integer
: integer	: integer
<b>\mathdoublescriptmode</b>	<b>\mathslackmode</b>
[=] integer	[=] integer
: integer	: integer
<b>\mathendclass</b>	<b>\mathspacingmode</b>
[=] integer	[=] integer
: integer	: integer
<b>\matheqnogapstep</b>	<b>\mathsurroundmode</b>
[=] integer	[=] integer
: integer	: integer
<b>\mathfontcontrol</b>	<b>\mathtolerance</b>
[=] integer	[=] integer
: integer	: integer
<b>\mathgluemode</b>	<b>t \maxdeadcycles</b>
[=] integer	[=] integer
: integer	: integer
<b>\mathgroupingmode</b>	<b>t \month</b>
[=] integer	[=] integer
: integer	: integer
<b>\mathinlinepenaltyfactor</b>	<b>t \newlinechar</b>
[=] integer	[=] integer
: integer	: integer
<b>\mathleftclass</b>	<b>\normalizelinemode</b>
[=] integer	[=] integer
: integer	: integer
<b>\mathlimitsmode</b>	<b>\normalizelparmode</b>
[=] integer	[=] integer
: integer	: integer

<b>l \nospaces</b> [=] integer : integer	<b>l \prerelpenalty</b> [=] integer : integer
<b>l \orphanpenalty</b> [=] integer : integer	<b>l \preshortinlinepenalty</b> [=] integer : integer
<b>l \outputbox</b> [=] integer : integer	<b>t \pretolerance</b> [=] integer : integer
<b>t \outputpenalty</b> [=] integer : integer	<b>l \protrudechars</b> [=] integer : integer
<b>l \overloadmode</b> [=] integer : integer	<b>t \relpenalty</b> [=] integer : integer
<b>l \parametermode</b> [=] integer : integer	<b>t \righthyphenmin</b> [=] integer : integer
<b>l \pardirection</b> [=] integer : integer	<b>e \savingshyphcodes</b> [=] integer : integer
<b>t \pausing</b> [=] integer : integer	<b>e \savingsdiscards</b> [=] integer : integer
<b>t \postdisplaypenalty</b> [=] integer : integer	<b>l \setfontid</b> [=] integer : integer
<b>l \postinlinepenalty</b> [=] integer : integer	<b>t \setlanguage</b> [=] integer : integer
<b>l \postshortinlinepenalty</b> [=] integer : integer	<b>l \shapingpenaltiesmode</b> [=] integer : integer
<b>l \prebinoppenalty</b> [=] integer : integer	<b>l \shapingpenalty</b> [=] integer : integer
<b>e \predisplaydirection</b> [=] integer : integer	<b>l \shortinlineorphanpenalty</b> [=] integer : integer
<b>l \predisplaygapfactor</b> [=] integer : integer	<b>t \showboxbreadth</b> [=] integer : integer
<b>t \predisplaypenalty</b> [=] integer : integer	<b>t \showboxdepth</b> [=] integer : integer
<b>l \preinlinepenalty</b> [=] integer : integer	<b>t \shownodedetails</b> [=] integer : integer

<b>l \singlelinepenalty</b>	<b>e \tracingifs</b>
[=] integer	[=] integer
: integer	: integer
<b>l \spacefactor</b>	<b>l \tracinginserts</b>
[=] integer	[=] integer
: integer	: integer
<b>l \spacefactorshrinklimit</b>	<b>l \tracinglevels</b>
[=] integer	[=] integer
: integer	: integer
<b>l \spacefactorstretchlimit</b>	<b>l \tracinglists</b>
[=] integer	[=] integer
: integer	: integer
<b>l \supmarkmode</b>	<b>t \tracinglostchars</b>
[=] integer	[=] integer
: integer	: integer
<b>l \textrdirection</b>	<b>t \tracingmacros</b>
[=] integer	[=] integer
: integer	: integer
<b>t \time</b>	<b>l \tracingmarks</b>
[=] integer	[=] integer
: integer	: integer
<b>t \tolerance</b>	<b>l \tracingmath</b>
[=] integer	[=] integer
: integer	: integer
<b>l \tracingadjusts</b>	<b>e \tracingnesting</b>
[=] integer	[=] integer
: integer	: integer
<b>l \tracingalignments</b>	<b>l \tracingnodes</b>
[=] integer	[=] integer
: integer	: integer
<b>e \tracingassigns</b>	<b>t \tracingonline</b>
[=] integer	[=] integer
: integer	: integer
<b>t \tracingcommands</b>	<b>t \tracingoutput</b>
[=] integer	[=] integer
: integer	: integer
<b>l \tracingexpressions</b>	<b>t \tracingpages</b>
[=] integer	[=] integer
: integer	: integer
<b>l \tracingfullboxes</b>	<b>t \tracingparagraphs</b>
[=] integer	[=] integer
: integer	: integer
<b>e \tracinggroups</b>	<b>l \tracingpasses</b>
[=] integer	[=] integer
: integer	: integer
<b>l \tracinghyphenation</b>	<b>l \tracingpenalties</b>
[=] integer	[=] integer
: integer	: integer

**t \tracingrestores**  
     [=] *integer*  
     : *integer*

**t \tracingstats**  
     [=] *integer*  
     : *integer*

**t \uchyph**  
     [=] *integer*  
     : *integer*

**l \variablefam**  
     [=] *integer*  
     : *integer*

**t \vbadness**  
     [=] *integer*  
     : *integer*

**t \widowpenalty**  
     [=] *integer*  
     : *integer*

**t \year**  
     [=] *integer*  
     : *integer*

## 48 internalmuglue

**t \medmuskip**  
     [=] *muglue*  
     : *muglue*

**l \pettymuskip**  
     [=] *muglue*  
     : *muglue*

**t \thickmuskip**  
     [=] *muglue*  
     : *muglue*

**t \thinmuskip**  
     [=] *muglue*  
     : *muglue*

**l \tinymuskip**  
     [=] *muglue*  
     : *muglue*

## 49 internaltoks

**t \errhelp**  
     [=] *toks*  
     : *toks*

**l \everybeforepar**  
     [=] *toks*  
     : *toks*

**t \everycr**  
     [=] *toks*  
     : *toks*

**t \everydisplay**  
     [=] *toks*  
     : *toks*

**e \everyeof**  
     [=] *toks*  
     : *toks*

**t \everyhbox**  
     [=] *toks*  
     : *toks*

**t \everyjob**  
     [=] *toks*  
     : *toks*

**t \everymath**  
     [=] *toks*  
     : *toks*

**l \everymathatom**  
     [=] *toks*  
     : *toks*

**t \everypar**  
     [=] *toks*  
     : *toks*

**l \everytab**  
     [=] *toks*  
     : *toks*

**t \everyvbox**  
     [=] *toks*  
     : *toks*

**t \output**  
     [=] *toks*  
     : *toks*

## 50 italiccorrection

**t \**  
**l \explicititaliccorrection**  
     TODO

**l \forcedleftcorrection**  
     TODO

**l \forcedrightcorrection**  
     TODO

## 51 kern

**t \hkern**  
     *dimension*

**t \kern**  
*dimension*

**t \vkern**  
*dimension*

## 52 leader

**t \cleaders**  
*( box | rule | glyph ) glue*

**l \gleaders**  
 see \cleaders

**t \leaders**  
 see \cleaders

**l \uleaders**  
*[ callback integer ] ( box | rule | glyph ) glue*

**t \xleaders**  
 see \cleaders

## 53 legacy

**t \shipout**  
*{ tokens }*

## 54 let

**l \futuredef**  
*\cs \cs*  
**t \futurelet**  
*\cs [=] \cs*  
**l \glet**  
*\cs*  
**l \gletcsname**  
*tokens\endcsname*

**l \glettonothing**  
*\cs*

**t \let**  
*\cs*

**l \letcharcode**  
*\cs*

**l \letcsname**  
*tokens\endcsname*

**l \letfrozen**  
*\cs*

**l \letprotected**  
*\cs*

**l \lettolastnamedcs**  
*\cs*

**l \lettonothing**  
*\cs*

**l \swapcsvalues**  
*\cs \cs*

**l \unletfrozen**  
*\cs*

**l \unletprotected**  
*\cs*

## 55 localbox

**l \localleftbox**  
*box*

**l \localmiddlebox**  
*box*

**l \localrightbox**  
*box*

## 56 luafunctioncall

**l \luabytecodecall**  
*integer*

**l \luafunctioncall**  
*integer*

## 57 makebox

**t \box**  
*( index | box )*

**t \copy**  
 see \box

**l \dbox**  
*[ target integer ] [ to dimension ]*  
*[ adapt ] [ attr integer integer ]*  
*[ anchor integer ] [ axis integer ]*  
*[ shift dimension ] [ spread dimension ]*  
*[ source integer ] [ direction integer ]*  
*[ delay ] [ orientation integer ]*  
*[ xoffset dimension ] [ xmove*  
*dimension ] [ yoffset dimension ]*  
*[ ymove dimension ] [ reverse ] [ retain ]*  
*[ container ] [ class integer ] { tokens }*

