

# Inhalt

<b>Erklärung der Help-Funktion</b>	<b>12</b>
<b>EAGLE konfigurieren</b>	<b>13</b>
<b>Schnelle Einführung</b>	<b>15</b>
<b>Control Panel und Editor-Fenster</b>	<b>16</b>
<b>Eingabe von Parametern und Werten</b>	<b>17</b>
<b>Schaltplan entwerfen</b>	<b>18</b>
<b>Schaltung überprüfen</b>	<b>19</b>
<b>Platine aus Schaltplan erzeugen</b>	<b>20</b>
<b>Layout überprüfen</b>	<b>21</b>
<b>Bauelement in Bibliothek definieren</b>	<b>22</b>
<b>Control Panel</b>	<b>23</b>
<b>Kontext Menüs</b>	<b>25</b>
<b>Verzeichnisse</b>	<b>27</b>
<b>Backup</b>	<b>28</b>
<b>User Interface</b>	<b>29</b>
<b>Tastatur und Maus</b>	<b>30</b>
<b>Benachbarte Objekte selektieren</b>	<b>31</b>
<b>Editor-Fenster</b>	<b>32</b>
<b>Bibliotheks-Editor</b>	<b>33</b>
<b>Bibliotheks-Editier-Modus</b>	<b>34</b>
<b>Layout-Editor</b>	<b>35</b>
<b>Schaltplan-Editor</b>	<b>36</b>
<b>Text-Editor</b>	<b>37</b>
<b>Editor-Befehle</b>	<b>38</b>
<b>Befehlseingabe</b>	<b>40</b>
<b>ADD</b>	<b>42</b>
<b>ARC</b>	<b>45</b>
<b>ASSIGN</b>	<b>46</b>
<b>AUTO</b>	<b>48</b>

---

<b>BOARD</b>	<b>49</b>
<b>BUS</b>	<b>50</b>
<b>CHANGE</b>	<b>51</b>
<b>CIRCLE</b>	<b>52</b>
<b>CLASS</b>	<b>53</b>
<b>CLOSE</b>	<b>54</b>
<b>CONNECT</b>	<b>55</b>
<b>COPY</b>	<b>56</b>
<b>CUT</b>	<b>57</b>
<b>DELETE</b>	<b>58</b>
<b>DESCRIPTION</b>	<b>60</b>
<b>DISPLAY</b>	<b>61</b>
<b>DRC</b>	<b>63</b>
<b>EDIT</b>	<b>64</b>
<b>ERC</b>	<b>65</b>
<b>ERRORS</b>	<b>67</b>
<b>EXPORT</b>	<b>68</b>
<b>GATESWAP</b>	<b>70</b>
<b>GRID</b>	<b>71</b>
<b>GROUP</b>	<b>73</b>
<b>HELP</b>	<b>74</b>
<b>HOLE</b>	<b>75</b>
<b>INFO</b>	<b>76</b>
<b>INVOKE</b>	<b>77</b>
<b>JUNCTION</b>	<b>78</b>
<b>LABEL</b>	<b>79</b>
<b>LAYER</b>	<b>80</b>
<b>MARK</b>	<b>82</b>
<b>MENU</b>	<b>83</b>
<b>MIRROR</b>	<b>85</b>

---

<b>MOVE</b>	<b>86</b>
<b>NAME</b>	<b>87</b>
<b>NET</b>	<b>88</b>
<b>OPEN</b>	<b>89</b>
<b>OPTIMIZE</b>	<b>90</b>
<b>PACKAGE</b>	<b>91</b>
<b>PAD</b>	<b>92</b>
<b>PASTE</b>	<b>94</b>
<b>PIN</b>	<b>95</b>
<b>PINSWAP</b>	<b>98</b>
<b>POLYGON</b>	<b>99</b>
<b>PREFIX</b>	<b>101</b>
<b>PRINT</b>	<b>102</b>
<b>QUIT</b>	<b>103</b>
<b>RATSNEST</b>	<b>104</b>
<b>RECT</b>	<b>105</b>
<b>REDO</b>	<b>106</b>
<b>REMOVE</b>	<b>107</b>
<b>RENAME</b>	<b>108</b>
<b>REPLACE</b>	<b>109</b>
<b>RIPUP</b>	<b>110</b>
<b>ROTATE</b>	<b>111</b>
<b>ROUTE</b>	<b>112</b>
<b>RUN</b>	<b>113</b>
<b>SCRIPT</b>	<b>114</b>
<b>SET</b>	<b>115</b>
<b>SHOW</b>	<b>117</b>
<b>SIGNAL</b>	<b>118</b>
<b>SMASH</b>	<b>119</b>
<b>SMD</b>	<b>120</b>

<b>SPLIT</b>	<b>121</b>
<b>TECHNOLOGY</b>	<b>122</b>
<b>TEXT</b>	<b>123</b>
<b>UNDO</b>	<b>125</b>
<b>UPDATE</b>	<b>126</b>
<b>USE</b>	<b>127</b>
<b>VALUE</b>	<b>128</b>
<b>VIA</b>	<b>129</b>
<b>WINDOW</b>	<b>130</b>
<b>WIRE</b>	<b>131</b>
<b>WRITE</b>	<b>133</b>
<b>Ausgabedaten erzeugen</b>	<b>134</b>
<b>Drucken</b>	<b>135</b>
<b>Drucken einer Zeichnung</b>	<b>136</b>
<b>Drucken eines Textes</b>	<b>137</b>
<b>Seiteneinrichtung</b>	<b>138</b>
<b>CAM-Prozessor</b>	<b>139</b>
<b>CAM-Prozessor-Hauptmenü</b>	<b>140</b>
<b>CAM-Prozessor-Job</b>	<b>141</b>
<b>Ausgabetreiber (Output Device)</b>	<b>142</b>
<b>Device-Parameter</b>	<b>143</b>
<b>Blenden-Konfigurationsdatei</b>	<b>144</b>
<b>Blenden-Emulation</b>	<b>145</b>
<b>Blenden-Toleranzen</b>	<b>146</b>
<b>Bohrer-Konfigurationsdatei</b>	<b>147</b>
<b>Bohrer-Toleranzen</b>	<b>148</b>
<b>Offset</b>	<b>149</b>
<b>Bedruckbarer Bereich</b>	<b>150</b>
<b>Stiftdaten</b>	<b>151</b>
<b>Eigenen Device-Treiber definieren</b>	<b>152</b>

---

<b>Ausgabedaten</b>	<b>153</b>
<b>Flag-Options</b>	<b>154</b>
<b>Layer und Farben</b>	<b>155</b>
<b>Konturdaten</b>	<b>156</b>
<b>Autorouter</b>	<b>157</b>
<b>Überprüfen des Designs</b>	<b>158</b>
<b>Design Rules</b>	<b>159</b>
<b>User Language</b>	<b>161</b>
<b>Schreiben eines ULP</b>	<b>162</b>
<b>ULP ausführen</b>	<b>163</b>
<b>Syntax</b>	<b>164</b>
<b>Whitespace</b>	<b>165</b>
<b>Kommentare</b>	<b>166</b>
<b>Direktiven</b>	<b>167</b>
<b>#INCLUDE</b>	<b>168</b>
<b>#usage</b>	<b>169</b>
<b>Schlüsselwörter (Keywords)</b>	<b>170</b>
<b>Identifizier</b>	<b>171</b>
<b>Konstanten</b>	<b>172</b>
<b>Character-Konstanten</b>	<b>173</b>
<b>Integer-Konstanten</b>	<b>174</b>
<b>Real-Konstanten</b>	<b>175</b>
<b>String-Konstanten</b>	<b>176</b>
<b>Escape-Sequenzen</b>	<b>177</b>
<b>Punctuator-Zeichen</b>	<b>178</b>
<b>Eckige Klammern</b>	<b>179</b>
<b>Runde Klammern</b>	<b>180</b>
<b>Geschweifte Klammern</b>	<b>181</b>
<b>Komma</b>	<b>182</b>
<b>Semicolon</b>	<b>183</b>

<b>Doppelpunkt</b>	<b>184</b>
<b>Gleichheitszeichen</b>	<b>185</b>
<b>Datentypen</b>	<b>186</b>
<b>char</b>	<b>187</b>
<b>int</b>	<b>188</b>
<b>real</b>	<b>189</b>
<b>string</b>	<b>190</b>
<b>Typ-Umwandlung</b>	<b>191</b>
<b>Typecast</b>	<b>192</b>
<b>Objekt-Typen</b>	<b>193</b>
<b>UL_ARC</b>	<b>196</b>
<b>UL_AREA</b>	<b>197</b>
<b>UL_BOARD</b>	<b>198</b>
<b>UL_BUS</b>	<b>199</b>
<b>UL_CIRCLE</b>	<b>200</b>
<b>UL_CLASS</b>	<b>201</b>
<b>UL_CONTACT</b>	<b>202</b>
<b>UL_CONTACTREF</b>	<b>203</b>
<b>UL_DEVICE</b>	<b>204</b>
<b>UL_DEVICESET</b>	<b>205</b>
<b>UL_ELEMENT</b>	<b>206</b>
<b>UL_GATE</b>	<b>207</b>
<b>UL_GRID</b>	<b>208</b>
<b>UL_HOLE</b>	<b>209</b>
<b>UL_INSTANCE</b>	<b>210</b>
<b>UL_JUNCTION</b>	<b>211</b>
<b>UL_LAYER</b>	<b>212</b>
<b>UL_LIBRARY</b>	<b>213</b>
<b>UL_NET</b>	<b>214</b>
<b>UL_PACKAGE</b>	<b>215</b>

<b>UL_PAD</b>	<b>216</b>
<b>UL_PART</b>	<b>217</b>
<b>UL_PIN</b>	<b>218</b>
<b>UL_PINREF</b>	<b>220</b>
<b>UL_POLYGON</b>	<b>221</b>
<b>UL_RECTANGLE</b>	<b>223</b>
<b>UL_SCHEMATIC</b>	<b>224</b>
<b>UL_SEGMENT</b>	<b>225</b>
<b>UL_SHEET</b>	<b>226</b>
<b>UL_SIGNAL</b>	<b>227</b>
<b>UL_SMD</b>	<b>228</b>
<b>UL_SYMBOL</b>	<b>229</b>
<b>UL_TEXT</b>	<b>230</b>
<b>UL_VIA</b>	<b>231</b>
<b>UL_WIRE</b>	<b>232</b>
<b>Definitionen</b>	<b>233</b>
<b>Konstanten-Definitionen</b>	<b>234</b>
<b>Variablen-Definitionen</b>	<b>235</b>
<b>Funktions-Definitionen</b>	<b>236</b>
<b>Operatoren</b>	<b>237</b>
<b>Bitweise Operatoren</b>	<b>238</b>
<b>Logische Operatoren</b>	<b>239</b>
<b>Vergleichs-Operatoren</b>	<b>240</b>
<b>Evaluation-Operatoren</b>	<b>241</b>
<b>Arithmetische Operatoren</b>	<b>242</b>
<b>String-Operatoren</b>	<b>243</b>
<b>Ausdrücke</b>	<b>244</b>
<b>Arithmetischer Ausdruck</b>	<b>245</b>
<b>Zuweisungs-Ausdruck</b>	<b>246</b>
<b>String-Ausdruck</b>	<b>247</b>

---

<b>Komma-Ausdruck</b>	<b>248</b>
<b>Bedingter Ausdruck</b>	<b>249</b>
<b>Funktionsaufruf</b>	<b>250</b>
<b>Statements</b>	<b>251</b>
<b>Compound-Statement (Verbundanweisung)</b>	<b>252</b>
<b>Expression-Statement (Ausdrucksanweisung)</b>	<b>253</b>
<b>Control-Statements (Steueranweisungen)</b>	<b>254</b>
<b>break</b>	<b>255</b>
<b>continue</b>	<b>256</b>
<b>do...while</b>	<b>257</b>
<b>for</b>	<b>258</b>
<b>if...else</b>	<b>259</b>
<b>return</b>	<b>260</b>
<b>switch</b>	<b>261</b>
<b>while</b>	<b>262</b>
<b>Builtins</b>	<b>263</b>
<b>Builtin-Constants</b>	<b>264</b>
<b>Builtin Variablen</b>	<b>265</b>
<b>Builtin-Functions</b>	<b>266</b>
<b>Character-Funktionen</b>	<b>268</b>
<b>is...()</b>	<b>269</b>
<b>to...()</b>	<b>270</b>
<b>File-Handling-Funktionen</b>	<b>271</b>
<b>fileerror()</b>	<b>272</b>
<b>fileglob()</b>	<b>273</b>
<b>Filename-Funktionen</b>	<b>274</b>
<b>Filedata-Funktionen</b>	<b>275</b>
<b>File-Input-Funktionen</b>	<b>276</b>
<b>fileread()</b>	<b>277</b>
<b>Mathematische Funktionen</b>	<b>278</b>



<b>Absolutwert-, Maximum- und Minimum-Funktion</b>	<b>279</b>
<b>Rundungs-Funktionen</b>	<b>280</b>
<b>Trigonometrische Funktionen</b>	<b>281</b>
<b>Exponential-Funktionen</b>	<b>282</b>
<b>Sonstige Funktionen</b>	<b>283</b>
<b>exit()</b>	<b>284</b>
<b>lookup()</b>	<b>285</b>
<b>sort()</b>	<b>287</b>
<b>Einheiten-Konvertierung</b>	<b>288</b>
<b>Print-Funktionen</b>	<b>289</b>
<b>printf()</b>	<b>290</b>
<b>sprintf()</b>	<b>293</b>
<b>String-Funktionen</b>	<b>294</b>
<b>strchr()</b>	<b>295</b>
<b>strjoin()</b>	<b>296</b>
<b>strlen()</b>	<b>297</b>
<b>strlwr()</b>	<b>298</b>
<b>strrchr()</b>	<b>299</b>
<b>strrstr()</b>	<b>300</b>
<b>strsplit()</b>	<b>301</b>
<b>strstr()</b>	<b>302</b>
<b>strsub()</b>	<b>303</b>
<b>strtod()</b>	<b>304</b>
<b>strtol()</b>	<b>305</b>
<b>strupr()</b>	<b>306</b>
<b>Zeit-Funktionen</b>	<b>307</b>
<b>time()</b>	<b>308</b>
<b>Zeit-Konvertierungen</b>	<b>309</b>
<b>Builtin-Statements</b>	<b>310</b>
<b>board()</b>	<b>311</b>

<b>device()</b>	<b>312</b>
<b>library()</b>	<b>313</b>
<b>output()</b>	<b>314</b>
<b>package()</b>	<b>315</b>
<b>schematic()</b>	<b>316</b>
<b>sheet()</b>	<b>317</b>
<b>symbol()</b>	<b>318</b>
<b>Dialoge</b>	<b>319</b>
<b>Vordefinierte Dialoge</b>	<b>320</b>
<b>dlgDirectory()</b>	<b>321</b>
<b>dlgFileOpen(), dlgFileSave()</b>	<b>322</b>
<b>dlgMessageBox()</b>	<b>323</b>
<b>Dialog-Objekte</b>	<b>324</b>
<b>dlgCell</b>	<b>325</b>
<b>dlgCheckBox</b>	<b>326</b>
<b>dlgComboBox</b>	<b>327</b>
<b>dlgDialog</b>	<b>328</b>
<b>dlgGridLayout</b>	<b>329</b>
<b>dlgGroup</b>	<b>330</b>
<b>dlgHBoxLayout</b>	<b>331</b>
<b>dlgIntEdit</b>	<b>332</b>
<b>dlgLabel</b>	<b>333</b>
<b>dlgListBox</b>	<b>334</b>
<b>dlgListView</b>	<b>335</b>
<b>dlgPushButton</b>	<b>336</b>
<b>dlgRadioButton</b>	<b>337</b>
<b>dlgRealEdit</b>	<b>338</b>
<b>dlgSpacing</b>	<b>339</b>
<b>dlgSpinBox</b>	<b>340</b>
<b>dlgStretch</b>	<b>341</b>

---

<b>dlgStringEdit</b>	<b>342</b>
<b>dlgTabPage</b>	<b>343</b>
<b>dlgTabWidget</b>	<b>344</b>
<b>dlgTextEdit</b>	<b>345</b>
<b>dlgTextView</b>	<b>346</b>
<b>dlgVBoxLayout</b>	<b>347</b>
<b>Layout-Information</b>	<b>348</b>
<b>Dialog-Funktionen</b>	<b>349</b>
<b>dlgAccept()</b>	<b>350</b>
<b>dlgRedisplay()</b>	<b>351</b>
<b>dlgReset()</b>	<b>352</b>
<b>dlgReject()</b>	<b>353</b>
<b>Escape-Zeichen</b>	<b>354</b>
<b>Ein vollständiges Beispiel</b>	<b>355</b>
<b>Rich Text</b>	<b>356</b>
<b>Automatischer Backup</b>	<b>358</b>
<b>Forward&amp;Back-Annotation</b>	<b>359</b>
<b>Konsistenzprüfung</b>	<b>360</b>
<b>Einschränkungen</b>	<b>362</b>
<b>Technische Unterstützung</b>	<b>363</b>
<b>Lizenz</b>	<b>364</b>
<b>Produkt-Registrierung</b>	<b>365</b>
<b>EAGLE-Editionen</b>	<b>366</b>

## Erklärung der Help-Funktion

Ist in einem Layout-Editor »Seite 35-, Schaltplan-Editor »Seite 36-, oder Bibliotheks-Editor »Seite 33-Fenster ein Befehl aktiviert, dann wird nach Drücken der F1-Taste oder Eintippen von `HELP` die Help-Seite für diesen Befehl aufgerufen.

Die Erklärung eines Befehls kann auch durch die Eingabe von

`HELP befehl`

aufgerufen werden.

Ersetzen Sie "befehl" beispielsweise durch `MOVE`, dann rufen Sie die Help-Seite für den `MOVE`-Befehl auf.

Von jeder anderen Stelle aus öffnet F1 eine "kontextsensitive" Help-Seite.

Folgende Help-Seiten geben Auskunft für den Einstieg in das Programm.

- Schnelle Einführung »Seite 15
- EAGLE konfigurieren »Seite 13
- Control Panel »Seite 23

## EAGLE konfigurieren

Die folgenden Editor-Befehle können dazu verwendet werden, EAGLE individuell anzupassen. Sie können entweder direkt von der Kommandozeile eines Editor-Fensters eingegeben werden oder man kann Sie in die Datei eagle.scr »Seite 114 eintragen.

Für die Befehle ASSIGN und SET existieren auch Dialoge, die über das Options-Menü der Editor-Fenster aufgerufen werden.

### Benutzer-Interface

Befehlsmenü	MENU »Seite 83-Befehl..;
Tastenbelegung	ASSIGN »Seite 46 function_key-Befehl..;
Snap-Funktion	SET »Seite 115 SNAP_LENGTH number;
	SET »Seite 115 SNAP_BENDED ON   OFF;
	SET »Seite 115 SELECT_FACTOR value;
Inhalt von Menüs	SET »Seite 115 USED_LAYERS name   number;
	SET »Seite 115 WIDTH_MENU value..;
	SET »Seite 115 DIAMETER_MENU value..;
	SET »Seite 115 DRILL_MENU value..;
	SET »Seite 115 SMD_MENU value..;
	SET »Seite 115 SIZE_MENU value..;
Wire Bend	SET »Seite 115 WIRE_BEND bend_nr;
Beep ein/aus	SET »Seite 115 BEEP ON   OFF;

### Bildschirmdarstellung

Farbe für Grid-Linien	SET »Seite 115 COLOR_GRID color;
Farbe für Layer	SET »Seite 115 COLOR_LAYER layer color;
Fill Style für Layer	SET »Seite 115 FILL_LAYER layer fill;
Grid-Parameter	SET »Seite 115 GRID_REDRAW ON   OFF;
	SET »Seite 115 MIN_GRID_SIZE pixels;
min. dargest. Textgröße	SET »Seite 115 MIN_TEXT_SIZE size;
Darst. der Netzklinien	SET »Seite 115 NET_WIRE_WIDTH width;
Darst. der Pads	SET »Seite 115 DISPLAY_MODE REAL   NODRILL;
	SET »Seite 115 PAD_NAMES ON   OFF;
Darst. der Buslinien	SET »Seite 115 BUS_WIRE_WIDTH width;
DRC-Füllmuster	SET »Seite 115 DRC_FILL fill_name;
Polygon-Berechnung	SET »Seite 115 POLYGON_RATSNEST ON   OFF;
Vector Font	SET »Seite 115 VECTOR_FONT ON   OFF;

### Mode-Parameter

Package-Check	SET »Seite 115 CHECK_CONNECTS ON   OFF;
Grid-Parameter	GRID »Seite 71 options;
Replace-Modus	SET »Seite 115 REPLACE_SAME NAMES   COORDS;
UNDO-Buffer	SET »Seite 115 UNDO_LOG ON   OFF;
Wire-Optimierung	SET »Seite 115 OPTIMIZING ON   OFF;
Net-Wire beenden	SET »Seite 115 AUTO_END_NET ON   OFF;
Automatische Junctions	SET »Seite 115 AUTO_JUNCTION ON   OFF;

### Voreinstellungen (Default-Werte)

Pad-Form	CHANGE »Seite 51 SHAPE shape;
Wire-Breite	CHANGE »Seite 51 WIDTH value;
Pad/Via-Durchmesser	CHANGE »Seite 51 DIAMETER diameter;
Pad/Via/Hole-Bohrd.	CHANGE »Seite 51 DRILL value;
Smd-Größe	CHANGE »Seite 51 SMD width height;
Text-Höhe	CHANGE »Seite 51 SIZE value;
Text-Linienbreite	CHANGE »Seite 51 RATIO ratio;
Text-Font	CHANGE »Seite 51 FONT font;
Polygon-Parameter	CHANGE »Seite 51 THERMALS ON   OFF;
Polygon-Parameter	CHANGE »Seite 51 ORPHANS ON   OFF;
Polygon-Parameter	CHANGE »Seite 51 ISOLATE distance;
Polygon-Parameter	CHANGE »Seite 51 POUR SOLID   HATCH;
Polygon-Parameter	CHANGE »Seite 51 RANK value;
Polygon-Parameter	CHANGE »Seite 51 SPACING distance;



## Schnelle Einführung

Um schnell mit EAGLE zurechtzukommen, sollten Sie mehr über folgende Themen wissen:

- Control Panel und Editor-Fenster »Seite 16
- Editor-Befehle benutzen »Seite 40
- Eingabe von Parametern und Werten »Seite 17
- Schaltplan entwerfen »Seite 18
- Schaltung überprüfen »Seite 19
- Platine aus Schaltplan erzeugen »Seite 20
- Layout überprüfen »Seite 21
- Bauelement in Bibliothek definieren »Seite 22
- Autorouter benutzen »Seite 157
- Ausdrucken auf den System-Drucker »Seite 135
- Daten ausgeben mit dem CAM-Prozessor »Seite 139

Bei Problemen wenden Sie sich bitte an unseren kostenlosen Technischen Support »Seite 363.

## **Control Panel und Editor-Fenster**

Vom Control Panel »Seite 23 aus können Sie über das File-Menü oder durch Anklicken eines Icons die Fenster des Schaltplan-, Layout- und Bibliotheks-Editors öffnen.



## Eingabe von Parametern und Werten

Parameter und Werte können über die EAGLE-Kommandozeile oder, wesentlich bequemer, über die Parameter-Toolbar eingegeben werden, die erscheint, wenn ein Befehl aktiviert ist. Da dies keiner großen Erklärung bedarf, wird an anderen Stellen im Help-Text nicht explizit darauf hingewiesen.

## Schaltplan entwerfen

### Schaltplan anlegen

Neuen Schaltplan mit File/New anlegen und mit Save as unter neuem Namen abspeichern.

### Zeichnungsrahmen laden

Bibliothek FRAMES mit USE »Seite 127 laden und Rahmen mit ADD »Seite 42 platzieren.

### Symbole platzieren

Bibliotheken mit USE »Seite 127 laden und Symbole platzieren (siehe ADD »Seite 42, MOVE »Seite 86, DELETE »Seite 58, ROTATE »Seite 111, NAME »Seite 87, VALUE »Seite 128). Fehlt ein bestimmtes Bauelement, dann mit Bibliotheks-Editor definieren.

### Busse einzeichnen

Busse mit BUS »Seite 50 einzeichnen. Geben Sie den Bussen Namen (NAME »Seite 87), aus denen hervorgeht, welche Signale sich herausführen lassen.

### Netze einzeichnen

Die Verbindungen zwischen den Pins der Elemente definiert man mit NET »Seite 88. Dargestellt werden Netze im Net-Layer. Mit dem Befehl JUNCTION »Seite 78 kennzeichnet man Verbindungen sich kreuzender Netze.

## Schaltung überprüfen

Electrical Rule Check (ERC »Seite 65) durchführen und anhand der Meldungen Fehler korrigieren. Eventuell Netz- und Pin-Liste ausgeben (EXPORT »Seite 68). Mit dem SHOW »Seite 117-Befehl Netze am Bildschirm verfolgen.

## Platine aus Schaltplan erzeugen

Mit dem BOARD »Seite 49-Befehl bzw. durch Anklicken des Switch-to-Board-Icons erzeugen Sie eine Platine aus dem geladenen Schaltplan (falls noch keine Platine mit demselben Namen existiert).

Es entsteht eine Leerplatine, neben der die mit Luftlinien verbundenen Bauelemente platziert sind. Versorgungs-Pins werden mit den Signalen verbunden, die ihrem Namen entsprechen, falls nicht explizit ein anderes Netz mit ihnen verbunden wurde.

Die Platine ist über die Forward&Back-Annotation »Seite 359 mit der Schaltung verbunden. Damit ist gewährleistet, daß beide übereinstimmen. Um die Forward&Back-Annotation aufrechtzuerhalten, sollten Sie immer beide Dateien geladen haben, wenn Sie Änderungen an der Schaltung oder an der Platine durchführen.

### Platinenumrisse und Platzierung festlegen

Gegebenenfalls die Leerplatine in Größe und Form verändern (MOVE »Seite 86, SPLIT »Seite 121). Elemente an gewünschte Position verschieben (MOVE »Seite 86) und überprüfen, ob die Platzierung günstig oder ungünstig ist (RATSNEST »Seite 104).

### Sperrflächen definieren

Falls gewünscht, zeichnet man Sperrflächen für den Autorouter als Rechtecke, Polygone oder Kreise in die Layer tRestrict, bRestrict und vRestrict (siehe RECT »Seite 105, POLYGON »Seite 99, CIRCLE »Seite 52). Für den Autorouter begrenzen auch geschlossene Wire-Züge im Dimension-Layer die Route-Fläche.

### Routen

Mit dem ROUTE »Seite 112-Befehl lassen sich jetzt die Luftlinien in Leitungen umwandeln. Diese Aufgabe kann man auch dem Autorouter (siehe Befehl AUTO »Seite 48) zuweisen.

## Layout überprüfen

Design Rule Check (DRC »Seite 63) durchführen und Fehler korrigieren (ERRORS »Seite 67). Eventuell Netz-, Bauteile- und Pin-Liste ausgeben (EXPORT »Seite 68).

## Bauelement in Bibliothek definieren

Die Definition eines Bauelements erfordert drei Schritte, die aufeinander aufbauen.

Öffnen Sie eine Bibliothek mit Open oder New im File-Menü (nicht mit dem USE-Befehl).

### Package definieren

Packages sind die Gehäuse der Bauelemente, die im Board dargestellt werden.

Klicken Sie das Edit-Package-Icon an, und tragen Sie den gewünschten Namen in das New-Feld ein.

Legen Sie das Raster fest (GRID »Seite 71).

Plazieren Sie die Pads (PAD »Seite 92) und legen Sie deren Namen (NAME »Seite 87) und Parameter fest (CHANGE »Seite 51).

Plazieren Sie mit dem TEXT »Seite 123-Befehl die Strings >NAME und >VALUE (repräsentieren den aktuellen Namen und Wert in der Platine), und zeichnen Sie das Gehäuse (WIRE »Seite 131-Befehl) in die entsprechenden Layer.

### Symbol definieren

Symbole sind der Teil eines Device, der im Schaltplan dargestellt wird.

Klicken Sie das Edit-Symbol-Icon an, und tragen Sie den gewünschten Namen in das New-Feld ein.

Plazieren Sie die Pins (PIN »Seite 95) und legen Sie deren Namen (NAME »Seite 87) und Parameter fest (CHANGE »Seite 51).

Plazieren Sie mit dem TEXT »Seite 123-Befehl die Strings >NAME und >VALUE (repräsentieren den aktuellen Namen und Wert im Schaltplan), und zeichnen Sie das Symbol (WIRE »Seite 131-Befehl) in die entsprechenden Layer.

### Device definieren

Ein Device enthält die Definition eines gesamten Bauelements einschließlich Gehäuse und Schaltplan-Symbol(e).

Klicken Sie das Edit-Device-Icon an, und tragen Sie den gewünschten Namen in das New-Feld ein.

Weisen Sie dem Device ein Gehäuse zu (PACKAGE »Seite 91-Befehl).

Verwenden Sie den ADD »Seite 42-Befehl, um das Symbol oder die Symbole in das Device zu holen.

Klicken Sie auf das CONNECT »Seite 55-Icon, um festzulegen, welche Pins mit welchen Gehäuse-Pads verbunden sind.

Speichern Sie die Bibliothek, und sie kann anschließend vom Schaltplan- oder vom Board-Editor aus mit USE »Seite 127 geladen werden.

## Control Panel

Das Control Panel ist EAGLEs Steuerzentrale. Es enthält in der linken Fensterhälfte eine Baumstruktur und ein Informationsfenster in der rechten Hälfte.

### Verzeichnisse einstellen

Die Haupteinträge in der Baumansicht repräsentieren die verschiedenen EAGLE-Dateitypen. Jeder der Einträge kann auf ein oder mehrere Verzeichnisse zeigen, die Dateien dieses Typs enthalten. Die Verzeichnisse werden im Directories-Dialog »Seite 27 definiert. Wenn einer der Haupteinträge auf ein Verzeichnis zeigt, sehen Sie nach dem Aufklappen des Eintrags (entweder durch einen Klick auf das kleine Symbol links oder durch einen Doppelklick auf den Eintrag selbst) direkt den Inhalt des Verzeichnisses. Wenn ein Haupteintrag auf mehrere Verzeichnisse zeigt, werden nach dem Aufklappen alle Verzeichniseinträge aufgelistet.

### Kontext-Menü

Das Kontext-Menü »Seite 25 eines Eintrags in der Baustuktur erreichen Sie mit einem rechten Mausklick auf den entsprechenden Eintrag. Es enthält dann spezielle Punkte zu diesem Eintrag.

### Descriptions

Die *Description*-Spalte der Baumansicht enthält eine Kurzbeschreibung des Eintrags (wenn vorhanden). Diese Beschreibungen werden aus der ersten nicht leeren Textzeile der folgenden Quellen erzeugt:

Verzeichnisse	Die Datei mit dem Namen DESCRIPTION darin
Libraries	DESCRIPTION-Befehl in der Bibliothek
Devices	DESCRIPTION-Befehl im Device-Editor
Packages	DESCRIPTION-Befehl im Package-Editor
Design Rules	Die Beschreibung der Design-Rule-Datei im DRC-Dialog
User Language Programs	Text, durch die #usage-Anweisung gekennzeichnet
Scripts	Der Kommentar am Anfang der Script-Datei
CAM Jobs	DESCRIPTION-Befehl im CAM-Prozessor-Job

### Drag&Drop

Sie können mit Hilfe von *Drag&Drop* Dateien und Verzeichnisse innerhalb der Baumstruktur kopieren oder bewegen. Es ist auch möglich, ein Device oder ein Package in das Schaltplan-Fenster bzw. in das Layout-Fenster zu ziehen und dort zu platzieren. Zieht man User-Language-Programme und Script-Dateien auf ein Editor-Fenster, werden sie darin ausgeführt. Design Rules werden einem Layout zugeordnet, wenn Sie einen entsprechenden Eintrag in das Layout-Editor-Fenster ziehen. Ziehen Sie eine Bibliothek in ein Board- oder Schaltplan-Fenster, erfolgt ein Library-Update »Seite 126. All diese Funktionen erreichen Sie auch über das *Kontext-Menü* des entsprechenden Eintrags in der Baumstruktur.

### Informations-Fenster

In der rechten Hälfte des Control Panels werden Informationen zum selektierten Punkt in der Baumstruktur angezeigt. Diese Informationen werden aus den Quellen erzeugt, die unter *Description* angegeben sind. Bei Device und Package erhält man eine Voransicht des Elements.

### Menü-Leiste

Die *Menü-Leiste* des Control Panels enthält folgende Punkte:

#### File

<u>N</u> ew	Erzeugt eine neue Datei
<u>O</u> pen	Öffnet existierende Datei oder legt neue Datei an
<u>S</u> ave all	Speichert alle modifizierten Editor-Dateien
<u>R</u> efresh Tree	Aktualisiert den Inhalt der Baumansicht
<u>C</u> lose project	Schließt das aktuelle Projekt
<u>E</u> xit	Beendet das Programm

#### Options

<u>D</u> irectories...	Öffnet den Datei-Dialog »Seite 27
<u>B</u> ackup...	Öffnet den Backup-Dialog »Seite 28
<u>U</u> ser interface...	Öffnet den User-Interface-Dialog »Seite 29

**Window**

Control Panel Alt+0 Zum Control Panel wechseln  
1 Schematic - ... Zu Fenster 1 wechseln  
2 Board - ... Zu Fenster 2 wechseln

**Help**

General help Öffnet eine allgemeine Help-Seite  
Contents Öffnet die Inhaltsseite der Help-Funktion  
Control panel Öffnet diese Help-Seite  
Product registration Öffnet den Produkt-Registrierungs »Seite 365-Dialog  
Product information Öffnet das Produkt-Informations-Fenster, das Details zu Ihrer EAGLE-Lizenz »Seite 364 enthält

**Statuszeile**

Die Statuszeile unten im Control Panel enthält den vollständigen Namen des gegenwärtig selektierten Objektes.



## Kontext Menüs

Ein Klick mit der rechten Maustaste auf ein Objekt des Control Panels »Seite 23 öffnet ein Kontext Menü das folgende Aktionen ermöglicht (nicht alle davon sind für alle Objekte zutreffend):

### **New Folder**

Erzeugt ein neues Verzeichnis unterhalb des selektierten Ordners und schaltet den neu erzeugten Zweig der Baumansicht in den *Rename*-Modus, so dass der gewünschte Name vergeben werden kann.

### **Edit Description**

Lädt die DESCRIPTION-Datei in den Rich-Text-Editor.

### **Rename**

Schaltet den Eintrag der Baumansicht in den Editier-Modus, so dass er umbenannt werden kann. Das kann man auch durch einen Mausklick auf den Text des selektierten Eintrags erreichen.

### **Copy**

Öffnet einen File-Dialog in dem Sie den neuen Namen der zu kopierenden Datei bzw. des Verzeichnisses angeben. Dateien oder Verzeichnisse kann man auch mit Hilfe von *Drag&Drop* kopieren.

### **Delete**

Löscht die Datei oder das Verzeichnis. Sie werden vor dem Löschen gefragt, ob wirklich gelöscht werden soll.

### **Use**

Ist eine Bibliothek als *in use* markiert, werden die Devices und Packages darin von der Suchfunktion berücksichtigt. Sie können auch durch einen Mausklick auf den Marker in der zweiten Spalte der Baumansicht die Bibliothek freigeben oder nicht.

### **Use all**

Alle Bibliotheken im Bibliothekspfad werden bei der Suche nach Devices und Packages berücksichtigt, d. h. alle Bibliotheken sind *in use*.

### **Use none**

Keine der Bibliotheken ist *in use* (einschließlich der Bibliotheken, die nicht im Bibliothekspfad stehen).

### **Update**

Tauscht alle Bauteile eines Schaltplans oder Layouts, die aus dieser Bibliothek genommen wurden gegen die aktuelle Bauteile-Definition. Das kann auch mit Hilfe von *Drag&Drop* erledigt werden.

### **Add to schematic**

Startet den ADD »Seite 42-Befehl im Schaltplan-Editor für dieses Device. Das ist auch über *Drag&Drop* möglich.

### **Add to board**

Startet den ADD »Seite 42-Befehl im Layout-Editor für dieses Package. Das ist auch über *Drag&Drop* möglich.

### **Open/Close Project**

Öffnet oder schließt ein Projekt. Dazu können Sie auch auf den Marker rechts vom Projekt-Eintrag in der Baumansicht klicken.

### **New**

Öffnet ein Fenster mit der neuen Datei des entsprechenden Typs.

### **Open**

Öffnet ein Editor-Fenster mit dieser Datei.

### **Print...**

Druckt die Datei auf dem System-Drucker aus. Weitere Informationen zur Benutzung der Druck-Dialoge entnehmen Sie bitte dem Kapitel Drucken auf dem System-Drucker »Seite 135.

Wird eine Datei über dies Kontext-Menü Option ausgedruckt, so wird immer die Datei von der Platte gelesen, auch wenn Sie ein offenes Editor Fenster haben in dem Sie die Datei editieren! Benutzen Sie den PRINT »Seite 102-Befehl um eine Zeichnung aus einem offenen Editor Fenster heraus zu drucken.

**Bitte beachten Sie, daß Polygone in Platinen beim Ausdrucken über das Kontext-Menü nicht automatisch freigerechnet werden! Es werden lediglich die Umrisse dargestellt. Um die Polygone freigerechnet auszudrucken, laden Sie die Zeichnung in ein Editor-Fenster, geben Sie RATSNEST »Seite 104 ein und dann PRINT »Seite 102.**

### **Run in ...**

Startet das gewählte User-Language-Programm im aktuellen Schaltplan, Board oder in der Bibliothek. Das ist auch über *Drag&Drop* möglich.

### **Execute in ...**

Führt diese Script-Datei im aktuellen Schaltplan, Layout oder in der Bibliothek aus. Das ist auch über *Drag&Drop* möglich.

### **Load into Board**

Lädt diesen Satz von Design Rules für das aktuelle Board. Das ist auch über *Drag&Drop* möglich.

## Verzeichnisse

Mit dem *Directories*-Dialog definiert man die Pfade, in denen nach Dateien gesucht werden soll.

Alle Felder können ein oder mehrere Verzeichnisse enthalten (getrennt durch ' ; ' ) enthalten, in denen nach den verschiedenen Dateitypen gesucht wird. Wird einer der Befehle OPEN »Seite 89, USE »Seite 127, SCRIPT »Seite 114 oder RUN »Seite 113 eingegeben, dann werden diese Pfade durchsucht, mit Priorität von links nach rechts. Wird der Datei-Dialog benutzt um eine Datei eines dieser Typen anzusprechen, so wird das Verzeichnis, in das der Anwender mittels des Datei-Dialogs navigiert hat, implizit an das Ende des jeweiligen Pfades angehängt.