**l \dpack**  
 see \dbox

**l \dsplit**  
*[ attr ] [ to ] [ upto ] { tokens }*

**t \hbox**  
 see \dbox

**l** `\hpack`  
     see `\dbox`  
**l** `\insertbox`  
     *integer*  
**l** `\insertcopy`  
     *integer*  
**t** `\lastbox`  
**l** `\localleftboxbox`  
**l** `\localmiddleboxbox`  
**l** `\localrightboxbox`  
**l** `\tpack`  
     see `\dbox`  
**l** `\tsplit`  
     see `\dsplit`  
**t** `\vbox`  
     see `\dbox`  
**l** `\vpack`  
     see `\dbox`  
**t** `\vsplit`  
     see `\dsplit`  
**t** `\vtop`  
     see `\dbox`

## 58 mark

**l** `\clearmarks`  
     *integer*  
**l** `\flushmarks`  
**t** `\mark`  
     { *tokens* }  
**e** `\marks`  
     *integer* { *tokens* }

## 59 mathaccent

**l** `\Umathaccent`  
     [ *attr integer integer* ] [ *center* ]  
     [ *class integer* ] [ *exact* ] [ *source*  
     *integer* ] [ *stretch* ] [ *shrink* ]  
     [ *fraction integer* ] [ *fixed* ]  
     [ *keepbase* ] [ *nooverflow* ] [ *base* ]  
     ( *both* [ *fixed* ] *character* [ *fixed* ]  
     *character* | *bottom* [ *fixed* ]  
     *character* | *top* [ *fixed* ]  
     *character* | *overlay*  
     *character* | *character* )  
**t** `\mathaccent`  
     { *tokens* }

## 60 mathcharnumber

**l** `\Umathchar`  
     *integer*  
**t** `\mathchar`  
     *integer*  
**l** `\mathclass`  
     *integer*  
**l** `\mathdictionary`  
     *integer mathchar*

## 61 mathchoice

**t** `\mathchoice`  
     { *tokens* } { *tokens* } { *tokens* } { *tokens* }  
**l** `\mathdiscretionary`  
     [ *class integer* ] { *tokens* } { *tokens* }  
     { *tokens* }  
**l** `\mathstack`  
     { *tokens* }

## 62 mathcomponent

**l** `\mathatom`  
     [ *attr integer integer* ] [ *all integer* ]  
     [ *leftclass integer* ] [ *limits* ]  
     [ *rightclass integer* ] [ *class integer* ]  
     [ *unpack* ] [ *unroll* ] [ *single* ] [ *source*  
     *integer* ] [ *textfont* ] [ *mathfont* ]  
     [ *options integer* ] [ *nolimits* ]  
     [ *nooverflow* ] [ *void* ] [ *phantom* ]  
     [ *continuation* ] [ *integer* ]  
**t** `\mathbin`  
     { *tokens* }  
**t** `\mathclose`  
     { *tokens* }  
**t** `\mathinner`  
     { *tokens* }  
**t** `\mathop`  
     { *tokens* }  
**t** `\mathopen`  
     { *tokens* }  
**t** `\mathord`  
     { *tokens* }  
**t** `\mathpunct`  
     { *tokens* }  
**t** `\mathrel`  
     { *tokens* }

**t** `\overline`  
     {tokens}

**t** `\underline`  
     {tokens}

## 63 mathfence

**l** `\Uleft`  
     [auto] [attr integer integer] [axis]  
     [bottom dimension] [depth dimension]  
     [factor integer] [height dimension]  
     [noaxis] [nocheck] [nolimits]  
     [nooverflow] [leftclass integer]  
     [limits] [exact] [void] [phantom]  
     [class integer] [rightclass integer]  
     [scale] [source integer] [top]  
     delimiter

**l** `\Umiddle`  
     see `\Uleft`

**l** `\Uoperator`  
     see `\Uleft`

**l** `\Uright`  
     see `\Uleft`

**l** `\Uvextensible`  
     see `\Uleft`

**t** `\left`  
     see `\Uleft`

**t** `\middle`  
     see `\Uleft`

**t** `\right`  
     see `\Uleft`

## 64 mathfraction

**l** `\Uabove`  
     dimension [attr integer integer]  
     [class integer] [center] [exact]  
     [proportional] [noaxis]  
     [nooverflow] [style mathstyle]  
     [source integer] [hfactor integer]  
     [vfactor integer] [font] [thickness  
     dimension]

**l** `\Uabovewithdelims`  
     delimiter delimiter dimension [attr  
     integer integer] [class integer]  
     [center] [exact] [proportional]  
     [noaxis] [nooverflow] [style  
     mathstyle] [source integer] [hfactor  
     integer] [vfactor integer] [font]

[thickness dimension]

**l** `\Uatop`  
     see `\Uabove`

**l** `\Uatopwithdelims`  
     see `\Uabovewithdelims`

**l** `\Uover`  
     [attr integer integer] [class  
     integer] [center] [exact]  
     [proportional] [noaxis]  
     [nooverflow] [style mathstyle]  
     [source integer] [hfactor integer]  
     [vfactor integer] [font] [thickness  
     dimension]

**l** `\Uoverwithdelims`  
     delimiter delimiter [attr integer  
     integer] [class integer] [center]  
     [exact] [proportional] [noaxis]  
     [nooverflow] [style mathstyle]  
     [source integer] [hfactor integer]  
     [vfactor integer] [font] [thickness  
     dimension]

**l** `\Uskewed`  
     delimiter [attr integer integer]  
     [class integer] [center] [exact]  
     [proportional] [noaxis]  
     [nooverflow] [style mathstyle]  
     [source integer] [hfactor integer]  
     [vfactor integer] [font] [thickness  
     dimension]

**l** `\Uskewedwithdelims`  
     delimiter delimiter delimiter [attr  
     integer integer] [class integer]  
     [center] [exact] [proportional]  
     [noaxis] [nooverflow] [style  
     mathstyle] [source integer] [hfactor  
     integer] [vfactor integer] [font]  
     [thickness dimension]

**l** `\Ustretched`  
     see `\Uskewed`

**l** `\Ustretchedwithdelims`  
     see `\Uskewedwithdelims`

**t** `\above`  
     dimension

**t** `\abovewithdelims`  
     delimiter delimiter dimension

**t** `\atop`  
     dimension

**t** `\atopwithdelims`  
     delimiter delimiter dimension

**t** `\over`  
**t** `\overwithdelims`  
     delimiter delimiter

## 65 mathmodifier

**l** `\Umathadapttleft`  
**l** `\Umathadapttright`  
**l** `\Umathlimits`  
**l** `\Umathnoaxis`  
**l** `\Umathnolimits`  
**l** `\Umathopenupdepth`  
     dimension  
**l** `\Umathopenupheight`  
     dimension  
**l** `\Umathphantom`  
**l** `\Umathsource`  
     [nucleus] integer  
**l** `\Umathuseaxis`  
**l** `\Umathvoid`  
**t** `\displaylimits`  
**t** `\limits`  
**t** `\nolimits`

## 66 mathparameter

**l** `\Umathaccentbasedepth`  
     mathstyle [=] dimension  
     > mathstyle: dimension  
**l** `\Umathaccentbaseheight`  
     mathstyle [=] dimension  
     > mathstyle: dimension  
**l** `\Umathaccentbottomovershoot`  
     mathstyle [=] dimension  
     > mathstyle: dimension  
**l** `\Umathaccentbottomshiftdown`  
     mathstyle [=] dimension  
     > mathstyle: dimension  
**l** `\Umathaccentextendmargin`  
     mathstyle [=] dimension  
     > mathstyle: dimension  
**l** `\Umathaccentsuperscriptdrop`  
     mathstyle [=] dimension  
     > mathstyle: dimension  
**l** `\Umathaccentsuperscriptpercent`  
     mathstyle [=] integer  
     > mathstyle: integer  
**l** `\Umathaccenttopovershoot`  
     mathstyle [=] dimension

    > mathstyle: dimension  
**l** `\Umathaccenttopshiftup`  
     mathstyle [=] dimension  
     > mathstyle: dimension  
**l** `\Umathaccentvariant`  
     [=] mathstyle  
     : mathstyle  
**l** `\Umathaxis`  
     mathstyle [=] dimension  
     > mathstyle: dimension  
**l** `\Umathbottomaccentvariant`  
     [=] mathstyle  
     : mathstyle  
**l** `\Umathconnectoroverlapmin`  
     mathstyle [=] dimension  
     > mathstyle: dimension  
**l** `\Umathdegreevariant`  
     [=] mathstyle  
     : mathstyle  
**l** `\Umathdelimiterextendmargin`  
     mathstyle [=] dimension  
     > mathstyle: dimension  
**l** `\Umathdelimiterovervariant`  
     [=] mathstyle  
     : mathstyle  
**l** `\Umathdelimiterpercent`  
     mathstyle [=] integer  
     > mathstyle: integer  
**l** `\Umathdelimitershortfall`  
     mathstyle [=] dimension  
     > mathstyle: dimension  
**l** `\Umathdelimiterundervariant`  
     [=] mathstyle  
     : mathstyle  
**l** `\Umathdenominatorvariant`  
     [=] mathstyle  
     : mathstyle  
**l** `\Umathexheight`  
     mathstyle [=] dimension  
     > mathstyle: dimension  
**l** `\Umathextrasubpreshift`  
     mathstyle [=] dimension  
     > mathstyle: dimension  
**l** `\Umathextrasubprespace`  
     mathstyle [=] dimension  
     > mathstyle: dimension  
**l** `\Umathextrasubshift`  
     mathstyle [=] dimension  
     > mathstyle: dimension



```

\Umathextrasubspace
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathextrasuppresshift
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathextrasupprespace
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathextrasupshift
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathextrasupspace
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathflattenedaccentbasedepth
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathflattenedaccentbaseheight
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathflattenedaccentbottomshiftdown
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathflattenedaccenttopshiftup
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathfractiondelsize
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathfractiondenomdown
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathfractiondenomvgap
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathfractionnumup
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathfractionnumvgap
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathfractionrule
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathfractionvariant
  [=] mathstyle
  : mathstyle

\Umathhextensiblevariant
  [=] mathstyle
  : mathstyle
\Umathlimitabovebgap
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathlimitabovekern
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathlimitabovevgap
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathlimitbelowbgap
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathlimitbelowkern
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathlimitbelowvgap
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathnolimitsubfactor
  mathstyle [=] integer
  > mathstyle: integer
\Umathnolimitsupfactor
  mathstyle [=] integer
  > mathstyle: integer
\Umathnumeratorvariant
  [=] mathstyle
  : mathstyle
\Umathoperatorsize
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathoverbarkern
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathoverbarrule
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathoverbarvgap
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathoverdelimiterbgap
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathoverdelimitervariant
  [=] mathstyle
  : mathstyle

```

```

\Umathoverdelimitervgap
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathoverlayaccentvariant
  [=] mathstyle
  : mathstyle
\Umathoverlinevariant
  [=] mathstyle
  : mathstyle
\Umathpresubshiftdistance
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathpresupshiftdistance
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathprimeraise
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathprimeraisecomposed
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathprimeshiftdrop
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathprimeshiftup
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathprimespaceafter
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathprimevariant
  [=] mathstyle
  : mathstyle
\Umathprimewidth
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathquad
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathradicaldegreeafter
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathradicaldegreebefore
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathradicaldegreeraise
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathradicalextensibleafter
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathradicalextensiblebefore
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathradicalkern
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathradicalrule
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathradicalvariant
  [=] mathstyle
  : mathstyle
\Umathradicalvgap
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathruleddepth
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathruleheight
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathskeweddelimitertolerance
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathskewedfractionhgap
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathskewedfractionvgap
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathspaceafterscript
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathspacebeforescript
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathstackdenomdown
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathstacknumup
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathstackvariant
  [=] mathstyle
  : mathstyle

```

```

\Umathstackvgap
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathsubscriptsnap
  TODO
\Umathsubscriptvariant
  [=] mathstyle
  : mathstyle
\Umathsubshiftdistance
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathsubshiftdown
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathsubshiftdrop
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathsubsupshiftdown
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathsubsupvgap
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathsubtopmax
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathsupbottommin
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathsupscriptsnap
  TODO
\Umathsupscriptvariant
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathsupshiftdistance
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathsupshiftdrop
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathsupshiftup
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathsupsubbottommax
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathtopaccentvariant
  [=] mathstyle
  : mathstyle
\Umathunderbarkern
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathunderbarrule
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathunderbarvgap
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathunderdelimiterbgap
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathunderdelimitervariant
  [=] mathstyle
  : mathstyle
\Umathunderdelimitervgap
  mathstyle [=] dimension
  > mathstyle: dimension
\Umathunderlinevariant
  [=] mathstyle
  : mathstyle
\Umathvextensiblevariant
  [=] mathstyle
  : mathstyle
\Umathxscale
  mathstyle [=] integer
  > mathstyle: integer
\Umathyscale
  mathstyle [=] integer
  > mathstyle: integer
\copymathatomrule
  integer integer
\copymathparent
  integer integer
\copymathspacing
  integer integer
\letmathatomrule
  integer integer integer integer
  integer
\letmathparent
  integer integer
\letmathspacing
  see \letmathatomrule
\resetmathspacing
\setdefaultmathcodes
\setmathatomrule
  integer integer mathstyle integer
  integer

```

**l** `\setmathdisplaypostpenalty`  
     *integer* [=] *integer*

**l** `\setmathdisplayprepenalty`  
     *integer* [=] *integer*

**l** `\setmathignore`  
     mathparameter *integer*

**l** `\setmathoptions`  
     *integer* [=] *integer*

**l** `\setmathpostpenalty`  
     *integer* [=] *integer*

**l** `\setmathprepenalty`  
     *integer* [=] *integer*

**l** `\setmathspacing`  
     *integer integer mathstyle glue*

## 67 mathradical

**l** `\Udelimited`  
     [ attr *integer integer* ] [ bottom ]  
     [ exact ] [ top ] [ style *mathstyle* ]  
     [ source *integer* ] [ stretch ] [ shrink ]  
     [ width *dimension* ] [ height *dimension* ]  
     [ depth *dimension* ] [ left ] [ middle ]  
     [ right ] [ nooverflow ] delimiter  
     delimiter [ delimiter ] [ delimiter ]  
     ( mathatom | { *tokens* } )

**l** `\Udelimiterover`  
     [ attr *integer integer* ] [ bottom ]  
     [ exact ] [ top ] [ style *mathstyle* ]  
     [ source *integer* ] [ stretch ] [ shrink ]  
     [ width *dimension* ] [ height *dimension* ]  
     [ depth *dimension* ] [ left ] [ middle ]  
     [ right ] [ nooverflow ] delimiter  
     [ delimiter ] [ delimiter ]  
     ( mathatom | { *tokens* } )

**l** `\Udelimiterunder`  
     see `\Udelimiterover`

**l** `\Uhexensible`  
     see `\Udelimiterover`

**l** `\Uoverdelimiter`  
     see `\Udelimiterover`

**l** `\Uradical`  
     see `\Udelimiterover`

**l** `\Uroot`  
     [ attr *integer integer* ] [ bottom ]  
     [ exact ] [ top ] [ style *mathstyle* ]  
     [ source *integer* ] [ stretch ] [ shrink ]  
     [ width *dimension* ] [ height *dimension* ]  
     [ depth *dimension* ] [ left ] [ middle ]

[ right ] [ nooverflow ] delimiter  
     [ delimiter ] [ delimiter ]  
     ( mathatom | { *tokens* } )  
     ( mathatom | { *tokens* } )

**l** `\Urooted`  
     [ attr *integer integer* ] [ bottom ]  
     [ exact ] [ top ] [ style *mathstyle* ]  
     [ source *integer* ] [ stretch ] [ shrink ]  
     [ width *dimension* ] [ height *dimension* ]  
     [ depth *dimension* ] [ left ] [ middle ]  
     [ right ] [ nooverflow ] delimiter  
     delimiter [ delimiter ] [ delimiter ]  
     ( mathatom | { *tokens* } )  
     ( mathatom | { *tokens* } )

**l** `\Uunderdelimiter`  
     see `\Udelimiterover`

**t** `\radical`  
     see `\Uroot`

## 68 mathscript

**l** `\noatomruling`

**t** `\nonscript`

**l** `\nosubprescript`

**l** `\nosubscript`

**l** `\nosuperprescript`

**l** `\nosuperscript`

**l** `\primescript`  
     ( mathatom | { *tokens* } )

**l** `\shiftedsubprescript`  
     see `\primescript`

**l** `\shiftedsubscript`  
     see `\primescript`

**l** `\shiftedsuperprescript`  
     see `\primescript`

**l** `\shiftedsuperscript`  
     see `\primescript`

**l** `\subprescript`  
     see `\primescript`

**l** `\subscript`  
     see `\primescript`

**l** `\superprescript`  
     see `\primescript`

**l** `\superscript`  
     see `\primescript`

## 69 mathshiftcs

**l** `\Ustartdisplaymath`

`\Ustartmath`  
`\Ustartmathmode`  
`\Ustopdisplaymath`  
`\Ustopmath`  
`\Ustopmathmode`