Die Variablen \$HOME and \$EAGLEDIR werden verwendet, um das Home-Verzeichnis des Benutzers bzw. das EAGLE-Programm-Verzeichnis anzugeben. Unter Windows gilt für \$HOME entweder die Umgebungsvariable HOME (falls gesetzt), oder der Wert des Schlüssels

"HKEY\_CURRENT\_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders\Personal" in der Windows-Registry, der den gültigen Pfad des Verzeichnisses "Eigene Dateien" enthält.

## Backup

Der *Backup*-Dialog ermöglicht es Ihnen, den automatischen Backup »Seite 358 individuell einzustellen.

### **Maximum backup level**

Definiert wie viele Backup-Kopien Ihrer EAGLE-Dateien "aufgehoben" werden wenn eine Datei normal mit dem WRITE-Befehl abgespeichert wird (Default: 9).

### **Auto backup interval (minutes)**

Bestimmt das Zeitintervall nachdem EAGLE automatisch eine Sicherungskopie aller modifizierten Zeichnungsdateien erzeugt (default ist 5 min.).

### **Automatically save project file**

Ist diese Option gewählt, wird Ihr Projekt automatisch gesichert, wenn Sie das Programm verlassen, vorausgesetzt Sie haben mit File/Open/Project... oder Options/Save as... vom Control Panel »Seite 23 aus eine Projekt-Datei angelegt.

EAGLE wird standardmäßig mit einer leeren Projekt-Datei geliefert (*EAGLE.EPF*), so daß Ihr Desktop automatisch gesichert wird.

## User Interface

Der *User interface* Dialog ermöglicht es, das Erscheinungsbild der Editor-Fenster »Seite 32 für Layout, Schaltplan und Bibliothek den eigenen Vorstellungen anzupassen.

### Controls

Pulldown menu	aktiviert das Pulldown-Menü am oberen Rand des Editor-Fensters
Action toolbar	aktiviert die Toolbar mit Buttons für "File", "Print" etc.
Parameter toolbar	aktiviert die dynamische Parameter-Toolbar, die alle Parameter des gerade aktiven Befehls enthält
Command buttons	aktiviert die Kommando-Toolbar
Command texts	aktiviert das Text-Menü

### Layout

Background	wählt die Hintergrundfarbe schwarz oder weiß im Layout-Editor
Cursor	wählt einen kleinen Kreuz oder ein großes Fadenkreuz als Cursor im Layout-Editor

### Schematic

Background	wählt die Hintergrundfarbe schwarz oder weiß im Schaltplan-Editor
Cursor	wählt ein kleines Kreuz oder ein großes Fadenkreuz als Cursor im Schaltplan-Editor

### Help

Bubble help	aktiviert die "Bubble Help" Funktion, die einen kurzen Hinweis über die Bedeutung der Buttons gibt, wenn man den Cursor über einen solchen bewegt
User guidance	aktiviert die "User Guidance" Funktion, die einen Hilfetext anzeigt, der dem Benutzer bei aktivem Befehl jederzeit den nächsten sinnvollen Schritt angibt

### Misc

Always vector font	Texte in Zeichnungen werden immer im EAGLE-eigenen Vektor-Font dargestellt, unabhängig welche Schriftart für einen Text ursprünglich gewählt wurde.
Mouse wheel zoom	definiert den Zoom-Faktor für das Hinein- bzw. Herauszoomen mit einer Rädchenmaus in einem Editor-Fenster ('0' schaltet diese Funktion aus, das Vorzeichen bestimmt die Drehrichtung des Rädchens)

## Tastatur und Maus

### Tastatur

Wird bei einem aktiven Befehl die ESC-Taste gedrückt, so wird die aktuelle Aktion dieses Befehls beendet, ohne daß der gesamte Befehl abgebrochen wird.

Für den MOVE-Befehl zum Beispiel bedeutet dies, daß ein am Cursor befindliches Objekt "fallengelassen" wird und ein anderes Objekt selektiert werden kann.

Die Tasten `Crsr-Up` (Pfeil nach oben) bzw. `Crsr-Down` (Pfeil nach unten) erlauben es in der Kommandozeile des Editor-Fensters früher eingegeben Befehlszeilen wieder herzuholen ("History-Funktion").

Siehe auch ASSIGN »Seite 46-Befehl.

### Maustasten

Verwenden Sie die linke Maustaste für alle Aktionen, die nicht in den folgenden Tabellen aufgeführt sind.

#### Verwendung der mittleren Maustaste

ARC »Seite 45   Aktiven Layer wechseln  
CIRCLE »Seite 52   Aktiven Layer wechseln  
LABEL »Seite 79 Aktiven Layer wechseln  
POLYGON »Seite 99   Aktiven Layer wechseln  
RECT »Seite 105 Aktiven Layer wechseln  
ROUTE »Seite 112   Aktiven Layer wechseln  
SMD »Seite 120 Aktiven Layer wechseln  
TEXT »Seite 123 Aktiven Layer wechseln  
WIRE »Seite 131 Aktiven Layer wechseln

Falls sich mit Ihrer Maus bei den angegebenen Befehlen mit der mittleren Maustaste das Layer-Menü nicht öffnet, verwenden Sie zum Wechseln des Layers den LAYER »Seite 80-Befehl.

#### Verwendung der rechten Maustaste

GROUP »Seite 73   Polygon schließen  
ADD »Seite 42   Element rotieren  
INVOKE »Seite 77   Gate rotieren  
LABEL »Seite 79 Text rotieren  
MOVE »Seite 86 Element rotieren  
PAD »Seite 92   Pad rotieren  
PIN »Seite 95 Pin rotieren  
PASTE »Seite 94 Paste Buffer rotieren  
ROTATE »Seite 111   Gruppe rotieren  
SMD »Seite 120 Smd-Pad rotieren  
TEXT »Seite 123 Text rotieren  
ARC »Seite 45   Richtung d. Kreisbogens ändern  
MIRROR »Seite 85   Gruppe spiegeln  
POLYGON »Seite 99   Wire Bend ändern  
ROUTE »Seite 112   Wire Bend ändern  
SPLIT »Seite 121 Wire Bend ändern  
WIRE »Seite 131 Wire Bend ändern

## Benachbarte Objekte selektieren

Wenn Sie an einer Stelle ein Objekt selektieren wollen, an der mehrere Objekte eng beieinander liegen, nimmt der Cursor die Form eines Vierfach-Pfeils an, und in der Kommandozeile erscheint die Frage

"select highlighted object? (left=yes, right=no)"

Drücken Sie die rechte Maustaste, um zyklisch durch alle in Frage kommenden Objekte "durchzutasten".

Mit der linken Maustaste wählen Sie das gewünschte Objekt aus.

Um die Auswahl ganz abubrechen, drücken Sie die Esc-Taste.

Mit dem Befehl

```
SET »Seite 115 Select_Factor select_radius;
```

können Sie beeinflussen, wie groß der "Selektionsradius" sein soll.

## Editor-Fenster

EAGLE kennt unterschiedliche Typen von Daten-Files. Jeder davon wird in einem eigenen Editor-Fenster-Typ bearbeitet. Wenn Sie eines der Objekte durch Doppelklick selektieren oder vom Control Panel »Seite 23 aus eine Datei mit **File/Open** laden, dann öffnet sich ein Editor-Fenster für diesen Dateityp.

- Bibliotheks-Editor »Seite 33
- Schaltplan-Editor »Seite 36
- Layout-Editor »Seite 35
- Text-Editor »Seite 37



## Bibliotheks-Editor

Der *Bibliotheks-Editor* dient dazu, Bauelemente-Bibliotheken (\* .`lib`) zu editieren.

Nachdem Sie ein neues Bibliotheks-Editor-Fenster geöffnet haben, erscheint eine leere Arbeitsfläche, und Sie müssen mit dem EDIT »Seite 64 Befehl angeben, welches Gehäuse (Package), Symbol oder Device Sie laden oder neu anlegen wollen.

## Bibliotheks-Editier-Modus

Im Bibliotheks-Editier-Modus können Sie Packages, Symbole und Devices editieren.

Package: Die Gehäuse-Definition.

Symbol: Schaltzeichen, wie es im Schaltplan erscheinen soll.

Device: Definition des realen Bauteils. Enthält eine oder mehrere Package-Varianten und ein oder mehrere Symbole (Gates). Es dürfen unterschiedliche Symbole kombiniert werden.

Klicken Sie den Button **Dev**, **Pac** oder **Sym**, um Devices, Packages oder Symbole zu selektieren.

Wenn Sie ein neues Objekt anlegen wollen, schreiben Sie den Namen des neuen Objekts in das Feld **New** Sie können auch ein existierendes Objekt laden, indem Sie seinen Namen in dieses Feld eintippen. Wenn Sie die Extension des File-Namens nicht angeben, wird ein Objekt geladen, dessen Typ vom **Choose...** Prompt bestimmt wird. Anderenfalls bestimmt die Extension den Typ.

Falls Ihre Lizenz »Seite 364 das Schaltplan-Modul nicht einschließt, erscheinen die Object-Type-Buttons (**Dev...**) nicht im Menü.

## Layout-Editor

Der *Layout-Editor* dient dazu, Layouts (\* .brd) zu editieren.

Wenn im selben Verzeichnis eine Schaltplan-Datei (\* .sch) mit demselben Namen existiert, wird beim Öffnen eines Layouts automatisch auch ein Schaltplan-Editor »Seite 36-Fenster mit dieser Datei als Icon auf den Desktop gelegt. Das ist notwendig, damit die Schaltplan-Datei geladen ist, wenn Änderungen an der Platine die Back-Annotation »Seite 359 zum Schaltplan erforderlich machen.

## Schaltplan-Editor

Der *Schaltplan-Editor* dient dazu, Schaltpläne (\* .sch) zu editieren.

Wenn im selben Verzeichnis eine Layout-Datei (\* .brd) mit demselben Namen existiert, wird beim Öffnen eines Layouts automatisch auch ein Layout-Editor »Seite 35-Fenster mit dieser Datei als Icon auf den Desktop gelegt. Das ist notwendig, damit die Layout-Datei geladen ist, wenn Änderungen an der Platine die Forward-Annotation »Seite 359 zum Layout erforderlich machen.

Mit Hilfe der Combo-Box in der Action-Toolbar des Schaltplan-Editor-Fensters können Sie zwischen den einzelnen Schaltplan-Blättern (Sheets) wechseln oder neue Blätter anlegen. Sie können dazu auch den EDIT »Seite 64-Befehl verwenden.

## Text-Editor

Der *Text-Editor* dient dazu, Textdateien zu editieren.

Die Textdatei muß eine reine ASCII-Datei sein und darf keine Steuerzeichen enthalten. Als Anwendungen sind in erster Linie vorgesehen: das Schreiben von User-Language-Programmen »Seite 161 und Script-Dateien »Seite 114 und die Anzeige der Ergebnisse des Electrical Rule Checks »Seite 65.

## Editor-Befehle

### EAGLE-Befehle und ihre Bedeutung

#### Wechsel der Betriebsart/Dateibefehle

EDIT »Seite 64 Bibliothek laden/anlegen  
WRITE »Seite 133 Zeichnung/Bibliothek speichern  
OPEN »Seite 89 Bibliothek zum Editieren laden  
CLOSE »Seite 54 Bibliothek nach Editieren schließen  
QUIT »Seite 103 EAGLE verlassen  
EXPORT »Seite 68 ASCII-Liste erzeugen (z.B. Netzliste)  
SCRIPT »Seite 114 Befehlsdatei ausführen  
USE »Seite 127 Bibl. zum Plazieren von Elementen laden  
REMOVE »Seite 107 Dateien/Bibl.-Elemente löschen

#### Zeichnungen/Bibliotheken anlegen und editieren

ARC »Seite 45 Kreisbogen zeichnen  
CIRCLE »Seite 52 Kreis zeichnen  
POLYGON »Seite 99 Polygon zeichnen  
RECT »Seite 105 Rechteck zeichnen  
WIRE »Seite 131 Linie oder geroutete Verbindung zeichnen  
TEXT »Seite 123 Text zu einer Zeichnung hinzufügen  
ADD »Seite 42 Element in Zeichnung / Symbol in Device einfügen  
COPY »Seite 56 Objekte/Elemente kopieren  
GROUP »Seite 73 Gruppe für spätere Operation definieren  
CUT »Seite 57 Vorher definierte Gruppe in Paste-Buffer laden  
PASTE »Seite 94 Paste-Buffer in Zeichnung einfügen  
DELETE »Seite 58 Objekt löschen  
MIRROR »Seite 85 Objekt spiegeln  
MOVE »Seite 86 Objekt bewegen oder rotieren  
ROTATE »Seite 111 Objekt rotieren  
NAME »Seite 87 Objekt mit Namen versehen  
VALUE »Seite 128 Wert für Element definieren/ändern  
SMASH »Seite 119 NAME/VALUE zum Bewegen vorbereiten  
SPLIT »Seite 121 Wires/Linien (Netze etc.) knicken  
LAYER »Seite 80 Layer definieren/wechseln

#### Spezielle Befehle für Platinen

SIGNAL »Seite 118 Signal (Luftlinie) definieren  
ROUTE »Seite 112 Signal routen  
RIPUP »Seite 110 Signal auflösen  
DELETE »Seite 58 Ein Segment eines Signals auflösen  
VIA »Seite 129 Durchkontaktierung (Via) plazieren  
HOLE »Seite 75 Bohrung (Hole) definieren  
RATSNEST »Seite 104 Kürzeste Luftlinien anzeigen  
REPLACE »Seite 109 Bauteil ersetzen  
DRC »Seite 63 Design Rule Check durchführen  
ERRORS »Seite 67 DRC-Fehler anzeigen

#### Spezielle Befehle für Schaltpläne

NET »Seite 88 Netz definieren  
BUS »Seite 50 Buslinie zeichnen  
JUNCTION »Seite 78 Verbindungspunkt plazieren  
INVOKE »Seite 77 'Gate' aus Device plazieren  
LABEL »Seite 79 Label für Bus oder Netz plazieren  
GATESWAP »Seite 70 Äquivalente 'Gates' tauschen  
PINSWAP »Seite 98 Äquivalente Pins tauschen  
ERC »Seite 65 Electrical Rule Check ausführen  
BOARD »Seite 49 Platine aus einem Schaltplan erzeugen

#### Spezielle Befehle für Bibliotheken

RENAME »Seite 108 Symbol/Package/Device neu benennen

CONNECT »Seite 55      Pin/Pad-Zuordnung festlegen  
PACKAGE »Seite 91      Package für Device definieren  
PREFIX »Seite 101      Default-Präfix für Device festlegen  
VALUE »Seite 128      Definieren, ob Value-Text änderbar  
PAD »Seite 92      Pad in Package einfügen  
SMD »Seite 120      Smd-Pad in Package einfügen  
PIN »Seite 95      Pin in Symbol einfügen  
HOLE »Seite 75      Nichtleitende Bohrung definieren  
REMOVE »Seite 107      Elemente aus Bibl. löschen

#### Befehle für Bildschirmdarstellung und Benutzer-Interface

WINDOW »Seite 130      Bildausschnitt verändern  
DISPLAY »Seite 61      Layer anzeigen/ausblenden  
ASSIGN »Seite 46      Tasten belegen  
CHANGE »Seite 51      EAGLE-Parameter ändern  
GRID »Seite 71      Raster/Einheit definieren  
MENU »Seite 83      Befehls-Menü konfigurieren  
SET »Seite 115      Programm-Parameter einstellen

#### Weitere Befehle

AUTO »Seite 48      Autorouter starten  
HELP »Seite 74      Help-Seite anzeigen  
INFO »Seite 76      Information über Objekt zeigen  
MARK »Seite 82      Meßmarke setzen/entfernen  
OPTIMIZE »Seite 90      Wire-Segmente zusammenfassen  
RUN »Seite 113      User-Language-Programm ausführen  
SHOW »Seite 117      Objekt hell darstellen  
UNDO »Seite 125      Befehle zurücknehmen  
REDO »Seite 106      Zurückgenommene Befehle ausführen  
PRINT »Seite 102      Ausdrucken auf dem System-Drucker  
UPDATE »Seite 126      Bibliotheks-Objekte aktualisieren

## Befehlseingabe

### Befehlseingabe

EAGLE-Befehle können auf vier verschiedene Arten eingegeben werden:

- mit der Tastatur als Text
- mit der Maus, durch Anklicken von Menüpunkten oder Icons
- mit belegten Tasten (siehe ASSIGN »Seite 46-Befehl)
- mit einer Script-Datei (siehe SCRIPT »Seite 114-Befehl)

Diese Eingabearten können auch gemischt verwendet werden.

Für die Befehlsbeschreibungen gelten folgende Regeln:

Befehle und Parameter in GROSSBUCHSTABEN werden direkt eingegeben (bzw. aus dem Befehlsmenü mit der Maus selektiert). Bei der Eingabe werden Groß- und Kleinbuchstaben nicht unterschieden.

Parameter in Kleinbuchstaben werden durch Namen, Zahlenwerte oder Schlüsselwörter ersetzt. Beispiel:

Syntax: GRID grid\_size grid\_multiple;  
Input: GRID 1 10;

### Verkürzte Eingabe

Befehle und andere Schlüsselwörter können beliebig abgekürzt werden, solange sie nicht mit anderen Schlüsselwörtern verwechselt werden können.

### Alternative Parameter

Das Zeichen | bedeutet, daß Parameter alternativ angegeben werden können. Beispiel:

Syntax: SET BEEP ON | OFF;  
Input: SET BEEP ON;  
oder  
SET BEEP OFF;

### Wiederholungspunkte

Die Zeichen .. bedeuten, daß die Funktion mehrfach ausgeführt werden kann bzw. daß mehrere Parameter vom gleichen Typ erlaubt sind. Beispiel:

Syntax: DISPLAY option layer\_name..  
Input: DISPLAY TOP PINS VIAS

### Koordinatenangaben

Das Zeichen • bedeutet normalerweise, daß an dieser Stelle im Befehl mit der linken Maustaste ein Objekt anzuklicken ist. Beispiel:

Syntax: MOVE • •..  
Input: MOVE  
Mausklick auf erstes zu bewegendes Element  
Mausklick auf das Ziel  
Mausklick auf zweites zu bewegendes Element  
etc.

An diesem Beispiel sehen Sie auch, wie die Wiederholungspunkte bei Befehlen mit Mausklicks zu verstehen sind.

Jeder Mausklick stellt eine Koordinatenangabe dar. Will man den Befehl textuell eingeben, dann kann man anstelle des Mausklicks die Koordinaten über die Tastatur in folgender Form eingeben:

(x y)

Dabei sind x und y Zahlen in der mit dem GRID-Befehl gewählten Einheit. Die textuelle Eingabemöglichkeit ist



insbesondere für Script-Dateien erforderlich.

Beispiel für die Koordinateneingabe in Textform: Sie wollen Platinenumrisse mit exakten Maßen eingeben.

```
GRID 1 MM;  
CHANGE LAYER DIMENSION;  
WIRE 0 (0 0) (160 0) (160 100) (0 100) (0 0);  
GRID LAST;
```

### **Strichpunkt**

Der Strichpunkt (';') schließt einen Befehl ab. Ein Befehl muß dann mit einem Strichpunkt abgeschlossen werden, wenn er weniger als die maximal mögliche Zahl von Parametern enthält. Der Befehl

```
WINDOW;
```

frischt beispielsweise das Zeichenfenster auf, während

```
WINDOW FIT
```

das Zeichenfenster so skaliert, daß die gesamte Zeichnung sichtbar ist. Im zweiten Fall ist kein Strichpunkt erforderlich, weil bereits klar ist, daß kein weiterer Parameter folgen kann.

## ADD

### Funktion

Element in eine Zeichnung einfügen.  
Symbol in ein Device einfügen.

### Syntax

```
ADD 'name'                package_name[@library_name] orientation •..
ADD 'name' ['gate'] device_name[@library_name] orientation •..
ADD 'name'                symbol_name                options      •..
```

### Maus

Rechte Maustaste rotiert das Element.

**Siehe auch** UPDATE »Seite 126, USE »Seite 127, INVOKE »Seite 77

Der ADD-Befehl holt ein Schaltplan-Symbol (Gate) oder ein Package aus der aktiven Bibliothek und platziert es in der Zeichnung.

Bei der Device-Definition holt der ADD-Befehl ein Symbol in das Device.

Üblicherweise klickt man den ADD-Befehl an und selektiert das Package/Symbol aus dem sich öffnenden Menü. Nun können die Parameter (falls erforderlich) per Tastatur eingegeben werden.

Wenn `device_name` Platzhalter enthält ('\*' oder '?') und mehr als ein Device gefunden wird, öffnet sich der ADD-Dialog. Daraus kann dann das gewünschte Device gewählt werden.

Plaziert wird das Package/Symbol mit der linken Maustaste, rotiert wird es mit der rechten. Nachdem es platziert wurde, hängt sofort eine weitere Kopie am Cursor.

Wenn bereits ein Device oder Package mit gleichem Namen (aus derselben Bibliothek) in der Zeichnung existiert und die Bibliothek seit dem Plazieren des ursprünglichen Elements modifiziert wurde, startet EAGLE automatisch einen Library-Update »Seite 126 bei dem Sie gefragt werden, ob die Bauteile durch die neuere Bibliotheksdefinition ersetzt werden sollen. **Achtung: Nach einem Library-Update sollten Sie immer den Design Rule Check »Seite 63 (DRC) und den Electrical Rule Check »Seite 65 (ERC) laufen lassen!**

### Package oder Symbol in Zeichnung holen

#### Platzhalter

Der ADD-Befehl kann mit Platzhaltern ('\*' oder '?') arbeiten, um ein bestimmtes Element zu finden. Der ADD-Dialog zeigt alle gefundenen Elemente in einer Baumannsicht und dazu auch eine Voransicht des Devices und der Package-Variante.

Um ein Element direkt zu plazieren, verwenden Sie die Syntax:

```
ADD devicename@libraryname
```

`devicename` darf Platzhalter enthalten und `libraryname` darf entweder direkt der Bibliotheksname (wie "ttl" oder "ttl.lbr") oder der volle Name mit Pfadangabe (wie "/home/mydir/myproject/ttl.lbr" oder "../lbr/ttl") sein.

#### Namen

Der Parameter `package_name`, `device_name` bzw. `symbol_name` ist der Name, unter dem das Package/Device/Symbol in der Bibliothek abgelegt ist. Er wird üblicherweise aus einem Menü selektiert. Der Parameter `name` ist der Name, den das Element in der Zeichnung erhalten soll. Er muß in Hochkommas eingeschlossen sein. Wird er nicht explizit angegeben, erhält das Element einen generierten Namen.

Beispiel:

```
ADD 'IC1' DIL14 •
```

holt das Package DIL14 in die Platine und gibt ihm den Namen IC1.

Wird im Schaltplan kein Name angegeben, erhält das Gate als Namen den bei der Device-Definition mit PREFIX »Seite 101 festgelegten Präfix, ergänzt um eine fortlaufende Zahl (z.B. IC1).

Beispiel:

```
ADD 7400 • • • • •
```

Hier werden der Reihe nach fünf Gatter aus Bausteinen des Typs 7400 platziert. Sofern als Präfix "IC" definiert wurde und die Einzelgatter innerhalb eines 7400 die Namen A...D haben, erhalten die Gatter in der Schaltung die Namen IC1A, IC1B, IC1C, IC1D, IC2A (falls schon Elemente mit demselben Präfix platziert wurden, wird die Zählung mit der nächsten laufenden Nummer fortgesetzt. Siehe auch INVOKE »Seite 77.

#### Bestimmte Gatter

Um ein bestimmtes Gatter eines neu hinzugefügten Bausteins zu holen kann nach dem Bauteilnamen der Name des Gatters in einfachen Hochkommas angegeben werden:

Beispiel:

```
ADD 'IC1' 'A' DIL14 •
```

Dies ist vor allem dafür gedacht, wenn ein Schematic über ein Script generiert werden soll. Beachten Sie bitte, daß wenn ein bestimmtes Gatter geholt wird keine anderen Gatter mit Add-Level MUST oder ALWAYS automatisch mit geholt werden, und Sie müssen zumindest die MUST-Gatter mit dem INVOKE »Seite 77-Befehl aktivieren (ansonsten wird sie der Electrical Rule Check »Seite 65 als fehlend melden).

#### Orientation

Dieser Parameter gibt die Ausrichtung des Bibliothekselements in der Zeichnung an. Normalerweise rotiert man das Element mit der rechten Maustaste. In Script »Seite 114-Dateien verwendet man die textuellen Angaben für diesen Parameter:

Als Orientation sind zulässig:

R0	keine Rotation
R90	einmal rotiert (90 Grad gegen den Uhrzeigersinn)
R180	zweimal rotiert (180 Grad gegen den Uhrzeigersinn)
R270	dreimal rotiert
MR0	an der y-Achse gespiegelt
MR90	einmal rotiert und an y-Achse gespiegelt
MR180	zweimal rotiert und an der y-Achse gespiegelt
MR270	dreimal rotiert und an y-Achse gespiegelt

Default: R0

Beispiel:

```
ADD DIL16 R90 (0 0);
```

platziert ein 16poliges DIL-Gehäuse, das um 90 Grad nach links gedreht ist, an den Koordinaten (0 0).

#### Fehlermeldungen

Soll ein Gate aus einem unvollständig definierten Device geholt werden, erscheint eine Fehlermeldung (siehe BOARD »Seite 49-Befehl). Dies läßt sich mit dem Befehl "SET »Seite 115 CHECK\_CONNECTS OFF;" verhindern. Vorsicht: Der BOARD-Befehl führt diese Prüfung auf alle Fälle durch. Sie abzuschalten ist also nur sinnvoll, wenn man keine Platine erzeugen will.

#### **Symbol in Device holen**

Bei der Device-Definition holt der ADD-Befehl ein vorher definiertes Symbol in das Device. Als Optionen sind zwei Parameter (Swaplevel und Addlevel) möglich, die in beliebiger Reihenfolge eingegeben werden können. Beide lassen sich mit dem CHANGE »Seite 51-Befehl voreinstellen und ändern. Auch der im ADD-Befehl angegebene Wert bleibt als Voreinstellung erhalten.

#### Swaplevel

Der Swaplevel ist eine Zahl im Bereich 0..255, wobei gilt:

0: Das Symbol (Gate) kann in der Schaltung nicht mit einem anderen vertauscht werden.

- 1..255 Das Symbol (Gate) kann in der Schaltung mit jedem anderen Symbol dieses Typs vertauscht werden, das denselben Swaplevel hat (auch zwischen verschiedenen Devices).

Default: 0

#### Addlevel

Für diesen Parameter gibt es folgende Möglichkeiten:

- |         |  |
|---------|--|
| Next    | Wenn ein Device mehr als ein Gate aufweist, werden in den Schaltplan der Reihe nach die Symbole mit Addlevel Next geholt.  |
| Must    | Wird ein beliebiges Symbol eines Device in den Schaltplan geholt, dann muß auch ein mit dem Addlevel Must definiertes Symbol im Schaltplan erscheinen. Dies geschieht automatisch. Es kann nicht gelöscht werden, bevor nicht alle anderen Symbole aus diesem Device gelöscht sind. Falls nur noch Must-Symbole aus einem Device vorhanden sind, löscht der DELETE-Befehl das ganze Device.  |
| Always  | Wie Must, allerdings kann ein Symbol mit Addlevel Always gelöscht und mit INVOKE »Seite 77 wieder in den Schaltplan geholt werden.   |
| Can     | Gibt es in einem Device Next-Gates, dann werden Can-Gates nur geholt, wenn sie explizit mit INVOKE angefordert werden. Ein Symbol mit Addlevel Can wird mit ADD nur dann in den Schaltplan geholt, wenn das Device nur Can- und Request-Gates enthält.   |
| Request | Diese Eigenschaft wird sinnvollerweise für Versorgungs-Symbole von Bausteinen verwendet. Request-Gates können nur explizit in die Schaltung geholt werden (INVOKE) und werden intern nicht mitgezählt. Das hat zur Folge, daß in Bausteinen mit nur einem Gatter und einem Versorgungsspannungs-Symbol der Gatter-Name nicht zum Bauteil-Namen hinzugefügt wird. Im Falle eines 7400 mit vier Gattern (plus Versorgungsspannung) heißen die einzelnen Gatter im Schaltplan beispielsweise IC1A, IC1B, IC1C und IC1D. Ein 68000 mit nur einem <i>Gate</i> , dem Prozessor-Symbol, heißt dagegen im Schaltplan z. B. IC1, da sein separates Spannungsversorgungs-Symbol als Gate nicht mitzählt. |

Beispiel:

```
ADD PWR 0 REQUEST •
```

holt das Symbol PWR (z. B. ein Versorgungssymbol) und definiert dafür den Swaplevel 0 (nicht tauschbar) und den Addlevel *Request*.

## ARC

### Funktion

Zeichnen von Kreisbögen.

### Syntax

```
ARC • • •  
ARC width • • •  
ARC CW width • • •  
ARC CCW width • • •
```

### Maus

Rechte Maustaste ändert Richtungssinn (CW, CCW).  
Mittlere Maustaste wechselt den aktiven Layer.

**Siehe auch** CHANGE »Seite 51, WIRE »Seite 131, CIRCLE »Seite 52

Mit dem ARC-Befehl zeichnet man Kreisbögen. Der erste und zweite Mausklick (linke Maustaste) definieren zwei gegenüberliegende Punkte auf dem Kreisumfang. Danach läßt sich mit der rechten Maustaste festlegen, ob der Bogen im Uhrzeigersinn oder im Gegenuhrzeigersinn dargestellt werden soll. Mit dem abschließenden Mausklick legt man den Winkel des Bogens fest.

Mit den Parametern CW (Clockwise) und CCW (Counterclockwise) kann man ebenfalls festlegen, ob der Bogen im Uhrzeigersinn oder im Gegenuhrzeigersinn dargestellt werden soll. Das ist insbesondere für Script-Dateien nützlich.

### Strichstärke

Der Parameter width gibt die Strichstärke an, er läßt sich mit dem Befehl

```
CHANGE WIDTH width;
```

voreinstellen oder verändern und ist identisch mit der aktuellen Strichstärke für Wires.

Kreisbögen mit einem Winkel von 0 oder 360 Grad oder einem Radius von 0 werden nicht akzeptiert.

Arcs in den Signal-Layern Top, Bottom, oder ROUTE2...15 gehören nicht zu Signalen. Der DRC meldet daher Fehler, wenn diese mit Wires, Pads usw. überlappen.

Beispiel für textuelle Eingabe:

```
GRID inch 1;  
ARC CW (0 1) (0 -1) (1 0);
```

erzeugt einen Viertelkreis im ersten Quadranten mit Mittelpunkt im Ursprung.

## ASSIGN

### Funktion

Tastenbelegung zuweisen.

### Syntax

```

ASSIGN
ASSIGN function_key befehl..;
ASSIGN function_key;

function_key = modifier+key
modifier      = jede Kombination aus S (Shift), C (Control) und A (Alt)
key           = F1..F12, A-Z, 0-9, BS (Backspace)

```

**Siehe auch** SCRIPT »Seite 114, Tastatur und Maus »Seite 30

Mit dem ASSIGN-Befehl kann man die Funktionstasten F1 bis F12, die Buchstabentasten A bis Z, die (oberen) Zifferntasten 0 bis 9 und die Backspace-Taste (jeweils auch in Kombination mit Shift, Ctrl und/oder Alt) mit einzelnen oder mehreren Befehlen belegen.

Der ASSIGN-Befehl ohne Parameter listet die aktuelle Tastenbelegung in einem Dialog auf, in dem die Einstellungen auch verändert werden können.

Die beim Betätigen der Taste auszuführende Befehlssequenz sollte man der Klarheit wegen in Hochkommas einschließen.

Soll als key eine der Tasten A-Z oder 0-9 verwendet werden, so muß der modifier mindestens A oder C enthalten.

Bitte beachten Sie, daß eine eventuell auf einer Taste liegende Betriebssystem-Funktion durch den ASSIGN-Befehl überschrieben wird (je nach verwendetem Betriebssystem kann es sein, daß bestimmte Tastenkombinationen nicht mit dem ASSIGN-Befehl überschrieben werden können).

Falls Sie eine Buchstabentaste zusammen mit dem Modifier A belegen (zum Beispiel A+F), so steht ein eventueller Hotkey im Pulldown-Menü nicht mehr zur Verfügung.

Um eine Tastenbelegung wieder zu entfernen geben Sie ASSIGN nur mit dem function\_key Code (ohne Befehl) ein.

### Beispiele

```

ASSIGN F7 'change layer top; route';
ASS A+F7 'cha lay to; rou';
ASSIGN C+F10 menu add mov rou '';'' edit;
ASSIGN CA+R 'route';

```

Die beiden ersten Eingaben bewirken das gleiche, da EAGLE nicht nur bei Befehlen, sondern auch bei den Parametern Abkürzungen zuläßt, solange sie eindeutig sind.

Beachten Sie, daß hier z. B. der Befehl "CHANGE layer top" mit Strichpunkt abgeschlossen ist und der ROUTE-Befehl nicht. Im ersten Fall enthält der Befehl nämlich alle Parameterangaben, während im zweiten Fall noch Koordinatenangaben fehlen (die dann sinnvollerweise mit der Maus eingegeben werden). Der ROUTE-Befehl darf also nicht mit Strichpunkt abgeschlossen werden.

### Befehlsmenü einstellen

Will man mit ASSIGN eine Taste so belegen, daß sie ein neues Menü einstellt, dann muß das im MENU-Befehl enthaltene Trennzeichen (Strichpunkt) von jeweils drei Hochkommas eingeschlossen sein, wie im dritten Beispiel zu sehen.

### Voreinstellung der Tastenbelegung

F1	HELP	Help-Funktion
Alt+F2	WINDOW FIT	Zeichnung formatfüllend darst.
F2	WINDOW;	Bildschirminhalt auffrischen
F3	WINDOW 2	In das Bild hineinzoomen, Fakt. 2
F4	WINDOW 0.5	Herauszoomen um Faktor 2
F5	WINDOW (@)	Neues Zentrum an Cursor-Pos.

F6	GRID;	Raster ein-/ausblenden
F7	MOVE	MOVE-Befehl
F8	SPLIT	SPLIT-Befehl
F9	UNDO	Befehl(e) zurücknehmen
F10	REDO	Befehl erneut ausführen
Alt+BS	UNDO	Befehl(e) zurücknehmen
Shift+Alt+BS	REDO	Befehl erneut ausführen

## AUTO

### Funktion

Autorouter »Seite 157 aktivieren.

### Syntax

```
AUTO;  
AUTO signal_name..;  
AUTO ! signal_name..;  
AUTO •..;
```

**Siehe auch** SIGNAL »Seite 118, ROUTE »Seite 112, WIRE »Seite 131, RATSNEST »Seite 104, SET »Seite 115

Der AUTO-Befehl aktiviert den integrierten Autorouter. Werden Signalnamen angegeben oder Signale mit der Maus selektiert, werden nur diese Signale verlegt. Ohne weitere Parameter verlegt der Autorouter alle Signale. Das Zeichen "!" gibt an, daß alle Signale außer den angegebenen zu routen sind. Es muß vor allen Signalnamen stehen und darf nur einmal vorkommen.

### Beispiel

```
AUTO ! GND VCC;
```

Der abschließende Strichpunkt ist in jedem Fall erforderlich, es sei denn, der Autorouter wird vom Menü aus mit "Start" gestartet. Die Aktivitäten des Autorouters können Sie am Bildschirm mitverfolgen.

Das Menü des Autorouter-Befehls erscheint nach Eingabe von AUTO ohne abschließenden Strichpunkt.

### Polygone

Beim Starten des Autorouters werden alle Polygone »Seite 99 neu freigerechnet, außer denjenigen im Urzustand, deren Signal vom Routen ausgenommen wurde (AUTO ! signal\_name..;).

### Protokolldatei

Informationen zum Routing-Vorgang enthält die Datei name.pro, die ebenfalls automatisch angelegt wird.

### Routing-Fläche

Der Autorouter legt ein umschließendes Rechteck um alle Wires im Dimension-Layer und nimmt die Größe dieses Rechtecks als maximale Route-Fläche. Wires im Dimension-Layer stellen für den Autorouter Sperrlinien dar. Das heißt, mit geschlossenen Polygonzügen kann man den Route-Bereich begrenzen.

### Signale

Als Signale erkennt der Autorouter Wires, Polygone und die mit SIGNAL definierten Signale aus den Layern Top, Bottom und Route2...15.

Achtung: Freigerechnete Polygone führen dazu, daß der Autorouter innerhalb dieser Flächen keine Vias mehr setzen kann. Polygone, mit denen etwa Masseflächen realisiert werden, sollten deshalb nach dem Routen aller Signale außer Masse eingefügt werden.

### Sperrflächen

Rechtecke, Polygone und Kreise aus den Layern bRestrict, tRestrict und vRestrict werden als Sperrflächen für Löt- und Bestückungsseite sowie für Durchkontaktierungen (Vias) behandelt.

Falls der Autorouter keine Signale in einem Layer verlegen soll, ist in das Feld für die Vorzugsrichtung 0 einzutragen.

### Wahl des Rasters

Bei der Wahl des Rasters ist zu beachten, daß möglichst keine Pads für den Router "unsichtbar" werden. Das heißt, jedes Pad soll mindestens einen Routing-Rasterpunkt belegen, sonst kann es passieren, daß der Autorouter eine Verbindung nicht legen kann, die ansonsten ohne Probleme zu verlegen wäre - einfach weil er das entsprechende Pad nicht auf seinem Raster darstellen kann.



## BOARD

### Funktion

Erzeugt eine Layout-Datei aus einer Schaltung.

### Syntax

BOARD

**Siehe auch** EDIT »Seite 64

Der BOARD-Befehl erzeugt eine Layout-Datei aus einer Schaltung.

Wenn die Platine bereits existiert, wird sie in ein Layout-Editor-Fenster geladen.

Wenn die Platine nicht existiert, werden Sie gefragt, ob Sie eine neue Datei anlegen wollen.

Der BOARD-Befehl überschreibt niemals eine existierende Platinen-Datei. Wenn eine Datei mit diesem Namen existiert, muß sie erst mit REMOVE »Seite 107 gelöscht werden, bevor der BOARD-Befehl sie neu anlegen kann.