## 70 mathstyle

`\allcrampedstyles`  
`\alldisplaystyles`  
`\allmainstyles`  
`\allmathstyles`  
`\allscriptscriptstyles`  
`\allscriptstyles`  
`\allsplitstyles`  
`\alltextstyles`  
`\alluncrampedstyles`  
`\allunsplitstyles`  
`\crampeddisplaystyle`  
`\crampedscriptscriptstyle`  
`\crampedscriptstyle`  
`\crampedtextstyle`  
`\displaystyle`  
`\givenmathstyle`  
       
`\scaledmathstyle`  
      $integer$   
      $> mathstyle : integer$   
`\scriptscriptstyle`  
`\scriptstyle`  
`\textstyle`

## 71 message

`\errmessage`  
      $\{tokens\}$   
`\message`  
      $\{tokens\}$

## 72 mkern

`\mkern`  
      $dimension$

## 73 mskip

`\mathatomskip`  
      $muglue$

`\mskip`  
      $muglue$

## 74 noexpand

`\noexpand`  
      $token$

## 75 pageproperty

`\deadcycles`  
      $[=] integer$   
      $: integer$   
`\insertdepth`  
      $integer [=] dimension$   
      $> integer : dimension$   
`\insertdistance`  
      $integer [=] dimension$   
      $> integer : dimension$   
`\insertheight`  
      $integer [=] dimension$   
      $> integer : dimension$   
`\insertheights`  
      $[=] dimension$   
      $: dimension$   
`\insertlimit`  
      $integer [=] dimension$   
      $> integer : dimension$   
`\insertmaxdepth`  
      $integer [=] dimension$   
      $> integer : dimension$   
`\insertmultiplier`  
      $integer [=] integer$   
      $> integer : integer$   
`\insertpenalties`  
      $[=] integer$   
      $: integer$   
`\insertpenalty`  
      $integer [=] integer$   
      $> integer : integer$   
`\insertstorage`  
      $integer [=] integer$   
      $> integer : integer$   
`\insertstoring`  
      $[=] integer$   
      $: integer$   
`\insertwidth`  
      $integer [=] dimension$   
      $> integer : dimension$

**l** **\pagedepth**  
     [=] *dimension*  
     : *dimension*  
**l** **\pageexcess**  
     [=] *dimension*  
     : *dimension*  
**t** **\pagefillllstretch**  
     [=] *dimension*  
     : *dimension*  
**t** **\pagefillstretch**  
     [=] *dimension*  
     : *dimension*  
**t** **\pagefilstretch**  
     [=] *dimension*  
     : *dimension*  
**l** **\pagefistretch**  
     [=] *dimension*  
     : *dimension*  
**t** **\pagegoal**  
     [=] *dimension*  
     : *dimension*  
**l** **\pagelastdepth**  
     [=] *dimension*  
     : *dimension*  
**l** **\pagelastfillllstretch**  
     [=] *dimension*  
     : *dimension*  
**l** **\pagelastfillstretch**  
     [=] *dimension*  
     : *dimension*  
**l** **\pagelastfilstretch**  
     [=] *dimension*  
     : *dimension*  
**l** **\pagelastfistretch**  
     TODO  
**l** **\pagelastheight**  
     [=] *dimension*  
     : *dimension*  
**l** **\pagelastshrink**  
     [=] *dimension*  
     : *dimension*  
**l** **\pagelaststretch**  
     [=] *dimension*  
     : *dimension*  
**t** **\pageshrink**  
     [=] *dimension*  
     : *dimension*  
**t** **\pagestretch**  
     [=] *dimension*

    : *dimension*  
**t** **\pagetotal**  
     [=] *dimension*  
     : *dimension*  
**l** **\pagevsize**  
     [=] *dimension*  
     : *dimension*

## 76 parameter

**l** **\alignmark**  
**l** **\parametermark**

## 77 penalty

**l** **\hpenalty**  
     *integer*  
**t** **\penalty**  
     *integer*  
**l** **\vpenalty**  
     *integer*

## 78 prefix

**l** **\aliased**  
**l** **\constant**  
**l** **\constrained**  
**l** **\deferred**  
**l** **\enforced**  
**l** **\frozen**  
**t** **\global**  
**l** **\immediate**  
**l** **\immutable**  
**l** **\inherited**  
**l** **\instance**  
**t** **\long**  
**l** **\mutable**  
**l** **\noaligned**  
**t** **\outer**  
**l** **\overloaded**  
**l** **\permanent**  
**e** **\protected**  
**l** **\retained**  
**l** **\semiprotected**  
**l** **\tolerant**  
**l** **\untraced**

**79 register**

```

l \attribute
    ( index | box ) [=] integer
    > ( index | box ) : integer
t \count
    see \attribute
t \dimen
    ( index | box ) [=] dimension
    > ( index | box ) : dimension
l \float
    ( index | box ) [=] float
    > ( index | box ) : float
t \muskip
    ( index | box ) [=] muglue
    > ( index | box ) : muglue
t \skip
    ( index | box ) [=] glue
    > ( index | box ) : glue
t \toks
    ( index | box ) [=] { tokens }
    > ( index | box ) : { tokens }

```

**80 relax**

```

l \norelax
t \relax

```

**81 removeitem**

```

t \unboundary
t \unkern
t \unpenalty
t \unskip

```

**82 setbox**

```

t \setbox
    ( index | box ) [=]

```

**83 setfont**

```

t \nullfont

```

**84 shorthanddef**

```

l \Umathchardef
    \cs integer
l \Umathdictdef
    \cs integer integer
l \attributedef
    \cs integer
t \chardef
    \cs integer
t \countdef
    \cs integer
t \dimendef
    \cs integer
l \dimensiondef
    \cs integer
l \floatdef
    \cs integer
l \fontspecdef
    \cs ( font | integer )
l \gluespecdef
    \cs integer
l \integerdef
    \cs integer
l \luadef
    \cs integer
t \mathchardef
    \cs integer
l \mugluespecdef
    \cs integer
t \muskipdef
    \cs integer
l \parameterdef
    \cs integer
l \positdef
    \cs integer
t \skipdef
    \cs integer
t \toksdef
    \cs integer

```

**85 someitem**

```

t \badness
    [=] integer
    : integer
e \currentgrouplevel
    [=] integer
    : integer

```

```

e \currentgrouptype
    [=] integer
    : integer
e \currentifbranch
    [=] integer
    : integer
e \currentiflevel
    [=] integer
    : integer
e \currentiftype
    [=] integer
    : integer
l \currentloopiterator
    [=] integer
    : integer
l \currentloopnesting
    [=] integer
    : integer
e \currentstacksize
    [=] integer
    : integer
e \dimexpr
    tokens\relax [=] dimension
    > tokens\relax : dimension
l \dimexpression
    tokens\relax [=] dimension
    > tokens\relax : dimension
l \floatexpr
    tokens\relax [=] float
    > tokens\relax : float
l \fontcharba
    integer [=] dimension
    > integer : dimension
e \fontchardp
    integer [=] dimension
    > integer : dimension
e \fontcharht
    integer [=] dimension
    > integer : dimension
e \fontcharic
    integer [=] dimension
    > integer : dimension
l \fontcharta
    integer [=] dimension
    > integer : dimension
e \fontcharwd
    integer [=] dimension
    > integer : dimension

l \fontid
    ( font | integer ) [=] integer
    > ( font | integer ) : integer
l \fontmathcontrol
    see \fontid
l \fontspecid
    see \fontid
l \fontspecifiedsize
    see \fontid
l \fontspecscale
    see \fontid
l \fontspecslant
    see \fontid
l \fontspecweight
    see \fontid
l \fontspecxscale
    see \fontid
l \fontspecyscale
    see \fontid
l \fonttextcontrol
    see \fontid
e \glueexpr
    tokens\relax [=] glue
    > tokens\relax : glue
e \glueshrink
    glue [=] dimension
    > glue : dimension
e \glueshrinkorder
    glue [=] dimension
    > glue : dimension
e \gluestretch
    glue [=] integer
    > glue : integer
e \gluestretchorder
    glue [=] integer
    > glue : integer
e \gluetomu
    glue [=] glue
    > glue : glue
l \glyphxscaled
    [=] integer
    : integer
l \glyphyscaled
    [=] integer
    : integer
l \indexofcharacter
    integer [=] integer
    > integer : integer