### Platine aus Schaltplan erzeugen

Wird eine Platine zum erstenmal geladen, prüft das Programm, ob im selben Verzeichnis ein Schaltplan mit demselben Namen existiert. Wenn ja, fragt das Programm, ob aus dem Schaltplan die Platine erstellt werden soll. Wenn eine Schaltung geladen ist, können Sie die zugehörige Platine erzeugen, indem Sie

```
edit .brd
```

in die Kommandozeile des Editor-Fensters eintippen.

Alle relevanten Daten des Schematic-Files (name.sch) werden dann in ein Board-File (name.brd) konvertiert. Das neue Board wird automatisch mit einer Größe von 160x100mm (Light Edition »Seite 366: 100x80mm) angelegt, wobei die Umrisse so gelegt sind, daß das Board mittig im 50mil Raster liegt. Alle Packages mit den im Schaltplan definierten Verbindungen sind links neben der leeren Platine plaziert. Supply-Pins sind bereits verbunden (siehe PIN »Seite 95-Befehl).

Falls Sie andere als die standardmäßig angelegten Platinenumrisse benötigen, brauchen Sie einfach nur die entsprechenden Linien zu löschen und die gewünschten Umrisse mit dem WIRE »Seite 131-Befehl in den Layer *Dimension* zu zeichnen. Wählen Sie dazu bitte eine Strichbreite von 0, da es sich hierbei nur um Hilfslinien handelt.

Ein Board-File kann nicht angelegt werden:

- Wenn sich Gates in der Schaltung befinden, die aus einem Device ohne Package stammen (Fehlermeldung: "device name has no package). Ausnahme: wenn nur Pins mit Direction "Sup" enthalten sind (Supply-Symbole).
- Wenn sich Gates in der Schaltung befinden, die aus einem Device stammen, für das nicht allen Pins ein Gehäuse-Pad zugeordnet ist (Fehlermeldung: "device name has no connects"). Ausnahme: Device ohne Pins (z. B. Zeichnungsrahmen).

## BUS

### Funktion

Zeichnen von Bussen im Schaltplan.

### Syntax

```
BUS • . .  
BUS bus_name • . .
```

**Siehe auch** NET »Seite 88, NAME »Seite 87, SET »Seite 115

Mit dem Befehl BUS zeichnet man Busse in den Bus-Layer eines Schaltplans. Der Busname hat die Form

SYNONYM:Teilbus, Teilbus, . .

wobei SYNONYM ein beliebiger Name sein darf. Teilbus ist entweder ein Netzname oder ein Busname mit Index in der Form:

Name [LowestIndex..HighestIndex]

Folgende Bedingungen müssen erfüllt sein:

0 <= LowestIndex <= HighestIndex <= 511

Wird ein Teilbus mit einem Index verwendet, darf der Name nicht mit einer Zahl enden, da sonst nicht klar wäre, welche Zahl zum Namen und welche zum Index gehörten.

Wenn der Bus auf einem anderen Bus abgesetzt wird, endet die Linie an dieser Stelle. Dieses Verhalten kann über "SET AUTO\_END\_NET OFF;" oder durch Deselektieren der Option "Auto end net and bus" unter "Options/Set/Misc" abgeschaltet werden.

### Beispiele für Busnamen

```
A[0..15]  
RESET  
DB[0..7], A[3..4]  
ATBUS:A[0..31], B[0..31], RESET, CLOCK, IOSEL[0..1]
```

Gibt man keinen Busnamen an, wird ein Name der Form B\$1 automatisch vergeben. Dieser Name lässt sich zu jeder Zeit mit dem NAME-Befehl verändern. Die Breite der Linien, die einen Bus darstellen, lässt sich z. B. mit

```
SET BUS_WIRE_WIDTH 40;
```

auf 40 Mil einstellen (Default: 30 Mil).

## CHANGE

### Funktion

Ändern von Parametern.

### Syntax

CHANGE option • •..

### Maus

Rechte Taste ändert Parameter einer vorher definierten Gruppe.

Der CHANGE-Befehl dient generell dazu, Eigenschaften von Objekten zu ändern oder voreinzustellen. Objekte, die schon in der Zeichnung vorhanden sind, werden einfach der Reihe nach mit der Maus selektiert, nachdem der Befehl und der entsprechende Parameter vorher eingegeben (bzw. aus einem Menü mit der Maus ausgewählt) wurden.

Nachfolgend platzierte Objekte erhalten die mit CHANGE geänderten Eigenschaften. Damit ist es möglich, mit diesem Befehl Parameter voreinzustellen.

Alle Zahlenangaben beziehen sich auf die aktuelle Maßeinheit (siehe GRID).

### Gruppe ändern mit rechter Maustaste

Will man den CHANGE-Befehl auf eine Gruppe ausführen, definiert man zuerst die Gruppe mit dem GROUP »Seite 73-Befehl, dann gibt man den CHANGE-Befehl mit den entsprechenden Parametern ein und klickt die Gruppe anschließend mit der rechten Maustaste an.

### Möglichkeiten des CHANGE-Befehls

Layer wechseln	CHANGE LAYER name   number
Text ändern	CHANGE TEXT
Texthöhe	CHANGE SIZE value
Textstärke	CHANGE RATIO ratio
Text Font	CHANGE FONT VECTOR   PROPORTIONAL   FIXED
Wire-Breite	CHANGE WIDTH value
Wire-Liniertyp	CHANGE STYLE value
Pad-Form	CHANGE SHAPE SQUARE   ROUND   OCTAGON   XLONGOCT   YLONGOCT
Pad-/Via-Durchmesser	CHANGE DIAMETER diameter
Pad-/Via/Hole-Bohrd.	CHANGE DRILL value
Smd-Maße	CHANGE SMD width height
Pin-Parameter	CHANGE DIRECTION NC   IN   OUT   I/O   OC   HIZ   SUP   PAS   PWR   SUP
	CHANGE FUNCTION NONE   DOT   CLK   DOTCLK
	CHANGE LENGTH POINT   SHORT   MIDDLE   LONG
	CHANGE VISIBLE BOTH   PAD   PIN   OFF
	CHANGE SWAPLEVEL number
Polygon-Parameter	CHANGE THERMALS ON   OFF
	CHANGE ORPHANS ON   OFF
	CHANGE ISOLATE distance
	CHANGE POUR SOLID   HATCH
	CHANGE RANK value
	CHANGE SPACING distance
Gate-Parameter	CHANGE SWAPLEVEL number
	CHANGE ADDLEVEL NEXT   MUST   ALWAYS   CAN   REQUEST
Netzklasse	CHANGE CLASS number   name
Package-Variante	CHANGE PACKAGE name [variant]   'variant' [name]
Technologie	CHANGE TECHNOLOGY name [variant]   'variant' [name]

## CIRCLE

### Funktion

Kreis in eine Zeichnung einfügen.

### Syntax

```
CIRCLE • •.. [Kreismitte, Radius]
CIRCLE width • •..
```

### Maus

Mittlere Maustaste wechselt den aktiven Layer.

**Siehe auch** CHANGE »Seite 51, WIRE »Seite 131

Mit dem CIRCLE-Befehl zeichnet man Kreise in den aktiven Layer.

Der CIRCLE-Befehl in den Layern tRestrict, bRestrict und vRestrict dient zum Anlegen von Sperrflächen. Dabei sollte eine Linienstärke (width) von 0 gewählt werden.

Der Parameter "width" gibt die Strichstärke des Kreises an. Er entspricht demselben Parameter des WIRE-Befehls und kann mit dem Befehl

```
CHANGE WIDTH breite;
```

geändert bzw. voreingestellt werden. Dabei ist *breite* der gewünschte Wert in der gegenwärtigen Einheit.

Kreise mit Strichstärke 0 werden gefüllt dargestellt.

### Beispiel für Parameterangabe in Textform

```
GRID inch 1;
CIRCLE (0 0) (1 0);
```

erzeugt einen Kreis mit einem Radius von 1 Zoll um den Ursprung (0 0). +

## CLASS

### Funktion

Definieren und Wählen von Netzklassen.

### Syntax

```
CLASS  
CLASS number|name  
CLASS number name [ width [ clearance [ drill ] ] ]
```

**Siehe auch** Design Rules »Seite 159, NET »Seite 88, SIGNAL »Seite 118, CHANGE »Seite 51

Der CLASS-Befehl wird zur Definition von Netzklassen verwendet.

Ohne Angabe von Parametern, wird ein Dialog geöffnet, der es erlaubt Netzklassen festzulegen.

Wird nur `number` oder `name` angegeben, wählt man die Netzkategorie mit der entsprechenden Nummer bzw. dem Namen für die folgenden NET- und SIGNAL-Befehle vor.

Wird beides `number` und `name` angegeben, werden dieser Netzkategorie die folgenden Werte für die Parameter (`width`, `clearance`, `drill`) zugeordnet. Diese Netzkategorie ist gleichzeitig für die folgenden NET- und SIGNAL-Befehle vorgewählt. Werden nach `name` nicht alle drei Parameter angegeben, gelten die Werte der Reihe nach für `width`, `clearance` bzw. `drill`. Soll beispielsweise nur `drill` geändert werden, müssen also auch die Parameter für `width` und `clearance` angegeben werden.

Wird `number` negativ gewählt, löscht man die Netzkategorie mit dem Absolutwert der angegebenen `number`. Die Default-Netzkategorie 0 kann man nicht löschen.

Bei den Namen der Netzkategorien wird nicht zwischen Groß- und Kleinbuchstaben unterschieden. SUPPLY hat z. B. dieselbe Bedeutung wie Supply oder SuPpLy.

Werden mehrere Netzkategorien in einer Zeichnung verwendet, brauchen der Design Rule Check und der Autorouter länger um ihre Arbeit zu erledigen. Daher ist es sinnvoll nur so viele Netzkategorien wie unbedingt nötig zu verwenden (die Anzahl der tatsächlich benutzten Netzkategorien ist ausschlaggebend, nicht die Anzahl der definierten Netzkategorien).

Um Probleme bei CUT-und-PASTE-Aktionen zwischen verschiedenen Zeichnungen zu vermeiden, ist es sinnvoll den unterschiedlichen Netzkategorien in verschiedenen Zeichnungen dieselben Nummern zu geben.

Der Autorouter verlegt die Signale in der Reihenfolge der benötigten Breite (`width` + `clearance`), beginnend mit denen, die am meisten Platz benötigen. Der Bus-Router verlegt nur Signale mit Netzkategorie 0.

Für bestehende Netze/Signale kann CLASS mit dem CHANGE-Befehl geändert werden. Änderungen der Netzkategorien mit dem CLASS-Befehl werden nicht im UNDO/REDO-Puffer gespeichert.

## CLOSE

### Funktion

Schließt ein Editor-Fenster.

### Syntax

CLOSE

**Siehe auch** OPEN »Seite 89, EDIT »Seite 64, WRITE »Seite 133, SCRIPT »Seite 114

Der CLOSE-Befehl schließt ein Editor-Fenster. Wenn die geladene Datei verändert worden ist, werden Sie gefragt, ob sie abgespeichert werden soll.

Dieser Befehl ist in erster Linie für Script-Dateien erforderlich.

## CONNECT

### Funktion

Zuordnung von Pins und Pads.

### Syntax

```
CONNECT
CONNECT symbol_name.pin_name pad_name..
CONNECT pin_name pad_name..
```

**Siehe auch** PREFIX »Seite 101, OPEN »Seite 89, CLOSE »Seite 54, SCRIPT »Seite 114

Dieser Befehl wird im Device-Editier-Modus angewendet. Er dient dazu, den Pins des Schaltplan-Symbols (Device), das gegenwärtig bearbeitet wird, die entsprechenden Pads des zugehörigen Gehäuses zuzuweisen. Zuvor muß mit dem PACKAGE-Befehl festgelegt worden sein, welches Package für das Device verwendet werden soll.

Wird der CONNECT-Befehl ohne Parameter aufgerufen, so erscheint ein Dialog in dem die Pad/Pin-Zuweisungen interaktiv definiert werden können.

### Device hat ein Symbol

Ist im Device nur ein Symbol vorhanden, kann der Parameter symbol\_name entfallen, z.B.

```
CONNECT gnd 1 rdy 2 phi1 3 irq\ 4 nc1 5 ...
```

### Device hat mehrere Symbole

Sind im Device mehrere Symbole vorhanden, sind als Parameter symbol\_name und pin\_name (mit Punkt als Trennzeichen) sowie pad\_name anzugeben, z.B.:

```
CONNECT A.I1      1    A.I2      2    A.O      3;
CONNECT B.I1      4    B.I2      5    B.O      6;
CONNECT C.I1     13    C.I2     12    C.O     11;
CONNECT D.I1     10    D.I2      9    D.O      8;
CONNECT PWR.GND    7;
CONNECT PWR.VCC   14;
```

In diesem Fall werden die Anschlüsse der vier NAND-Gatter eines 7400 zugewiesen. Das Device enthält fünf Symbole mit den Bezeichnungen A, B, C, D, PWR. Die Eingänge der Gatter heißen im Schaltplan I1 und I2; der Ausgang heißt O.

Der CONNECT-Befehl kann beliebig oft ausgeführt werden. Er kann alle Pin/Pad-Zuweisungen enthalten oder nur einen Teil davon. Jeder neue CONNECT-Befehl überschreibt die bisherigen Definitionen.

### Gate- oder Pin-Namen mit Punkten

Soll ein Gate- oder ein Pin-Name einen Punkt enthalten, kann dieser ohne weiteres verwendet werden. Es sind keine besonderen Zeichen (Esc-Character oder Anführungszeichen) nötig.

### Beispiel

```
ed 6502.dev;
prefix 'IC';
package dil40;
connect gnd 1 rdy 2 phi1 3 irq\ 4 nc1 5 nmi\ 6 \
      sync 7 vcc 8 a0 9 a1 10 a2 11 a3 12 a4 \
      13 a5 14 a6 15 a7 16 a8 17 a9 18 a10 19 \
      a11 20 p$0 21 a12 22 a13 23 a14 24 a15 \
      25 d7 26 d6 27 d5 28 d4 29 d3 30 d2 31 \
      d1 32 d0 33 r/w 34 nc2 35 nc3 36 phi0 37 \
      so 38 phi2 39 res\ 40;
```

Hier sorgt das Zeichen "\" am Zeilenende dafür, daß am Beginn der nächsten Zeile keine Zeichenfolge mit einem Befehl verwechselt werden kann. Als Bestandteil eines Pin-Namens drückt das Zeichen "\"" im allgemeinen aus, daß es sich um ein invertiertes Signal handelt (z.B. "irq").

Eine Verwechslung mit Befehlen kann man auch vermeiden, indem man Parameter in Hochkommas einschließt.

## COPY

### Funktion

Kopieren von Objekten und Elementen.

### Syntax

COPY • • . .

**Siehe auch** GROUP »Seite 73, CUT »Seite 57, PASTE »Seite 94, POLYGON »Seite 99

Mit dem COPY-Befehl lassen sich Objekte und Packages selektieren und anschließend an eine andere Stelle derselben Zeichnung kopieren. Beim Kopieren von Elementen generiert EAGLE einen neuen Namen und behält den Wert (Value) bei. Beim Kopieren von Signalen (Wires), Bussen und Netzen wird der Name beibehalten. In allen anderen Fällen wird ein neuer Name generiert.

### Leitungen kopieren

Kopiert man Wires oder Polygone, die zu einem Signal gehören, dann gehört die Kopie zum selben Signal. Bitte beachten Sie, daß aus diesem Grund der DRC keinen Fehler feststellt, wenn z. B. zwei Wires mit COPY überlappend platziert werden.

### Elemente duplizieren

Der COPY-Befehl ermöglicht es Ihnen, Bauelemente, die sich schon in der Zeichnung befinden, zu platzieren, ohne daß Sie dafür eine Bibliothek brauchen.



## CUT

### Funktion

Gruppe in den Paste-Puffer laden.

### Syntax

CUT ;  
CUT •

**Siehe auch** PASTE »Seite 94, GROUP »Seite 73

Teile einer Zeichnung (z. B. auch eine ganze Platine) lassen sich mit Hilfe der Befehle GROUP, CUT und PASTE in andere Zeichnungen übernehmen.

Zuerst definiert man eine Gruppe (GROUP). Dann gibt man den Befehl CUT, der den Paste-Puffer mit den selektierten Elementen und Objekten lädt. Jetzt kann man die Platine oder die Bibliothek wechseln und mit PASTE den Pufferinhalt in die neue Zeichnung kopieren. Falls erforderlich, werden neue Namen generiert. Der Pufferinhalt bleibt erhalten und kann mit weiteren PASTE-Befehlen erneut kopiert werden.

### Referenzpunkt

Wird beim CUT-Befehl mit der Maus ein Punkt angeklickt, dann befindet sich beim PASTE-Befehl der Mauscursor genau an dieser Stelle der Gruppe (genauer: am nächstgelegenen Rasterpunkt). Ansonsten befindet sich der Mauscursor bei PASTE etwa in der Mitte der Gruppe.

### Hinweis

Im Gegensatz zu anderen (Windows-) Programmen entfernt der CUT-Befehl von EAGLE die markierte Gruppe nicht physikalisch aus der Zeichnung, sondern kopiert die Gruppe lediglich in den Paste-Buffer.

## DELETE

### Funktion

Löschen von Objekten.

### Syntax

DELETE • . .  
DELETE SIGNALS

### Maus

Rechte Maustaste löscht Gruppe.

### Tasten

Shift löscht das übergeordnete Objekt in der Hierarchie (siehe Anmerkung).

**Siehe auch** RIPUP »Seite 110, DRC »Seite 63, GROUP »Seite 73

Der DELETE-Befehl löscht das Objekt aus der Zeichnung, das dem Cursor bzw. dem angegebenen Koordinatenpunkt am nächsten liegt.

Mit der rechten Maustaste wird eine zuvor mit GROUP »Seite 73 definierte Gruppe gelöscht.

Nach dem Löschen einer Gruppe können Luftlinien, die durch das Entfernen von Bauelementen neu entstanden sind, "übrigbleiben", da diese nicht in der ursprünglich definierten Gruppe enthalten waren. In solchen Fällen sollte mit RATSNEST »Seite 104 eine Neuberechnung der Luftlinien durchgeführt werden.

Bei aktiver Forward&Back-Annotation »Seite 359 können im Board keine Wires oder Vias aus Signalen gelöscht werden, die an Bauelemente angeschlossen sind. Ebenso können keine Bauelemente gelöscht werden, an die Signale angeschlossen sind. Änderungen dieser Art müssen im Schaltplan vorgenommen werden.

Um eine bereits verlegte Verbindung im Board wieder in eine Luftlinie zu verwandeln, verwenden Sie den RIPUP »Seite 110-Befehl.

Der DELETE-Befehl wirkt nur auf sichtbare Layer (siehe DISPLAY-Befehl).

### Löschen von Polygon-Kanten

Bei Polygonen löscht der DELETE-Befehl jeweils eine Ecke. Sind nur noch drei Ecken vorhanden, wird das ganze Polygon gelöscht.

### Löschen von Bauelementen

Bauelemente auf der Top-Seite lassen sich nur löschen, wenn der tOrigins-Layer sichtbar ist und wenn (bei aktiver Forward&Back-Annotation »Seite 359) keine Signale mit dem Element verbunden sind (siehe auch REPLACE »Seite 109). Entsprechend lassen sich Bauelemente auf der Bottom-Seite nur löschen, wenn der bOrigins-Layer eingeblendet ist.

### Löschen von Junction-Punkten, Netzen und Bussen

Es gelten folgende Regeln:

- Zerfällt ein Bus-Segment in zwei Teilsegmente, so behalten beide den ursprünglichen Namen.
- Zerfällt ein Netz-Segment in zwei Teilsegmente, so behält das größere der beiden Teilsegmente den ursprünglichen Namen, während das kleinere einen (neuen) generierten Namen erhält.
- Labels gehören nach der Trennung zum jeweils nächstgelegenen Segment.
- Wird ein Junction-Punkt gelöscht, so wird das Netz an dieser Stelle aufgetrennt. Prüfen Sie im Zweifelsfall mit SHOW nach, welches Segment welchen Namen erhalten hat; falls die umgekehrte Namensgebung gewünscht wird, lässt sie sich mit dem NAME-Befehl leicht erreichen.

### Löschen von Supply-Symbolen

Wird das letzte Supply-Symbol einer gegebenen Versorgungsspannung von einem Netzsegment, das den Namen dieses Supply-Symbols trägt, gelöscht, erhält dieses Netzsegment einen neuen, automatisch generierten Namen (sofern kein anderes Supply-Symbol mehr diesem Segment zugeordnet ist) oder den Namen eines noch verbleibenden anderen Supply-Symbols.

### Löschen von Signalen

Selektiert man mit dem DELETE-Befehl Wires oder Vias, die zu einem Signal gehören, dann sind drei Fälle zu unterscheiden:

- Das Signal wurde aufgetrennt und zerfällt in zwei Teile. EAGLE vergibt dann intern für das kleinere Teilsignal einen neuen Namen und behält den bisherigen Namen für das größere Teilsignal bei.
- Das Signal wurde von einem Ende her gelöscht. Das verbleibende Teilsignal behält seinen bisherigen Namen.
- Das Signal bestand nur noch aus einer Verbindung. Es ist dann ganz gelöscht, und der bisherige Name existiert nicht mehr.

Werden Wires oder Vias eines Signals gelöscht, das Polygone enthält, dann bleiben alle Polygone in dem Teil des Signals, der den ursprünglichen Namen behält (normalerweise der "größere" Teil).

### Löschen von übergeordneten Objekten

Wird ein Objekt mit gedrückter *Shift*-Taste angeklickt, so wird das in der Hierarchie nächsthöhere Objekt gelöscht. Im Einzelnen gilt dies für folgende Objekte:

Gatter	Löscht das gesamte Bauteil, in dem sich dieses Gatter befindet (auch wenn die Gatter auf mehrere Schaltplan-Seiten verteilt sind). Bei aktiver Forward&Backannotation werden die Leiterbahnen, die an dem Bauteil im Board angeschlossen sind, nicht in Luftlinien umgewandelt (was beim Löschen einzelner Gatter geschehen würde)
Polygon Wire	Löscht das gesamte Polygon
Net/Bus Wire	Löscht das gesamte Netz- bzw. Bus-Segment

### Alle Signale löschen

DELETE SIGNALS kann dazu verwendet werden, um alle Signale aus einer Platine zu entfernen, um z. B. eine neue oder geänderte Netzliste einzulesen. Es werden nur solche Signale entfernt, die an Pads angeschlossen sind. Andere (wie z.B. Eckwinkel im Top- und Bottom-Layer, die intern auch als Signal behandelt werden) bleiben unberührt.

Der DRC erzeugt unter Umständen Fehlerpolygone, die man nicht mit DELETE löschen kann, sondern mit DRC Clear.

## DESCRIPTION

### Funktion

Definiert die Beschreibung eines Devices, Packages oder einer Bibliothek.

### Syntax

```
DESCRIPTION  
DESCRIPTION description_string;
```

**Siehe auch** CONNECT »Seite 55, PACKAGE »Seite 91, VALUE »Seite 128

Mit diesem Befehl erzeugt oder editiert man die Beschreibungstexte eines Devices, eines Packages oder einer Bibliothek im Bibliotheks-Editor.

Description\_string kann Rich-Text »Seite 356-Steuerzeichen enthalten.

Die erste nicht leere Zeile von description\_string wird als Kurzbeschreibung (*headline*) im Control Panel angezeigt.

Der DESCRIPTION-Befehl ohne Angabe von Parametern öffnet einen Dialog, der das Editieren des Textes erlaubt. Der obere Teil des Dialogs zeigt den formatierten Text, sofern er Steuerzeichen des Rich-Text »Seite 356-Formats enthält, während der untere Teil für die Eingabe des Textes genutzt wird. Ganz oben im Dialog sehen Sie das Feld *headline*. In diesem wird die Kurzbeschreibung ohne Rich-Text-Steuerzeichen angezeigt.

Die Beschreibung der Bibliothek kann von der Kommandozeile aus nur editiert werden, wenn die Bibliothek neu geöffnet und bisher kein Device, Symbol oder Package editiert wurde. Sie kann aber jederzeit über das Pulldown-Menü "Library/Description..." verändert werden.

Die Beschreibung eines Devices oder eines Packages kann immer über die Kommandozeile oder das Pulldown-Menü "Edit/Description..." verändert werden.

### Beispiel

```
DESCRIPTION '<b>Quad NAND</b><p>\nFour NAND gates with 2 inputs each.';
```

Das Resultat sieht so aus:

### Quad NAND

Four NAND gates with 2 inputs each.

## DISPLAY

### Funktion

Auswählen der sichtbaren Layer.

### Syntax

```
DISPLAY
DISPLAY [option] layer_number..
DISPLAY [option] layer_name..
```

Gültige options sind: ALL und NONE, ? und ??

**Siehe auch** LAYER »Seite 80, PRINT »Seite 102

Mit dem DISPLAY-Befehl wählt man diejenigen Layer aus, die auf dem Bildschirm sichtbar sein sollen. Dabei darf als Parameter die Layer-Nummer oder der Layer-Name angegeben werden (auch gemischt). Gibt man den Parameter ALL an, werden alle Layer sichtbar. Mit dem Parameter NONE kann man alle Layer ausblenden. Beispiel:

```
DISPLAY NONE BOTTOM
```

Nach diesem Befehl ist nur der Bottom-Layer sichtbar.

Stellt man dem Layer-Namen oder der Layer-Nummer ein Minuszeichen voran, wird er ausgeblendet. Beispiel:

```
DISPLAY BOTTOM -TOP -3
```

In diesem Fall wird Bottom eingeblendet, und Top sowie der Layer mit der Nummer 3 werden ausgeblendet.

Zur Vereinfachung werden mit tPlace bzw. bPlace automatisch t/bNames, t/bValues, t/bOrigins und t/bDocu ein- und ausgeblendet. Für Schaltpläne gilt entsprechend: Symbols blendet Names und Values mit ein und aus.

Manche Befehle (PAD, SMD, SIGNAL, ROUTE) aktivieren automatisch bestimmte Layer.

Wird der DISPLAY-Befehl ohne Parameter aufgerufen, so erscheint ein Dialog in dem alle Einstellungen vorgenommen werden können.

### Nicht definierte Layer

Die Optionen '?' und '??' werden verwendet, zu kontrollieren, was passiert, wenn ein nicht definierter Layer über DISPLAY aufgerufen wird. Jeder nicht definierte Layer, der einem '?' folgt, erzeugt eine Warnmeldung, die der Benutzer entweder bestätigen kann oder mit Cancel den DISPLAY-Befehl abbrechen kann. Nicht definierte Layer, die nach '??' folgen, werden kommentarlos ignoriert. Diese Optionen sind besonders beim Schreiben von Script-Dateien nützlich, die in der Lage sein sollen beliebige Dateien zu modifizieren, unabhängig davon welche Layer in der Datei letztendlich definiert wurden.

```
DISPLAY TOP BOTTOM ? MYLAYER1 MYLAYER2 ?? OTHER WHATEVER
```

Im Beispiel oben müssen die Layer TOP und BOTTOM definiert sein. Ansonsten wird der Befehl mit einer Fehlermeldung abgebrochen. Das Fehlen von MYLAYER1 und MYLAYER2 wird in einer Warnung angezeigt. Die Aktion kann vom Benutzer abgebrochen werden. Die Layer OTHER und WHATEVER werden angezeigt, wenn vorhanden, ansonsten werden sie einfach ignoriert.

Die Optionen '?' und '??' dürfen in einer Befehlssequenz beliebig oft verwendet werden.

### Pads und Vias

Verwendet man Pads und Vias mit unterschiedlichen Formen in den einzelnen Layern, werden alle Formen, die in den sichtbaren (über DISPLAY aktivierten) Signal-Layern verwendet werden, übereinander dargestellt.

Wählt man für den Layer 17 (Pads) bzw. 18 (Vias) die Farbe 0 (das entspricht der Hintergrundfarbe schwarz oder weiß), werden Pads und Vias in der Farbe und dem Füllmuster des jeweiligen Signal-Layers gezeichnet. Ist kein Signal-Layer eingeblendet, werden auch keine Pads oder Vias dargestellt.

Wählt man für den Layer 17 (Pads) bzw. 18 (Vias) eine andere Farbe und es ist kein Signal-Layer sichtbar, werden Pads und Vias in der Form des obersten und untersten Signal-Layers dargestellt.

Das gilt auch für Ausdrücke mit PRINT.

**Objekte selektieren**

Um Objekte und Elemente selektieren zu können (z. B. mit MOVE, DELETE) muß der entsprechende Layer sichtbar sein. Elemente in Platinen lassen sich nur selektieren, wenn der tOrigins-Layer (bzw. bOrigins bei gespiegelten Elementen) sichtbar ist!

Vermeiden Sie die Layer-Namen ALL und NONE, sowie Namen, die mit einem Minuszeichen beginnen.

## DRC

### Funktion

Design Rule Check (Platine prüfen)

### Syntax

DRC

DRC • • ;

**Siehe auch** Design Rules »Seite 159, CLASS »Seite 53, SET »Seite 115, ERC »Seite 65, ERRORS »Seite 67

Der DRC-Befehl prüft das Layout gegenüber einem gültigen Satz von Design Rules »Seite 159.

Es werden nur die aktiven Signallayer geprüft. Darum ist es notwendig alle benutzten Layer für den DRC zu aktivieren, spätestens bei der letzten Überprüfung vor der Platinenfertigung.

Die gefundenen Fehler werden als Fehlerpolygone in den zugehörigen Layern dargestellt und können mit dem ERRORS »Seite 67-Befehl bearbeitet werden.

Geben Sie den DRC-Befehl ohne weitere Parameter an, öffnet sich der Design-Rules-Dialog. Von hier aus kann man die Design Rules einstellen und die Prüfung starten.

Wird der DRC-Befehl mit zwei Koordinatenpaaren angegeben (oder im DRC-Dialog der Button Select angeklickt), prüft der DRC nur innerhalb des angegebenen Rechtecks. Es werden nur die Fehler angezeigt, die innerhalb des Rechtecks liegen.

Um alle Fehlerpolygone zu löschen verwenden Sie

ERRORS CLEAR

### SET-Befehle, die den DRC beeinflussen

Der SET-Befehl kann dazu verwendet werden, um das Verhalten des DRC-Befehls zu verändern:

SET DRC\_FILL fill\_name;

Legt das für die DRC-Fehlerpolygone verwendete Füllmuster fest. Default: LtSlash.

## EDIT

### Funktion

Lade-Befehl.

### Syntax

```
EDIT name
EDIT name.ext
EDIT .ext
```

**Siehe auch** OPEN »Seite 89, CLOSE »Seite 54, BOARD »Seite 49

Der EDIT-Befehl wird verwendet, um eine Platine oder einen Schaltplan zu editieren bzw. neu anzulegen. Außerdem dient der Befehl dazu, Symbole, Devices und Packages zu laden, wenn man eine Bibliothek bearbeitet.

```
EDIT name.brd   lädt eine Platine
EDIT name.sch   lädt einen Schaltplan
EDIT name.pac   lädt ein Package
EDIT name.sym   lädt ein Symbol
EDIT name.dev   lädt ein Device
EDIT name.s1    lädt Blatt 1 eines Schaltplans
EDIT name.s99   lädt Blatt 99 eines Schaltplans
```

Wildcards in Namen sind erlaubt (z. B. edit \*.brd).

Gibt man EDIT ohne weitere Parameter ein, können Sie die Datei oder das Objekt mit Hilfe des sich öffnenden Menüs wählen.

Um von einem Schaltplan zu einer Platine mit dem gleichen Namen zu wechseln, kann man den Befehl

```
EDIT .brd
```

verwenden. Umgekehrt kommt man von der Platine in den entsprechenden Schaltplan mit

```
EDIT .sch
```

Eine anderes Blatt (Sheet) eines Schaltplans kann man mit

```
EDIT .sx
```

(x ist die Blattnummer) oder mit Hilfe der Combo-Box in der Action-Toolbar laden.

Will man Symbole, Devices oder Packages editieren, dann ist zuerst eine Bibliothek mit OPEN zu öffnen und dann der entsprechende EDIT-Befehl zu geben.

### Welches Verzeichnis?

Der EDIT-Befehl holt Dateien aus dem Projektverzeichnis »Seite 27.



## ERC

### Funktion

Electrical Rule Check (Prüfung auf elektrische Fehler).

### Syntax

ERC

**Siehe auch** DRC »Seite 63, Konsistenzprüfung »Seite 360

Dieser Befehl prüft Schaltpläne auf elektrische Fehler. Die Ergebnisse werden in eine Textdatei mit der Extension .ERC geschrieben.

Folgende Warnungen werden ausgegeben:

- 1. SUPPLY Pin Pin\_Name overwritten with Net\_Name
- 2. NC Pin Elem.\_Name Pin\_Name connected to Net\_Name
- 3. POWER Pin El.\_Name Pin\_N. connected to Net\_Name
- 4. only one Pin on net Net\_Name
- 5. no Pins on net Net\_Name
- 6. SHEET Sheet\_Nr.: unconnected Pin: Element\_N. Pin\_N.
- 7. SHEET Sheet\_Nr.: Junction at (x, y) appears to connect nets Net\_Name and Net\_Name
- 8. SHEET Sheet\_Nr., NET Net\_Name: missing Junction at (x, y)
- 9. SHEET Sheet\_Nr., NET Net\_Name: close but unconnected wires at (x, y)
- 10. SHEET Sheet\_Nr.: NETS Net\_Name and Net\_Name too close at (x, y)
- 11. SHEET Sheet\_Nr., NET Net\_Name: wire overlaps pin at (x, y)
- 12. SHEET Sheet\_Nr.: pins overlap at (x, y)

Folgende Fehler werden gemeldet:

- 13. no SUPPLY for POWER Pin Element\_Name Pin\_Name
- 14. no SUPPLY for implicit POWER Pin El.\_Name Pin\_Name
- 15. unconnected INPUT Pin: Element\_Name Pin\_Name
- 16. only INPUT Pins on net Net\_Name
- 17. OUTPUT and OC Pins mixed on net Net\_Name
- 18. n OUTPUT Pins on net Net\_Name
- 19. OUTPUT and SUPPLY Pins mixed on net OUTNET
- 20. uninvoked MUST gate Gate\_Name in part Part\_Name (Device\_Name)

### Bedeutung

- 1. SUPPLY-Pin Pin\_Name überschrieben mit Net\_Name
- 2. NC-Pin Elem.\_Name Pin\_Name verbunden mit Net\_Name
- 3. POWER-Pin El.\_Name Pin\_N. verbunden mit Net\_Name
- 4. nur ein Pin an Netz Net\_Name
- 5. keine Pins an Netz Net\_Name
- 6. offener Pin an Baustein mit ein oder zwei Anschlüssen
- 7. SHEET Sheet\_Nr.: Junction bei (x, y) scheint die Netze Net\_Name und Net\_Name zu verbinden
- 8. SHEET Sheet\_Nr., NET Net\_Name: fehlende Junction bei (x, y)
- 9. SHEET Sheet\_Nr., NET Net\_Name: nicht verbundene Wires zu nahe bei (x, y)
- 10. SHEET Sheet\_Nr.: NETS Net\_Name und Net\_Name zu nahe bei (x, y)
- 11. SHEET Sheet\_Nr., NET Net\_Name: Wire überlappt mit Pin bei (x, y)
- 12. SHEET Sheet\_Nr.: Pins überlappen bei (x, y)
- 13. SUPPLY für POWER-Pin des benutzten Gates Element\_Name Pin\_Name fehlt
- 14. SUPPLY für POWER-Pin des unbenutzten Gate Element\_Name Pin\_Name fehlt
- 15. offener INPUT-Pin: Element\_Name Pin\_Name
- 16. ausschließlich INPUT-Pins an Netz Net\_Name
- 17. OUTPUT- und OC-Pins an Netz Net\_Name gemischt
- 18. mehrere (n) OUTPUT-Pins an Netz Net\_Name
- 19. OUTPUT- und SUPPLY-Pins an Netz OUTNET gemischt
- 20. das Gatter mit Add-Level MUST ist nicht benutzt

### Konsistenzprüfung

Der ERC-Befehl führt auch eine Konsistenzprüfung »Seite 360 zwischen Schaltung und zugehöriger Platine durch, sofern die Board-Datei vor dem Start des ERC geladen worden ist. Als Ergebnis des ERC wird die automatische Forward&Back-Annotation »Seite 359 ein- oder ausgeschaltet, abhängig davon, ob die Dateien konsistent sind oder nicht.

Bitte beachten Sie, dass der ERC Unterschiede zwischen impliziten Power-Pins und Supply-Pins im Schaltplan und den tatsächlichen Signalverbindungen im Layout feststellen kann. Solche Unstimmigkeiten können entstehen, wenn Sie die Supply-Pins im Schaltplan modifizieren, nachdem Sie mit dem BOARD-Befehl eine Platinen-Datei erzeugt haben. Wenn die Power-Pins nur "implizit" verbunden sind, können diese Änderungen nicht immer in das Layout übertragen werden. Werden solche Fehler festgestellt, bleibt die Forward&Back-Annotation »Seite 359 weiterhin erhalten. Allerdings müssen die Supply-Pins überprüft werden!

## ERRORS

### Funktion

Zeigt Fehler, die vom DRC gefunden wurden.

### Syntax

ERRORS  
ERRORS CLEAR

### Maus

Linke Maustaste: zeigt Fehler an.  
Doppelklick bringt Fehler ins Bildzentrum.

### Siehe auch DRC »Seite 63

Zur Auswertung der vom DRC gefundenen Fehler dient der Befehl ERRORS. Wird er aktiviert, dann öffnet sich ein Fenster, in dem alle vom DRC gefundenen Fehler aufgelistet sind. Selektiert man einen Eintrag aus der Fehlerliste, wird der jeweilige Fehler durch eine Bezugslinie angezeigt.

Ein Doppelklick auf einen Fehlereintrag zentriert die Zeichenfläche auf die Position des Fehlers. Aktivieren Sie die Option "Centered", geschieht dies automatisch.

Der "Del"-Button im ERRORS-Dialog löscht einzelne Fehlerpolygone. Um alle Fehlerpolygone zu löschen benutzen Sie den Befehl

ERRORS CLEAR

Falls Sie DRC-Fehler erhalten, die auch nach einer entsprechenden Änderung der Design Rules »Seite 159 nicht verschwinden, überprüfen Sie bitte die Netzklasse »Seite 53 des beanstandeten Objektes. Möglicherweise wird der Fehler wegen eines der Parameter dieser Klasse gemeldet.

## EXPORT

### Funktion

ASCII- und Grafik-Dateien erzeugen.