```



```

l \indexofregister
    integer [=] integer
    > integer : integer
t \inputlineno
    [=] integer
    : integer
l \insertprogress
    integer [=] dimension
    > integer : dimension
l \lastarguments
    [=] integer
    : integer
l \lastatomclass
    [=] integer
    : integer
l \lastboundary
    [=] integer
    : integer
l \lastchkdimension
    [=] dimension
    : dimension
l \lastchknumber
    [=] integer
    : integer
t \lastkern
    [=] dimension
    : dimension
l \lastleftclass
    [=] integer
    : integer
l \lastloopiterator
    [=] integer
    : integer
l \lastnodesubtype
    [=] integer
    : integer
e \lastnodetype
    [=] integer
    : integer
l \lastpageextra
    [=] dimension
    : dimension
l \lastparcontext
    [=] integer
    : integer
l \lastpartrigger
    TODO
t \lastpenalty
    [=] integer

    : integer
l \lastrightclass
    [=] integer
    : integer
t \lastskip
    [=] glue
    : glue
l \leftmarginkern
    [=] dimension
    : dimension
l \luatexrevision
    [=] {tokens}
    : {tokens}
l \luatexversion
    [=] {tokens}
    : {tokens}
l \mathatomglue
    [=] glue
    : glue
l \mathcharclass
    integer [=] integer
    > integer : integer
l \mathcharfam
    integer [=] integer
    > integer : integer
l \mathcharslot
    integer [=] integer
    > integer : integer
l \mathmainstyle
    [=] integer
    : integer
l \mathparentstyle
    TODO
l \mathscale
    [=] integer
    : integer
l \mathstackstyle
    [=] integer
    : integer
l \mathstyle
    [=] integer
    : integer
l \mathstylefontid
    [=] integer
    : integer
e \muexpr
    tokens\relax [=] muglue
    > tokens\relax : muglue

```

- e** `\mutoglu`
    - `muglu [=] glue`
    - `> muglu : glue`
  - l** `\nestedloopiterator`
    - `[=] integer`
    - `: integer`
  - l** `\numericsscale`
    - `(integer | float) [=] integer`
    - `> (integer | float) : integer`
  - l** `\numericsscaled`
    - see `\numericsscale`
  - e** `\numexpr`
    - `tokens\relax [=] integer`
    - `> tokens\relax : integer`
  - l** `\numexpression`
    - `tokens\relax [=] integer`
    - `> tokens\relax : integer`
  - l** `\overshoot`
    - `[=] dimension`
    - `: dimension`
  - l** `\parametercount`
    - `[=] integer`
    - `: integer`
  - l** `\parameterindex`
    - `[=] integer`
    - `: integer`
  - e** `\parshapedimen`
    - `integer [=] dimension`
    - `> integer : dimension`
  - e** `\parshapeindent`
    - `integer [=] dimension`
    - `> integer : dimension`
  - e** `\parshaplength`
    - `[=] dimension`
    - `: dimension`
  - l** `\parshapewidth`
    - TODO
  - l** `\previousloopiterator`
    - `[=] integer`
    - `: integer`
  - l** `\rightmarginkern`
    - `[=] dimension`
    - `: dimension`
  - l** `\scaledemwidth`
    - `(font | integer) [=] dimension`
    - `> (font | integer) : dimension`
  - l** `\scaledexheight`
    - see `\scaledemwidth`
  - l** `\scaledextraspac`
    - see `\scaledemwidth`
  - l** `\scaledfontcharba`
    - `integer [=] dimension`
    - `> integer : dimension`
  - l** `\scaledfontchardp`
    - `integer [=] dimension`
    - `> integer : dimension`
  - l** `\scaledfontcharht`
    - `integer [=] dimension`
    - `> integer : dimension`
  - l** `\scaledfontcharic`
    - `integer [=] dimension`
    - `> integer : dimension`
  - l** `\scaledfontcharta`
    - `integer [=] dimension`
    - `> integer : dimension`
  - l** `\scaledfontcharwd`
    - `integer [=] dimension`
    - `> integer : dimension`
  - l** `\scaledinterwordshrink`
    - see `\scaledemwidth`
  - l** `\scaledinterwordspace`
    - see `\scaledemwidth`
  - l** `\scaledinterwordstretch`
    - see `\scaledemwidth`
  - l** `\scaledmathaxis`
    - `mathstyle [=] dimension`
    - `> mathstyle : dimension`
  - l** `\scaledmathemwidth`
    - `mathstyle [=] dimension`
    - `> mathstyle : dimension`
  - l** `\scaledmathexheight`
    - `mathstyle [=] dimension`
    - `> mathstyle : dimension`
  - l** `\scaledslantperpoint`
    - see `\scaledemwidth`
- ## 86 specification
- e** `\clubpenalties`
    - `[options] integer n * (integer)`
    - `: integer`
  - e** `\displaywidowpenalties`
    - see `\clubpenalties`
  - e** `\interlinepenalties`
    - see `\clubpenalties`
  - l** `\mathbackwardpenalties`
    - see `\clubpenalties`

**l \mathforwardpenalties**

see \clubpenalties

**l \orphanpenalties**

see \clubpenalties

**l \parpasses**

[ options ] n \* ( [ adjdemerits *integer* ]  
 [ adjustspacing *integer* ]  
 [ adjustspacingstep *integer* ]  
 [ adjustspacingshrink *integer* ]  
 [ adjustspacingstretch *integer* ]  
 [ badness *integer* ] [ classes *integer* ]  
 [ callback *integer* ]  
 [ doubleadjdemerits *integer* ]  
 [ doublehyphendemerits *integer* ]  
 [ emergencystretch *dimension* ]  
 [ extrahyphenpenalty *integer* ]  
 [ finalhyphendemerits *integer* ]  
 [ identifier *integer* ]  
 [ ifadjustspacing *integer* ] [ looseness  
*integer* ] [ linebreakcriterium  
*integer* ] [ linebreakoptional *integer* ]  
 [ linepenalty *integer* ] [ next ]  
 [ orphanpenalty *integer* ] [ quit ]  
 [ skip ] [ threshold *dimension* ]  
 [ tolerance *integer* ] )

: *integer***t \parshape**

[ options ] *integer* n \* ( *dimension*  
*dimension* )

: *integer***e \widowpenalties**

see \clubpenalties

**87 the****e \detokenize**{ *tokens* }**l \expandeddetokenize**{ *tokens* }**l \protecteddetokenize**{ *tokens* }**l \protectedexpandeddetokenize**{ *tokens* }**t \the***dimension***l \thewithoutunit***quantity***e \unexpanded**{ *tokens* }**88 unhbox****t \unhbox***integer***t \unhcopy***integer***l \unhpack***integer***89 unvbox****l \insertunbox***integer***l \insertuncopy***integer***e \pagediscards****e \splitdiscards****t \unvbox***integer***t \unvcopy***integer***l \unvpack***integer***90 vadjust****t \vadjust**

[ pre ] [ post ] [ baseline ] [ before ]  
 [ index *integer* ] [ after ] [ attr  
*integer integer* ] [ depth  
 ( after | before | check | last ) ]  
 { *tokens* }

**91 valign****t \valign**

[ attr *integer integer* ] [ callback  
*integer* ] [ discard ] [ noskips ]  
 [ reverse ] [ to *dimension* ] [ spread  
*dimension* ] { *tokens* }

**92 vcenter****t \vcenter**

[ target *integer* ] [ to *dimension* ]  
 [ adapt ] [ attr *integer integer* ]  
 [ anchor *integer* ] [ axis *integer* ]

[ *shift dimension* ] [ *spread dimension* ]  
 [ *source integer* ] [ *direction integer* ]  
 [ *delay* ] [ *orientation integer* ]  
 [ *xoffset dimension* ] [ *xmove dimension* ] [ *yoffset dimension* ]  
 [ *ymove dimension* ] [ *reverse* ] [ *retain* ]  
 [ *container* ] [ *class integer* ] { *tokens* }

### 93 vmove

**t \lower**  
     *dimension box*  
**t \raise**  
     *dimension box*

### 94 vrule

**l \novrule**  
     [ *attr integer* [=] *integer* ] [ *width dimension* ] [ *height dimension* ] [ *depth dimension* ] [ *left dimension* ] [ *right dimension* ] [ *top dimension* ] [ *bottom dimension* ] [ *xoffset dimension* ] [ *yoffset dimension* ] [ *font integer* ] [ *fam integer* ] [ *char integer* ]  
**l \srule**  
     see \novrule  
**l \virtualvrule**  
     [ *attr integer* [=] *integer* ] [ *width dimension* ] [ *height dimension* ] [ *depth dimension* ] [ *left dimension* ] [ *right dimension* ] [ *top dimension* ] [ *bottom dimension* ] [ *xoffset dimension* ] [ *yoffset dimension* ]  
**t \vrule**  
     see \novrule

### 95 vskip

**t \vfil**  
**t \vfill**  
**t \vfilneg**  
**t \vskip**  
     *dimension* [ *plus* ( *dimension* | *fi*[ *n\*l* ] ) ) ] [ *minus* ( *dimension* | *fi*[ *n\*l* ] ) ) ]  
**t \vss**

### 96 xray

**t \show**  
     *token*  
**t \showbox**  
     ( *index* | *box* )  
**l \showcodestack**  
     TODO  
**e \showgroups**  
**e \showifs**  
**t \showlists**  
**l \showstack**  
**t \showthe**  
     *quantity*  
**e \showtokens**  
     { *tokens* }

## Rationale

Some words about the why and how it came. One of the early adopters of ConT<sub>E</sub>Xt was Taco Hoekwater and we spent numerous trips to T<sub>E</sub>X meetings all over the globe. He was also the only one I knew who had read the T<sub>E</sub>X sources. Because ConT<sub>E</sub>Xt has always been on the edge of what is possible and at that time we both used it for rather advanced rendering, we also ran into the limitations. I'm not talking of T<sub>E</sub>X features here. Naturally old school T<sub>E</sub>X is not really geared for dealing with images of all kind, colors in all kind of color spaces, highly interactive documents, input methods like xml, etc. The nice thing is that it offers some escapes, like specials and writes and later execution of programs that opened up lots of possibilities, so in practice there were no real limitations to what one could do. But coming up with a consistent and extensible (multi lingual) user interface was non trivial, because it had an impact in memory usage and performance. A lot could be done given some programming, as ConT<sub>E</sub>Xt MkII proves, but it was not always pretty under the hood. The move to LuaT<sub>E</sub>X and MkIV transferred some action to Lua, and because LuaT<sub>E</sub>X effectively was a ConT<sub>E</sub>Xt related project, we could easily keep them in sync.