### Syntax

```
EXPORT SCRIPT      filename;  
EXPORT NETLIST     filename;  
EXPORT NETSCRIPT   filename;  
EXPORT PARTLIST    filename;  
EXPORT PINLIST     filename;  
EXPORT DIRECTORY   filename;  
EXPORT IMAGE       filename|CLIPBOARD [MONOCHROME] resolution;
```

**Siehe auch** SCRIPT »Seite 114, RUN »Seite 113

Der EXPORT-Befehl dient dazu, EAGLE-Daten in Form von Textdateien (ASCII-Dateien) zur Verfügung zu stellen oder Grafikdaten aus der aktuellen Zeichnung zu erzeugen.

Standardmäßig wird die erzeugte Datei in das **Projekt**-Verzeichnis geschrieben.

Der EXPORT-Befehl erzeugt folgende Textdateien:

### SCRIPT

Die mit OPEN geöffnete Bibliothek wird als Script-Datei ausgegeben. Damit besteht die Möglichkeit, Bibliotheken mit einem Texteditor zu bearbeiten und anschließend wieder einzulesen.

Bitte beachten Sie, daß beim Exportieren Koordinatenangaben in der gegenwärtigen Rastereinheit verwendet werden.

Wenn mit EXPORT eine Bibliothek in ein Script-File verwandelt und dieses anschließend wieder eingelesen wird, so sollte dafür eine NEUE (leere!) Bibliothek geöffnet werden, da es sonst vorkommen kann, daß Objekte mehrfach definiert werden! Der Vorgang des Script-Einlesens kann u. U. erheblich beschleunigt werden, wenn vorher

```
Set Undo_Log Off;
```

eingegeben wird (nicht vergessen, es nachher wieder einzuschalten, da sonst kein Undo möglich ist!).

### NETLIST

Gibt eine Netzliste des geladenen Schaltplans oder der geladenen Platine aus. Es werden nur Netze berücksichtigt, die mit Elementen verbunden sind.

### NETSCRIPT

Gibt die Netzliste des geladenen Schaltplans in Form eines Script-Files aus, das in eine Platine (mit bereits platzierten Elementen bzw. mit durch DELETE SIGNALS gelöschten Signalen) eingelesen werden kann.

### PARTLIST

Gibt eine Bauteile-Liste des Schaltplans oder der Platine aus. Es werden nur Bauteile mit Pins/Pads berücksichtigt.

### PINLIST

Gibt eine Liste mit den Pad- und Pin-Namen aller Bauteile aus, die zu jedem Pin die Direction sowie den Namen des angeschlossenen Netzes enthält.

### DIRECTORY

Gibt das Inhaltsverzeichnis der gerade geöffneten Bibliothek aus.

### IMAGE

Bei der Ausgabe eines *IMAGE* wird eine Grafikdatei mit der angegebenen Datei-Erweiterung erzeugt. Folgende

Formate sind verfügbar:

.bmp	Windows-Bitmap-Datei
.png	Portable-Network-Graphics-Datei
.pbm	Portable-Bitmap-Datei
.pgm	Portable-Grayscale-Bitmap-Datei
.ppm	Portable-Pixmap-Datei
.xbm	X-Bitmap-Datei
.xpm	X-Pixmap-Datei

Der *resolution*-Parameter definiert die Bildauflösung (in 'dpi').

Ist *filename* der besondere Name CLIPBOARD (egal ob groß- oder klein geschrieben), wird das Bild in die Zwischenablage des Systems kopiert.

Das optionale Schlüsselwort *MONOCHROME* erzeugt ein schwarz/weißes Bild.

## GATESWAP

### Funktion

Äquivalente Gates vertauschen.

### Syntax

```
GATESWAP • •..;  
GATESWAP gate_name gate_name..;
```

**Siehe auch** ADD »Seite 42

Mit diesem Befehl kann man Gates in einem Schaltplan vertauschen. Dabei müssen die beiden Gates identisch sein (dieselben Pins haben) und in der Device-Definition denselben Swaplevel (größer als 0) bekommen haben. Sind diese Bedingungen erfüllt, können auch Gates aus unterschiedlichen Devices vertauscht werden.

Der als Parameter anzugebende Name ist der im Schaltplan sichtbare Name (z. B. U1A für Gate A im Bauteil U1).

Wird ein Bauteil durch den GATESWAP-Befehl "unbenutzt", wird es automatisch aus der Schaltung entfernt.

## GRID

### Funktion

Rasterdarstellung und -einheit einstellen.

### Syntax

```
GRID option...;  
GRID;
```

### Tasten

F6: GRID; schaltet das Raster ein bzw. aus.

**Siehe auch** SCRIPT »Seite 114

Mit dem GRID-Befehl definiert man, ob und wie das Raster auf dem Bildschirm dargestellt wird. Außerdem legt dieser Befehl die verwendete Rastereinheit fest.

```
GRID;
```

schaltet das Raster ein bzw. aus.

Objekte und Elemente lassen sich nur auf dem eingestellten Raster plazieren. Für Platinen im Zollraster darf deshalb z. B. kein mm-Raster verwendet werden.

Folgende options sind möglich:

GRID ON;	Raster darstellen
GRID OFF;	Raster ausschalten
GRID DOTS;	Raster als Punkte darstellen
GRID LINES;	Raster als Linien darstellen
GRID MIC;	Rastereinheit ist Mikrometer
GRID MM;	Rastereinheit ist mm
GRID MIL;	Rastereinheit ist Mil (0.001 Inch)
GRID INCH;	Rastereinheit ist Inch (Zoll)
GRID FINEST;	Raster auf kleinstmöglichen Wert einstellen (1/10000 mm)
GRID grid_size;	Rasterabstand in der aktuellen Einheit
GRID LAST;	Setzt die Grid-Parameter auf die zuletzt eingestellten Werte
GRID DEFAULT;	Einstellung auf Standardwerte
GRID grid_size grid_multiple;	grid_size = Rasterabstand grid_multiple = Rasterfaktor

### Beispiele

```
Grid mm;  
Set Diameter_Menu 1.0 1.27 2.54 5.08;  
Grid Last;
```

In diesem Fall kann man zur zuletzt eingestellten Grid-Definition zurückkehren, ohne sie zu kennen.

```
GRID mm 1 10;
```

gibt an, daß ein Rasterabstand von 1 mm eingestellt und jede zehnte Rasterlinie angezeigt werden soll.

Beim GRID-Befehl sind auch mehrere Parameter zulässig:

```
GRID inch 0.05 mm;
```

Der erste Zahlenwert im GRID-Befehl entspricht dem Rasterabstand, der zweite (falls vorhanden) dem Rasterfaktor.

Hier wird das Raster zunächst auf 0.05 Zoll eingestellt, dann wird die Koordinatenanzeige auf mm umgestellt. Die Koordinaten werden in diesem Fall zwar in mm angezeigt, trotzdem ist das Raster nach wie vor auf 1/20 Zoll eingestellt!

GRID DEFAULT;

Setzt das Raster auf den Standardwert. Dieser Befehl ist äquivalent mit:

GRID OFF DOTS INCH 0.05 1;



## GROUP

### Funktion

Definieren einer Gruppe.

### Syntax

```
GROUP • . .  
GROUP ;
```

### Maus

Rechte Maustaste schließt das Polygon.

Mit gedrückter linker Taste ein Rechteck aufziehen definiert eine rechteckige Gruppe.

**Siehe auch** CHANGE »Seite 51, CUT »Seite 57, PASTE »Seite 94, MIRROR »Seite 85, DELETE »Seite 58

Mit dem GROUP-Befehl definiert man eine Gruppe von Objekten und Elementen, auf die man anschließend bestimmte Befehle anwenden kann. Natürlich kann auch eine ganze Zeichnung als Gruppe definiert werden.

Die Objekte und Elemente selektiert man, indem man nach Aktivieren des GROUP-Befehls mit der Maus ein Polygon zeichnet, das mit dem Betätigen der rechten Maustaste geschlossen wird. In die Gruppe werden nur Objekte aus den sichtbaren Layern übernommen.

Zur Gruppe gehören:

- alle Objekte mit einem Aufhängepunkt, deren Aufhängepunkt innerhalb des Polygons liegt;
- alle Objekte mit zwei Aufhängepunkten, von denen mindestens ein Endpunkt innerhalb des Polygons liegt;
- alle Kreise, deren Mittelpunkt innerhalb des Polygons liegt;
- alle Rechtecke, von denen mindestens ein Eckpunkt innerhalb des Polygons liegt.

### Gruppe bewegen

Um eine Gruppe zu bewegen, verwendet man den MOVE-Befehl mit der rechten Maustaste. Bewegt werden alle Objekte und Elemente, die vorher mit dem GROUP-Befehl selektiert worden sind. Wires, von denen nur ein Eckpunkt innerhalb des Polygons liegt, werden an diesem Ende bewegt, während das andere fest bleibt. "Hängt" die Gruppe am Cursor, läßt sie sich wiederum mit der rechten Maustaste rotieren.

Die Gruppendefinition bleibt wirksam, bis eine neue Zeichnung geladen wird oder bis der Befehl

GROUP ;

ausgeführt wird.

## HELP

### Funktion

Help-Seite anzeigen.

### Syntax

HELP  
HELP command

### Tasten

F1: HELP ruft die kontextsensitive Help-Funktion auf.

Dieser Befehl ruft ein Help-Fenster mit Hinweisen zum Programm auf, die vom momentanen Status abhängig sind (kontextsensitiv).

Wird im HELP-Befehl ein Befehlsname (command) angegeben, dann erhält man die Beschreibung dieses Befehls.

### Beispiel

HELP GRID;

Es erscheint die Beschreibung des GRID-Befehls.

## HOLE

### Funktion

Bohrloch in Platine oder Package einfügen.

### Syntax

HOLE drill •..

**Siehe auch** VIA »Seite 129, PAD »Seite 92, CHANGE »Seite 51

Mit diesem Befehl definiert man Bohrungen ohne Durchkontaktierung in Platinen oder Packages. Der Parameter drill gibt den Bohrdurchmesser in der aktuellen Einheit an. Er darf maximal 0.51602 Zoll (ca. 13.1 mm) betragen.

### Beispiel

HOLE 0.20 •

Falls die eingestellte Maßeinheit "Inch" ist, hat das Hole einen Durchmesser von 0.20 Zoll.

Der eingegebene Bohrdurchmesser (gilt auch für Pads und Vias) bleibt für nachfolgende Operationen erhalten. Er kann mit dem Befehl

CHANGE DRILL value •

verändert werden.

Ein Hole kann nur selektiert werden, wenn der Holes-Layer eingeblendet ist (DISPLAY).

Eine Bohrung (Hole) erzeugt das zugehörige Bohrsymbol im Layer Holes und einen Kreis mit dem entsprechenden Durchmesser im Layer Dimension. Die Zuordnung von Symbolen zu bestimmten Bohrdurchmessern kann im "Options/Set/Drill" Dialog geändert werden. Der Kreis im Dimension-Layer ist besonders für den Autorouter wichtig, der den eingestellten Mindestabstand zwischen Vias/Wires und Dimension-Linien damit auch zum Bohrloch einhält.

In Versorgungs-Layern erzeugen Holes Annulus-Symbole.

In den Layern tStop und bStop erzeugen Holes die LötStopmaske, deren Durchmesser sich aus dem Bohrdurchmesser plus dem mit der Option -B angegebenen Wert ergibt (Default: 10 Mil).

## INFO

### Funktion

Eigenschaften von Objekten anzeigen.

### Syntax

INFO • . .

**Siehe auch** CHANGE »Seite 51, SHOW »Seite 117

Der INFO-Befehl gibt zu einem Objekt umfassende Informationen, z.B. Wire-Breite, Layer und so weiter.

## INVOKE

### Funktion

Bestimmte Gates von Bauteilen holen.

### Syntax

```
INVOKE • orientation •  
INVOKE Part_Name Gate_Name orientation •
```

### Maus

Rechte Maustaste rotiert das Gate.

**Siehe auch** COPY »Seite 56

Addlevel und Orientation siehe ADD-Befehl.

Will man gezielt ein bestimmtes Gate eines Bauelements in die Schaltung holen (z.B. ein Power-Gate mit Addlevel Request), dann benutzt man den INVOKE-Befehl.

Ein Gate kann man aktivieren, indem man

- den Namen eines Bauelements angibt (etwa INVOKE IC5) und das gewünschte Gate aus einem Popup-Menü auswählt,
- den Elementnamen und den Gate-Namen angibt (etwa INVOKE IC5 POWER),
- das vorhandene Gate mit der Maus anklickt und das gewünschte Gate aus einem Popup-Menü auswählt.

Mit dem abschließenden Mausklick positioniert man das neue Gate.

### Gates auf verschiedenen Sheets

Soll ein Gate aus einem Bauelement geholt werden, das sich auf einem anderen Blatt des Schaltplans befindet, ist als Parameter der Name des Bauelements anzugeben. In diesem Fall zeigt die rechte Spalte des Popup-Menüs, auf welchem Blatt sich die verwendeten Gates befinden. Ein Gate auf dem Blatt, das gerade in Bearbeitung ist, wird durch ein Sternchen in der rechten Spalte des Popup-Menüs gekennzeichnet.

## JUNCTION

### Funktion

Markierungspunkt für zusammengehörige Netze setzen.

### Syntax

JUNCTION • . .

**Siehe auch** NET »Seite 88

Löschen von Junctions, siehe DELETE-Befehl.

Mit diesem Befehl lassen sich die Kreuzungspunkte zusammengehöriger Netze mit einem Punkt markieren. Ein Junction-Punkt läßt sich nur auf einem Netz plazieren. Wird ein Junction-Punkt an einer Stelle gesetzt, an der sich unterschiedliche Netze kreuzen, dann wird der Benutzer gefragt, ob er die Netze verbinden will.

Wird eine Netzlinie auf einem Punkt abgesetzt auf dem schon mindestens zwei weitere Netzlinien und/oder Pins liegen, wird automatisch ein Verbindungspunkt (Junction) gesetzt. Diese Funktion kann über "SET AUTO\_JUNCTION OFF ;" oder durch Deselektieren des Punkts "Auto set junction" im Menü "Options/Set/Misc" abgeschaltet werden.

Auf dem Bildschirm werden Junctions immer mit mindestens fünf Pixel Durchmesser dargestellt, damit sie auch in kleinen Zoom-Stufen noch sichtbar sind.

## **LABEL**

### **Funktion**

Busse und Netze beschriften.

### **Syntax**

LABEL • • •

### **Maus**

Rechte Maustaste rotiert den Label-Text.

Mittlere Maustaste selektiert Ziel-Layer für Label-Text.

**Siehe auch** NAME »Seite 87, BUS »Seite 50

Mit diesem Befehl kann man den Namen eines Busses oder Netzes im Schaltplan an eine beliebige Stelle plazieren. Der erste Mausklick sorgt dafür, daß der Name des selektierten Busses oder Netzes "am Cursor hängenbleibt". Der Text kann dann mit der rechten Maustaste rotiert werden. Mit der mittleren Maustaste wählt man den Ziel-Layer für den Label-Text aus. Der zweite Mausklick mit der linken Maustaste plaziert den Text an eine beliebige Stelle.

Es können beliebig viele Labels je Bus/Signal plaziert werden.

Label-Texte lassen sich nicht mit CHANGE TEXT ändern.

Labels werden vom Programm wie Texte behandelt, aber ihr "Wert" entspricht immer dem Namen des zugehörigen Busses oder Netzes. Ändert man den Namen eines Busses/Netzes mit dem NAME-Befehl, dann ändern sich automatisch alle zugehörigen Labels.

Selektiert man beim SHOW-Befehl einen Bus, ein Netz oder ein Label, dann werden alle zugehörigen Busse, Netze bzw. Labels hell dargestellt.

## LAYER

### Funktion

Wechseln und Definieren von Layern.

### Syntax

```
LAYER layer_number  
LAYER layer_name  
LAYER layer_number layer_name  
LAYER [??] -layer_number
```

**Siehe auch** DISPLAY »Seite 61

### Zeichenebene auswählen

Der LAYER-Befehl mit einem Parameter dient dazu, den (vorhandenen) Layer auszuwählen, in den gezeichnet werden soll. Wird der Befehl aus dem Menü ausgewählt, öffnet sich ein Popup-Menü, in dem man den gewünschten Layer selektieren kann. Bei Eingabe über die Tastatur ist als Parameter die Layer-Nummer oder der Layer-Name (wie er im Popup-Menü erscheint) zulässig.

Der ROUTE-Befehl ändert den aktiven Layer automatisch.

Bestimmte Layer stehen nicht in allen Betriebsarten zur Verfügung, da sie nicht überall einen Sinn haben.

### Layer definieren

Der LAYER-Befehl mit zwei Parametern dient dazu, einen neuen Layer zu definieren oder einen vorhandenen umzubenennen. Die Eingabe von z.B.:

```
LAYER 101 BEISPIEL;
```

erzeugt einen neuen Layer mit der Nummer 101 und dem Namen BEISPIEL.

Werden in einer Zeichnung Bibliothekselemente platziert (mit ADD oder REPLACE), die zusätzliche Layer enthalten, dann werden diese Layer automatisch in der Zeichnung angelegt.

Die vordefinierten Layer haben spezielle Bedeutung. Man kann zwar ihren Namen ändern, aber ihre besondere Funktion bleibt aufgrund ihrer Nummer erhalten.

Wenn Sie eigene Layer definieren, sollten Sie nur die Layer-Nummern ab 100 verwenden. Bei kleineren Nummern kann es sein, daß Sie in späteren EAGLE-Versionen spezielle Bedeutung bekommen.

### Layer löschen

Der LAYER-Befehl mit dem der Layer-Nummer vorangestellten Minuszeichen löscht den Layer mit dieser Nummer, z. B.:

```
LAYER -103;
```

löscht den Layer mit der Nummer 103. Voraussetzung ist, daß der Layer leer ist. Ist das nicht der Fall, wird die Fehlermeldung

"layer is not empty: #"

ausgegeben, wobei # die Layer-Nummer repräsentiert. Falls Sie jegliche Fehlermeldungen beim Löschen eines Layers vermeiden wollen, so können Sie die Option '??' angeben. Das kann in Scripts nützlich sein, die bestimmte Layer zu löschen versuchen, wo es aber keinen Fehler darstellt wenn ein Layer nicht leer oder überhaupt nicht vorhanden ist.

Die vordefinierten Layer lassen sich nicht löschen.

### Versorgungs-Layer

Die Layer 2...15 stellen für den Autorouter Versorgungs-Layer dar, wenn der Layer-Name mit dem Zeichen \$ beginnt und ein Signal mit gleichem Namen (ohne vorangestelltes \$) existiert.

### Vordefinierte EAGLE-Layer, nach Layer-Nummern geordnet



## Layout

1	Top	Leiterbahnen oben
2	Route2	Innenlage (Signal- oder Versorgungs-Layer)
3	Route3	Innenlage (Signal- oder Versorgungs-Layer)
4	Route4	Innenlage (Signal- oder Versorgungs-Layer)
5	Route5	Innenlage (Signal- oder Versorgungs-Layer)
6	Route6	Innenlage (Signal- oder Versorgungs-Layer)
7	Route7	Innenlage (Signal- oder Versorgungs-Layer)
8	Route8	Innenlage (Signal- oder Versorgungs-Layer)
9	Route9	Innenlage (Signal- oder Versorgungs-Layer)
10	Route10	Innenlage (Signal- oder Versorgungs-Layer)
11	Route11	Innenlage (Signal- oder Versorgungs-Layer)
12	Route12	Innenlage (Signal- oder Versorgungs-Layer)
13	Route13	Innenlage (Signal- oder Versorgungs-Layer)
14	Route14	Innenlage (Signal- oder Versorgungs-Layer)
15	Route15	Innenlage (Signal- oder Versorgungs-Layer)
16	Bottom	Leiterbahnen unten
17	Pads	Pads (bedrahtete Bauteile)
18	Vias	Vias (durchgehend)
19	Unrouted	Luftlinien
20	Dimension	Platinen-Umrisse (und Kreise für Holes)
21	tPlace	Bestückungsdruck oben
22	bPlace	Bestückungsdruck unten
23	tOrigins	Aufhängepunkte oben (Kreuz aut. gen.)
24	bOrigins	Aufhängepunkte unten (Kreuz aut. gen.)
25	tNames	Servicedruck oben (Bauteile-Namen, NAME)
26	bNames	Servicedruck unten (Bauteile-Namen, NAME)
27	tValues	Bauteile-Werte oben (VALUE)
28	bValues	Bauteile-Werte unten (VALUE)
29	tStop	LötStopmaske oben (aut. generiert)
30	bStop	LötStopmaske unten (aut. generiert)
31	tCream	Lotpaste oben
32	bCream	Lotpaste unten
33	tFinish	Veredelung oben
34	bFinish	Veredelung unten
35	tGlue	Klebmaske oben
36	bGlue	Klebmaske unten
37	tTest	Test- und Abgleichinformationen oben
38	bTest	Test- und Abgleichinformationen unten
39	tKeepout	Sperrflächen für Bauteile oben
40	bKeepout	Sperrflächen für Bauteile unten
41	tRestrict	Sperrflächen für Leiterbahnen oben
42	bRestrict	Sperrflächen für Leiterbahnen unten
43	vRestrict	Sperrflächen für Vias
44	Drills	Bohrungen, durchkontaktiert
45	Holes	Bohrungen, nicht durchkontaktiert
46	Milling	CNC-Fräser-Daten zum Schneiden der Plat.
47	Measures	Bemaßungen
48	Document	allgemeine Dokumentation
49	Reference	Passermarken
51	tDocu	Bauteiledokumentation oben
52	bDocu	Bauteiledokumentation unten

## Schaltplan

91	Nets	Netze
92	Busses	Busse
93	Pins	Anschlußpunkte für Schaltplansymbole mit Zusatzinformationen
94	Symbols	Umrisse der Schaltplansymbole
95	Names	Namen bei Schaltplansymbolen
96	Values	Werte/(Typenbez. bei Schaltplansymbolen)

Layer können immer mit ihrem Namen oder Nummer angegeben werden.

## MARK

### Funktion

Marke in der Zeichenfläche definieren.

### Syntax

MARK •  
MARK;

**Siehe auch** GRID »Seite 71

Mit dem MARK-Befehl definiert man einen Punkt in der Zeichenfläche, der als Bezugspunkt zum Ausmessen von Strecken dienen kann. Die Koordinaten relativ zu diesem Punkt werden in der gegenwärtig eingestellten Einheit (GRID) links oben auf dem Bildschirm mit vorangestelltem @-Zeichen angezeigt. Der Bezugspunkt wird als weißes Kreuz dargestellt. Um genau messen zu können, sollten Sie vorher ein Raster einstellen das fein genug ist.

Die Eingabe 'MARK;' schaltet die Marke ein oder aus.

## MENU

### Funktion

Text-Menüleiste verändern.

### Syntax

```
MENU option ..;  
MENU;
```

**Siehe auch** ASSIGN »Seite 46, SCRIPT »Seite 114

Mit dem MENU-Befehl kann man das Text-Menü nach eigenen Wünschen ändern.

Die vollständige Spezifikation für den option-Parameter ist:

```
option      := command | menu | delimiter  
command     := text [ ':' text ]  
menu        := text '{' option [ '|' option ] '}'  
delimiter   := '---'
```

Eine Option des MENU-Befehls kann entweder ein einfacher Befehl sein, wie z. B.

```
MENU Display Grid;
```

der die Befehle Display und Grid als Inhalt des Text-Menüs definiert; ein selbstdefinierter Befehl sein, wie z. B.

```
MENU 'MyDisp : Display None Top Bottom Pads Vias;' 'MyGrid : Grid mil 100  
lines on;';
```

Das Text-Menü enthält die beiden selbstdefinierten Befehle MyDisp und MyGrid. Nach Anklicken eines Buttons im Menü wird die nach ' : ' definierte Befehlssequenz der entsprechenden Option ausgeführt; oder eine Schaltfläche erzeugen, die Untermenüs enthalten kann, wie z. B.

```
MENU 'Grid { Fine : Grid inch 0.001; | Coarse : Grid inch 0.1; }';
```

Es wird ein Button mit dem Namen Grid erzeugt, der nach dem Anklicken ein Untermenü mit den beiden Optionen Fine und Coarse enthält.

Die besondere Option ' --- ' kann man verwenden, um einen Trennstrich im Menü zu erzeugen. Das fördert die Übersichtlichkeit im Menü.

Bitte beachten Sie, dass jede *option*, die mehr als ein Wort oder einen Text, der als Befehl interpretiert werden kann, enthält, in einfache Hochkommas eingeschlossen werden muss. Wenn Sie den MENU-Befehl in einer Script-Datei verwenden, um ein komplexes Text-Menü zu definieren und dabei die Menü-Definition zur besseren Lesbarkeit über mehrere Zeilen verteilen wollen, ist es notwendig die Zeilen mit einem Backslash ( '\ ' ) zu beenden, wie z. B.

```
MENU 'Grid {\n    Fine : Grid inch 0.001; |\n    Coarse : Grid inch 0.1;\n}';
```

### Beispiel

```
MENU Move Delete Rotate Route ';' Edit;
```

erzeugt ein Menü, das die Befehle Move bis Route, den Strichpunkt als Befehlsabschlußzeichen und den Edit-Befehl enthält.

Der Befehl

```
MENU;
```

stellt wieder das Standard-Menü ein.

Beachten Sie, dass der Eintrag ';' immer im Menü enthalten sein sollte. Er wird zum Beenden vieler Befehle

benötigt.

## MIRROR

### Funktion

Objekte spiegeln.

### Syntax

MIRROR • . .

### Maus

Rechte Maustaste selektiert Gruppe.

**Siehe auch** ROTATE »Seite 111, TEXT »Seite 123

Mit dem MIRROR-Befehl können Objekte an der y-Achse gespiegelt und damit z.B. auf der Lötseite der Platine platziert werden.

Das Spiegeln von Packages ist nur mit eingeblendetem tOrigins- bzw. bOrigins-Layer möglich.

Beim Spiegeln von Elementen werden die angeschlossenen Wires mitgespiegelt (Achtung auf Kurzschlüsse!). Vias werden dabei nicht automatisch gesetzt.

### Gruppe spiegeln

Will man eine Gruppe spiegeln, definiert man zuerst die Gruppe mit dem GROUP-Befehl, dann selektiert man den MIRROR-Befehl und klickt mit der rechten Maustaste die Gruppe an. Sie wird dann um die senkrechte Achse durch den dem Mauscursor nächstgelegenen Rasterpunkt gespiegelt.

Wires, Circles, Pads, Rectangles, Polygone und Labels lassen sich nicht explizit spiegeln (als Bestandteile von Gruppen werden sie mitgespiegelt).

### Text spiegeln

Text auf der Lötseite (Layer 'Bottom' und 'bPlace') wird automatisch gespiegelt, so daß er dann später, wenn man die Platine von der Lötseite her betrachtet, richtig lesbar ist.

Im Schaltplan lassen sich Texte nicht spiegeln.

## MOVE

### Funktion

Bewegen von Objekten und Elementen.

### Syntax

MOVE • • . .

MOVE package\_name • . .

### Maus

Rechte Maustaste dreht das Element oder selektiert Gruppe.

Mit gedrückter linker Taste kann das Objekt sofort bewegt werden.

### Tasten

F7: MOVE aktiviert den MOVE-Befehl.

**Siehe auch** GROUP »Seite 73, RATSNEST »Seite 104

Der MOVE-Befehl bewegt das Objekt, das dem Cursor bzw. dem angegebenen Koordinatenpunkt am nächsten liegt. Im Layout-Editor lassen sich Packages auch mit ihrem Namen selektieren. Das ist vor allem dann nützlich, wenn sich das Package nicht im gerade dargestellten Bildausschnitt befindet.

Das Bewegen von Packages ist nur mit eingeblendetem tOrigins- bzw. bOrigins-Layer möglich.

Der MOVE-Befehl wirkt nur auf sichtbare Layer (siehe DISPLAY).

Wires, die an einem Element hängen, lassen sich an diesem Ende nicht bewegen. Beim Bewegen von Elementen bewegen sich die angeschlossenen Wires mit, sofern sie Bestandteil eines Signals sind, (Achtung auf Kurzschlüssel!).

Wird ein Objekt mit der linken Maustaste angeklickt und die Taste danach nicht wieder losgelassen, so kann das Objekt sofort bewegt werden ("click&drag"). Es ist dann allerdings nicht möglich das Objekt während des Bewegens zu drehen oder zu spiegeln.

### Leitungen bewegen

Bewegt man mit MOVE Wires übereinander, dann werden diese Wires nicht zu einem Signal verbunden (Kurzschluß, den der DRC-Befehl feststellt).

### Gruppe bewegen

Will man eine Gruppe bewegen, definiert man zuerst die Gruppe mit dem GROUP-Befehl, dann selektiert man den MOVE-Befehl und klickt mit der rechten Maustaste die Gruppe an. Jetzt läßt sich die ganze Gruppe bewegen und mit der rechten Maustaste (um den Mauscursor) rotieren.

### Hinweise für den Schaltplan

Wird ein Supply-Pin (Pin-Direction Sup) auf ein Netz platziert, erhält dieses Netz-Segment den Namen des Supply-Pins. Werden Pins direkt aufeinander platziert (ohne explizite Netz-Linie), sind sie verbunden.

Wird ein Element bewegt, werden beim Absetzen offene Pins dieses Elements an eventuell vorhandene Netze bzw. andere Pins angeschlossen.

Bewegt man ein Netz auf einen Pin, werden Netz und Pin nicht logisch verbunden, obwohl sie optisch verbunden sind.

## NAME

### Funktion

Anzeigen und Ändern von Namen.

### Syntax

```
NAME • . .  
NAME new_name •  
NAME old_name new_name
```

**Siehe auch** SHOW »Seite 117, SMASH »Seite 119, VALUE »Seite 128

### Zeichnung

Beim Editieren von Zeichnungen kann man mit den NAME-Befehl den Namen des selektierten Elements, Signals, Netzes oder Busses anzeigen und (in einem Popup-Menü) ändern.

Die Syntax mit *old\_name new\_name* ist nur in einer Platine erlaubt und kann dazu benutzt werden ein Element umzubenennen.

### Bibliothek

Im Bibliotheks-Editier-Modus gilt das gleiche für Pad-, Smd-, Pin- und Gate(Symbol)-Namen.

### Automatische Namensvergabe

EAGLE vergibt automatisch Namen (E\$. für Elemente; S\$. für Signale; P\$. für Pads, Pins und Smds; G\$. für Gates). Zumindest in Packages und Symbolen sollte man die Pad- und Pin-Bezeichnungen durch gängige Namen (z.B. 1..14 bei einem 14pol. DIL-Gehäuse) bzw. die Signalbezeichnung ersetzen.

### Schaltplan

Beim Umbenennen von Netzen und Bussen in Schaltplänen werden drei verschiedene Fälle unterschieden, da Netze und Busse aus mehreren (nicht explizit verbundenen) Segmenten bestehen und über mehrere Seiten verteilt sein können. Das in einem solchen Fall erscheinende Menü

This Segment  
Every Segment on this Sheet  
All Segments on all Sheets

ermöglicht die Auswahl, ob

nur das selektierte Segment,  
alle Segmente auf dieser Seite,  
alle Segmente auf allen Seiten

mit der Maus oder durch Eingabe des ersten Buchstaben (Hot-Key = T, E, A) umbenannt werden sollen. Je nachdem ob das Netz bzw. der Bus nur auf dieser Seite definiert ist oder aus einem oder mehreren Segmenten besteht, entfallen einzelne Punkte des Menüs bzw. das gesamte Menü. Existiert der neue Name bereits (auf dieser oder einer anderen Seite), wird vorher noch abgefragt, ob die beiden Netze bzw. Busse verbunden werden sollen.

## NET

### Funktion

Zeichnen von Netzen im Schaltplan.

### Syntax

NET • • . .

NET net\_name • • . .

**Siehe auch** BUS »Seite 50, NAME »Seite 87, CLASS »Seite 53, SET »Seite 115

Mit dem NET-Befehl zeichnet man Einzelverbindungen (Netze) in den Net-Layer eines Schaltplans. Der erste Mausklick gibt den Startpunkt des Netzes an, der zweite setzt die Linie ab. Zwei Mausklicks am selben Punkt beenden das Netz.

Wird ein Netz an einem Punkt abgesetzt, an dem schon ein anderes Netz, ein Bus oder ein Pin liegt, endet die Netzlinie hier. Diese Funktion kann über "SET AUTO\_END\_NET OFF;" oder durch Deselektieren der Option "Auto end net and bus" im Menü "Options/Set/Misc" abgeschaltet werden.

Wird eine Netzlinie an einem Punkt abgesetzt an dem mindestens zwei Netzlinien und/oder Pins liegen, wird automatisch ein Verknüpfungspunkt (Junction) gesetzt. Diese Funktion can über "SET AUTO\_JUNCTION OFF;" oder durch Deselektieren der Option "Auto set junction" im Menü "Options/Set/Misc" abgeschaltet werden.

### Bussignal auswählen

Startet man das Netz auf einem Bus, öffnet sich ein Popup-Menü, aus dem man ein Signal des Busses auswählen kann. Das Netz erhält dann den entsprechenden Namen und gehört damit zu diesem Signal. Enthält der Bus mehrere Teilbusse, öffnet sich erst ein Popup-Menü, in dem man den gewünschten Teilbus auswählen kann.

### Netz-Namen

Gibt man einen Namen (maximal acht Zeichen) im NET-Befehl an, dann erhält das Netz diesen Namen.

Gibt man keinen Namen im NET-Befehl an und startet man auch nicht auf einem Bus, dann wird ein Name der Form N\$1 für das Netz automatisch vergeben.

Netze oder Teile davon, die auf verschiedenen Seiten eines Schaltplans liegen und denselben Namen haben, sind miteinander verbunden.

### Linienbreite

Die Breite der Linien, die ein Netz darstellen, läßt sich mit dem Befehl

```
SET NET_WIRE_WIDTH width;
```

voreinstellen (Default: 6 Mil).



## OPEN

### Funktion

Öffnen einer Bibliothek.

### Syntax

```
OPEN library_name
```

**Siehe auch** CLOSE »Seite 54, USE »Seite 127, EDIT »Seite 64, SCRIPT »Seite 114

Der OPEN-Befehl öffnet eine existierende Bibliothek oder legt eine neue an, falls noch keine mit dem angegebenen Namen existiert. Danach kann ein existierendes oder ein neues Symbol, Device oder Package editiert werden.

Dieser Befehl ist in erster Linie für Script-Dateien erforderlich.

## OPTIMIZE

### Funktion

Zusammenfassen von Wire-Segmenten.

### Syntax

```
OPTIMIZE;  
OPTIMIZE signal_name ..  
OPTIMIZE •..
```

### Maus

Rechte Maustaste: Befehl wirkt auf vorher definierte Gruppe.

**Siehe auch** SET »Seite 115, SPLIT »Seite 121, MOVE »Seite 86, ROUTE »Seite 112

Der OPTIMIZE-Befehl faßt Wire-Segmente in den Signal-Layern, die in einer Linie liegen, zu einem Segment zusammen. Voraussetzung dafür ist, daß sich die Segmente im selben Layer befinden und daß Sie dieselbe Breite haben.

### Automatische Optimierung

Diese Wire-Optimierung geschieht auch automatisch nach dem MOVE-, SPLIT und ROUTE-Befehl für den damit selektieren Wire, es sei denn, sie wurde mit dem Befehl

```
SET OPTIMIZING OFF;
```

abgeschaltet, oder beim SPLIT-Befehl wurden zwei Mausklicks auf denselben Punkt eines Wires gesetzt.

Der OPTIMIZE-Befehl arbeitet unabhängig von der Einstellung der Set-Variablen Optimizing, d.h. er funktioniert auch, wenn

```
SET OPTIMIZING OFF;
```

einggegeben wurde.

## PACKAGE

### Funktion

Zuweisung einer Package-Variante für ein Device.

### Syntax

```
PACKAGE
PACKAGE pname vname
PACKAGE name
PACKAGE -old_name new_name
PACKAGE -name
```

**Siehe auch** CONNECT »Seite 55, TECHNOLOGY »Seite 122, PREFIX »Seite 101

Dieser Befehl wird im Device-Editor verwendet um eine Gehäuse-Variante zu definieren, zu löschen oder umzubenennen.

Ohne Angabe von Parametern öffnet sich ein Dialog, der es erlaubt ein Gehäuse zu wählen und dessen Varianten-Namen zu definieren.

Der PACKAGE-Befehl löscht bereits existierende Pad/Pin-Verbindungen. Die Parameter `pname vname` verknüpfen das Gehäuse `pname` mit der neuen Variante `vname`.

Der einzelne Parameter `name` ruft die schon vorhandenen Package-Variante auf. Wurde bisher noch keine Package-Variante definiert und existiert ein Package mit dem angegebenen Namen in der Bibliothek, wird mit diesem Package eine neue Package-Variante mit Namen " (ein "leerer" Name) erzeugt (aus Kompatibilitätsgründen zu Version 3.5 notwendig).

Gibt man `-old_name new_name` an, wird die Package-Variante `old_name` umbenannt in `new_name`.

Der einzelne Parameter `-name` löscht die angegebene Package-Variante.

Der Name der Package-Variante wird dem Device-Set-Namen hinzugefügt, so dass ein eindeutiger Device-Name entsteht. Enthält der Device-Set-Name das Zeichen ' ? ', wird dieser Platzhalter durch den Package-Varianten-Namen ersetzt. Bitte beachten Sie, dass die Package-Variante erst nach der Technology-Funktion bearbeitet wird. Wenn der Device-Set-Name weder ' \* ' noch ' ? ' enthält, setzt sich der Device-Name aus *Device-Set-Name+Technology+Package-Variante* zusammen.

Anschließend ist der CONNECT-Befehl zu verwenden, damit festgelegt werden kann, welcher Pin (im Schaltplan-Symbol) welchem Pad des Gehäuses entspricht.

Führt man im Schaltplan-Modus den BOARD »Seite 49-Befehl aus, so entsteht für jedes Device dasjenige Package, das mit dem PACKAGE-Befehl festgelegt wurde.

## PAD

### Funktion

Plazieren von Pads in Packages.

### Syntax

```
PAD •...  
PAD pad_diameter pad_shape 'name' •...
```

### Maus

Rechte Maustaste rotiert das Pad (bei länglicher Form)

**Siehe auch** SMD »Seite 120, CHANGE »Seite 51, DISPLAY »Seite 61, SET »Seite 115, NAME »Seite 87, VIA »Seite 129, Design Rules »Seite 159

Ein Pad ist ein Bauelemente-Anschluß mit Durchkontaktierung.

Der PAD-Befehl platziert ein Pad in einem Package. Die Eingabe eines Durchmessers vor dem Plazieren ändert die Größe des Pads. Der Durchmesser wird in der aktuellen Maßeinheit angegeben. Er darf maximal 0.51602 Zoll (ca. 13.1 mm) betragen.

Pads erzeugen Bohrsymbole im Layer Drills und die LötStopmaske in den Layern tStop/bStop.

### Beispiel

```
PAD 0.06 •
```

Falls die eingestellte Maßeinheit "Inch" ist, hat das Pad einen Durchmesser von 0.06 Zoll. Die eingegebene Größe bleibt für nachfolgende Operationen erhalten.

### Pad-Formen

Ein Pad kann eine der folgenden Formen (pad\_shape) haben:

Square	quadratisch
Round	rund
Octagon	achteckig
XLongOct	längliches Achteck in x-Richtung
YLongOct	längliches Achteck in y-Richtung

Bei den länglichen Pads ist der kleinere der beiden Durchmesser als Parameter anzugeben. Das Seitenverhältnis ist fest auf 2:1 eingestellt.

Die Pad-Form kann entweder (wie der Durchmesser) eingegeben werden, während der Pad-Befehl aktiv ist, oder sie kann mit dem CHANGE-Befehl verändert werden, z. B.:

```
CHANGE SHAPE OCTAGON •
```

Die eingegebene Form bleibt für nachfolgende Operationen erhalten.

Da die Darstellung verschiedener Pad-Formen und der Bohrlöcher den Bildaufbau etwas verlangsamt, kann mit dem Befehl

```
SET DISPLAY_MODE REAL | NODRILL;
```

von realer auf schnelle Darstellung umgeschaltet werden.

Beachten Sie bitte, daß die tatsächlichen Werte für Pad-Form und -Durchmesser durch die Design Rules »Seite 159 des Boards bestimmt werden, in dem das Bauteil verwendet wird.

### Pad-Namen

Pad-Namen werden vom Programm automatisch erzeugt und können mit dem NAME-Befehl geändert werden. Sie enthalten maximal acht Zeichen. Der Name kann als Parameter auch im PAD-Befehl mit angegeben werden (muß in Hochkommas eingeschlossen sein).

Die Namen der Pads kann man mit dem Befehl

```
SET PAD_NAMES ON/OFF
```

ein- bzw. ausblenden. Die Änderung wird erst nach dem nächsten Bildaufbau sichtbar.

#### Einzelne Pads

Einzelne Pads in Platinen müssen als Package definiert und in die Platine geladen werden. Durchkontaktierungen lassen sich aber mit dem VIA-Befehl direkt in Platinen plazieren. Solche Durchkontaktierungen haben aber keinen Elementnamen und können deshalb auch nicht in der Netzliste geführt werden.

#### Package verändern

Es ist nicht möglich, in einem Package, das schon in einem Device verwendet wird, nachträglich ein Pad hinzuzufügen oder zu löschen, da dies die im Device definierten Pin-/Pad-Zuordnungen (CONNECT-Befehl) verändern würde.

## PASTE

### Funktion

Inhalt des PASTE-Puffers einfügen.

### Syntax

PASTE •

### Maus

Rechte Maustaste rotiert PASTE-Puffer-Inhalt.

**Siehe auch** CUT »Seite 57, GROUP »Seite 73

Mit CUT und PASTE lassen sich Teile einer Zeichnung/Bibliothek kopieren, auch in eine andere Zeichnung/Bibliothek.

Dabei ist folgendes zu beachten:

- Von und nach Devices ist kein CUT/ PASTE möglich.
- Elemente und Signale können nur von Platine zu Platine kopiert werden.
- Elemente, Busse und Netze können nur von Schaltplan zu Schaltplan kopiert werden.
- Pads und SmDs können nur von Package zu Package kopiert werden.
- Pins können nur von Symbol zu Symbol kopiert werden.
- Elemente, Signale, Pads, SmDs und Pins erhalten einen neuen Namen, falls ihr bisheriger Name in der neuen Zeichnung (bzw. im Symbol oder Package) schon existiert.
- Busse behalten ihren Namen.
- Netze behalten ihren Namen, falls es ein Label oder ein Supply-Symbol an einem der Netz-Segmente gibt. Anderenfalls wird ein neuer Name generiert, sofern der bisherige Name schon vorhanden ist.

Befinden sich in der mit PASTE einzufügenden Zeichnung modifizierte Devices oder Packages, die in einer älteren Version schon im Schaltplan oder im Layout verwendet wurden, wird automatisch ein Library-Update »Seite 126 gestartet, um die Elemente durch die neueren aus dem PASTE-Puffer zu ersetzen. **Achtung: Nach einem Library Update sollten Sie immer einen Design Rule Check »Seite 63 (DRC) und einen Electrical Rule Check »Seite 65 (ERC) durchführen!**

## PIN

### Funktion

Anschlußpunkte in Symbolen definieren.

### Syntax

PIN 'name' options •...

### Maus

Rechte Maustaste rotiert den Pin.

**Siehe auch** NAME »Seite 87, SHOW »Seite 117, CHANGE »Seite 51

Die options teilen sich in folgende Gruppen auf:

Direction  
Function  
Length  
Orientation  
Visible  
Swaplevel

#### Direction

Die logische Richtung des Signalfusses. Sie ist für den Electrical Rule Check (siehe ERC-Befehl) und für die automatische Verdrahtung der Stromversorgungs-Pins von Bedeutung. Möglich sind:

NC	not connected
In	Eingang
Out	Ausgang (totem-pole)
I/O	Ein-/Ausgang (bidirektional)
OC	Open Collector oder Open Drain
Hiz	High-Impedance(3-State)-Ausgang
Pas	passiv (für Widerstände, Kondensatoren etc.)
Pwr	Power-Pin (Vcc, Gnd, Vss ...), Stromvers.-Eing.
Sup	Stromversorgungs-Ausgang, z.B. Massesymbol.

Default: I/O

Wenn Pwr-Pins in einem Symbol vorhanden sind und im Schaltplan ein entsprechender Sup-Pin existiert, werden die Netze automatisch eingefügt. Sup-Pins werden nicht in Bauelementen verwendet.

#### Function

Die grafische Darstellung des Pins. Möglich sind:

None	keine spezielle Funktion
Dot	Invertier-Symbol
Clk	Taktsymbol
DotClk	Invertiertes Taktsymbol

Default: None

#### Length

Die Länge des Pin-Symbols. Möglich sind:

Point	Pin wird ohne Linie und Beschriftung dargestellt
Short	Linie ist 0.1 Zoll lang
Middle	Linie ist 0.2 Zoll lang
Long	Linie ist 0.3 Zoll lang

Default: Long

#### Orientation

Die Lage des Pins. Beim Plazieren lassen sich Pins mit der rechten Maustaste rotieren. Der Parameter "orientation" ist für die textuelle Eingabe des Pin-Befehls erforderlich, z. B. in Script-Dateien.

R0	Pin-Symbol rechts
R90	Pin-Symbol oben
R180	Pin-Symbol links
R270	Pin-Symbol unten

Default: R0

#### Visible

Dieser Parameter bestimmt, ob der Pin- und/oder Pad-Name im Schaltplan sichtbar sein soll.

Both	Pin- und Pad-Name sind im Schaltplan sichtbar
Pad	nur der Pad-Name ist im Schaltplan sichtbar
Pin	nur der Pin-Name ist im Schaltplan sichtbar
Off	weder Pin- noch Pad-Name im Schaltplan sichtbar

Default: Both

#### Swaplevel

Zahl zwischen 0 und 255. Die Zahl 0 bedeutet, daß der Pin nicht gegen einen anderen desselben Gates ausgetauscht werden darf. Jede Zahl, die größer als 0 ist, bedeutet, daß der Pin mit solchen Pins ausgetauscht werden kann, die den gleichen Swaplevel haben und im selben Symbol definiert sind. Beispiel: Die Eingangs-Pins eines NAND-Gatters können beide denselben Swaplevel bekommen, da sie äquivalent sind.

Default: 0

### **Anwendung des PIN-Befehls**

Der PIN-Befehl dient dazu, in einem Symbol die Anschlußpunkte für Netze (Pins) zu definieren. Pins werden im Symbols-Layer dargestellt. Zusätzliche Informationen erscheinen im Pins-Layer. Mit den options lassen sich jedem Pin individuelle Eigenschaften mitgeben. Die options dürfen in jeder beliebigen Reihenfolge eingegeben werden, man kann sie aber auch ganz weglassen.

Gibt man im PIN-Befehl einen Namen an, dann muß er in Hochkommas eingeschlossen sein. Pin-Namen kann man im Symbol-Editier-Modus mit dem NAME-Befehl ändern.

#### Namen "hochzählen"

Will man beispielsweise Pins mit den Namen D0 bis D7 in einem Symbol plazieren, dann setzt man den ersten Pin mit dem Befehl

```
PIN 'D0' •
```

und alle weiteren nur noch mit je einem Mausklick ab. Der numerische Teil des Namens wird dann automatisch weitergezählt.

#### Options mit CHANGE einstellen

Alle options lassen sich auch mit dem CHANGE-Befehl voreinstellen. Sie bleiben so lange erhalten, bis sie entweder mit dem PIN- oder dem CHANGE-Befehl wieder geändert werden.

Der SHOW-Befehl zeigt den Namen des Pins sowie Direction und Swaplevel an.

#### Pin rotieren

Bewegt man einen Pin mit dem MOVE-Befehl, dann rotiert die rechte Maustaste den Pin.

#### Gleiche Pin-Namen

Wenn Sie Bausteine definieren wollen, die mehrere Pins mit gleichem Namen haben, dann gehen Sie folgendermaßen vor:

Drei Pins sollen z.B. GND heißen. Sie geben den Pins bei der Symbol-Definition die Namen GND@1, GND@2



und GND@3. Im Schaltplan sind nur die Zeichen vor dem "@" sichtbar, und die Pins werden dort auch so behandelt, als hießen Sie GND.

Es ist nicht möglich, in einem Symbol, das in einem Device verwendet wird, nachträglich einen Pin hinzuzufügen oder zu löschen, da dies die im Device definierten Pin-/Pad-Zuordnungen (CONNECT-Befehl) verändern würde.

Pins können Namen mit maximal acht Zeichen haben.

#### Pin-Beschriftung

Die Position der Pin- und Pad-Namen in einem Schaltplansymbol ist relativ zum Pin-Aufhängepunkt festgelegt und kann nicht verändert werden. Ebenso ist die Schriftgröße für Pin- und Pad-Namen fest eingestellt (60 Mil). Bitte orientieren Sie sich beim Definieren neuer Symbole an den Größenverhältnissen der in den mitgelieferten Bibliotheken vorhandenen Bausteine.

## PINSWAP

### Funktion

Äquivalente Pins/Pads vertauschen.

### Syntax

PINSWAP • • . .

**Siehe auch** PIN »Seite 95

In einem Schaltplan kann man mit diesem Befehl Pins vertauschen, die zum selben Device gehören und bei der Symbol-Definition denselben Swaplevel erhalten haben (Swaplevel > 0). Swaplevel siehe PIN-Befehl. Ist eine Platine über die Back-Annotation »Seite 359 einem Schaltplan verbunden, dann lassen sich Pads nur dann vertauschen, wenn die zugehörigen Pins vertauscht werden können.

In einer Platine, zu der es keinen Schaltplan gibt, lassen sich mit zwei Pads desselben Package vertauschen. Der Swaplevel wird dabei nicht geprüft.

Die an den vertauschten Pads angeschlossenen Leitungen wandern mit, so daß es zu Kurzschlüssen kommen kann. Bitte DRC durchführen und, falls erforderlich, Fehler korrigieren.

## POLYGON

### Funktion

Zeichnen von Polygon-Flächen.

### Syntax

POLYGON signal\_name wire\_width • • • ..

### Maus

Doppelklick mit linker Maustaste schließt das Polygon.

Mittlere Maustaste wählt den Layer.

Rechte Maustaste ändert den Knickwinkel (siehe SET Wire\_Bend »Seite 115).

**Siehe auch** CHANGE »Seite 51, DELETE »Seite 58, RATSNEST »Seite 104, RIPUP »Seite 110

Der POLYGON-Befehl dient zum Zeichnen von Polygonflächen. Polygone in den Layern Top, Bottom und Route2..15 werden als Signale behandelt. Polygone in den Layern t/b/vRestrict sind Sperrflächen für den Autorouter.

### Anmerkung

Sie sollten es vermeiden, sehr kleine Werte für die *width* eines Polygons zu verwenden, da dies zu extrem großen Datenmengen führen kann, wenn die Zeichnung mit dem CAM Prozessor »Seite 139 ausgegeben wird.

Die Polygon-*width* sollte immer größer sein als die physikalische Auflösung des Ausgabegerätes. Zum Beispiel sollte bei einem Gerber Fotoplotter mit einer typischen Auflösung von 1 Mil die Polygon *width* nicht kleiner als zum Beispiel 6 Mil gewählt werden. Im allgemeinen sollte die Polygon *width* in der selben Größenordnung liegen wie die der übrigen Wires.

Falls Sie dem Polygon einen Namen geben wollen, der mit einer Ziffer beginnt (zum Beispiel 0V), so müssen Sie diesen Namen in Hochkommas einschließen, um ihn von einem *width*-Wert zu unterscheiden.

Die Parameter Isolate und Rank sind nur für Polygone in den Signallayern Top...Bottom relevant.

### Urzustand und freigerechneter Zustand

Für Polygone, die Bestandteil eines Signals sind, gibt es zwei verschiedene Zustände:

1. Outlines: "Urzustand", also die Form in der sie vom Benutzer definiert worden sind (Umrisslinien)
2. Real mode: "freigerechneter" Zustand, also die Form wie sie vom Programm berechnet wird.

In der Board-Datei (name.BRD) ist nur der Urzustand abgespeichert.

Standardmäßig werden alle Polygone am Bildschirm im Urzustand dargestellt, da das Freirechnen ein rechenintensiver und damit zeitaufwendiger Vorgang ist. Es werden dabei nur die vom Benutzer definierten Umrisslinien dargestellt.

Bei der Ausgabe mit dem CAM-Prozessor werden auf jeden Fall alle Polygone freigerechnet.

Die Berechnung des Polygons kann mit einem Klick auf das Stop-Icon abgebrochen werden. Alle bis dahin freigerechneten Polygone liegen dann im freigerechneten Zustand vor, alle anderen (auch das gerade in der Berechnung befindliche!) liegen im Urzustand vor.

Das Freirechnen von Polygonen wird mit dem RATSNEST »Seite 104 -Befehl ausgelöst. Man kann das mit SET »Seite 115 POLYGON\_RATSNEST OFF; verhindern.

Ein freigerechnetes Polygon kann durch Anklicken mit dem RIPUP »Seite 110-Befehl wieder in den Urzustand zurückversetzt werden.

Bei CHANGE-Operationen wird ein Polygon neu freigerechnet, wenn es vor dem CHANGE bereits freigerechnet war.

### Andere Befehle und Polygone

Polygone werden an den Kanten selektiert (wie normale Wires)

SPLIT: fügt neue Polygonkanten ein.

DELETE: löscht eine Polygon-Ecke (falls nur noch drei Ecken vorhanden sind, wird das ganze Polygon gelöscht).

CHANGE LAYER: ändert den Layer des gesamten Polygons.

CHANGE WIDTH: ändert den Parameter Width des gesamten Polygons.

MOVE: bewegt Polygonkante oder -ecke (wie bei normalen Wire-Zügen).

COPY: kopiert ganzes Polygon.

NAME: Falls das Polygon in einem Signal-Layer liegt, wird der Name des Signals geändert.

### Polygon-Parameter

Width: Linienbreite der Polygonkanten. Wird auch zum Ausfüllen verwendet.

Layer: Polygone können in jeden Layer gezeichnet werden. Polygone in Signal-Layern sind Bestandteil eines Signals und werden 'freigestellt', d.h. potentialfremde Anteile werden 'abgezogen'. Polygone in Signallayern gehören zu einem Signal und halten Mindestabstände zu anderen Signalen, die in den Design Rules oder über die Netzklasse definiert wurden, ein. Von Polygonen im Top-Layer werden auch Objekte im Layer tRestrict abgezogen (entsprechendes gilt für Bottom und bRestrict). Damit ist es z. B. möglich, eine negative Beschriftung innerhalb einer Massefläche zu erzeugen.

Pour: Füllmodus (Solid = ganz gefüllt [Default], Hatch = schraffiert).

Rank: Legt fest wie Polygone voneinander subtrahiert werden. Polygone mit einem niedrigeren 'Rank' "erscheinen zuerst" (haben eine höhere Priorität) und werden somit von Polygonen mit einem höheren 'Rank' abgezogen.

Für Polygone in Signallayern (im Layout gezeichnet) sind die Werte 1 . . 6 erlaubt, für Polygone in Packages die Werte 0 oder 7. Polygone mit gleichem Rank werden vom Design Rule Check »Seite 63 gegeneinander geprüft. Der Parameter 'Rank' ist nur für Polygone in Signallayern (1 . . 16) relevant und wird von Polygone in anderen Layern ignoriert. Der Default-Wert ist 1 für Signal-Polygone und 7 für Package-Polygone.

Thermals: Bestimmt wie potentialgleiche Pads und Smds angeschlossen werden (On = es werden Thermals generiert [default], Off = keine Thermals).

Spacing: Abstand der Füll-Linien bei Pour=Hatch (Default: 50 Mil).

Isolate: Abstand der freigestellten Polygonkanten zu potentialfremdem Kupfer bzw. Objekten im Dimension-Layer (default: 0). Dieser Wert ist nur dann maßgeblich, wenn er größer ist als der jeweilige Wert in den Design Rules. Siehe auch Design Rules »Seite 159 unter **Distance**.

Orphans: Beim Freistellen von Polygonen kann es passieren, daß das ursprüngliche Polygon in mehrere Teile zerfällt. Falls sich in einem solchen Teil kein Aufhängepunkt eines Objektes des zugehörigen Signals befindet, entsteht eine 'Insel' ohne elektrische Verbindung zum zugehörigen Signal. Sollen solche Inseln (oder 'verwaiste' Flächen) erhalten bleiben, ist der Parameter Orphans auf On zu setzen. Bei Orphans = Off [default] werden sie eliminiert. Hat keine der Polygoneiteilflächen eine solche Verbindung, werden alle Teile unabhängig von der Einstellung des Parameters Orphans dargestellt.

Unter gewissen Umständen, insbesondere mit Orphans = Off, kann ein Polygon vollständig verschwinden.

### Stegbreite bei Thermals

Die Breite der Stege bei Thermals ist:

- bei Pads gleich dem halben Bohrdurchmesser des Pads,
- bei Smds gleich der Hälfte der kleineren Kante,
- mindestens gleich der Linienbreite (Width) des Polygons,
- maximal zweimal die Linienbreite (Width) des Polygons.

### Konturdaten

Der Signalname \_OUTLINES\_ gibt dem Polygon besondere Eigenschaften, die man zur Erzeugung von Konturdaten »Seite 156 (z. B. zum Fräsen von Prototypen) benötigt. Dieser Name sollte ansonsten nicht verwendet werden.

## PREFIX

### Funktion

Präfix für Schaltzeichen festlegen.

### Syntax

```
PREFIX prefix_string;
```

**Siehe auch** CONNECT »Seite 55, PACKAGE »Seite 91, VALUE »Seite 128

Dieser Befehl wird im Device-Editier-Modus angewendet. Er legt fest, mit welchem Zeichen oder welcher Zeichenfolge der automatisch vergebene Name beginnen soll, wenn das Element mit dem ADD-Befehl im Schaltplan platziert wird.

### Beispiel

```
PREFIX U;
```

Wird dieser Befehl ausgeführt, während man das Device 7400 editiert, dann bekommen später die mit ADD im Schaltplan platzierten NAND-Gatter die Namen U1, U2, U3 und so weiter. Diese Namen lassen sich mit dem NAME-Befehl ändern.

## PRINT

### Funktion

Druckt eine Zeichnung auf dem System-Drucker aus.

### Syntax

```
PRINT [factor] [-limit] [options] [;]
```

**Siehe auch** CAM-Prozessor »Seite 139, Drucken auf dem System-Drucker »Seite 135

Der PRINT-Befehl druckt die gerade editierte Zeichnung auf dem System-Drucker aus.

Farben und Füllmuster werden aus dem Editor-Fenster übernommen, falls nicht die Optionen SOLID oder BLACK angegeben werden.

Wenn Sie Pads und Vias "ausgefüllt" drucken wollen (ohne sichtbare Bohrlöcher), benutzen Sie den Befehl

```
SET »Seite 115 DISPLAY_MODE NODRILL;
```

**Bitte beachten Sie, daß Polygone in Platinen beim Ausdrucken mit dem PRINT-Befehl nicht automatisch freigerechnet werden! Es werden lediglich die Umrisse dargestellt. Um die Polygone freigerechnet auszudrucken führen Sie bitte vorher den RATSNEST »Seite 104-Befehl aus.**

Es kann ein `factor` angegeben werden um die Ausgabe zu skalieren.

Mit dem Parameter `limit` kann die maximale Anzahl von Blättern angegeben werden, die für die Ausgabe verwendet werden soll. Diese Zahl muß mit einem vorangestellten '-' angegeben werden, um sie vom `factor` unterscheiden zu können. Sollte die Zeichnung nicht auf die vorgegebene Anzahl von Blättern passen, so wird der `factor` so lange verkleinert, bis sie gerade noch paßt.

Wird der PRINT-Befehl nicht mit einem ';' abgeschlossen, so erscheint ein Druck-Dialog »Seite 136 in dem alle Druck-Optionen eingestellt werden können. Bitte beachten Sie, daß Optionen, die über die Kommandozeile eingegeben wurden, nur dann dauerhaft in den Druckeinstellungen gespeichert werden, wenn sie über den Druck-Dialog »Seite 136 bestätigt wurden (d.h. wenn der Befehl nicht mit einem ';' abgeschlossen wurde).

Folgende `options` stehen zur Verfügung:

MIRROR	spiegelt die Ausgabe
ROTATE	dreht die Ausgabe um 90°
UPSIDEDOWN	dreht die Ausgabe um 180°. Zusammen mit ROTATE, wird die Zeichnung um insgesamt 270° gedreht
BLACK	ignoriert die Farbeinstellungen der Layer und zeichnet alles in Schwarz
SOLID	ignoriert die Füllmuster der Layer und zeichnet alles voll ausgefüllt

Wird einer `option` ein '-' vorangestellt, so wird diese Option ausgeschaltet, falls sie zur Zeit eingeschaltet ist (von einem vorhergehenden PRINT). Ein '-' allein schaltet alle `options` aus.

### Beispiele

PRINT	öffnet den Print Dialog »Seite 136 für die Eingabe der Druckeinstellungen
PRINT;	druckt die Zeichnung ohne weiteren Dialog, mit default Einstellungen
PRINT - MIRROR BLACK SOLID;	druckt die Zeichnung gespiegelt, alles in Schwarz und voll ausgefüllt
PRINT 2.5 -1;	druckt die Zeichnung um den Faktor 2.5 vergrößert, wobei aber sichergestellt wird, daß nicht mehr als <b>ein</b> Blatt verwendet wird

## QUIT

### Funktion

Beendet die Arbeit mit EAGLE

### Syntax

QUIT

Dieser Befehl beendet die Arbeit mit EAGLE. Sind seit dem letzten Abspeichern der Zeichnung oder der Bibliothek Änderungen vorgenommen worden, erscheint ein Popup-Menü, das nachfragt, ob die Zeichnung vorher abgespeichert werden soll. Beantwortet man diese Frage mit no (n), dann wird das Programm ohne Abspeichern der Zeichnung beendet. Bei Cancel bleibt man im Programm.

Sie können das Programm von jeder Stelle aus mit *Alt+X* verlassen.

## RATSNEST

### Funktion

Neuberechnen der Luftlinien und Polygone.

### Syntax

RATSNEST

**Siehe auch** SIGNAL »Seite 118, MOVE »Seite 86, POLYGON »Seite 99, RIPUP »Seite 110

Der RATSNEST-Befehl berechnet alle Luftlinien neu, damit man z. B. nach einer Änderung der Bauelemente-Plazierung wieder die kürzesten Verbindungen erhält. Auch nach dem Einlesen einer Netzliste (mit dem SCRIPT-Befehl) ist es sinnvoll, den RATSNEST-Befehl aufzurufen, da dabei im allgemeinen nicht die kürzesten Luftlinien entstehen.

Der RATSNEST-Befehl berechnet auch alle Polygone neu, die zu einem Signal gehören. Dies ist notwendig, damit für bereits durch Polygon-Flächen verbundene Pads keine Luftlinien mehr erzeugt werden. Alle zugehörigen Flächen sind dann in realer Darstellung zu sehen. Der Bildaufbau wird dadurch verlangsamt. Auf Umriß-Darstellung kann mit dem RIPUP-Befehl gewechselt werden. Die automatische Berechnung der Polygone kann mit SET »Seite 115 POLYGON\_RATSNEST OFF; verhindert werden.

RATSNEST berechnet keine Luftlinien für Signale, für die es einen eigenen Versorgungs-Layer gibt (z.B. Layer \$GND für Signal GND), es sei denn für Smd-Bauelemente, die an das nächstgelegene GND-Pad angeschlossen werden.

### Luftlinien der Länge Null

Enden zwei oder mehr Wires desselben Signals am selben Punkt, aber auf unterschiedlichen Layern, und sind die Signale nicht über eine Durchkontaktierung verbunden, dann wird eine *Luftline der Länge Null* erzeugt und als dicker Punkt im Unrouted-Layer dargestellt. Dasselbe gilt für gegenüberliegende SMDs (auf Top- und Bottom-Layer), die zum selben Signal gehören.

Solche *Luftlinien der Länge Null* können mit dem ROUTE »Seite 112-Befehl wie andere Luftlinien angeklickt werden. Top- und Bottom-Seite können an diesen Stellen auch durch Plazieren eines Vias »Seite 129 verbunden werden.

### Überprüfen, ob alles geroutet ist

Wenn kein unverdrahtetes Signal mehr vorhanden ist, gibt der RATSNEST-Befehl die Meldung

```
Ratsnest: Nothing to do!
```

aus. Anderenfalls erscheint die Meldung

```
Ratsnest: xx airwires.
```

wobei xx die Zahl der ungerouteten Luftlinien repräsentiert.



## RECT

### Funktion

Rechteck in eine Zeichnung einfügen.

### Syntax

RECT • • . . [diagonale Eckpunkte]

### Maus

Mittlere Maustaste wechselt den aktiven Layer.

**Siehe auch** CIRCLE »Seite 52

Mit diesem Befehl zeichnet man Rechtecke in den aktiven Layer. Die beiden Punkte legen gegenüberliegende Ecken des Rechtecks fest. Die mittlere Maustaste wechselt den aktiven Layer.

Rechtecke werden mit der Farbe des aktiven Layers ausgefüllt. Nach dem Löschen eines Rechtecks sollte deshalb das Bild mit dem WINDOW-Befehl neu aufgefrischt werden, damit die unter der Fläche liegenden Bildteile wieder sichtbar werden.

### Sperrflächen

Der RECT-Befehl in den Layern tRestrict, bRestrict und vRestrict dient zum Anlegen von Sperrflächen für den Autorouter.

### Nicht Bestandteil von Signalen

Rechtecke im Top- und Bottom-Layer gehören nicht zu Signalen. Der DRC meldet deshalb Fehler, wenn sie mit Wires, Pads usw. überlappen.

## REDO

### Funktion

Befehl erneut ausführen.

### Syntax

REDO;

### Tasten

F10: REDO REDO-Befehl ausführen. Shift+Alt+BS: REDO

**Siehe auch** UNDO »Seite 125, Forward&Back-Annotation »Seite 359

Die mit UNDO rückgängig gemachten Befehle lassen sich mit REDO erneut ausführen. Damit kann man ganze Abläufe rekonstruieren. Die Befehle EDIT, OPEN, AUTO und REMOVE löschen die Vorgeschichte.

UNDO/REDO ist vollkommen in den Mechanismus der Forward&Back-Annotation integriert.

## REMOVE

### Funktion

Löschen von Dateien, Devices, Symbolen, Packages und Schaltplanseiten (Sheets).

### Syntax

```
REMOVE name  
REMOVE name .Sxx
```

**Siehe auch** OPEN »Seite 89, RENAME »Seite 108

### Dateien löschen

Der REMOVE-Befehl löscht im Platinen- und Schaltplan-Editier-Modus die mit name angegebene Datei.

### Devices, Symbole, Packages

Im Bibliotheks-Editier-Modus löscht der REMOVE-Befehl aus der aktiven Bibliothek das unter dem Namen name gespeicherte Device, Symbol oder Package. Der Name kann mit einer Erweiterung (z. B. REMOVE name.pac) angegeben werden. Wenn Sie den Namen ohne Erweiterung angeben, müssen Sie sich im entsprechenden Editier-Modus des Bibliotheks-Editors befinden um ein Objekt zu löschen (z. B. im Package-Editor um ein Package zu löschen).

Symbole/Packages lassen sich nur löschen, wenn sie in keinem Device verwendet werden.

Ist ein Symbol, Device oder Package zum Editieren geladen, wirkt der Befehl auf den entsprechenden Typ. Ansonsten werden die Namen als Device-Namen interpretiert, falls das Schaltplan-Modul vorhanden ist, und als Package-Namen, falls kein Schaltplan-Modul vorhanden ist.

Der Name darf mit der Namens Erweiterung angegeben werden (zum Beispiel RENAME name1.pac name2[.pac] - die Erweiterung im zweiten Parameter ist optional). Wird der erste Parameter ohne Erweiterung angegeben, müssen Sie sich im entsprechenden Editier-Modus (z. B. im Package-Modus um Packages umzubenennen) befinden.

### Schaltplan-Seiten

Der REMOVE-Befehl kann auch zum Löschen von Seiten aus einem Schaltplan verwendet werden, dabei kann der Name der geladenen Zeichnung entfallen.

### Beispiel

```
REMOVE .S3
```

löscht Blatt Nr. 3 aus dem gegenwärtig geladenen Schaltplan.

In der Syntax-Beschreibung entspricht xx der Nummer des zu löschenden Sheets (1 bis 99). Falls xx gleich der Nummer des gerade editierten Sheets ist, wird Sheet Nummer 1 geladen. Alle Sheets mit Nummern, die größer sind als xx, erhalten eine um 1 verminderte Nummer. Das Löschen eines Sheets löscht auch den Undo-Puffer und kann daher nicht rückgängig gemacht werden! Da alle Sheets eines Schaltplans in einer Datei gespeichert sind (name.SCH), kann beim versehentlichen Löschen eines Sheets aber immer noch auf die alte Version zurückgegriffen werden - falls die gerade editierte Version nach dem Löschen des Sheets nicht zurückgeschrieben wurde!

REMOVE löscht den Undo-Puffer.

## RENAME

### Funktion

Symbole, Devices oder Packages in einer Bibliothek umbenennen.

### Syntax

```
RENAME old_name new_name;
```

**Siehe auch** OPEN »Seite 89

Mit RENAME kann der Name eines Symbols, Device oder Package geändert werden. Die Bibliothek muß vorher mit OPEN geöffnet worden sein. Please note:

- Symbole können umbenannt werden, wenn ein Symbol geladen ist.
- Packages können umbenannt werden, wenn ein Package geladen ist.
- Devices können umbenannt werden, wenn ein Device geladen ist.

RENAME löscht den Undo-Puffer.

## REPLACE

### Funktion

Element in Platine austauschen.

### Syntax

```
REPLACE package_name • . .
```

**Siehe auch** SET »Seite 115, UPDATE »Seite 126

Der REPLACE-Befehl kennt zwei verschiedene Betriebsarten, die mit dem SET-Befehl eingestellt werden. In beiden ist es möglich, ein Bauelement auf der Platine durch ein anderes aus einer beliebigen Bibliothek zu ersetzen.

REPLACE funktioniert nur mit eingeblendetem tOrigins- bzw. bOrigins-Layer.

### Gleiche Namen

Die erste Betriebsart wird mit

```
SET REPLACE_SAME NAMES ;
```

aktiviert; sie ist beim Programmstart eingestellt. In dieser Betriebsart kann man einem Element in einer Schaltung ein anderes Package zuweisen, bei dem dieselben Pad- und Smd-Namen vorhanden sind.

Das neue Package kann aus einer anderen Bibliothek stammen, und es darf zusätzliche Pads und Smds enthalten. Die Lage der Anschlüsse ist beliebig.

Anschlüsse des alten Package, die mit Signalen verbunden sind, müssen entsprechend auch im neuen Package vorhanden sein. Das neue Package darf auch weniger Anschlüsse haben, solange diese Bedingung erfüllt ist.

### Gleiche Koordinaten

Die zweite Betriebsart wird mit

```
SET REPLACE_SAME COORDS ;
```

aktiviert. Sie erlaubt es, einem Element in einer Schaltung ein anderes Package zuweisen, bei dem auf denselben Koordinaten (relativ zum Ursprung des Package) Pads oder Smds liegen müssen. Die Namen dürfen unterschiedlich sein.

Das neue Package kann aus einer anderen Bibliothek stammen, und es darf zusätzliche Pads und Smds enthalten. Anschlüsse des alten Package, die mit Signalen verbunden sind, müssen entsprechend auch im neuen Package vorhanden sein. Das neue Package darf auch weniger Anschlüsse haben, solange diese Bedingung erfüllt ist.

Der REPLACE-Befehl kann nicht verwendet werden, wenn die Forward&Back Annotation »Seite 359 aktiv ist. Verwenden Sie dann den Befehl CHANGE »Seite 51 PACKAGE oder den UPDATE »Seite 126-Befehl.

Existiert bereits ein Package mit demselben Namen (aus derselben Bibliothek) in der Zeichnung, und wurde die Bibliothek seit dem Plazieren des Bauteils modifiziert, wird automatisch ein Library-Update »Seite 126 gestartet. Dabei werden Sie gefragt, ob die Objekte in der Zeichnung durch die neueren aus der Bibliothek ersetzt werden sollen. **Achtung: Starten Sie den Design Rule Check »Seite 63 (DRC) und den Electrical Rule Check »Seite 65 (ERC) nach jedem Library-Update!**

## RIPUP

### Funktion

Verdrahtete in unverdrahtete Signale (Luftlinien) verwandeln.  
Polygondarstellung auf "Umriss".

### Syntax

```
RIPUP ;  
RIPUP •  
RIPUP name ..  
RIPUP ! name ..
```

Der RIPUP-Befehl funktioniert auch für Gruppen (siehe GROUP-Befehl).

**Siehe auch** DELETE »Seite 58, GROUP »Seite 73, POLYGON »Seite 99, RATSNEST »Seite 104

Der Befehl RIPUP wird dazu verwendet, bereits verlegte Signale wieder in Luftlinien zu verwandeln. Der Befehl kann verwendet werden für:

- alle Signale (Ripup;),
- alle Signale außer den genannten (Ripup ! name name..;),
- für bestimmte Signale (z. B. Ripup D0, D1, D2;),
- für bestimmte Signalabschnitte (mit der Maus anzuklicken)

Im letzteren Fall werden verlegte Wires und Vias in Luftlinien verwandelt. Wird mit RIPUP auf eine Luftlinie geklickt, so werden die an dieser Luftlinie angrenzenden verlegten Wires und Vias bis hin zum nächsten Pad, SMD oder der nächsten Luftlinie in Luftlinien verwandelt.

```
RIPUP name ..
```

Wirkt auf das gesamte Signal "name" (mehrere Signale sind möglich, z.B. Ripup D0 D1 D2;)

```
RIPUP • ..
```

Wirkt auf das durch den Mausklick selektierte Segment (bis zum nächsten Pad/Smd). Bei

```
RIPUP ;
```

werden nur solche Signale berücksichtigt, die an Elementen angeschlossen sind (z.B. die Eckwinkel zur Platinenbegrenzung bleiben erhalten).

### Polygone

Falls der RIPUP-Befehl auf ein Signal angewendet wird, zu dem ein Polygon gehört, wird das Polygon anschließend mit seinen Umrissen dargestellt (schnellerer Bildaufbau!). Soll das Polygon wieder freigerechnet werden, ist der Befehl RATSNEST auszuführen.

## ROTATE

### Funktion

Drehen von Objekten und Elementen.

### Syntax

ROTATE • . .

### Maus

Rechte Maustaste: Befehl wirkt auf vorher selektierte Gruppe.

**Siehe auch** ADD »Seite 42, MIRROR »Seite 85, MOVE »Seite 86, GROUP »Seite 73

Mit dem ROTATE-Befehl kann man die Orientierung von Objekten und Elementen in 90-Grad-Schritten ändern.

### Packages

Beim Drehen von Packages bewegen sich die angeschlossenen Leitungen mit (Achtung auf Kurzschlüsse!).

Packages lassen sich nur drehen, wenn der tOrigins- bzw. der bOrigins-Layer sichtbar ist.

### Objekte

Wires, Circles, Pads, Rectangles, Polygone und Labels lassen sich nicht explizit drehen, aber als Bestandteile von Gruppen (siehe GROUP-Befehl).

### Texte

Text wird immer so dargestellt, daß er von vorne oder von rechts zu lesen ist - auch wenn er rotiert wird. Nach zweimaligem Rotieren erscheint er deshalb wieder gleich, aber der Aufhängepunkt liegt nicht mehr links unten, sondern rechts oben. Denken Sie daran, wenn sich ein Text scheinbar nicht mehr selektieren läßt.

## ROUTE

### Funktion

Unverdrahtete in verdrahtete Signale umwandeln.

### Syntax

```
ROUTE • • ..  
ROUTE wire_width • • ..  
ROUTE • wire_width • ..
```

### Maus

Rechte Maustaste wechselt den Knickwinkel (siehe SET Wire\_Bend »Seite 115).

Mittlere Maustaste wechselt die Ebene (nur zwischen Signal-Layern).

**Siehe auch** AUTO »Seite 48, UNDO »Seite 125, WIRE »Seite 131, SIGNAL »Seite 118, SET »Seite 115, RATSNEST »Seite 104

Ist der ROUTE-Befehl aktiviert, kann die Breite (wire\_width) des entstehenden Wires unmittelbar von der Tastatur aus eingegeben werden.

Der ROUTE-Befehl dient dazu, die unverdrahteten Signale - dargestellt als Luftlinien im Unrouted-Layer - in verdrahtete - also Wires in den Signal-Layern - umzuwandeln.

Nachdem man den ROUTE-Befehl aktiviert hat, setzt man den ersten Punkt an einem Ende der Luftlinie an und bewegt den Cursor in die Richtung, in die man die Leitung legen will. EAGLE ersetzt dann die Luftlinie durch einen Wire oder zwei Wire-Stücke (je nach eingestelltem Knickwinkel) im gerade aktiven Signal-Layer.

Die linke Maustaste erneut betätigt, setzt das Leitungsstück ab.