Our traveling together, meeting several times per year, and eventually email and intense LuaT<sub>E</sub>X developments (lots of Skype sessions) for a couple of years, gave us enough opportunity to discuss all kind of nice features not present in the engine. The previous century we discussed lots of them, rejected some, stayed with others, and I admit that forgot about most of the arguments already. Some that we did was already explored in e<sub>e</sub>t<sub>e</sub>x, some of those ended up in LuaT<sub>E</sub>X, and eventually what we have in LuaMetaT<sub>E</sub>X can be seen as the result of years of programming in T<sub>E</sub>X, improving macros, getting more performance and efficiency out of existing ConT<sub>E</sub>Xt code and inspiration that we got out of the ConT<sub>E</sub>Xt community, a demanding lot, always willing to experiment with us.

Once I decided to work on LuaMetaT<sub>E</sub>X and bind its source to the ConT<sub>E</sub>Xt distribution so that we can be sure that it won't get messed up and might interfere with the ConT<sub>E</sub>Xt expectations, some more primitives saw their way into it. It is very easy to come up with all kind of bells and whistles but it is equally easy to hurt performance of an engine and what might go unnoticed in simple tests can really affect a macro package that depends on stability. So, what I did was mostly looking at the ConT<sub>E</sub>Xt code and wondering how to make some of the low level macros look more natural, also because I know that there are users who look into these sources. We spend a lot of time making them look consistent and nice and the nicer the better. Getting a better performance was seldom an argument because much is already as fast as can be so there is not that much to gain, but less clutter in tracing was an argument for some new primitives. Also, the fact that we soon might need to fall back on our phones to use T<sub>E</sub>X a smaller memory footprint and less byte shuffling also was a consideration. The LuaMetaT<sub>E</sub>X memory footprint is somewhat smaller than the LuaT<sub>E</sub>X footprint. By binding LuaMetaT<sub>E</sub>X to ConT<sub>E</sub>Xt we can also guarantee that the combinations works as expected.

I'm aware of the fact that ConT<sub>E</sub>Xt is in a somewhat unique position. First of all it has always been kind of cutting edge so its users are willing to experiment. There are users who immediately update and run tests, so bugs can and will be fixed fast. Already for a long time the community has a convenient infrastructure for updating and the build farm for generating binaries (also for other engines) is running smoothly.

Then there is the ConT<sub>E</sub>Xt user interface that is quite consistent and permits extensions with staying backward compatible. Sometimes users run into old manuals or examples and then complain that ConT<sub>E</sub>Xt is not compatible but that then involves obsolete technology: we no longer need font and input encodings and font definitions are different for OpenType fonts. We always had an abstract backend model, but nowadays pdf is kind of dominant and drives a lot of expectations. So, some of the MkII commands are gone and MkIV has some more. Also, as MetaPost evolved that department

in ConT<sub>E</sub>Xt also evolved. Think of it like cars: soon all are electric so one cannot expect a hole to poor in some fluid but gets a (often incompatible) plug instead. And buttons became touch panels. There is no need to use much force to steer or brake. Navigation is different, as are many controls. And do we need to steer ourselves a decade from now?

So, just look at T<sub>E</sub>X and ConT<sub>E</sub>Xt in the same way. A system from the nineties in the previous century differs from one three decades later. Demands differ, input differs, resources change, editing and processing moves on, and so on. Manuals, although still being written are seldom read from cover to cover because online searching replaced them. And who buys books about programming? So LuaMetaT<sub>E</sub>X, while still being T<sub>E</sub>X also moves on, as do the way we do our low level coding. This makes sense because the original T<sub>E</sub>X ecosystem was not made with a huge and complex macro package in mind, that just happened. An author was supposed to make a style for each document. An often used argument for using another macro package over ConT<sub>E</sub>Xt was that the later evolved and other macro packages would work the same forever and not change from the perspective of the user. In retrospect those arguments were somewhat strange because the world, computers, users etc. do change. Standards come and go, as do software politics and preferences. In many aspects the T<sub>E</sub>X community is not different from other large software projects, operating system wars, library devotees, programming language addicts, paradigm shifts. But, don't worry, if you don't like LuaMetaT<sub>E</sub>X and its new primitives, just forget about them. The other engines will be there forever and are a safe bet, although LuaT<sub>E</sub>X already stirred up the pot I guess. But keep in mind that new features in the latest greatest ConT<sub>E</sub>Xt version will more and more rely on LuaMetaT<sub>E</sub>X being used; after all that is where it's made for. And this manual might help understand its users why, where and how the low level code differs between MkII, MkIV and LMTX.

Can we expect more new primitives than the ones introduced here? Given the amount of time I spent on experimenting and considering what made sense and what not, the answer probably is “no”, or at least “not that much”. As in the past no user ever requested the kind of primitives that were added, I don't expect users to come up with requests in the future either. Of course, those more closely related to ConT<sub>E</sub>Xt development look at it from the other end. Because it's there where the low level action really is, demands might still evolve.

Basically there are two areas where the engine can evolve: the programming part and the rendering. In this manual we focus on the programming and writing the manual sort of influences how details get filled in. Rendering is more complex because there heuristics and usage plays a more dominant role. Good examples are the math, par and page builder. They were extended and features were added over time but improved rendering came later. Not all extensions are critical, some are there (and got added) in order to write more readable code but there is only so much one can do in that area. Occasionally a feature pops up that is a side effect of a challenge. No matter what gets added it might not affect complexity too much and definitely not impact performance significantly!

Hans Hagen  
Hasselt NL

**To be checked primitives (new)**

## To be checked primitives (math)

Uabove	Umathfractiondenomvgap
Udelcode	Umathfractionnumup
Udelimited	Umathfractionnumvgap
Udelimiter	Umathfractionrule
Udelimiterover	Umathfractionvariant
Udelimiterunder	Umathhextensiblevariant
Uhextensible	Umathlimitabovebgap
Uleft	Umathlimitabovekern
Umathaccentbasedepth	Umathlimitabovevgap
Umathaccentbaseheight	Umathlimitbelowbgap
Umathaccentbottomovershoot	Umathlimitbelowkern
Umathaccentbottomshiftdown	Umathlimitbelowvgap
Umathaccentextendmargin	Umathlimits
Umathaccentsuperscriptdrop	Umathnoaxis
Umathaccentsuperscriptpercent	Umathnolimits
Umathaccenttopovershoot	Umathnumeratorvariant
Umathaccenttopshiftup	Umathopenupdepth
Umathaccentvariant	Umathopenupheight
Umathadapttoleft	Umathoperatorsize
Umathadapttoright	Umathoverdelimiterbgap
Umathaxis	Umathoverdelimitervariant
Umathbottomaccentvariant	Umathoverdelimitervgap
Umathcode	Umathoverlayaccentvariant
Umathconnectoroverlapmin	Umathphantom
Umathdegreevariant	Umathpresubshiftdistance
Umathdelimiterextendmargin	Umathpresupshiftdistance
Umathdelimiterovervariant	Umathprimeraise
Umathdelimiterpercent	Umathprimeraisecomposed
Umathdelimitershortfall	Umathprimeshiftup
Umathdelimiterundervariant	Umathprimespaceafter
Umathdenominatorvariant	Umathprimevariant
Umathdictdef	Umathprimewidth
Umathexheight	Umathquad
Umathextrasubpreshift	Umathradicaldegreeafter
Umathextrasubprespace	Umathradicaldegreebefore
Umathextrasubshift	Umathradicaldegreeraise
Umathextrasubspace	Umathradicalextensibleafter
Umathextrasuppreshift	Umathradicalextensiblebefore
Umathextrasupprespace	Umathradicalkern
Umathextrasupshift	Umathradicalrule
Umathextrasupspace	Umathradicalvariant
Umathflattenedaccentbasedepth	Umathradicalvgap
Umathflattenedaccentbaseheight	Umathruledepth
Umathflattenedaccentbottomshiftdown	Umathruleheight
Umathflattenedaccenttopshiftup	Umathskeweddelimitertolerance
Umathfractiondelsize	Umathskewedfractionhgap
Umathfractiondenomdown	



Umathskewedfractionvgap	Umathunderdelimitervariant
Umathsource	Umathunderdelimitervgap
Umathspaceafterscript	Umathuseaxis
Umathspacebeforescript	Umathvextensiblevariant
Umathstackdenomdown	Umathvoid
Umathstacknumup	Umathxscale
Umathstackvariant	Umathyscale
Umathstackvgap	Umiddle
Umathsubscriptsnap	Uoperator
Umathsubscriptvariant	Uoverdelimiter
Umathsubshiftdistance	Uroot
Umathsubshiftdown	Urooted
Umathsubshiftdrop	Uskewed
Umathsubsupshiftdown	Uskewedwithdelims
Umathsubsupvgap	Ustartdisplaymath
Umathsubtopmax	Ustartmath
Umathsupbottommin	Ustartmathmode
Umathsupscriptsnap	Ustopdisplaymath
Umathsupscriptvariant	Ustopmath
Umathsupshiftdistance	Ustopmathmode
Umathsupshiftdrop	Ustretched
Umathsupshiftdown	Ustretchedwithdelims
Umathsupsubbottommax	Uunderdelimiter
Umathtopaccentvariant	Uvextensible
Umathunderdelimiterbgap	

Many primitives starting with `Umath` are math parameters that are discussed elsewhere, if at all.

**To be checked primitives (old)**

## Indexed primitives

-  
 /  
 <space>  
 Uabovewithdelims  
 Uatop  
 Uatopwithdelims  
 Umathaccent  
 Umathchar  
 Umathchardef  
 Umathnolimitsubfactor  
 Umathnolimitsupfactor  
 Umathoverbarkern  
 Umathoverbarrule  
 Umathoverbarvgap  
 Umathoverlinevariant  
 Umathunderbarkern  
 Umathunderbarrule  
 Umathunderbarvgap  
 Umathunderlinevariant  
 Uover  
 Uoverwithdelims  
 Uradical  
 Uright  
  
 above  
 abovedisplayshortskip  
 abovedisplayskip  
 abovewithdelims  
 accent  
 additionalpageskip  
 adjdemerits  
 adjustspacing  
 adjustspacingshrink  
 adjustspacingstep  
 adjustspacingstretch  
 advance  
 advanceby  
 afterassigned  
 afterassignment  
 aftergroup  
 aftergrouped  
 aliased  
 aligncontent  
 alignmark  
 alignmentcellsource  
 alignmentwrapsource  
 aligntab  
 allcrampedstyles  
 alldisplaystyles  
 allmainstyles  
 allmathstyles  
 allscriptscriptstyles  
 allscriptstyles  
 allsplitstyles  
 alltextstyles  
 alluncrampedstyles  
 allunsplitstyles  
 amcode  
 associateunit  
 atendoffile  
 atendoffiled  
 atendofgroup  
 atendofgrouped  
 atop  
 atopwithdelims  
 attribute  
 attributedef  
 automaticdiscretionary  
 automatichyphenpenalty  
 automigrationmode  
 autoparagraphmode  
 badness  
 baselineskip  
 batchmode  
 begincsname  
 begingroup  
 beginlocalcontrol  
 beginmathgroup  
 beginsimplegroup  
 belowdisplayshortskip  
 belowdisplayskip  
 binoppenalty  
 botmark  
 botmarks  
 boundary  
 box  
 boxadapt  
 boxanchor  
 boxanchors  
 boxattribute  
 boxdirection  
 boxfinalize  
 boxfreeze

boxgeometry	currentgrouplevel
boxlimit	currentgrouptype
boxlimitate	currentifbranch
boxlimitmode	currentiflevel
boxmaxdepth	currentifttype
boxorientation	currentloopiterator
boxrepack	currentloopnesting
boxshift	currentmarks
boxshrink	currentstacksize
boxsource	day
boxstretch	dbox
boxtarget	deadcycles
boxtotal	def
boxvadjust	defaultthyphenchar
boxxmove	defaultskewchar
boxxoffset	defcsname
boxymove	deferred
boxyoffset	delcode
brokenpenalty	delimiter
catcode	delimiterfactor
catcodetable	delimitershortfall
cdef	detokened
cdefcsname	detokenize
cf	detokenized
cfcode	dimen
char	dimendef
chardef	dimensiondef
cleaders	dimexpr
clearmarks	dimexpression
clubpenalties	directlua
clubpenalty	discretionary
constant	discretionaryoptions
constrained	displayindent
copy	displaylimits
copymathatomrule	displayskipmode
copymathparent	displaystyle
copymathspacing	displaywidowpenalties
correctionskip	displaywidowpenalty
count	displaywidth
countdef	divide
cr	divideby
crampeddisplaystyle	doubleadjdemerits
crampedscriptscriptstyle	doublehyphendemerits
crampedscriptstyle	dp
crampedtextstyle	dpack
crcr	dsplit
csactive	dump
csname	edef
csstring	edefcsame

edefcsname	expanded
edivide	expandedafter
edivideby	expandeddetokenize
efcode	expandedendless
else	expandedloop
emergencyextrastretch	expandedrepeat
emergencyleftskip	expandparameter
emergencyrightskip	expandtoken
emergencystretch	expandtoks
end	explicitdiscretionary
endcsname	explicithyphenpenalty
endgroup	explicititaliccorrection
endinput	explicitSPACE
endlinechar	fam
endlocalcontrol	fi
endmathgroup	finalhyphendemerits
endsimplegroup	firstmark
enforced	firstmarks
eofinput	firstvalidlanguage
eqno	float
errhelp	floatdef
errmessage	floatexpr
errorcontextlines	floatingpenalty
errorstopmode	flushmarks
escapechar	font
etoks	fontcharba
etoksapp	fontchardp
etokspre	fontcharht
eufactor	fontcharic
everybeforepar	fontcharta
everycr	fontcharwd
everydisplay	fontdimen
everyeof	fontid
everyhbox	fontmathcontrol
everyjob	fontname
everymath	fontspecdef
everymathatom	fontspecid
everypar	fontspecifiedname
everytab	fontspecifiedsize
everyvbox	fontspecscale
exceptionpenalty	fontspecslant
exhyphenchar	fontspecweight
exhyphenpenalty	fontspecxscale
expand	fontspecyscale
expandactive	fonttextcontrol
expandafter	forcedleftcorrection
expandafterpars	forcedrightcorrection
expandafterspaces	formatname
expandcstoken	frozen

futurecsname	hfill
futuredef	hfilneg
futureexpand	hfuzz
futureexpandis	hj
futureexpandisap	hjcode
futurelet	hkern
gdef	hmcode
gdefcsname	holdinginserts
givenmathstyle	holdingmigrations
gleaders	hpack
glet	hpenalty
gletcsname	hrule
glettonothing	hsize
global	hskip
globaldefs	hss
glue	ht
glueexpr	hyphenation
glueshrink	hyphenationmin
glueshrinkorder	hyphenationmode
gluespecdef	hyphenchar
gluestretch	hyphenpenalty
gluestretchorder	if
gluetomu	ifabsdim
glyph	ifabsfloat
glyphdatafield	ifabsnum
glyphoptions	ifarguments
glyphscale	ifboolean
glyphscriptfield	ifcase
glyphscriptscale	ifcat
glyphscriptscriptscale	ifchkdim
glyphslant	ifchkdimension
glyphstatefield	ifchknum
glyphtextscale	ifchknumber
glyphweight	ifcmpdim
glyphxoffset	ifcmpnum
glyphxscale	ifcondition
glyphxscaled	ifcramped
glyphyoffset	ifcsname
glyphyscale	ifcstok
glyphyscaled	ifdefined
gtoksapp	ifdim
gtokspre	ifdimexpression
halign	ifdimval
hangafter	ifempty
hangindent	iffalse
hbadness	ifflags
hbox	iffloat
hccode	iffontchar
hfil	ifhaschar