Mit der mittleren Maustaste wechselt man die Ebene. Durchkontaktierungen (Vias) werden automatisch gesetzt.

Die rechte Maustaste ändert den Knickwinkel (siehe SET Wire\_Bend »Seite 115).

Ist eine Luftlinie komplett verdrahtet, ertönt ein Piepston.

### SMDs verdrahten

Wenn Sie an einem SMD zu routen beginnen, wird der SMD-Layer automatisch zum Routing-Layer.

Wenn Sie den ROUTE-Befehl an den Koordinaten eines SMDs beenden, wird der Route-Vorgang nur dann als beendet angesehen, wenn der letzte Wire auf dem Layer des SMDs platziert wurde.

### Fangfunktion

Der ROUTE-Befehl verfügt über eine Fangfunktion für den Zielpunkt der Luftlinie. Sobald die Länge der verbleibenden Luftlinie einen gewissen Wert unterschreitet, wird die Leiterbahn automatisch bis zum Zielpunkt fortgesetzt. Mit

```
SET SNAP_LENGTH number;
```

kann der Grenzwert für die Fangfunktion eingestellt werden, wobei number in der aktuellen Grid-Einheit anzugeben ist. Mit

```
SET SNAP_LENGTH 0;
```

wird die Fangfunktion abgeschaltet. Default ist 20 Mil.

### Winkelraster

Verlegt man mit dem ROUTE-Befehl Leitungen zu Pads, die nicht im Raster liegen, dann läßt sich das unter Umständen nicht mit 45-Grad-Winkeln erreichen. Mit

```
SET SNAP_BENDED OFF;
```

kann man dafür sorgen, daß (auch bei Wire\_Bend 1 und 3) das außerhalb des Rasters liegende Pad angeschlossen werden kann. In diesem Fall weicht ein Leitungssegment geringfügig vom 45-Grad-Winkelraster ab.



## RUN

### Funktion

Führt ein User-Language »Seite 161-Programm aus.

### Syntax

```
RUN file_name [argument ...]
```

**Siehe auch** SCRIPT »Seite 114

Der RUN-Befehl startet das User-Language-Programm mit dem Namen `file_name`.

Das optionale `Argument`-Liste ist für das ULP über die Builtin-Variablen »Seite 265 `argc` und `argv` verfügbar.

### ULP von Script-Datei ausführen

Wenn ein ULP von einer Script-Datei aufgerufen wird und das Programm einen Wert ungleich 0 zurückgibt (durch einen Aufruf der Funktion `exit()` »Seite 284 oder weil das STOP-Symbol angeklickt wurde), wird die Ausführung der Script-Datei beendet.

### Editor-Befehle von einem ULP ausführen

Ein ULP kann die `exit()` »Seite 284-Funktion auch mit einem `string`-Parameter verwenden, um einen Befehl direkt in einem Editor-Fenster auszuführen.

## SCRIPT

### Funktion

Befehls-Datei ausführen.

### Syntax

```
SCRIPT file_name;
```

**Siehe auch** SET »Seite 115, MENU »Seite 83, ASSIGN »Seite 46, EXPORT »Seite 68, RUN »Seite 113

Der SCRIPT-Befehl stellt eine Möglichkeit dar, Befehlssequenzen auszuführen, die in einer Datei abgelegt sind. Gibt man den Befehl über die Tastatur ein, so wird, falls keine Extension angegeben wurde, ".scr" als Default verwendet.

### Beispiele

```
SCRIPT nofill      ruft nofill.scr auf
SCRIPT myscr.      ruft myscr (ohne Suffix!) auf
SCRIPT myscr.old   ruft myscr.old auf
```

Bitte beachten Sie die Möglichkeiten des EXPORT-Befehls im Zusammenhang mit Script-Dateien!

Wählt man den Befehl mit Hilfe der Maus, dann zeigt ein Popup-Menü alle Dateien mit der Erweiterung .scr an. Man kann dann die gewünschte Datei auswählen.

Der SCRIPT-Befehl bietet die Möglichkeit, das Programm an individuelle Bedürfnisse anzupassen. Unter anderem kann man damit:

- das Befehlsmenü einstellen,
- Tasten belegen,
- Platinenumrisse laden,
- Farben festlegen.

Die SCRIPT-Dateien enthalten Befehle, entsprechend den Syntax-Regeln.

### Fortsetzungszeilen

Bei manchen Befehlen kann es erforderlich sein, mehrere Zeilen zu belegen. In diesem Fall gilt: Jedes öffnende Hochkomma muß in derselben Zeile wieder geschlossen werden. Das Zeichen "\" am Ende einer Kommandozeile sorgt dafür, daß das erste Wort der nächsten Zeile nicht als Befehl interpretiert wird. Damit lassen sich in vielen Fällen Hochkommas vermeiden.

### Start-Parameter setzen

Die SCRIPT-Datei eagle.scr wird jedesmal ausgeführt wenn eine Zeichnung neu in einem Editor-Fenster geladen wird, oder der Editier-Modus in einer Bibliothek gewechselt wird, sofern eagle.scr im Projekt-Verzeichnis oder im Script-Pfad »Seite 27 steht.

### Script-Datei im Bibliotheks-Editor ausführen

Alle Layer werden nur dann erkannt, wenn vorher der Bibliotheks-Editor neu geöffnet worden ist.

## SET

### Funktion

Systemparameter verändern.

### Syntax

```
SET
SET options;
```

Mit dem SET-Befehl können Parameter festgelegt werden, die das Verhalten des Programms, die Bildschirmdarstellung und die Bedieneroberfläche betreffen. Die genaue Syntax ist im folgenden beschrieben.

Wird der SET-Befehl ohne Parameter aufgerufen, so erscheint ein Dialog in dem alle Parameter eingestellt werden können.

### Benutzer-Interface

Fangfunktion	<pre>SET SNAP_LENGTH number;</pre> <p>Damit läßt sich der Grenzwert für die Fangfunktion des ROUTE »Seite 112-Befehls in der aktuellen Einheit einstellen. Default: 20 Mil</p> <p>Verlegt man mit dem ROUTE »Seite 112-Befehl Leitungen zu Pads, die nicht im Raster liegen, dann sorgt die Fangfunktion dafür, daß man innerhalb der Snap_length zu diesem Pad routen kann.</p> <pre>SET SNAP_BENDED ON   OFF;</pre> <p>Bei <i>wire_bend</i> 1 und 3 liegt der Knickpunkt des gerouteten Wires auf dem Raster, falls <i>On</i> eingestellt ist.</p> <pre>SET SELECT_FACTOR value;</pre> <p>Damit stellt man ein, bis zu welchem Abstand vom Cursor benachbarte Objekte zur Auswahl »Seite 31 vorgeschlagen werden. Der Wert wird relativ zur Höhe des gegenwärtigen Bildausschnitts angegeben. Default: 0.02 (2%).</p>
Inhalt von Menüs	<pre>SET USED_LAYERS name   number;</pre> <p>Legt die Layer fest, die in den entsprechenden EAGLE-Menüs angezeigt werden. Siehe Beispieldatei <i>mylayers.scr</i>. Die Layer Top, Bottom, Pads, Vias, Unrouted, Dimension, Drills und Holes sowie die Schaltplan-Layer bleiben auf jeden Fall im Menü. Auch Signal-Layer, in denen Leitungen verlegt sind, bleiben aktiv. <i>SET Used_Layers All</i> aktiviert alle Layer.</p> <pre>SET WIDTH_MENU value..; SET DIAMETER_MENU value..; SET DRILL_MENU value..; SET SMD_MENU value..; SET SIZE_MENU value..;</pre> <p>Für die Parameter <i>width</i> etc. kann der Inhalt der entsprechenden Popup-Menüs mit obigen Befehlen konfiguriert werden. Je Menü sind max. 16 Werte möglich (beim Smd-Menü max. 16 Wertepaare). Beispiel: Grid Inch; Set Width Menu 0.1 0.2 0.3;</p>
Knickwinkel für Wires	<pre>SET WIRE_BEND bend_nr;</pre> <p><i>bend_nr</i> kann von 0 bis 4 variiert werden. Für die einzelnen Werte gilt folgender Verlauf des Wires:</p> <ul style="list-style-type: none"> <li>0: Startpunkt - waagrecht - senkrecht - Endpunkt</li> <li>1: Startpunkt - waagrecht - 45° - Endpunkt</li> <li>2: Startpunkt - Endpunkt (direkte Verbindung)</li> <li>3: Startpunkt - 45° - waagrecht - Endpunkt</li> <li>4: Startpunkt - senkrecht - waagrecht - Endpunkt</li> </ul>
Piepston ein/aus	<pre>SET BEEP ON   OFF;</pre>

### Bildschirmdarstellung

Farbe für Grid-Linien	<pre>SET COLOR_GRID color;</pre>
Farbe für Layer	<pre>SET COLOR_LAYER layer color;</pre>
Füllmuster für Layer	<pre>SET FILL_LAYER layer fill;</pre>
Raster-Parameter	<pre>SET GRID_REDRAW ON   OFF;</pre> <p>Bildschirm bei Grid-Änderung auffrischen oder nicht.</p> <pre>SET MIN_GRID_SIZE pixels;</pre>

	Das Grid wird nur dann gezeichnet, wenn der Rasterabstand größer ist als die eingestellte Zahl von Pixeln.
Min. darg. Textgröße	SET MIN_TEXT_SIZE size; Texte, die weniger als size Bildpunkte hoch sind, werden auf dem Bildschirm als Rechtecke dargestellt. Einstellung 0 bedeutet: alle Texte werden lesbar dargestellt.
Netz-Linien-Darstellung	SET NET_WIRE_WIDTH width;
Pad-Darstellung	SET DISPLAY_MODE REAL   NODRILL; REAL: Pads werden dargestellt, wie sie geplottet werden. NODRILL: Pads werden ohne Bohrung dargestellt. SET PAD_NAMES ON   OFF; Pad-Namen werden ein-/ausgeblendet.
Bus-Linien-Darstellung	SET BUS_WIRE_WIDTH width;
DRC »Seite 63-Parameter	SET DRC_FILL fill_name;
Polygon-Berechnung	SET POLYGON_RATSNEST ON   OFF; Siehe POLYGON »Seite 99-Befehl.
Vector Font	SET VECTOR_FONT ON   OFF; Siehe TEXT »Seite 123-Befehl.

### Mode-Parameter

Package-Check	SET CHECK_CONNECTS ON   OFF; Der ADD »Seite 42-Befehl prüft, ob bei einem Device jedem Pad ein Pin (mit CONNECT »Seite 55) zugewiesen ist. Diese Prüfung läßt sich abschalten. Allerdings kann keine Platine aus einem Schaltplan erzeugt werden, falls ein Device ohne Gehäuse gefunden wird.
REPLACE »Seite 109-Modus	SET REPLACE_SAME NAMES   COORDS;
UNDO »Seite 125-Puffer ein/aus	SET UNDO_LOG ON   OFF;
Wire-Optim. ein/aus	SET OPTIMIZING ON   OFF; Wires, die nach MOVE, ROUTE oder SPLIT in einer Linie liegen, werden zu einem Wire zusammengefaßt, falls On eingestellt ist. Siehe auch OPTIMIZE »Seite 90.

Color, geordnet nach Farbnummern, die anstelle von color verwendet werden können. Damit legt man die Farbe fest:

0	Black
1	Blue
2	Green
3	Cyan
4	Red
5	Magenta
6	Brown
7	LGray
8	DGray
9	LBlue
10	LGreen
11	LCyan
12	LRed
13	LMagenta
14	Yellow
15	White

Fill legt die Art fest, wie Wires und Rectangles in einem bestimmten Layer gefüllt werden sollen. Auch dieser Parameter kann durch die am Anfang der Zeile stehende Zahl ersetzt werden:

0	Empty
1	Solid
2	Line
3	LtSlash
4	Slash
5	BkSlash
6	LtBkSlash
7	Hatch
8	XHatch
9	Interleave
10	WideDot
11	CloseDot

## SHOW

### Funktion

Anzeigen von Namen und Objekten.

### Syntax

```
SHOW • . .  
SHOW name . .
```

**Siehe auch** INFO »Seite 76

Der SHOW-Befehl dient zum Anzeigen von Namen, Elementen und Objekten. Er listet Parameter von Elementen und Objekten in der linken oberen Bildschirmecke auf. Mit SHOW kann man auch ganze Signale und Netze hervorheben (auf dem Bildschirm heller dargestellt).

### Quervergleich zwischen Platine und Schaltung

Bei aktivierter Forward&Back-Annotation »Seite 359 wird ein Objekt, das mit Hilfe des SHOW-Befehls heller dargestellt wird, sowohl im Schaltplan als auch in der Platine heller dargestellt.

### Mehrere Objekte

Wenn Sie mehrere Objekte mit dem SHOW-Befehl selektieren, wird jedes einzelne (der Reihe nach) hell dargestellt.

Beispiele:

```
Show ic1 ic2;
```

ic1 wird hell und wieder dunkel, dann wird ic2 hell,

```
Show ic1;
```

ic1 wird hell und bleibt hell,

```
Show ic2;
```

ic2 wird ebenfalls hell.

## SIGNAL

### Funktion

Signale definieren.

### Syntax

```
SIGNAL • • ..  
SIGNAL signal_name • • ..  
SIGNAL signal_name element_name pad_name..;
```

**Siehe auch** AUTO »Seite 48, ROUTE »Seite 112, NAME »Seite 87, CLASS »Seite 53, WIRE »Seite 131, RATSNEST »Seite 104, EXPORT »Seite 68

Mit dem SIGNAL-Befehl definiert man Signale, also die Verbindungen zwischen den Anschlüssen der Packages. Es sind mindestens zwei Stützstellen anzugeben, da sonst keine Luftlinie entstehen kann.

### Mauseingabe

Man selektiert mit der Maus der Reihe nach die Anschlüsse, die miteinander verbunden werden sollen. EAGLE stellt die Signale als Luftlinien im Unrouted-Layer dar.

Gibt man signal\_name mit ein, dann erhält das Signal den angegebenen Namen.

### Texteingabe

Man kann ein Signal aber auch vollständig textuell definieren. Die Eingabe

```
SIG GND IC1 7 IC2 7 IC3 7;
```

würde z.B. die Pads mit dem Namen '7' der ICs 1...3 miteinander verbinden. Denken Sie an diese Möglichkeit im Zusammenhang mit Script-Dateien. Sie können beispielsweise komplette Netzlisten mit Hilfe von Script-Dateien eingeben.

### Kurzschluß-Check

Versucht man, mit SIGNAL zwei Pads zu verbinden, die bereits unterschiedlichen Signalen angehören, dann wird in einem Popup-Menü nachgefragt, ob die beiden Signale verbunden werden sollen und welchen Namen sie erhalten sollen.

### Konturdaten

Der Signalname \_OUTLINES\_ gibt dem Polygon besondere Eigenschaften, die man zur Erzeugung von Konturdaten »Seite 156 (z. B. zum Fräsen von Prototypen) benötigt. Dieser Name sollte ansonsten nicht verwendet werden.

## SMASH

### Funktion

Loslösen von >NAME- und >VALUE-Texten.

### Syntax

SMASH • . .

**Siehe auch** NAME »Seite 87, VALUE »Seite 128

Den SMASH-Befehl wendet man auf Elemente an, damit man anschließend die zugehörigen Texte, die den aktuellen Namen und Wert (Value) repräsentieren, separat bewegen kann (MOVE). Das ist vor allem für Schalt- und Bestückungspläne nützlich.

Wird ein mit SMASH losgelöster Text gelöscht, erscheint wieder die ursprüngliche Darstellung.

Nach dem SMASH-Befehl kann man die >NAME- und >VALUE-Texte behandeln wie alle anderen Texte. Allerdings läßt sich ihr Inhalt nicht mit CHANGE TEXT ändern.

## SMD

### Funktion

Plazieren von Smds in Packages.

### Syntax

```
SMD [ x_width y_width ] [ -roundness ] [ 'name' ] •..
```

### Maus

Mittlere Maustaste wechselt zwischen Top- und Bottom-Layer.

**Siehe auch** PAD »Seite 92, CHANGE »Seite 51, NAME »Seite 87, ROUTE »Seite 112, Design Rules »Seite 159

Smd: Anschlußfläche für SMD-Bauelemente.

Der SMD-Befehl platziert einen SMD-Anschluß in einem Package. Die Eingabe der Länge und Breite vor dem Plazieren ändert die Größe des Smds. Die Parameter werden in der aktuellen Maßeinheit angegeben. Sie dürfen maximal 0.51602 Zoll (ca. 13.1 mm) betragen. Es ist allerdings sinnvoll, nur geradzahlige Maße zu verwenden.

Die eingegebene Smd-Größe bleibt für nachfolgende Operationen erhalten. **Roundness**

Der Wert für `roundness` kann ganzzahlig - mit negativen Vorzeichen, um es vom `width`-Parameter zu unterscheiden - zwischen 0 und 100 angegeben werden. Der Wert 0 erzeugt rechteckige SMDs, während der Wert 100 die Ecken der SMDs vollständig rundet. Der Befehl

```
SMD 50 50 -100 '1' •
```

erzeugt zum Beispiel ein rundes SMD mit dem Namen '1' an der Position des Mausklicks. Dieses kann man für ein BGA-Gehäuse (Ball Grid Array) verwenden.

### Namen

Smd-Namen werden vom Programm automatisch erzeugt und können mit dem NAME-Befehl geändert werden. Der Name kann als Parameter auch im SMD-Befehl mit angegeben werden (muß in Hochkommas eingeschlossen sein).

### Einzelne Smds

Einzelne Smds in Platinen sind als Package zu realisieren und dann in die Platine zu holen.

### Package verändern

Es ist nicht möglich, in einem Package, das in einem Device verwendet wird, nachträglich ein Smd hinzuzufügen oder zu löschen, da dies die im Device definierten Pin-/Pad-Zuordnungen (CONNECT-Befehl) verändern würde.



## SPLIT

### Funktion

Knicke in Wires einfügen.

### Syntax

SPLIT • • •

### Maus

Rechte Maustaste ändert den Knickwinkel (siehe SET Wire\_Bend »Seite 115).

### Tasten

F8: SPLIT aktiviert den SPLIT-Befehl.

**Siehe auch** MOVE »Seite 86, OPTIMIZE »Seite 90, SET »Seite 115

Den SPLIT-Befehl benötigt man, wenn nachträglich in Wires oder Polygonen noch eine Abknickung erforderlich ist. SPLIT teilt Wires am Anklickpunkt. Das kürzere Stück verläuft gemäß dem eingestellten Knickwinkel (Wire\_Bend), das längere verläuft in gerader Linie zum nächsten Aufhängepunkt. Nach dem SPLIT-Befehl werden die betroffenen Wire-Segmente wieder optimiert (entsprechend dem OPTIMIZE-Befehl), sofern nicht zuvor der Befehl

SET OPTIMIZING OFF;

eingegeben wurde. Hat man diesen Befehl eingegeben, bleiben die Trennstellen in den Wires erhalten. Sie bleiben auch dann erhalten, wenn man im SPLIT-Befehl dieselbe Stelle zweimal mit der Maus anklickt.

### Leitung verjüngen

Dazu selektiert man den SPLIT-Befehl, markiert den zu verjüngenden Abschnitt mit zwei Mausklicks, gibt den Befehl

CHANGE WIDTH breite

ein und klickt mit der Maus das gewünschte Segment an.

## TECHNOLOGY

### Funktion

Definiert eventuell vorhandene *Technologien* für ein Device-Set.

### Syntax

```
TECHNOLOGY name . . ;  
TECHNOLOGY -name . . ;  
TECHNOLOGY -* . . ;
```

**Siehe auch** PACKAGE »Seite 91

Dieser Befehl wird im Device-Editor verwendet, um die verschiedenen *Technologien* eines Bauteils im Device-Namen zu bestimmen. Einer der Namen, die mit dem TECHNOLOGY-Befehl definiert wurden, ersetzen den Platzhalter '\*' im Device-Set-Namen, sobald man das Device in einem Schaltplan platziert. Der Begriff *Technology* stammt von der hauptsächlichen Verwendung dieser Funktion verschiedene Varianten eines Devices zu erzeugen, die alle dasselbe Schaltplan-Symbol, dieselbe(n) Package-Variante(n) und dieselben Pin/Pad-Zuordnungen haben. Die Devices unterscheiden sich nur im Namen, der sich beispielsweise für die TTL-Bausteine im Bezug auf Ihre Technologie, wie "L", "LS", oder "HCT" unterscheiden.

Der TECHNOLOGY-Befehl kann nur angewendet werden, wenn schon vorher eine Package-Variante über den PACKAGE »Seite 91-Befehl definiert wurde.

Ist kein '\*'-Platzhalter im Device-Set-Namen angegeben, wird der Device-Set-Name um die Technologie-Angabe zu einem vollständigen Device-Namen ergänzt. Bitte beachten Sie, dass die Technologie vor der Package-Variante eingefügt wird. Ist weder '\*' noch '?' angegeben, setzt sich der Device-Name aus *Device-Set-Name+Technology+Package-Variante* zusammen.

Die Namen (name), die mit dem TECHNOLOGY-Befehl angegeben sind, werden zu einer schon vorhandenen Liste des aktuellen Devices hinzugefügt. Geben Sie einen Namen mit '-' an, wird diese Bezeichnung aus der Liste entfernt. Sollte der Name mit einem '-' beginnen, muss dieser in einfachen Hochkommas angegeben werden. '-' löscht alle Technologien.

Es sind nur die ASCII-Zeichen 33..126 als Technologie-Bezeichnung erlaubt. Die maximale Anzahl von Technologie-Varianten pro Device ist 254.

Die besondere "leere" Technologie kann in mit zwei Hochkommas angegeben werden, also '' (ohne Namen).

### Beispiel

In einem Device mit dem Namen "74\*00" löscht der Befehl

```
TECHNOLOGY -* '' L LS S HCT;
```

zunächst alle bisherigen Technologien und erzeugt anschließend die folgenden Technologie-Varianten:

```
7400  
74L00  
74LS00  
74S00  
74HCT00
```

## TEXT

### Funktion

Plazieren von Text.

### Syntax

```
TEXT beliebige_zeichen orientation •..  
TEXT 'beliebige_zeichen' orientation •..
```

### Maus

Rechte Maustaste dreht den Text.  
Mittlere Maustaste wechselt den Layer.

**Siehe auch** CHANGE »Seite 51, MOVE »Seite 86, MIRROR »Seite 85, PIN »Seite 95, ROTATE »Seite 111

Der TEXT-Befehl platziert einen Text in einer Zeichnung, oder in einem Bibliothekselement. Bei der Eingabe mehrerer Texte geht man sinnvollerweise so vor, daß man zuerst den TEXT-Befehl aktiviert, dann tippt man den ersten Begriff ein und setzt ihn mit der linken Maustaste ab, dann den zweiten usw.

### Schreibrichtung

Mit der rechten Maustaste dreht man den Text.

Als Option kann die Schreibrichtung (orientation) auch textuell angegeben werden. Das ist vor allem für Script-Dateien sinnvoll. Die entsprechenden Schlüsselwörter sind im PIN-Befehl aufgeführt (R0, R90 usw.).

Text wird immer so dargestellt, daß er von vorne oder von rechts zu lesen ist - auch wenn er rotiert wird. Nach zweimaligem Rotieren erscheint er deshalb wieder gleich, aber der Aufhängepunkt liegt nicht mehr links unten, sondern rechts oben. Denken Sie daran, wenn sich ein Text scheinbar nicht mehr selektieren läßt.

### Text auf Lötseite

Texte in den Layern Bottom und bPlace werden automatisch gespiegelt.

### Leerzeichen oder Strichpunkt

Sollen in einem Text mehrere aufeinanderfolgende Leerzeichen oder ein Strichpunkt enthalten sein, dann setzt man den ganzen String in Hochkommas. Sollen Hochkommas gedruckt werden, dann ist jedes einzelne in Hochkommas einzuschließen.

### Schlüsselwörter

Ist der TEXT-Befehl aktiv und enthält der einzugebende Text Wörter, die EAGLE für Befehle oder Orientation-Schlüsselwörter hält (z. B. und für UNDO), dann sind diese Wörter oder der gesamte Text in Hochkommas einzuschließen.

### Texthöhe

Die Zeichengröße und die Strichstärke ändert man mit den Befehlen:

```
CHANGE SIZE text_size •..  
CHANGE RATIO ratio •..
```

Maximale Texthöhe: ca. 2 Zoll.  
Maximale Strichstärke: 0.51602 Zoll (ca. 13.1 mm)  
Parameter "ratio": 0...31 (% der Texthöhe).

### Schriftart

Texte können in drei Schriftarten verwendet werden:

Vector    der programm-interne Vektor-Font  
Proportional    ein Proportional-Pixel-Font (üblicherweise 'Helvetica')  
Fixed    ein Monospaced-Pixel-Font (üblicherweise 'Courier')

Die Schriftart wird mit CHANGE verändert:

CHANGE FONT VECTOR|PROPORTIONAL|FIXED • . .

Das Programm versucht die Nicht-Vector-Schriftarten so gut wie möglich auszugeben. Da diese jedoch vom Grafik-Interface Ihres Systems gezeichnet werden, können bei Proportional- und Fixed-Schriftart Abweichungen in der Größe bzw. Länge entstehen.

Setzen Sie die Option "Always vector font" im User-Interface-Dialog »Seite 29, werden alle Texte mit dem programm-internen Vektor-Font dargestellt und ausgegeben. Diese Einstellung ist dann sinnvoll, wenn vom System die anderen Schriftarten nicht korrekt angezeigt werden.

Beim Anlegen eines neuen Boards oder Schaltplans wird die aktuelle Einstellung in der Zeichnungsdatei gespeichert. So wird sicher gestellt (auch bei der Weitergabe an Dritte, die evtl. mit anderen Einstellungen arbeiten), dass die Datei mit Ihren Einstellungen ausgegeben wird.

Verwenden Sie SET »Seite 115 VECTOR\_FONT ON|OFF, um die Einstellungen für ein bestehendes Layout oder einen Schaltplan zu ändern.

Wenn Sie Daten mit dem CAM-Prozessor erzeugen, werden Texte immer mit Vector-Font ausgegeben. Andere Fonts werden nicht unterstützt.

Soll ein Text in einem Nicht-Vector-Font von einem Signal-Polygon subtrahiert werden, wird nur das umschließende Rechteck ausgespart. Aufgrund der oben angeführten Probleme bzgl. Einhalten von Größe und Länge der Texte, kann es sein, dass der Text über das umschließende Rechteck hinausgeht. Sollten Sie also Texte von Polygonen subtrahieren wollen, ist es höchst empfehlenswert nur den Vector-Font zu verwenden.

Der Parameter *Ratio* hat für Nicht-Vector-Fonts keine Bedeutung.

### Zeichensatz

Eine korrekte Darstellung wird nur für die Zeichen im ASCII-Code unter 128 garantiert. Alle anderen Zeichen darüber können systemabhängig zu unterschiedlichen Darstellungen in den unterschiedlichen Schriftarten führen.

### Spezielle Platzhalter-Texte

>NAME	Bauteilname (ggf.+Gate-Name)	1)
>VALUE	Bauteilwert/-typ	1)
>PART	Bauteilname	2)
>GATE	Gate-Name	2)
>DRAWING_NAME	Zeichnungsname	
>LAST_DATE_TIME	Datum/Zeit der letzten Änderung	
>PLOT_DATE_TIME	Datum/Zeit der Plot-Erstellung	
>SHEET	Blattnummer eines Schaltplans	3)

1) Nur im Package und Symbol

2) Nur im Symbol

3) Nur im Symbol oder Schaltplan

## UNDO

### Funktion

Vorhergehende Befehle zurücknehmen.

### Syntax

UNDO;

### Tasten

F9:            UNDO   UNDO-Befehl ausführen. Alt+BS: UNDO

**Siehe auch** REDO »Seite 106, SET »Seite 115, Forward&Back-Annotation »Seite 359

Mit dem Befehl UNDO kann man Befehle rückgängig machen. Das ist insbesondere dann nützlich, wenn man z. B. versehentlich etwas gelöscht hat. Die mehrmalige Eingabe von UNDO macht entsprechend viele Befehle rückgängig. Das geht bis zum Zustand nach dem letzten EDIT-, OPEN-, AUTO oder REMOVE-Befehl. Diese Befehle löschen die Vorgeschichte.

Die UNDO-Funktion benötigt Platz auf der Platte und kann die Eingabe von Script-Dateien erheblich verlangsamen. Sie läßt sich bei Bedarf mit dem Befehl

```
SET UNDO_LOG OFF;
```

abschalten.

Nach UNDO kann es vorkommen, daß die Bildschirmdarstellung teilweise durch gelöschte Objekte scheinbar zerstört ist. Mit der Funktionstaste F2 (WINDOW;) kann man den Bildschirminhalt wieder auffrischen.

UNDO/REDO ist vollkommen in den Mechanismus der Forward&Back-Annotation integriert. +

## UPDATE

### Funktion

Aktualisiert Bibliotheks-Elemente.

### Syntax

```
UPDATE  
UPDATE;  
UPDATE library_name..;  
UPDATE +@ | -@ [library_name..];
```

**Siehe auch** ADD »Seite 42, REPLACE »Seite 109

Der UPDATE-Befehl vergleicht Bauteile in einem Layout oder Schaltplan mit den zugehörigen Bibliothekselementen und aktualisiert diese, sofern Unterschiede festgestellt werden.

Wird der UPDATE-Befehl ohne Parameter aufgerufen, öffnet sich ein File-Dialog, aus dem man die Bibliothek wählt, dessen Bauteile mit den Definitionen in der Zeichnung verglichen werden soll.

Wird der Befehl ohne Parameter mit einem ' ; ' abgeschlossen, werden alle Bauteile überprüft.

Werden eine oder mehrere Bibliotheken angegeben, überprüft das Programm alle Bauteile aus diesen Bibliotheken. Der Bibliotheksname kann entweder als einfacher Name (wie "ttl" oder "ttl.lbr") oder mit voller Pfadangabe (wie "/home/mydir/myproject/ttl.lbr" oder "../lbr/ttl") angegeben werden.

Falls der erste Parameter ' +@ ' ist, werden die Namen der angegebenen Bibliotheken (oder aller Bibliotheken, falls keine angegeben wurden) um das ' @ ' -Zeichen gefolgt von einer Zahl erweitert. Dies kann dazu benutzt werden um sicherzustellen, daß die in einer Zeichnung enthaltenen Bibliotheken nicht verändert werden wenn ein Bauteil aus einer neueren Bibliothek gleichen Namens in die Zeichnung eingefügt wird. Bibliotheksnamen die bereits mit dem ' @ ' -Zeichen gefolgt von einer Zahl enden werden nicht verändert.

Falls der erste Parameter ' -@ ' ist, wird das ' @ ' -Zeichen (gefolgt von einer Zahl) vom Ende der angegebenen Bibliotheksnamen (oder aller Bibliotheksnamen, falls keine angegeben wurden) entfernt. Dies funktioniert natürlich nur dann, wenn sich noch keine Bibliothek mit diesem neuen Namen in der Zeichnung befindet.

Bitte beachten Sie, daß "UPDATE +@;" gefolgt von "UPDATE -@;" (und umgekehrt) nicht unbedingt die ursprüngliche Folge von Bibliotheksnamen ergeben muß, da die Reihenfolge, in der die Namen bearbeitet werden, von der Reihenfolge der Bibliotheken in der Zeichnungsdatei abhängt.

Die Bibliotheksdefinitionen, die in einem Schaltplan oder Board gespeichert sind, werden nur anhand des Bibliotheksnamens (z. B. "ttl") identifiziert. Bei der Entscheidung ob das Bauteil überprüft werden soll oder nicht, wird nur dieser Name berücksichtigt. Die Bibliotheken werden in den unter "Libraries" im Directories-Dialog »Seite 27 angegebenen Verzeichnissen, von links nach rechts, gesucht. Die erste Bibliothek mit dem angegebenen Namen, die in den Verzeichnissen gefunden wird, wird verwendet. Bitte beachten Sie, dass bei den Bibliotheksnamen nicht zwischen Groß- und Kleinschreibung unterschieden wird. Es ist nicht relevant ob die gesuchte Bibliothek zur Zeit "in use" ist oder nicht. Kann eine Bibliothek nicht gefunden werden, findet auch kein Update statt. In diesem Fall wird keine Fehlermeldung ausgegeben.

Wird der UPDATE-Befehl in einem Schaltplan oder Board gestartet, und sind diese über die Forward&Back Annotation »Seite 359 verbunden, aktualisiert EAGLE die Bauteile in beiden Dateien.

In manchen Fällen wird es notwendig sein anzugeben, ob Gates, Pins oder Pads aufgrund ihres Namen oder ihrer Koordinaten zugeordnet werden sollen. Das ist dann der Fall, wenn die zugehörigen Bibliothekselemente verschoben oder umbenannt wurden. Wenn zuviele Änderungen gemacht wurden (z. B. wurde ein Pin verschoben und umbenannt), ist ein automatisches Aktualisieren nicht möglich. In diesem Fall sollte man die Bibliotheksänderung entweder in mehreren Schritten machen (erst Umbenennen, dann Verschieben) oder das ganze Element umbenennen, so dass es nicht getauscht wird.

**Achtung:** Nach jedem Library Update sollten Sie unbedingt einen Design Rule Check »Seite 63 (DRC) und einen Electrical Rule Check »Seite 65 (ERC) durchführen!

## USE

### Funktion

Bibliothek zur Benutzung markieren.

### Syntax

```
USE
USE -*;
USE library_name..;
```

**Siehe auch** ADD »Seite 42, REPLACE »Seite 109

Der USE-Befehl markiert eine Bibliothek so, dass sie für die Befehle ADD »Seite 42 oder REPLACE »Seite 109 verfügbar ist.

Rufen Sie den USE-Befehl ohne Parameter auf, öffnet sich ein File-Dialog aus dem man eine Bibliotheksdatei auswählen kann. Falls für Bibliotheken im "Options/Directories"-Dialog ein Pfad definiert wurde, erscheinen im File-Dialog die Bibliotheken aus diesem Verzeichnis.

Der spezielle Parameter - \* bewirkt, daß alle bisher markierten Bibliotheken aufgegeben werden.

library\_name kann der volle Name einer Bibliothek oder ein teilqualifizierter Name sein. Falls library\_name der Name eines Verzeichnisses ist, werden alle Bibliotheken aus diesem Verzeichnis markiert.

Der Suffix .lbr braucht nicht angegeben zu werden.

EAGLE übernimmt die komplette Bibliotheksinformation in die Zeichnung, deshalb ist die Bibliothek zum Bearbeiten fertiger Platinen nicht mehr erforderlich.

Änderungen an einer Bibliothek wirken sich nicht auf Elemente in den schon bestehenden Zeichnungen aus. Siehe UPDATE »Seite 126-Befehl, um Bauteile durch aktualisierte Bibliothekselemente zu ersetzen.

### Auswählen der Bibliotheken über das Control Panel

Wenn eine Bibliothek, die Sie benutzen wollen im Bibliotheks-Editor verändert und noch nicht gespeichert wurde, werden Sie gefragt, ob die Datei jetzt gespeichert werden soll oder nicht. Beantworten Sie diese Frage mit *Yes*, wird die Datei gespeichert und Sie benutzen die modifizierte Datei. Antworten Sie mit *No*, wird die Datei nicht gespeichert und Sie benutzen die Bibliothek unverändert, so wie sie auf der Festplatte gespeichert ist. Die Schaltfläche *Cancel* bricht den Befehl ab, so dass weder die Datei gespeichert noch diese Bibliothek "in use" ist. Bibliotheken können im Control Panel »Seite 23 als "in use" markiert werden, indem Sie auf den Marker klicken, der seine Farbe ändert, um anzuzeigen, ob die Bibliothek "in use" ist, oder durch die Auswahl des Punkts "Use" im Kontext-Menü des Bibliothekseintrags in der Baumansicht. Im Kontext-Menü gibt es die Möglichkeit alle *all* oder keine *none* der Bibliotheken zu wählen.

### Bibliotheken "in use" und Projekte

Die Bibliotheken, die "in use" sind, werden in der Projekt-Datei (eagle.epf) gespeichert, sofern ein Projekt geladen ist.

### Beispiele

USE	öffnet den Datei-Dialog zur Auswahl einer Bibliothek
USE -*;	gibt alle vorher mit USE markierten Bibliotheken auf
USE demo trans*;	markiert die Bibliothek demo.lbr und alle Bibliotheken mit Namen trans*.lbr
USE -* C:\eagle\lbr;	gibt zunächst alle bereits markierten Bibliotheken auf und markiert dann alle Bibliotheken aus dem Verzeichnis C:\eagle\lbr

## VALUE

### Funktion

Elementwert eintragen und ändern.

### Syntax

```
VALUE •..  
VALUE wert •..  
VALUE name wert ..  
VALUE ON;  
VALUE OFF;
```

**Siehe auch** NAME »Seite 87, SMASH »Seite 119

### In Platine und Schaltplan

Elemente kann man mit einem Wert versehen, etwa 10k bei einem Widerstand. Bei ICs trägt man anstelle des Wertes sinnvollerweise den Typ ein (z. B. 7400). Den Wert bzw. Typ trägt man mit dem VALUE-Befehl ein. Der Befehl selektiert das nächstgelegene Element und öffnet ein Popup-Menü, in dem man einen neuen Wert festlegen oder den bisherigen verändern kann.

Gibt man wert an, bevor man das Element mit der Maus selektiert, dann erhalten alle nachfolgend selektierten Elemente diesen Wert. Das ist sehr praktisch, wenn man z. B. eine ganze Reihe von ICs auf denselben Wert setzen will.

Werden name und wert angegeben, so erhält das Element name den angegebenen Wert.

### Beispiel

```
VALUE R1 10k R2 100k
```

Hier wurde mehreren Elementen in einem Befehl je ein Wert zugewiesen. Diese Möglichkeit läßt sich auch in Script-Dateien nach folgendem Muster ausnutzen:

```
VALUE R1 10k \  
      R2 100k \  
      R3 5.6k \  
      C1 10uF \  
      C2 22nF \  
      ...
```

Der Backslash ("\") verhindert, daß in der nächsten Zeile ein Parameter mit einem Schlüsselwort verwechselt wird.

### Im Device: Wert oder Typ

Wendet man den VALUE-Befehl im Device-Editier-Modus an, dann sind die Parameter On und Off zulässig.

On: Anstelle des Platzhalters VALUE (im Symbol definiert) kann im Schaltplan der aktuelle Wert eingegeben werden.

Off: Anstelle des Platzhalters VALUE erscheint im Schaltplan der Device-Name (z.B. 74LS00N). Er läßt sich im Schaltplan nur nach Rückfrage mit dem VALUE-Befehl verändern.



## VIA

### Funktion

Plazieren von Durchkontaktierungen in Platinen.

### Syntax

```
VIA •...  
VIA 'signal_name' diameter shape •...
```

**Siehe auch** SMD »Seite 120, CHANGE »Seite 51, DISPLAY »Seite 61, SET »Seite 115, PAD »Seite 92, Design Rules »Seite 159

Der VIA-Befehl plaziert ein Via in einer Platine. Dabei fügt er das Via zu einem Signal hinzu (falls es auf einer Leitung plaziert wird) und führt eine Kurzschlußprüfung durch.

### Signalname

Der Parameter `signal_name` ist in erster Linie für die Anwendung in Script-Dateien gedacht, die generierte Daten einlesen. Wenn ein Signalname angegeben ist, werden alle folgenden Vias mit diesem Signal verbunden, und es wird keine automatische Prüfung durchgeführt.

**Diese Möglichkeit ist mit großer Vorsicht einzusetzen, da es zu Kurzschlüssen kommen kann, wenn ein Via so plaziert wird, daß es unterschiedliche Signale verbindet. Bitte führen Sie deshalb einen Design Rule Check »Seite 63 durch, nachdem Sie den VIA-Befehl mit dem Parameter `signal_name` benutzt haben!**

### Via-Durchmesser und Bohrdurchmesser

Die Eingabe eines Durchmessers vor dem Plazieren ändert die Größe des Vias. Der Durchmesser wird in der aktuellen Maßeinheit angegeben. Er darf maximal 0.51602 Zoll (ca. 13.1 mm) betragen.

Die eingegebene Größe bleibt für nachfolgende Operationen erhalten.

Der Bohrdurchmesser entspricht dem Durchmesser, der für Pads eingestellt ist. Er läßt sich mit

```
CHANGE DRILL durchmesser •
```

einstellen und verändern.

Vias erzeugen Bohrsymbole im Layer Drills und die LötStopmaske in den Layern tStop/bStop.

### Via-Form

Ein Via kann eine der folgenden Formen (shape) haben:

Square	quadratisch
Round	rund
Octagon	achteckig

Die Via-Form kann entweder (wie der Durchmesser) eingegeben werden, während der VIA-Befehl aktiv ist, oder sie kann mit dem Befehl

```
CHANGE SHAPE shape •
```

verändert werden.

Die eingegebene Form bleibt für nachfolgende Operationen erhalten.

Beachten Sie bitte, daß die tatsächlichen Werte für Via-Form und -Durchmesser durch die Design Rules »Seite 159 des Boards bestimmt werden, in dem das Via verwendet wird.

## WINDOW

### Funktion

Bildausschnitt festlegen oder Bild auffrischen.

### Syntax

```
WINDOW;  
WINDOW •;  
WINDOW • •;  
WINDOW • • •  
WINDOW scale_factor  
WINDOW FIT
```

### Maus

Mit gedrückter linker Taste ein Rechteck aufziehen entspricht "• •;".

### Tasten

Alt+F2:	WINDOW FIT	Zeichnung formatfüllend darstellen
F2:	WINDOW;	Bild auffrischen
F3:	WINDOW 2	Hineinzoomen um Faktor 2
F4:	WINDOW 0.5	Herauszoomen um Faktor 2
F5:	WINDOW (@);	Neues Zentrum an akt. Cursor-Pos. (falls Befehl aktiviert)

Der WINDOW-Befehl legt den sichtbaren Ausschnitt der Zeichnung fest. Ohne weitere Parameter frischt der Befehl das Bild auf.

### Neues Zentrum

Der WINDOW-Befehl mit einem Mausklick legt diesen Punkt in die Fenstermitte und läßt den Maßstab unverändert. Den Bildausschnitt können Sie auch mit den Slidern am Rand des Arbeitsbereichs verschieben. Mit F5 legen Sie die Position des Mausursors als neues Zentrum fest.

### Eckpunkte festlegen

Der WINDOW-Befehl mit zwei Mausklicks legt einen Ausschnitt fest, bei dem beide Punkte gerade noch innerhalb des Fensters liegen - eine sehr bequeme Möglichkeit für Ausschnittvergrößerungen. Das Seitenverhältnis der Zeichnung wird nicht geändert.

### Neues Zentrum und zoomen

Der WINDOW-Befehl mit drei Mausklicks legt einen Ausschnitt fest, bei dem der erste Punkt im Zentrum liegt. Das Verhältnis des Abstandes von Punkt 1 zu Punkt 2 und von Punkt 1 zu Punkt 3 legt den Vergrößerungsfaktor fest. Dabei gilt: Liegt der dritte Punkt weiter entfernt vom ersten als der zweite, dann erscheinen die Objekte größer.

### Zoomen

WINDOW 2

vergrößert die Darstellung der Objekte um Faktor zwei.

WINDOW 0.5

verkleinert die Darstellung der Objekte um Faktor zwei.

### Zeichnung formatfüllend

WINDOW FIT

stellt die gesamte Zeichnung im Fenster dar.

## WIRE

### Funktion

Wires (Linien) zeichnen.

### Syntax

```
WIRE • •..  
WIRE 'signal_name' wire_width • •..  
WIRE • wire_width •..
```

### Maus

Rechte Maustaste ändert den Knickwinkel (siehe SET Wire\_Bend »Seite 115).  
Mittlere Maustaste wählt den Layer aus.

**Siehe auch** SIGNAL »Seite 118, ROUTE »Seite 112, CHANGE »Seite 51, NET »Seite 88, BUS »Seite 50, DELETE »Seite 58, RIPUP »Seite 110

Der WIRE-Befehl platziert Wires (Linien) in einer Zeichnung, und zwar zwischen erstem und zweitem Koordinatenpunkt. Jeder weitere Punkt (Mausklick) wird mit dem vorhergehenden verbunden. Dabei werden jeweils zwei Koordinatenpunkte mit einer geraden Linie verbunden oder mit zwei, die in einem bestimmten Winkel abknicken. Dieser Knickwinkel lässt sich mit der rechten Maustaste weiterschalten.

Zwei Mausklicks an derselben Stelle setzen das Leitungsstück ab.

### Signalname

Der Parameter `signal_name` ist in erster Linie für die Anwendung in Script-Dateien gedacht, die generierte Daten einlesen. Wenn ein Signalname angegeben ist, werden alle folgenden Wires mit diesem Signal verbunden, und es wird keine automatische Prüfung durchgeführt.

**Diese Möglichkeit ist mit großer Vorsicht einzusetzen, da es zu Kurzschlüssen kommen kann, wenn ein Wire so platziert wird, daß er unterschiedliche Signale verbindet. Bitte führen Sie deshalb einen Design Rule Check »Seite 63 durch, nachdem Sie den WIRE-Befehl mit dem Parameter `signal_name` benutzt haben!**

### Strichstärke

Gibt man den Befehl mit dem Parameter `wire_width` (z. B. 0.1) ein, dann wird dadurch die Linienbreite in der aktuellen Maßeinheit festgelegt. Zulässig ist maximal 0.51602 Zoll (ca. 13.1 mm). Die Linienbreite bleibt für nachfolgende Operationen erhalten.

Die Breite lässt sich auch zu jeder Zeit mit dem Befehl

```
CHANGE WIDTH breite •
```

ändern oder voreinstellen.

Bitte verwenden Sie den WIRE-Befehl nicht für Netze und Busse sowie für Luftlinien. Siehe NET »Seite 88, BUS »Seite 50 und SIGNAL »Seite 118.

### Linienarten

Linien können in folgenden Arten (*styles*) gezeichnet werden::

- Continuous - durchgezogen
- LongDash - (lang) gestrichelt
- ShortDash - (kurz) gestrichelt
- DashDot - Strich-Punkt-Linie

Die Linienart kann mit dem CHANGE »Seite 51-Befehl verändert werden.

DRC und Autorouter behandeln alle Linienarten als durchgezogen (Continuous). Andere Linienarten werden hauptsächlich für elektrische oder mechanische Zeichnungen verwendet und sollten nicht in Signallayern benutzt werden. Der DRC meldet eigens einen Fehler, wenn Sie eine Nicht-continuous-Linie als Teil einer signalführenden Leiterbahn mit einem Pad verbinden.

### Signale in Top-, Bottom und Route-Layern

Wires in den Layern Top, Bottom, Route2.. werden als Signale behandelt. Wird ein Wire in einem der Signal-Layer an einem bestehenden Signal angesetzt, so gehört der gesamte gezeichnete Wire-Zug zu diesem Signal (nur, wenn die Wire-Enden bzw. das Wire-Ende und der Pad-Mittelpunkt genau übereinstimmen). Setzt man das Ende eines Wires an einem anderen Signal ab, fragt EAGLE zur Bestätigung nach, ob die beiden Signale wirklich miteinander verbunden werden sollen. Jedes Geradenstück wird von EAGLE (z. B. beim RIPUP-Befehl) als eigenes Objekt behandelt.

## WRITE

### Funktion

Abspeichern einer Zeichnung oder Bibliothek.

### Syntax

```
WRITE;  
WRITE drawing_name  
WRITE @drawing_name
```

Der WRITE-Befehl sichert eine Zeichnung oder eine Bibliothek im Projektverzeichnis »Seite 27. Man kann einen neuen Namen wählen oder denjenigen beibehalten, unter dem die Zeichnung/Bibliothek geladen wurde.

Dem Namen kann man auch einen Pfadnamen voranstellen, wenn die Datei in ein bestimmtes Verzeichnis gesichert werden soll.

Wird dem neuen Namen ein @ vorangestellt, so wird auch der Name der geladenen Zeichnung entsprechend geändert. Die zugehörige Platine/Schaltung wird dann automatisch ebenfalls unter diesem Namen abgespeichert, und der Undo-Puffer wird gelöscht.

Um die Konsistenz der Forward&Back-Annotation »Seite 359 zwischen Platine und Schaltung zu gewährleisten, verhält sich der WRITE-Befehl wie folgt:

- Wenn eine Platine/Schaltung unter demselben Namen gespeichert wird, wird die zugehörige Schaltung/Platine ebenfalls gespeichert, sofern sie geändert wurde.
- wenn eine Platine/Schaltung unter einem unterschiedlichen Namen gespeichert wurde, fragt das Programm, ob die zugehörige Schaltung/Platine ebenfalls unter diesem Namen gespeichert werden soll.
- Beim Speichern unter einem unterschiedlichen Namen wird das "Modified"-Flag nicht gelöscht.

## **Ausgabedaten erzeugen**

- Ausdruck mit PRINT »Seite 135
- CAM-Prozessor »Seite 139
- Konturdaten »Seite 156

## Drucken

Die Parameter für das Drucken auf den Systemdrucker können mit folgenden drei Dialogen eingestellt werden:

- Drucken einer Zeichnung »Seite 136
- Drucken eines Textes »Seite 137
- Seiteneinrichtung »Seite 138

**Siehe auch** PRINT »Seite 102

## Drucken einer Zeichnung

Wenn Sie den PRINT »Seite 102-Befehl ohne abschließenden ' ; ' eingeben, oder wenn Sie **Print** aus dem Kontext-Menü »Seite 25 des Icons einer Zeichnung im Control Panel »Seite 23 auswählen, erhalten Sie einen Dialog mit folgenden Optionen:

### **Mirror**

Spiegelt die Ausgabe.

### **Rotate**

Dreht die Ausgabe um 90°.

### **Upside down**

Dreht die Ausgabe um 180°. Zusammen mit **Rotate**, wird die Zeichnung um insgesamt 270° gedreht.

### **Black**

Ignoriert die Farbeinstellungen der Layer und druckt alles in Schwarz.

### **Solid**

Ignoriert die Füllmuster der Layer und druckt alles voll ausgefüllt.

### **Scale factor**

Skaliert die Zeichnung mit dem gegebenen Wert.

### **Page limit**

Gibt an wieviele Blätter der Ausdruck maximal haben soll. Falls die Zeichnung nicht auf die angegebene Zahl von Blättern paßt, wird der Scale factor so lange verkleinert, bis sie paßt. Der Defaultwert von 0 bedeutet "kein Limit".

### **All**

Alle Sheets des Schaltplans werden ausgedruckt (das ist der Defaultwert wenn **Print** aus dem Kontext-Menü »Seite 25 eines Schaltplan-Icons ausgewählt wird).

### **From...to**

Nur die angegebenen Sheets werden ausgedruckt.

### **This**

Es wird nur das Sheet ausgedruckt, das gerade editiert wird (das ist der Defaultwert wenn der PRINT »Seite 102-Befehl in einem Schaltplan-Editor Fenster verwendet wird).

### **Printer...**

Ruft den System-Druckerdialog auf, in dem der Drucker ausgewählt werden kann sowie druckerspezifische Parameter (wie etwa Papiergröße und Blattausrichtung) eingestellt werden können.

### **Page...**

Ruft den Dialog zur Seiteneinrichtung »Seite 138 auf.



## Drucken eines Textes

Wenn Sie **P**rint aus dem Kontext Menü »Seite 25 des Icons einer Textdatei im Control Panel »Seite 23 oder aus dem **F**ile Menü des Text Editors »Seite 37 auswählen, erhalten Sie einen Dialog mit folgenden Optionen:

### **Wrap**

Schaltet den Zeilenumbruch für zu lange Zeilen ein.

### **Printer...**

Ruft den System-Druckerdialog auf, in dem der Drucker ausgewählt werden kann sowie druckerspezifische Parameter (wie etwa Papiergröße und Blattausrichtung) eingestellt werden können.

### **Page...**

Ruft den Dialog zur Seiteneinrichtung »Seite 138 auf.

## Seiteneinrichtung

Der Seiteneinrichtungs-Dialog bestimmt wie eine Zeichnung oder ein Text auf dem Papier erscheinen soll.

### Border

Definiert den linken, oberen, rechten und unteren Rand. Die Werte werden entweder in Millimeter oder Inch angegeben, je nachdem, welche Einheit weniger Dezimalstellen ergibt.

Die Defaultwerte für die Ränder werden vom Druckertreiber übernommen und definieren die maximal bedruckbare Fläche. Sie können hier auch kleinere Werte angeben, wobei es von Ihrem Drucker abhängt, ob die angegebenen Ränder dann eingehalten werden können oder nicht.

Nach der Auswahl eines anderen Druckers kann es sein, daß neue gerätespezifische Grenzen wirksam werden; die vorgegebenen Ränder werden dann automatisch vergrößert, falls der neue Drucker dies erfordert. Beachten Sie bitte, daß die Werte nicht automatisch verkleinert werden, auch wenn der neue Drucker kleinere Werte zulassen würde. Um die kleinstmöglichen Werte für die Ränder zu ermitteln, geben Sie in jedes Feld 0 ein. Dieser Wert wird dann durch das gerätespezifische Minimum ersetzt.

### Calibrate

Falls Sie mit Ihrem Drucker Produktionsvorlagen erstellen wollen, kann es nötig sein, den Drucker zu kalibrieren um exakte 1:1 Ausdrücke Ihrer Layouts zu erhalten.

Der Wert im X Feld gibt den Kalibrierungsfaktor in der Richtung an, in der sich der Druckkopf bewegt. Der Wert im Y Feld kalibriert die Koordinaten in Papiervorschubrichtung.

**ACHTUNG: Wenn Sie mit Ihrem Drucker Produktionsvorlagen erzeugen, prüfen Sie bitte immer das Druckergebnis auf Einhaltung der exakten Maße!**

Die Defaultwerte von 1 gehen davon aus, daß der Drucker in beiden Richtungen exakt druckt.

### Vertical

Definiert die vertikale Ausrichtung der Zeichnung: Top (oben), Center (mittig) oder Bottom (unten).

### Horizontal

Definiert die horizontale Ausrichtung der Zeichnung: Left (links), Center (mittig) or Right (rechts).

### Caption

Aktiviert die Ausgabe einer Bildunterschrift mit Datum und Uhrzeit des Ausdrucks sowie dem Dateinamen.

Bei gespiegelter Ausgabe enthält die Bildunterschrift das Wort "mirrored", und falls der Vergrößerungsfaktor nicht 1.0 ist, wird er als **f=...** mit angegeben (der Vergrößerungsfaktor wird mit 4 Nachkommastellen ausgegeben, so daß auch eine Angabe von **f=1.0000** nicht bedeutet, daß der Faktor *exakt* 1.0 ist).

### Printer...

Ruft den System-Druckerdialog auf, in dem der Drucker ausgewählt werden kann sowie druckerspezifische Parameter (wie etwa Papiergröße und Blattausrichtung) eingestellt werden können.

## CAM-Prozessor

Mit dem CAM-Prozessor können Sie jede Layer-Kombination an ein Peripheriegerät oder in eine Datei ausgeben.

Die folgenden Help-Themen führen Sie durch die erforderlichen Schritte, von der Auswahl des Daten-Files bis zur Konfiguration des Ausgabegeräts (Device).

- Datei auswählen »Seite 140
- Device auswählen »Seite 142
- Output-File wählen »Seite 153
- Plot-Layer wählen »Seite 155
- Device-Parameter einstellen »Seite 143
- Flag-Optionen einstellen »Seite 154

Sie können verschiedene Parametersätze zu einem CAM-Prozessor-Job »Seite 141 zusammenstellen, mit dessen Hilfe Sie einen kompletten Satz von Ausgabedateien durch Anklicken eines Buttons erzeugen können.

**Siehe auch** Drucken auf dem System-Drucker »Seite 135

## CAM-Prozessor-Hauptmenü

Im *CAM-Prozessor-Hauptmenü* können Sie wählen, von welcher Datei die Ausgabe generiert werden soll, Sie können Blenden- und Bohrer-Konfigurationsdateien bearbeiten oder Job-Dateien laden und sichern.

### File

<u>O</u> pen	<u>B</u> oard... Board-Datei für Ausgabe laden
	<u>D</u> rill rack... Bohrer-Konfigurationsdatei zum Editieren laden
	<u>W</u> heel... Blenden-Konfigurationsdatei zum Editieren laden
	<u>J</u> ob... Anderen Job laden oder neuen erzeugen
<u>S</u> ave job...	Gegenwärtigen Job sichern
<u>C</u> lose	CAM-Prozessor-Fenster schließen
<u>E</u> xit	Programm beenden

### Layer

<u>D</u> eselect all	Alle Layer deselektieren
<u>S</u> how selected	Nur die selektierten Layer anzeigen
<u>S</u> how <u>a</u> ll	Alle Layer anzeigen

### Help

<u>G</u> eneral help	Allgemeine Help-Seite öffnen
<u>C</u> ontents	Inhaltsverzeichnis der Help-Funktion öffnen
<u>C</u> AM Processor	CAM-Prozessor-Help öffnen
<u>J</u> ob help	Help zum Job-Mechanismus anzeigen
<u>D</u> evice help	Help zu Ausgabe-Devices anzeigen

## CAM-Prozessor-Job

Ein CAM-Prozessor-*Job* besteht aus unterschiedlichen *Sections*, von denen jede einen kompletten Satz von CAM-Prozessor-Parametern mit einer bestimmten Layer-Auswahl darstellt.

Ein typischer CAM-Prozessor-Job könnte zum Beispiel zwei *Sections* enthalten: eine, die die Fotoplot-Dateien für die Bestückungsseite erzeugt, und eine weitere, die die entsprechenden Daten für die Lötseite erzeugt.

### Section

Der *Section*-Selektor zeigt die gegenwärtig aktive Job-*Section* an. Durch Anklicken des Buttons können Sie jede der vorher mit dem *Add*-Button definierten *Sections* auswählen.

### Prompt

Wenn Sie in dieses Feld einen Text eintragen, gibt der CAM-Prozessor diese Meldung aus, bevor er die zugehörige Job-*Section* bearbeitet. Wenn Sie zum Beispiel vor jeder Ausgabe das Papier in den Stiftplotter einlegen wollen, könnte die Meldung "Bitte Papier wechseln!" lauten. Jede *Section* kann ihre eigene Meldung haben. Wenn keine Meldung definiert ist, wird die *Section* ohne vorherige Unterbrechung ausgeführt.

### Add

Klicken Sie den *Add*-Button an, um dem Job eine neue *Section* hinzuzufügen. Sie werden dann nach dem Namen der neuen *Section* gefragt. Für die neue *Section* gelten die Parametereinstellungen, die im Menü zu sehen sind. Bitte achten Sie darauf, wenn Sie eine neue Job-*Section* anlegen, daß Sie **zuerst mit 'Add' die neue Section anlegen** und erst **danach die Parameter modifizieren**. Wenn Sie zuerst die Parameter der gegenwärtigen *Section* modifizieren und erst danach mit 'Add' die neue *Section* anlegen, werden Sie vom Programm gefragt, ob Sie die Änderungen an der gegenwärtigen *Section* abspeichern wollen oder nicht.

### Del

Durch Anklicken des *Del*-Buttons löschen Sie die gegenwärtige Job-*Section*. Bevor die *Section* gelöscht wird, müssen Sie die Rückfrage, ob sie wirklich gelöscht werden soll, bestätigen.

### Process Section

Der *Process Section*-Button startet die Datenausgabe für die gegenwärtig angezeigte *Section*.

### Process Job

Der *Process Job*-Button startet die Datenausgabe für den gesamten Job. Dabei wird die zuerst definierte *Section* zuerst bearbeitet. Es entstehen die gleichen Ausgabedaten, als würden Sie der Reihe nach die unterschiedlichen *Sections* auswählen und mit dem *Process Section*-Button starten.

## Ausgabetreiber (Output Device)

Der Ausgabetreiber (*Output Device*) legt fest, welche Art von Daten der CAM-Prozessor erzeugt. Sie können aus den unterschiedlichsten Treiber den geeigneten auswählen, z.B. für Foto-Plotter, Bohrstationen etc.

### Device

Durch Anklicken des Device-Selectors öffnen Sie eine Liste aller verfügbaren Device-Treiber.

### Scale

Bei Geräten, die eine Skalierung erlauben, können Sie in dieses Feld einen Skalierfaktor eintragen. Werte über 1 führen zu einer Vergrößerung, Werte unter 1 verkleinern die Ausgabe.

Sie können die Größe der Ausgabe auf eine bestimmte Anzahl von Seiten beschränken, indem Sie einen negativen Wert im Scale-Feld eingeben. In diesem Fall wird der Skalierfaktor auf 1.0 voreingestellt und so lange verkleinert, bis die Zeichnung gerade noch auf die angegebene Anzahl von Seiten passt. Wird zum Beispiel "-2" eingegeben, so entsteht eine Zeichnung die nicht mehr als zwei Seiten benötigt. Beachten Sie bitte, daß die zur Verfügung stehende Blattgröße (Width und Height Parameter) Ihres Ausgabegerätes richtig eingestellt sein muß, damit dieser Mechanismus funktioniert. Diese Größen können in den Width- und Height-Feldern oder durch Editieren der Datei EAGLE.DEF eingestellt werden.

### File

Sie können in dieses Feld den Namen der Ausgabedatei »Seite 153 direkt eingeben, oder Sie klicken den File-Button an, um einen Dialog für die Definition der Ausgabedatei zu öffnen.

Wenn Sie den Dateinamen aus dem Namen der Schaltplan- oder Platinen-Datei ableiten wollen, können Sie den Namen teilweise angeben (mindestens die Extension, z.B. .GBR). In diesem Fall wird der Rest des Dateinamens von der Quelldatei abgeleitet.

### Wheel

Sie können in dieses Feld den Namen der Blenden-Konfigurationsdatei »Seite 144 direkt eingeben, oder Sie klicken den Wheel-Button an, um einen Datei-Dialog zu öffnen und die Datei zu selektieren.

Wenn Sie den Dateinamen aus dem Namen der Schaltplan- oder Platinen-Datei ableiten wollen, können Sie den Namen teilweise angeben (mindestens die Extension, z.B. .WHL). In diesem Fall wird der Rest des Dateinamens von der Quelldatei abgeleitet.

### Rack

Sie können in dieses Feld den Namen der Bohrer-Konfigurationsdatei »Seite 147 direkt eingeben, oder Sie klicken den Rack-Button an, um einen Datei-Dialog zu öffnen und die Datei zu selektieren.

Wenn Sie den Dateinamen aus dem Namen der Schaltplan- oder Platinen-Datei ableiten wollen, können Sie den Namen teilweise angeben (mindestens die Extension, z.B. .DRL). In diesem Fall wird der Rest des Dateinamens von der Quelldatei abgeleitet.

## Device-Parameter

Abhängig vom gewählten Ausgabetreiber »Seite 142 gibt es verschiedene treiberspezifische Parameter, mit denen Sie die Ausgabe an Ihre Bedürfnisse anpassen können.

- Blenden-Konfigurationsdatei »Seite 144
- Blenden-Emulation »Seite 145
- Blenden-Toleranzen »Seite 146
- Bohrer-Konfigurationsdatei »Seite 147
- Bohrer-Toleranzen »Seite 148
- Offset »Seite 149
- Seitengröße »Seite 150
- Stiftdaten »Seite 151

## Blenden-Konfigurationsdatei

Dem Fotoplotter muß bekannt sein, welche Blenden den Codes in der Ausgabedatei entsprechen. Diese Zuordnung ist in der Blenden-Konfigurationsdatei definiert.

### Beispiel

D010	annulus	0.004 x 0.000
D010	round	0.004
D040	square	0.004
D054	thermal	0.090 x 0.060
D100	rectangle	0.060 x 0.075
D104	oval	0.030 x 0.090
D110	draw	0.004



## Blenden-Emulation

Wenn die Option "Apertures" gewählt ist, werden nicht vorhandene Blenden mit kleineren Blenden emuliert. Ist sie ausgeschaltet, werden keine Blenden emuliert, auch nicht Thermal- oder Annulus-Blenden.

Die Optionen "Annulus" und "Thermal" werden gewählt, wenn bei eingeschalteter Blendenemulation zusätzlich Annulus- und/oder Thermal-Symbole emuliert werden sollen.

Achtung: Die Blendenemulation kann zu sehr langen Plot-Zeiten führen (hohe Kosten!).

## Blenden-Toleranzen

Falls Sie Toleranzen für Draw- bzw. Blitz-Blenden (Flash) angeben, verwendet der CAM-Prozessor Blenden innerhalb dieser Toleranz, falls keine mit dem exakten Maß verfügbar ist.

Toleranzen werden in Prozent angegeben.

**Bitte beachten Sie, daß dadurch Ihre "Design Rules" unter Umständen nicht mehr eingehalten werden!**

## Bohrer-Konfigurationsdatei

Der Bohrstation muß bekannt sein, welche Bohrer den Codes in der Ausgabedatei entsprechen. Diese Zuordnung ist in der Bohrer-Konfigurationsdatei definiert.

Die Datei kann mit Hilfe eines User-Language-Programms DRILLCFG.ULP, das sich im ULP-Verzeichnis Ihrer EAGLE-Installation befindet, erzeugt werden. Verwenden Sie dazu den Befehl RUN »Seite 113.

### Beispiel

T01	0.010
T02	0.016
T03	0.032
T04	0.040
T05	0.050
T06	0.070

## **Bohrer-Toleranzen**

Falls Sie eine Toleranz für Bohrer angeben, verwendet der CAM-Prozessor Bohrer innerhalb dieser Toleranz, falls keiner mit dem exakten Maß verfügbar ist.

Toleranzen werden in Prozent angegeben.

## **Offset**

Offset in x- und y-Richtung (Inch, Dezimalzahl)

Kann dazu verwendet werden den Nullpunkt von großformatigen Plottern in die linke untere Ecke zu verlegen.

## **Bedruckbarer Bereich**

### **Height**

Bedruckbarer Bereich in y-Richtung (Inch).

### **Width**

Bedruckbarer Bereich in x-Richtung (Inch).

Bitte beachten Sie, daß der CAM-Prozessor die Zeichnung auf mehrere Teile aufteilt, falls das umschließende Rechteck um alle in der Datei enthaltenen Objekte (auch in Layern, die nicht ausgegeben werden) nicht auf die bedruckbare Fläche paßt.

## Stiftdaten

### Diameter

Stift-Durchmesser in mm: Wird beim Füllen von Flächen zur Berechnung der notwendigen Anzahl von Linien benutzt.

### Velocity

Stiftgeschwindigkeit in cm/s (bei Stift-Plottern, die unterschiedliche Geschwindigkeiten unterstützen). Die Plotter-Default-Geschwindigkeit wählt man mit dem Wert 0.

## **Eigenen Device-Treiber definieren**

Die Ausgabetreiber sind in der Textdatei EAGLE.DEF definiert. Dort finden Sie Details, wie man einen eigenen Treiber definiert. Am besten kopieren Sie einen Block eines existierenden Treibers für denselben Gerätetyp und passen dann die Parameter an.

Bitte verwenden Sie einen Texteditor »Seite 37, der keine Steuerzeichen in die Datei schreibt.



## Ausgabedaten

Das *Output File* enthält die Daten, die vom CAM-Prozessor erzeugt werden.

Folgende Dateinamen sind üblich:

=====		
Datei- name	Selekt. Layer	Bedeutung
=====		
*.cmp	Top, Via, Pad	Bauteilseite
*.ly2	Route2, Via, Pad	Multilayer-Innenlage
*.ly3	Route3, Via, Pad	Multilayer-Innenlage,
*.ly4	\$User1	Multilayer-Versorgungslage
...		...
*.sol	Bot, Via, Pad	Lötseite
*.plc	tPl, Dim, tName,	Bestückungspl. Bauteilseite
*.pls	bPl, Dim, bName,	Bestückungsplan Lötseite
*.stc	tStop	LötStopmaske Bauteilseite
*.sts	bStop	LötStopmaske Lötseite
*.drd	Drills, Holes	Bohrdaten für NC-Bohrm.
=====		

### Alternative Benennung der Ausgabedateien

Falls bei Output ".EXT" eingetragen ist, entsteht die Ausgabedatei "boardname.EXT".

Um zu vermeiden, daß in einem Job von der nachfolgenden Section die Gerber-Info-Datei überschrieben wird, können Sie als Output-File-Name ".\*" wählen (dabei steht "\*" für beliebige in Dateinamen erlaubte Zeichen). Das Output-File erhält dann den Namen "boardname.\*x" und die Info-Datei erhält den Namen "boardname.xxi". Wenn z.B. die Platine "myboard.brd" geladen ist, und als Output-File-Name ".cp#" gewählt wurde, dann heißt das Output-File "myboard.cpx" und die Gerber-Info-Datei "myboard.cpi".

## Flag-Options

### **Mirror**

Ausgabe spiegeln. Achtung, dabei entstehen negative Koordinaten, deshalb sollte gleichzeitig Funktion "pos. Coord." eingeschaltet sein.

### **Rotate**

Die Zeichnung wird um 90 Grad gedreht. Achtung, dabei entstehen negative Koordinaten, deshalb sollte gleichzeitig die Funktion "pos. Coord." eingeschaltet sein.

### **Upside down**

Die Zeichnung wird um 180 Grad gedreht. Zusammen mit Rotate wird die Zeichnung um insgesamt 270 Grad gedreht. Achtung, dabei entstehen negative Koordinaten, deshalb sollte gleichzeitig Funktion "pos. Coord." eingeschaltet sein.

### **Pos. Coord**

Die Zeichnung wird so ausgegeben, daß keine negativen Koordinaten vorkommen. Sie wird an die Koordinatenachsen herangeschoben. Achtung: negative Koordinaten führen bei vielen Peripheriegeräten zu Fehlern!

### **Quickplot**

Beschleunigte Ausgabe, bei der nur die Umrisse von Objekten erscheinen.

### **Optimize**

Mit dieser Option schalten Sie die Wegoptimierung für die Plotterausgabe ein oder aus.

### **Fill pads**

Pads füllen. Diese Funktion ist nur mit Treiber des Typs "generic", wie z. B. mit PostScript möglich. Wird die Option deselektiert, sind die Bohrungen in Pads und Vias sichtbar.

## Layer und Farben

Wählen Sie die auszugebenden Layer, indem Sie die Check-Boxes in der Layer-Liste anklicken.

Bei Peripheriegeräten, die unterschiedliche Farben unterstützen, geben Sie die Farbnummer für den jeweiligen Layer an.

Folgende Layer-Kombinationen und Output-File-Namen »Seite 153 sind üblich:

=====		
Datei-	Selekt. Layer	Bedeutung
name		
=====		
*.cmp	Top, Via, Pad	Bauteilseite
*.ly2	Route2, Via, Pad	Multilayer-Innenlage
*.ly3	Route3, Via, Pad	Multilayer-Innenlage,
*.ly4	\$User1	Multilayer-Versorgungslage
...		...
*.sol	Bot, Via, Pad	Lötseite
*.plc	tPl, Dim, tName,	Bestückungspl. Bauteilseite
*.pls	bPl, Dim, bName,	Bestückungsplan Lötseite
*.stc	tStop	LötStopmaske Bauteilseite
*.sts	bStop	LötStopmaske Lötseite
*.drd	Drills, Holes	Bohrdaten für NC-Bohrm.
=====		

## Konturdaten

EAGLE kann Konturdaten erzeugen, die beispielsweise zum Fräsen von Prototyp-Platinen verwendet werden können.

Das User-Language-Programm *outlines.ulp* enthält alle Schritte um diese Daten zu erzeugen. Die folgende ausführliche Beschreibung zeigt was zu tun ist, um Konturdaten zu generieren.

### Board vorbereiten

Zuerst definieren Sie ein POLYGON »Seite 99 in dem Layer für den die Konturdaten erzeugt werden sollen. Das Polygon muss folgende Eigenschaften haben:

- es muss den Namen `_OUTLINES_` haben
- es muss das **einzigste** Element mit dem Signalnamen `_OUTLINES_` sein
- der Wert für *Rank* muss '6' sein
- der Wert für *Width* muss dem Durchmesser des Fräswerkzeugs entsprechen
- es muss groß genug sein, um die ganze Board-Fläche zu bedecken

Wenn sich ein solches Polygon in Ihrem Layout befindet, berechnet es der RATSNEST »Seite 104-Befehl in der Weise, dass seine *Konturen* genau den Linien entsprechen, die mit dem Fräswerkzeug gefahren werden müssen, um die einzelnen Signale voneinander freizufräsen. Die *Füllung* des berechneten Polygons bestimmt, was weggefräst werden muss, um alle unnötigen Kupferflächen zu entfernen.

### Ausgeben der Daten

Die Konturdaten können über ein User Language Program »Seite 161 aus dem Board extrahiert werden. Die Datei *outlines.ulp*, die mit EAGLE geliefert wird, enthält diesen Prozess vollständig. Wenn Sie ein eigenes ULP schreiben wollen, verwenden Sie am besten *outlines.ulp* als Ausgangsbasis. Sehen Sie die Hilfe-Seite zu `UL_POLYGON` »Seite 221 für weitere Details über das Erzeugen von Konturdaten mit Hilfe eines Polygons.

### Durchmesser des Fräswerkzeugs

Der Durchmesser des Fräasers (und somit die Strichbreite *width* des Polygons) muss so klein sein, dass es möglich ist zwischen zwei unterschiedlichen Signalen durchzukommen, um diese zu isolieren. **Führen Sie in jedem Fall einen Design Rule Check »Seite 63 (DRC) mit den Werten für Mindestabstände zwischen unterschiedlichen Signalen (*Clearance*) durch, die dem Durchmesser des Fräswerkzeugs entsprechen!**

Werte ungleich 0 für den Isolate-Parameter können dafür verwendet werden, beim sequentiellen Arbeiten mit unterschiedlichen Fräserdurchmessern bereits gefräste Bereiche auszusparen.

### Herstellen des ursprünglichen Zustands im Board

Stellen Sie sicher, dass Sie nach dem Erzeugen der Konturdaten das Polygon mit dem Namen `_OUTLINES_` löschen. Dieses spezielle Polygon verursacht ansonsten Kurzschlüsse im Board, da es nicht den üblichen Design Rules »Seite 159 entspricht!

## **Autorouter**

Der integrierte Autorouter kann vom Board-Fenster aus mit dem Befehl AUTO »Seite 48 gestartet werden.

Bitte überprüfen Sie Ihre Lizenz »Seite 364, um festzustellen, ob Sie Zugriff zum Autorouter haben.

## Überprüfen des Designs

EAGLE hat zwei Befehle zur Überprüfung Ihres Designs:

- Electrical Rule Check (ERC »Seite 65)
- Design Rule Check (DRC »Seite 63)

Der ERC wird in einem Schaltungs-Fenster ausgeführt. Er überprüft das Design auf elektrische Konsistenz.

Der DRC wird in einem Platinen-Fenster ausgeführt. Er überprüft das Design auf Überlappung von unterschiedlichen Potentialen, Abstandsverletzungen etc.

## Design Rules

*Design Rules* legen alle Parameter fest denen ein Platinen-Layout entsprechen muss.

Der Design Rule Check »Seite 63 prüft das Board gegenüber diesen Regeln und meldet Verstöße gegen sie.

Die Design Rules eines Boards können mit Hilfe des Design-Rule-Dialogs modifiziert werden. Der Dialog öffnet sich, wenn man den DRC »Seite 63-Befehl ohne abschließendes ' ; ' aufruft.

Neu angelegte Boards übernehmen ihre Design Rules aus der Datei 'default.dru', die in dem ersten Verzeichnis, das im "Options/Directories/Design rules"-Pfad aufgeführt ist, gesucht wird. Ist keine solche Datei vorhanden, so gelten die programminternen Defaultwerte.

**Hinweis** zu den Werten für **Clearance** und **Distance**: da die interne Auflösung der Koordinaten 1/10000mm beträgt kann der DRC nur Fehler größer als 1/10000mm zuverlässig melden.

### File

Das *File*-Tab zeigt eine Beschreibung des aktuellen Satz von Design Rules und erlaubt über *Change* seine Beschreibung zu verändern (das ist empfohlen, wenn Sie eigene Regeln definieren). Über *Load* kann man einen anderen Satz von Design Rules aus einer Datei laden, *Save as..* speichert die aktuellen Einstellungen in einer dru-Datei.

Bitte beachten Sie, dass die Design Rules immer im Board-File gespeichert werden, so dass diese Regeln auch für die Produktion der Platine bei Weitergabe der brd-Datei an den Leiterplatten-Hersteller gelten.

### Clearance

Im *Clearance*-Tab definiert man verschiedene Mindestabstände zwischen Objekten in den Signallayern. Das sind üblicherweise Mindestwerte, die vom Fertigungsprozess beim Leiterplatten-Hersteller vorgegeben werden. Sprechen Sie sich dazu mit dem Hersteller ab.

Der aktuelle Mindestabstand zwischen Objekten, die zu unterschiedlichen Signalen gehören, werden auch von den Werten der unterschiedliche Netzklassen beeinflusst.