ifhastok	initialpageskip
ifhastoks	initialtopskip
ifhasxtoks	input
ifhbox	inputlineno
ifhmode	insert
ifinalignment	insertbox
ifincsname	insertcopy
ifinner	insertdepth
ifinsert	insertdistance
ifintervaldim	insertheight
ifintervalfloat	insertheights
ifintervalnum	insertlimit
iflastnamedcs	insertmaxdepth
ifmathparameter	insertmode
ifmathstyle	insertmultiplier
ifmmode	insertpenalties
ifnum	insertpenalty
ifnumexpression	insertprogress
ifnumval	insertstorage
ifodd	insertstoring
ifparameter	insertunbox
ifparameters	insertuncopy
ifrelax	insertwidth
iftok	instance
iftrue	integerdef
ifvbox	interactionmode
ifvmode	interlinepenalties
ifvoid	interlinepenalty
ifx	jobname
ifzerodim	kern
ifzerofloat	language
ifzeronum	lastarguments
ignorearguments	lastatomclass
ignoredepthcriterion	lastboundary
ignorenestedupto	lastbox
ignorepars	lastchkdimension
ignorereset	lastchknumber
ignorespaces	lastkern
ignoreupto	lastleftclass
immediate	lastlinefit
immediateassigned	lastloopiterator
immediateassignment	lastnamedcs
immutable	lastnodesubtype
indent	lastnodetype
indentskip	lastpageextra
indexofcharacter	lastparcontext
indexofregister	lastpartrigger
inherited	lastpenalty
initcatcodetable	lastrightclass

lastskip	luacopyinputnodes
lccode	luadef
leaders	luaescapestring
left	luafunction
lefthangskip	luafunctioncall
lefthyphenmin	luatexbanner
leftmarginkern	luatexrevision
leftskip	luatexversion
leqno	mark
let	marks
latcharcode	mathaccent
letcsname	mathatom
letfrozen	mathatomglue
letmathatomrule	mathatomskip
letmathparent	mathbackwardpenalties
letmathspacing	mathbeginclass
letprotected	mathbin
lettolastnamedcs	mathboundary
lettonothing	mathchar
limits	mathcharclass
linebreakcriterion	mathchardef
linebreakoptional	mathcharfam
linebreakpasses	mathcharslot
linedirection	mathcheckfencesmode
linepenalty	mathchoice
lineskip	mathclass
lineskiplimit	mathclose
localbrokenpenalty	mathcode
localcontrol	mathdictgroup
localcontrolled	mathdictionary
localcontrolledendless	mathdictproperties
localcontrolledloop	mathdirection
localcontrolledrepeat	mathdiscretionary
localinterlinepenalty	mathdisplaymode
lcalleftbox	mathdisplaypenaltyfactor
lcalleftboxbox	mathdisplayskipmode
lcalmiddlebox	mathdoublescriptmode
lcalmiddleboxbox	mathendclass
lcalpretolerance	matheqnogapstep
lcalrightbox	mathfontcontrol
lcalrightboxbox	mathforwardpenalties
lcaltolerance	mathgluemode
long	mathgroupingmode
looseness	mathinlinepenaltyfactor
lower	mathinner
lowercase	mathleftclass
lpcode	mathlimitsmode
luabytecode	mathmainstyle
luabytecodecall	mathnolimitsmode



mathop	newlinechar
mathopen	noalign
mathord	noaligned
mathparentstyle	noatomruling
mathpenaltiesmode	noboundary
mathpretolerance	noexpand
mathpunct	nohrule
mathrel	noindent
mathrightclass	nolimits
mathrulesfam	nonscript
mathrulesmode	nonstopmode
mathscale	norelax
mathscriptsmode	normalizelinemode
mathslackmode	normalizeparmode
mathspacingmode	normalunexpanded
mathstack	nospaces
mathstackstyle	nosubprescript
mathstyle	nosubscript
mathstylefontid	nosuperprescript
mathsurround	nosuperscript
mathsurroundmode	novrule
mathsurroundskip	nulldelimiterspace
maththreshold	nullfont
mathtolerance	number
maxdeadcycles	numericsscale
maxdepth	numericsscaled
meaning	numexpr
meaningasis	numexpression
meaningful	omit
meaningfull	open
meaningles	optionalboundary
meaningless	or
medmuskip	orelse
message	orphanpenalties
middle	orphanpenalty
mkern	orunless
month	outer
moveleft	output
moveright	outputbox
mskip	outputpenalty
muexpr	over
mugluespecdef	overfullrule
multiply	overline
multiplyby	overloaded
muskip	overloadmode
muskipdef	overshoot
mutable	overwithdelims
mutoglu	pageboundary
nestedloopiterator	pagedepth

pagediscards	posthyphenchar
pageexcess	postinlinepenalty
pageextragoal	postshortinlinepenalty
pagefillllstretch	prebinoppenalty
pagefillstretch	predisplaydirection
pagefilstretch	predisplaygapfactor
pagefistretch	predisplaypenalty
pagegoal	preplaysize
pagelastdepth	preexhyphenchar
pagelastfillllstretch	prehyphenchar
pagelastfillstretch	preinlinepenalty
pagelastfilstretch	prerelpenalty
pagelastfistretch	preshortinlinepenalty
pagelastheight	presuperscript
pagelastshrink	pretolerance
pagelaststretch	prevdepth
pageshrink	prevgraf
pagestretch	previousloopiterator
pagetotal	primescript
pagevsize	protected
par	protecteddetokenize
parametercount	protectedexpandeddetokenize
parameterdef	protrudechars
parameterindex	protrusionboundary
parametermark	pxdimen
parametermode	quitloop
parattribute	quitloopnow
pardirection	quitvmode
parfillleftskip	radical
parfillrightskip	raise
parfillskip	rdivide
parindent	rdivideby
parinitleftskip	realign
parinitrightskip	relax
parpasses	relpenalty
parshape	resetmathspacing
parshapedimen	restorecatcodes
parshapeindent	restorecatcodetable
parshapelength	retained
parshapewidth	retokenized
parskip	right
patterns	righthangskip
pausing	rightthyphenmin
penalty	rightmarginkern
permanent	rightskip
pettymuskip	romannumeral
positdef	rpcode
postdisplaypenalty	savecatcodetable
postexhyphenchar	savinghyphcodes

savingvdiscards	shiftedsuperscript
scaledemwidth	shipout
scaledexheight	shortinlinemaththreshold
scaledextraspaces	shortinlineorphanpenalty
scaledfontcharba	show
scaledfontcharbd	showbox
scaledfontcharht	showboxbreadth
scaledfontcharic	showboxdepth
scaledfontcharta	showcodestack
scaledfontcharwd	showgroups
scaledfontdimen	showifs
scaledinterwordshrink	showlists
scaledinterwordspace	shownodedetails
scaledinterwordstretch	showstack
scaledmathaxis	showthe
scaledmathemwidth	showtokens
scaledmathexheight	singlelinepenalty
scaledmathstyle	skewchar
scaledslantperpoint	skip
scantextokens	skipdef
scantokens	snapshotpar
scriptfont	spacefactor
scriptscriptfont	spacefactormode
scriptscriptstyle	spacefactorshrinklimit
scriptspace	spacefactorstretchlimit
scriptstyle	spaceskip
scrollmode	span
semiexpand	special
semiexpanded	splitbotmark
semiprotected	splitbotmarks
semprotected	splitdiscards
setbox	splitfirstmark
setdefaultmathcodes	splitfirstmarks
setfontid	splitmaxdepth
setlanguage	splittopskip
setmathatomrule	srule
setmathdisplaypostpenalty	string
setmathdisplayprepenalty	subprescript
setmathignore	subscript
setmathoptions	superprescript
setmathpostpenalty	superscript
setmathprepenalty	supmarkmode
setmathspacing	swapcsvalues
sfcode	tabsize
shapingpenaltiesmode	tabskip
shapingpenalty	tabskips
shiftedsuperscript	texdirection
shiftedsuperscript	textfont
shiftedsuperscript	textstyle

the	tracingrestores
thewithoutunit	tracingstats
thickmuskip	tsplit
thinmuskip	uccode
time	uchyph
tinymuskip	uleaders
tocharacter	unboundary
todimension	undent
tohexadecimal	underline
tointeger	unexpanded
tokenized	unexpandedendless
toks	unexpandedloop
toksapp	unexpandedrepeat
toksdef	unhbox
tokspre	unhcopy
tolerance	unhpack
tolerant	unkern
tomathstyle	unless
topmark	unletfrozen
topmarks	unletprotected
topskip	unpenalty
toscaled	unskip
tosparsedimension	untraced
tosparsescaled	unvbox
tpack	unvcopy
tracingadjusts	unvpack
tracingalignments	uppercase
tracingassigns	vadjust
tracingcommands	valign
tracingexpressions	variablefam
tracingfullboxes	vbadness
tracinggroups	vbox
tracinghyphenation	vcenter
tracingifs	vfil
tracinginserts	vfill
tracinglevels	vfilneg
tracinglists	vfuzz
tracinglostchars	virtualhrule
tracingmacros	virtualvrule
tracingmarks	vkern
tracingmath	vpack
tracingnesting	vpenalty
tracingnodes	vrule
tracingonline	vsize
tracingoutput	vskip
tracingpages	vsplit
tracingparagraphs	vss
tracingpasses	vtop
tracingpenalties	wd

widowpenalties  
widowpenalty  
wordboundary  
wrapuppar  
write  
xdef  
xdefcsname

xleaders  
xspaceskip  
xtoks  
xtoksapp  
xtokspre  
year