Bitte beachten Sie, dass ein Polygon mit dem besonderen Namen `_OUTLINES_` dazu verwendet wird Kontur-Daten »Seite 156 zu erzeugen und dieses die Design Rules **nicht** einhält.

### Distance

Im *Distance*-Tab legt man die Mindestabstände zwischen Objekten in den Signallayern und dem Platinenrand (Dimension) und zwischen Bohrungen (Holes) fest. Achtung: Es werden nur Signale gegenüber Dimension geprüft, die auch tatsächlich an mindestens einem Pad oder Smd angeschlossen sind. So ist es erlaubt, Eckwinkel zur Markierung der Platinenbegrenzung in den Signallayern zu zeichnen ohne dass der DRC Fehler meldet.

Aus Gründen der Kompatibilität zu Version 3.5x gilt: Wird der Parameter für den Mindestabstand zwischen Kupfer und Dimension auf 0 gesetzt, so werden Objekte im Dimension-Layer beim Freirechnen der Polygone nicht mehr berücksichtigt (ausgenommen Holes, die immer berücksichtigt werden). Es findet dann auch keine Abstandsprüfung zwischen Kupfer und Dimension mehr statt.

### Sizes

Unter *Sizes* legt man die Mindestbreite von Objekten in Signallayern und den Mindestbohrdurchmesser fest. Diese Werte sind absolute Minimalmaße, die vom Herstellungsprozess der Platine bestimmt werden. Sprechen Sie sich hierzu mit dem Leiterplatten-Hersteller ab.

Die Mindestbreite von Leiterbahnen und der Mindestbohrdurchmesser von Durchkontaktierungen kann außerdem für unterschiedliche Netzklassen festgelegt werden.

### Restring

Im *Restring*-Tab definiert man die Mindestbreite des Kupferrings, die nach dem Bohren eines Pads oder Vias um die Bohrung herum stehen bleibt. Die Werte werden in Prozent des Bohrdurchmessers angegeben. Außerdem kann ein Minimal- und ein Maximalwert festgelegt werden. Die Restringbreiten für Pads können im Top-, Bottom- und in den Innen-Layern unterschiedlich sein, während bei Durchkontaktierungen (Vias) nur zwischen Außen- und Innenlagen unterschieden wird.

Wenn für den tatsächliche Durchmesser eines Pads (in der Bibliothek festgelegt) oder eines Vias ein größerer Wert vorgegeben wird, wird dieser in den Außenlagen verwendet. Pads werden beim Anlegen von Packages in der Regel mit dem Durchmesser 0 gezeichnet, so dass der Restring vollständig in Abhängigkeit des

Bohrdurchmessers berechnet werden kann.

### Shapes

Unter *Shapes* definiert man die Formen der Smds und Pads.

Smds werden üblicherweise als Rechtecke (mit "Roundness" = 0) in der Bibliothek definiert. Wenn Sie in Ihrem Design gerundete Smds verwenden wollen, kann man hier einen Rundungsfaktor (Roundness) angeben.

Pads werden normalerweise als Octagon (längliche Octagons wo sinnvoll) in der Bibliothek festgelegt. In den drei Combo-Boxen können Sie festlegen, ob die Pads im Layout so verwendet werden wie sie auch in der Bibliothek definiert wurden, oder ob alle rechteckig, rund oder octagonal sein sollen. Das kann man für Top- und Bottom-Layer separat definieren.

### Supply

Unter *Supply* legt man die Abmessungen für Thermal- und Annulus-Symbole fest, die in Versorgungslagen verwendet werden.

Bitte beachten Sie, dass das tatsächliche Aussehen dieser Symbole von ihrer Definition abweichen kann, wenn man Daten mit Photoplottern erzeugt, die besondere Thermal/Annulus-Blenden verwenden.

### Masks

Im *Masks*-Tab legt man die Abmessungen von Lötstop- (solder stop) und Lotpasten-Symbolen (cream mask) fest. Sie werden in Prozent der kleineren Seite eines Smds, Pads oder Vias angegeben und werden durch einen Minimal- bzw. Maximalwert begrenzt.

Eine Lötstopmaske wird automatisch für Smds, Pads und solche Vias erzeugt, die den angegebenen Wert für den Bohrdurchmesser im Parameter "Limit" überschreiten.

Die Lotpastenmaske (cream frame) wird nur für Smds erzeugt.

### Misc

Unter *Misc* kann man die Rasterprüfung (grid) bzw. die Winkelprüfung (angle) aktivieren und die Anzahl der angezeigten Fehler begrenzen.



## User Language

Die EAGLE-User-Language gestattet den Zugriff auf die EAGLE-Datenstrukturen und kann beliebige Ausgabedateien erzeugen.

Um diese Eigenschaft zu Nutzen, müssen Sie ein User-Language-Programm (ULP) schreiben »Seite 162 und anschließend ausführen »Seite 163.

Die folgenden Abschnitte beschreiben die User Language im Detail:

Syntax »Seite 164	Syntax-Regeln
Daten-Typen »Seite 186	Definiert die grundlegenden Datentypen (Data types)
Objekt-Typen »Seite 193	Definiert die EAGLE-Objekte (Objects)
Definitionen »Seite 233	Zeigt, wie man eine Definition schreibt
Operatoren »Seite 237	Liste der gültigen Operatoren (Operators)
Ausdrücke »Seite 244	Zeigt, wie man einen Ausdruck (Expression) schreibt
Statements »Seite 251	Definiert die gültigen Statements
Builtins »Seite 263	Liste der Builtin-Constants, -Functions etc.
Dialogs »Seite 319	zeigt wie man grafische Dialoge in ein ULP integriert

## Schreiben eines ULP

Ein User-Language-Programm ist eine reine Textdatei und wird in einer C-ähnlichen Syntax »Seite 164 geschrieben. User-Language-Programme verwenden die Extension `.ULP`. Sie können ein ULP mit jedem beliebigen Texteditor schreiben, vorausgesetzt, er fügt keine Steuerzeichen ein, oder Sie können den EAGLE-Texteditor »Seite 37 verwenden.

Ein User-Language-Programm besteht aus zwei wesentlichen Bestandteilen: Definitionen »Seite 233 und Statements »Seite 251.

Definitionen »Seite 233 werden verwendet, um Konstanten, Variablen und Funktionen zu definieren, die wiederum in Statements »Seite 251 verwendet werden.

Ein einfaches ULP könnte so aussehen:

```
#usage "Add the characters in the word 'Hello'\n"
      "Usage: RUN sample.ulp"
// Definitions:
string hello = "Hello";
int count(string s)
{
    int c = 0;
    for (int i = 0; s[i]; ++i)
        c += s[i];
    return c;
}
// Statements:
output("sample") {
    printf("Count is: %d\n", count(hello));
}
```

Der Wert der `#usage` »Seite 169-Directive zeigt im Control Panel »Seite 23 die Beschreibung des Programms an.

Soll das Ergebnis des ULPs ein Befehl sein, der im Editor-Fenster ausgeführt werden soll, kann man die Funktion `exit()` »Seite 284 verwenden um den Befehl an das Editor-Fenster zu schicken.

## ULP ausführen

User-Language-Programme werden mit Hilfe des RUN »Seite 113-Befehls von der Kommandozeile eines Editor-Fensters aus ausgeführt.

Ein ULP kann die Information zurückgeben, ob es erfolgreich abgeschlossen wurde oder nicht. Sie können die `exit()` »Seite 284-Funktion verwenden, um das Programm zu beenden und den Rückgabewert (return value) zu setzen.

Ein "return value" von 0 bedeutet, das ULP wurde normal beendet (erfolgreich), während jeder andere Wert einen unnormalen Programmabbruch anzeigt.

Der Default-Rückgabewert jedes ULP ist 0.

Wird der RUN »Seite 113-Befehl als Teil einer Script-Datei »Seite 114, ausgeführt, dann wird die Script-Datei abgebrochen, wenn das ULP mit einem "return value" ungleich 0 beendet wurde.

Eine spezielle Variante der Funktion `exit()` »Seite 284 kann verwendet werden, um einen Befehl als Ergebnis des ULPs an ein Editor-Fenster zu schicken.

## Syntax

Die Grundbausteine eines User-Language-Programms sind:

- Whitespace »Seite 165
- Kommentare »Seite 166
- Direktiven »Seite 167
- Schlüsselwörter (Keywords) »Seite 170
- Identifier »Seite 171
- Konstanten »Seite 172
- Punctuator-Zeichen »Seite 178

Alle unterliegen bestimmten Regeln, die in den entsprechenden Abschnitten beschrieben werden.

## Whitespace

Bevor ein User-Language-Programm ausgeführt werden kann, muß es von einer Datei eingelesen werden. Während dieses Vorgangs wird der Inhalt der Datei zerlegt (*parsed*) in Tokens und in *Whitespace*.

Leerzeichen (blanks), Tabulatoren, Newline-Zeichen und Kommentar »Seite 166 werden als *Whitespace* behandelt und nicht berücksichtigt.

Die einzige Stelle, an der ASCII-Zeichen, die *Whitespace* repräsentieren, berücksichtigt werden, ist innerhalb von Literal Strings »Seite 172, wie in

```
string s = "Hello World";
```

wo das Leerzeichen zwischen 'o' und 'W' ein Teil des Strings bleibt.

Wenn dem abschließenden Newline-Zeichen einer Zeile ein Backslash (\), vorausgeht, werden Backslash und Newline nicht berücksichtigt.

```
"Hello \  
World"
```

wird als "Hello World" interpretiert.

## Kommentare

Wenn man ein ULP schreibt, sollte man möglichst erklärenden Text hinzufügen, der einen Eindruck davon vermittelt, was dieses Programm tut. Sie können auch Ihren Namen und, falls verfügbar, Ihre Email-Adresse hinzufügen, damit die Anwender Ihres Programms die Möglichkeit haben, mit Ihnen Kontakt aufzunehmen, wenn Sie Probleme oder Verbesserungsvorschläge haben.

Es gibt zwei Möglichkeiten, Kommentare einzufügen. Die erste verwendet die Syntax

```
/* some comment text */
```

bei der alle Zeichen zwischen (und einschließlich) den Zeichen `/*` und `*/` als Kommentar interpretiert wird. Solche Kommentare können über mehrere Zeilen gehen, wie in

```
/* This is a  
   multi line comment  
*/
```

aber sie lassen sich nicht verschachteln. Das erste `*/` das einem `/*` folgt, beendet den Kommentar.

Die zweite Möglichkeit, einen Kommentar einzufügen, verwendet die Syntax

```
int i; // some comment text
```

Dabei werden alle Zeichen nach (und einschließlich) dem `//` bis zum Newline-Zeichen (aber nicht einschließlich) am Ende der Zeile als Kommentar interpretiert.

## Direktiven

Folgende *Direktiven* sind verfügbar:

```
#include »Seite 168  
#usage »Seite 169
```

## #include

Ein ULP kann Befehle aus einem anderen ULP durch die `#include`-Direktive ausführen. Die Syntax lautet

```
#include "filename"
```

Die Datei `filename` wird zuerst im selben Verzeichnis in dem sich auch die Source-Datei (das ist die Datei mit der `#include`-Direktive) befindet, gesucht. Wird sie dort nicht gefunden, wird in den angegebenen ULP-Verzeichnissen gesucht.

Die maximale "include-Tiefe" ist 10.

Jede `#include`-Direktive wird nur **einmal** ausgeführt. So wird sichergestellt, dass keine Mehrfachdefinitionen von Variablen oder Funktionen entstehen, die Fehler verursachen könnten.

### Hinweis zur Kompatibilität zwischen den Betriebssystemen

Enthält `filename` eine Pfadangabe, ist es das beste als Trennzeichen immer den **Forward-Slash (/)** zu verwenden (auch unter Windows!). Laufwerksbuchstaben unter Windows sollten vermieden werden. Wird das berücksichtigt, läuft das ULP unter allen Betriebssystemen.



## #usage

Jedes User-Language-Programm sollte Informationen über seine Funktion, die Benutzung und evtl. auch über den Autor enthalten.

Die Direktive

```
#usage text
```

ist die übliche Methode diese Information verfügbar zu machen.

Wird die #usage-Direktive verwendet, wird ihr Text (der eine String-Konstante »Seite 176 sein muss) im Control Panel »Seite 23 verwendet, um die Beschreibung des Programms anzuzeigen.

Für den Fall, dass das ULP diese Information z. B. in einer dlgMessageBox() »Seite 323 benötigt, ist dieser Text durch die Builtin-Konstante »Seite 264 `usage` im ULP verfügbar.

Es wird nur die #usage-Direktive des Hauptprogramms (das ist die Datei, die mit dem RUN »Seite 113-Befehl gestartet wurde) berücksichtigt. Deshalb sollten reine include »Seite 168-Dateien auch eine eigene #usage-Directive enthalten.

Am besten ist die #usage-Direktive an den Anfang der Datei zu stellen, so muss das Control Panel nicht den ganzen Text der Datei durchsuchen, um die Informationen, die angezeigt werden sollen, zu finden.

### Beispiel

```
#usage "A sample ULP\n"
      "Implements an example that shows how to use the EAGLE User
Language\n"
      "Usage: RUN sample.ulp\n"
      "Author: john@home.org"
```

## Schlüsselwörter (Keywords)

Die folgenden *Schlüsselwörter* sind für spezielle Zwecke reserviert und dürfen nicht als normale Identifier-Namen verwendet werden:

```
break »Seite 255
case »Seite 261
char »Seite 187
continue »Seite 256
default »Seite 261
do »Seite 257
else »Seite 259
enum »Seite 234
for »Seite 258
if »Seite 259
int »Seite 188
numeric »Seite 235
real »Seite 189
return »Seite 260
string »Seite 190
switch »Seite 261
void »Seite 186
while »Seite 262
```

Zusätzlich sind die Namen von Builtins »Seite 263 und Objekt-Typen »Seite 193 reserviert und dürfen nicht als Identifier-Namen verwendet werden.

## Identifizier

Ein *Identifizier* ist ein Name, der dazu benutzt wird, eine benutzerdefinierte Konstante »Seite 234, Variable »Seite 235 oder Funktion »Seite 236 einzuführen.

Identifizier bestehen aus einer Sequenz von Buchstaben (a b c..., A B C...), Ziffern (1 2 3...) und Unterstreichungszeichen (\_). Das erste Zeichen eines Identifiziers **muß** ein Buchstabe oder ein Unterstreichungszeichen sein.

Identifizier sind case-sensitive, das bedeutet, daß

```
int Number, number;
```

zwei unterschiedliche Integer-Variablen definieren würde.

Die maximale Länge eines Identifiziers ist 100 Zeichen, von denen alle signifikant sind.

## Konstanten

Konstanten sind gleichbleibende Daten, die in ein User-Language-Programm geschrieben werden. Analog zu den verschiedenen Datentypen »Seite 186 gibt es auch unterschiedliche Typen von Konstanten.

- Character-Konstanten »Seite 173
- Integer-Konstanten »Seite 174
- Real-Konstanten »Seite 175
- String-Konstanten »Seite 176

## Character-Konstanten

Eine *Character-Konstante* besteht aus einem einzelnen Buchstaben oder einer Escape-Sequenz »Seite 177, eingeschlossen in einfachen Hochkommas, wie in

```
'a'  
'='  
'\n'
```

Der Typ der Character-Konstante ist `char` »Seite 187.

## Integer-Konstanten

Abhängig vom ersten (eventuell auch vom zweiten) Zeichen wird eine *Integer-Konstante* unterschiedlich interpretiert:

erstes	zweites	Konstante interpretiert als
0	1-7	oktal (Basis 8)
0	x, X	hexadezimal (Basis 16)
1-9		dezimal (Basis 10)

Der Typ einer Integer-Konstante ist `int` »Seite 188.

### Beispiele

16	dezimal
020	oktal
0x10	hexadezimal

## Real-Konstanten

Eine *Real-Konstante* folgt dem allgemeinen Muster

`[ - ] int . frac [ e | E [ ± ] exp ]`

wobei die einzelnen Teile für

- Vorzeichen (optional)
- Dezimal-Integer
- Dezimalpunkt
- Dezimalbruch
- e oder E und ein Integer-Exponent mit Vorzeichen

stehen.

Sie können entweder Dezimal-Integer-Zahl oder Dezimalbruch weglassen (aber nicht beides). Sie können entweder den Dezimalpunkt oder den Buchstaben e oder E und den Integer-Exponenten mit Vorzeichen weglassen (aber nicht beides).

Der Typ einer Real-Konstante ist `real` »Seite 189.

### Beispiele

Konstante	Wert
23 . 45e6	23.45 x 10 <sup>6</sup>
.0	0.0
0 .	0.0
1 .	1.0
-1 . 23	-1.23
2e-5	2.0 x 10 <sup>-5</sup>
3E+10	3.0 x 10 <sup>10</sup>
.09E34	0.09 x 10 <sup>34</sup>

## String-Konstanten

Eine *String-Konstante* besteht aus einer Sequenz von Buchstaben oder einer Escape-Sequenz »Seite 177, eingeschlossen in doppelte Anführungszeichen, wie in

```
"Hello world\n"
```

Der Typ einer String-Konstante ist `string` »Seite 190.

String-Konstanten können jede beliebige Länge haben, vorausgesetzt es steht genügend Speicher zur Verfügung.

String-Konstanten können mit dem einfach aneinandergereiht werden um längere Strings zu bilden:

```
string s = "Hello" " world\n";
```

Es ist auch möglich, eine String-Konstante über mehrere Zeilen zu schreiben, indem man das Newline-Zeichen mit Hilfe des Backslash (\) "ausblendet":

```
string s = "Hello \  
world\n";
```



## Escape-Sequenzen

Eine *Escape-Sequenz* besteht aus einem Backslash (\), gefolgt von einem oder mehreren Sonderzeichen:

Sequenz	Bedeutung
\a	audible bell
\b	backspace
\f	form feed
\n	new line
\r	carriage return
\t	horizontal tab
\v	vertical tab
\\	backslash
\'	single quote
\"	double quote
\O	O = bis 3 octal digits
\xH	H = bis 2 hex digits

Jedes Zeichen nach dem Backslash, das nicht in der Liste aufgeführt ist, wird als dieses Zeichen (ohne Backslash) behandelt.

Escape-Sequenzen können in Character-Konstanten »Seite 173 und String-Konstanten »Seite 176 verwendet werden.

### Beispiele

```
'\n'  
"A tab\tinside a text\n"  
"Ring the bell\a\n"
```

## Punctuator-Zeichen

Die *Punctuator-Zeichen*, die in einem User-Language-Programm benutzt werden können, sind

[ ]	Eckige Klammern (Brackets) »Seite 179
( )	Runde Klammern (Parentheses) »Seite 180
{ }	Geschweifte Klammern (Braces) »Seite 181
,	Komma »Seite 182
;	Semicolon »Seite 183
:	Doppelpunkt (Colon) »Seite 184
=	Gleichheitszeichen »Seite 185

Andere Sonderzeichen werden als Operatoren »Seite 237 verwendet.

## Eckige Klammern

Eckige Klammern (*Brackets*) werden verwendet in Array-Definitionen:

```
int ai[];
```

in Array-Subscripts

```
n = ai[2];
```

und in String-Subscripts, um auf die einzelnen Zeichen eines Strings zuzugreifen

```
string s = "Hello world";  
char c = s[2];
```

## Runde Klammern

Runde Klammern (*Parentheses*) gruppieren Ausdrücke »Seite 244 (ändern eventuell die Priorität der Operatoren »Seite 237), isolieren bedingte Ausdrücke und bezeichnen Funktionsaufrufe »Seite 250 und Funktionsparameter:

```
d = c * (a + b);  
if (d == z) ++x;  
func();  
void func2(int n) { ... }
```

## Geschweifte Klammern

Geschweifte Klammern (*Braces*) bezeichnen den Beginn und das Ende einer Verbundanweisung (*Compound Statement*)

```
if (d == z) {  
    ++x;  
    func();  
}
```

und werden auch verwendet, um die Werte für die Array-Initialisierung zu gruppieren:

```
int ai[] = { 1, 2, 3 };
```

## Komma

Das *Komma* trennt die Elemente einer Funktionsargument-Liste oder die Parameter eines Funktionsaufrufs:

```
int func(int n, real r, string s) { ... }  
int i = func(1, 3.14, "abc");
```

Es trennt auch die Wertangaben bei der Array-Initialisierung:

```
int ai[] = { 1, 2, 3 };
```

und es begrenzt die Elemente einer Variablen-Definition:

```
int i, j, k;
```

## Semicolon

Der *Semicolon* schließt ein Statement »Seite 251 ab, wie in

```
i = a + b;
```

und er begrenzt die Init-, Test- und Inkrement-Ausdrücke eines for »Seite 258 Statements:

```
for (int n = 0; n < 3; ++n) {  
    func(n);  
}
```

## Doppelpunkt

Der *Doppelpunkt* bezeichnet das Ende eines Labels in einem Switch »Seite 261-Statement:

```
switch (c) {  
  case 'a': printf("It was an 'a'\n"); break;  
  case 'b': printf("It was a  'b'\n"); break;  
  default:  printf("none of them\n");  
}
```



## Gleichheitszeichen

Das *Gleichheitszeichen* trennt Variablen-Definitionen von Initialisierungsliste:

```
int i = 10;  
char c[] = { 'a', 'b', 'c' };
```

Es wird auch als Zuweisungsoperator »Seite 242 verwendet.

## Datentypen

Ein User-Language-Programm kann Variablen unterschiedlicher Typen definieren, die unterschiedliche Arten von EAGLE-Daten repräsentieren.

Die vier grundlegenden Datentypen sind

```
char »Seite 187    für Einzelzeichen  
int  »Seite 188    für Ganzzahlen  
real »Seite 189    für Gleitkommazahlen  
string »Seite 190  für Textinformation
```

Neben diesen grundlegenden Datentypen gibt es auch High-level-Objekt-Typen »Seite 193, die die Datenstrukturen repräsentieren, wie sie in den EAGLE-Dateien gespeichert sind.

Der Datentyp `void` wird nur als Return-Typ einer Funktion »Seite 236 verwendet. Er zeigt an, daß diese Funktion keinen Wert zurückgibt.

## **char**

Der Datentyp `char` speichert Einzelzeichen, wie die Buchstaben des Alphabets oder kleine Zahlen ohne Vorzeichen.

Eine Variable des Typs `char` belegt 8 bit (1 Byte), und kann jeden Wert im Bereich 0 . . 255 speichern.

**Siehe auch** Operatoren »Seite 237, Character-Konstanten »Seite 173

## **int**

Der Datentyp `int` speichert Ganzzahlen mit Vorzeichen, wie die Koordinaten eines Objekts.

Eine Variable vom Typ `int` belegt 32 bit (4 Byte), und kann jeden Wert im Bereich -2147483648 .. 2147483647 speichern.

**Siehe auch** Integer-Konstanten »Seite 174

## **real**

Der Datentyp `real` speichert Gleitkommazahlen mit Vorzeichen, z.B. den Rasterabstand.

Eine Variable vom Typ `real` belegt 64 bit (8 Byte), und kann jeden Wert im Bereich  $\pm 2.2 \times 10^{-308}$  ..  $\pm 1.7 \times 10^{+308}$  mit einer Genauigkeit von 15 Digits speichern.

**Siehe auch** Real-Konstanten »Seite 175

## string

Der Datentyp `string` speichert Textinformation, z.B. den Namen eines Bauteils oder eines Netzes.

Eine Variable des Typs `string` ist nicht auf eine bestimmte Länge beschränkt, vorausgesetzt, es steht genügend Speicher zur Verfügung.

Variablen des Typs `string` sind ohne explizite Länge definiert. Sie "wachsen" automatisch, soweit erforderlich, während der Programmausführung.

Die Elemente einer `String`-Variablen sind vom Typ `char` »Seite 187, und man kann auf sie individuell zugreifen, indem man `[index]` benutzt.

Das erste Zeichen eines Strings hat den Index 0:

```
string s = "Layout";  
printf("Third char is: %c\n", s[2]);
```

Hier würde das Zeichen 'y' ausgedruckt. Beachten Sie, daß `s[2]` das dritte Zeichen des Strings `s` liefert!

**Siehe auch** Operatoren »Seite 237, Builtin-Functions »Seite 266, String-Konstanten »Seite 176

### Implementierungs-Details

Der Datentyp `string` ist implementiert mit "Zero-terminated-Strings", wie von C her bekannt (also mit `char[]`). Betrachtet man die folgende Variablen-Definition

```
string s = "abcde";
```

dann ist `s[4]` das Zeichen 'e', und `s[5]` ist das Zeichen '\0', oder der Integer-Wert 0x00. Diese Tatsache kann dazu ausgenutzt werden, das Ende eines Strings ohne die Funktion `strlen()` »Seite 297 festzustellen, wie in

```
for (int i = 0; s[i]; ++i) {  
    // do something with s[i]  
}
```

Es ist auch völlig in Ordnung, einen String "abzuschneiden", indem man den Wert "0" an der gewünschten Stelle einfügt:

```
string s = "abcde";  
s[3] = 0;
```

Als Ergebnis erhält man für den String `s` den Wert "abc".

## Typ-Umwandlung

Der Typ des Ergebnisses eines arithmetischen Ausdrucks »Seite 244, wie z.B.  $a + b$ , wobei  $a$  und  $b$  unterschiedliche arithmetische Typen sind, ist gleich dem "größeren" der beiden Operanden-Typen.

Arithmetische Typen sind `char` »Seite 187, `int` »Seite 188 und `real` »Seite 189 (in dieser Reihenfolge). Ist zum Beispiel  $a$  vom Typ `int` »Seite 188 und  $b$  vom Typ `real` »Seite 189, dann ist das Ergebnis  $a + b$  vom Typ `real` »Seite 189.

**Siehe auch** `Typecast` »Seite 192

## Typecast

Der Ergebnis-Typ eines arithmetischen Ausdrücke »Seite 244 kann explizit in einen anderen arithmetischen Typ umgewandelt werden, indem man einen *Typecast* darauf anwendet.

Die allgemeine Syntax eines Typecast ist

`type (expression)`

wobei `type` `char` »Seite 187, `int` »Seite 188 oder `real` »Seite 189 ist und `expression` jeder arithmetische Ausdruck »Seite 244 sein kann.

Wenn man mit Typecast einen Ausdruck vom Typ `real` »Seite 189 in `int` »Seite 188 umwandelt, wird der Dezimalbruch des Wertes abgeschnitten.

**Siehe auch** Typ-Umwandlung »Seite 191



## Objekt-Typen

Die EAGLE-Datenstruktur ist in drei Binärdatei-Typen gespeichert:

- Library (\*.lbr)
- Schematic (\*.sch)
- Board (\*.brd)

Diese Dateien enthalten Objekte, die hierarchisch gegliedert sind. In einem User-Language-Programm kann man auf die Hierarchiestufen mit Hilfe der entsprechenden Builtin-Zugriffs-Statements zugreifen:

```
library »Seite 313 (L) { ... }
schematic »Seite 316 (S) { ... }
board »Seite 311 (B) { ... }
```

Diese Zugriffs-Statements schaffen einen Kontext, innerhalb dessen Sie auf alle Objekte in Bibliotheken, Schaltplänen oder Platinen zugreifen können.

Auf die "Properties" dieser Objekte kann mit Hilfe von *Members* zugegriffen werden.

Es gibt zwei Arten von Members:

- Data members
- Loop members

Data members liefern die Objektdaten unmittelbar. Zum Beispiel in

```
board(B) {
    printf("%s\n", B.name);
}
```

liefert Data member *name* des Board-Objekts *B* den Board-Namen. Data members können auch andere Objekte zurückgeben, wie in

```
board(B) {
    printf("%f\n", B.grid.size);
}
```

wo Data member *grid* des Boards ein Grid-Objekt zurückliefert, dessen Data member *size* dann Grid-Size (Rastergröße) zurückgibt.

Loop members werden verwendet, um auf Mehrfach-Objekte derselben Art zuzugreifen, die in einem Objekt einer höheren Hierarchiestufe enthalten sind:

```
board(B) {
    B.elements(E) {
        printf("%-8s %-8s\n", E.name, E.value);
    }
}
```

Dieses Beispiel verwendet Loop member *elements()* des Boards, um eine Schleife durch alle Board-Elemente zu realisieren. Der Block nach dem *B.elements(E)*-Statement wird der Reihe nach für jedes Element ausgeführt, und das gegenwärtige Element kann innerhalb des Blocks unter dem Namen *E* angesprochen werden.

Loop members behandeln Objekte in alpha-numerisch sortierter Reihenfolge, falls die Objekte einen Namen haben.

Eine Loop-member-Funktion erzeugt eine Variable vom erforderlichen Typ, um die Objekte zu speichern. Sie dürfen jeden gültigen Namen für eine derartige Variable verwenden, so daß das obige Beispiel auch so lauten könnte:

```
board(MyBoard) {  
    B.elements(TheCurrentElement) {  
        printf("%-8s %-8s\n", TheCurrentElement.name, TheCurrentElement.value);  
    }  
}
```

Das Ergebnis wäre identisch mit dem vorhergehenden Beispiel. Der Gültigkeitsbereich einer Variablen, die von einer Loop-member-Funktion angelegt wird, ist auf das Statement oder den Block unmittelbar nach dem Loop-Funktionsaufruf beschränkt.

Objekt-Hierarchie einer (Bibliothek) Library:

```
LIBRARY »Seite 213  
  GRID »Seite 208  
  LAYER »Seite 212  
  DEVICESSET »Seite 205  
    DEVICE »Seite 204  
    GATE »Seite 207  
  PACKAGE »Seite 215  
    PAD »Seite 216  
    SMD »Seite 228  
    ARC »Seite 196  
    CIRCLE »Seite 200  
    HOLE »Seite 209  
    RECTANGLE »Seite 223  
    TEXT »Seite 230  
    WIRE »Seite 232  
    POLYGON »Seite 221  
      WIRE »Seite 232  
  SYMBOL »Seite 229  
    PIN »Seite 218  
    ARC »Seite 196  
    CIRCLE »Seite 200  
    RECTANGLE »Seite 223  
    TEXT »Seite 230  
    WIRE »Seite 232  
    POLYGON »Seite 221  
      WIRE »Seite 232
```

Objekt-Hierarchie eines Schaltplans (Schematic):

```
SCHEMATIC »Seite 224  
  GRID »Seite 208  
  LAYER »Seite 212  
  LIBRARY »Seite 213  
  SHEET »Seite 226  
    ARC »Seite 196  
    CIRCLE »Seite 200  
    RECTANGLE »Seite 223  
    TEXT »Seite 230  
    WIRE »Seite 232  
    POLYGON »Seite 221  
      WIRE »Seite 232  
  PART »Seite 217  
    INSTANCE »Seite 210  
  BUS »Seite 199  
    SEGMENT »Seite 225  
      TEXT »Seite 230  
      WIRE »Seite 232  
  NET »Seite 214  
    SEGMENT »Seite 225  
      JUNCTION »Seite 211  
      PINREF »Seite 220  
      TEXT »Seite 230  
      WIRE »Seite 232
```

Objekt-Hierarchie einer Platine (Board):

BOARD »Seite 198  
GRID »Seite 208  
LAYER »Seite 212  
LIBRARY »Seite 213  
ARC »Seite 196  
CIRCLE »Seite 200  
HOLE »Seite 209  
RECTANGLE »Seite 223  
TEXT »Seite 230  
WIRE »Seite 232  
POLYGON »Seite 221  
    WIRE »Seite 232  
ELEMENT »Seite 206  
SIGNAL »Seite 227  
    CONTACTREF »Seite 203  
    POLYGON »Seite 221  
        WIRE »Seite 232  
    VIA »Seite 231  
    WIRE »Seite 232

## UL\_ARC

### Data members

angle1	real »Seite 189 (Startwinkel, 0.0...359.9°)
angle2	real »Seite 189 (Endwinkel, 0.0...719.9°)
layer	int »Seite 188
radius	int »Seite 188
width	int »Seite 188
x1, y1	int »Seite 188 (Startpunkt)
x2, y2	int »Seite 188 (Endpunkt)
xc, yc	int »Seite 188 (Mittelpunkt)

**Siehe auch** UL\_BOARD »Seite 198, UL\_PACKAGE »Seite 215, UL\_SHEET »Seite 226, UL\_SYMBOL »Seite 229

### Anmerkung

Start- und Endwinkel werden im mathematisch positiven Sinne ausgegeben (also gegen den Uhrzeigersinn, "counterclockwise"), wobei gilt `angle1 < angle2`.

### Beispiel

```
board(B) {  
    B.arcs(A) {  
        printf("Arc: (%d %d), (%d %d), (%d %d)\n",  
              A.x1, A.y1, A.x2, A.y2, A.xc, A.yc);  
    }  
}
```

## UL\_AREA

### Data members

x1, y1           int »Seite 188 (linke untere Ecke)  
x2, y2           int »Seite 188 (rechte obere Ecke)

**Siehe auch** UL\_BOARD »Seite 198, UL\_DEVICE »Seite 204, UL\_PACKAGE »Seite 215, UL\_SHEET »Seite 226, UL\_SYMBOL »Seite 229

UL\_AREA ist ein Pseudo-Objekt, das Informationen über die Fläche liefert, die ein Objekt einnimmt. Für UL\_DEVICE, UL\_PACKAGE und UL\_SYMBOL ist die Fläche definiert als umschließendes Rechteck der Objekt-Definition in der Bibliothek. Deshalb geht der Offset in einem Board nicht in die Fläche ein, obwohl UL\_PACKAGE von UL\_ELEMENT abgeleitet wird.

### Beispiel

```
board(B) {  
    printf("Area: (%d %d), (%d %d)\n",  
          B.area.x1, B.area.y1, B.area.x2, B.area.y2);  
}
```

## UL\_BOARD

### Data members

area	UL_AREA »Seite 197
grid	UL_GRID »Seite 208
name	string »Seite 190

### Loop members

arcs()	UL_ARC »Seite 196
circles()	UL_CIRCLE »Seite 200
classes()	UL_CLASS »Seite 201
elements()	UL_ELEMENT »Seite 206
holes()	UL_HOLE »Seite 209
layers()	UL_LAYER »Seite 212
libraries()	UL_LIBRARY »Seite 213
polygons()	UL_POLYGON »Seite 221
rectangles()	UL_RECTANGLE »Seite 223
signals()	UL_SIGNAL »Seite 227
texts()	UL_TEXT »Seite 230
wires()	UL_WIRE »Seite 232

**Siehe auch** UL\_LIBRARY »Seite 213, UL\_SCHEMATIC »Seite 224

### Beispiel

```
board(B) {  
    B.elements(E) printf("Element: %s\n", E.name);  
    B.signals(S)  printf("Signal: %s\n", S.name);  
}
```

## UL\_BUS

### Data members

name                    string »Seite 190 (BUS\_NAME\_LENGTH)

### Loop members

segments()            UL\_SEGMENT »Seite 225

### Konstanten

BUS\_NAME\_LENGTH        max. Länge eines Busnamens (obsolet - ab Version 4 können Bus-Namen beliebig lang sein)

**Siehe auch** UL\_SHEET »Seite 226

### Beispiel

```
schematic(SCH) {  
    SCH.sheets(SH) {  
        SH.busses(B) printf("Bus: %s\n", B.name);  
    }  
}
```

## UL\_CIRCLE

### Data members

layer	int »Seite 188
radius	int »Seite 188
width	int »Seite 188
x, y	int »Seite 188 (Mittelpunkt)

**Siehe auch** UL\_BOARD »Seite 198, UL\_PACKAGE »Seite 215, UL\_SHEET »Seite 226, UL\_SYMBOL »Seite 229

### Beispiel

```
board(B) {  
    B.circles(C) {  
        printf("Circle: (%d %d), r=%d, w=%d\n",  
               C.x, C.y, C.radius, C.width);  
    }  
}
```

+



## UL\_CLASS

### Data members

clearance	int »Seite 188
drill	int »Seite 188
name	string »Seite 190 (siehe Anmerkung)
number	int »Seite 188
width	int »Seite 188

**Siehe auch** Design Rules »Seite 159, UL\_NET »Seite 214, UL\_SIGNAL »Seite 227, UL\_SCHEMATIC »Seite 224, UL\_BOARD »Seite 198

### Anmerkung

Wenn `name` einen leeren String liefert, ist die Netzklasse nicht definiert und wird somit auch nicht von einem Signal oder Netz benutzt.

### Beispiel

```
board(B) {  
    B.signals(S) {  
        printf("%-10s %d %s\n", S.name, S.class.number, S.class.name);  
    }  
}
```

## UL\_CONTACT

### Data members

name	string »Seite 190 (CONTACT_NAME_LENGTH)
pad	UL_PAD »Seite 216
signal	string »Seite 190
smd	UL_SMD »Seite 228
x, y	int »Seite 188 (Mittelpunkt, siehe Anmerkung)

### Konstanten

CONTACT\_NAME\_LENGTH max. empfohlene Länge eines "Contact"-Namens (wird nur für formatierte Ausgaben benutzt)

**Siehe auch** UL\_PACKAGE »Seite 215, UL\_PAD »Seite 216, UL\_SMD »Seite 228, UL\_CONTACTREF »Seite 203, UL\_PINREF »Seite 220

### Anmerkung

Die Koordinaten (x, y) des "Contacts" hängen vom Kontext ab aus dem sie aufgerufen werden:

- Wird "Contact" aus einem UL\_LIBRARY-Kontext aufgerufen, sind die Koordinaten dieselben, wie in der Package-Zeichnung
- In allen anderen Fällen gelten Sie die aktuellen Werte in der Board-Datei

### Beispiel

```
library(L) {  
  L.packages(PAC) {  
    PAC.contacts(C) {  
      printf("Contact: '%s', (%d %d)\n",  
            C.name, C.x, C.y);  
    }  
  }  
}
```

## UL\_CONTACTREF

### Data members

contact           UL\_CONTACT »Seite 202  
element           UL\_ELEMENT »Seite 206

**Siehe auch** UL\_SIGNAL »Seite 227, UL\_PINREF »Seite 220

### Beispiel

```
board(B) {  
    B.signals(S) {  
        printf("Signal '%s'\n", S.name);  
        S.contactrefs(C) {  
            printf("\t%s, %s\n", C.element.name, C.contact.name);  
        }  
    }  
}
```

## UL\_DEVICE

### Data members

area	UL_AREA »Seite 197
description	string »Seite 190
headline	string »Seite 190
library	string »Seite 190
name	string »Seite 190 (DEVICE_NAME_LENGTH)
package	UL_PACKAGE »Seite 215
prefix	string »Seite 190 (DEVICE_PREFIX_LENGTH)
technologies	string »Seite 190 (siehe Anmerkung)
value	string »Seite 190 ("On" oder "Off")

### Loop members

gates()           UL\_GATE »Seite 207

### Konstanten

DEVICE_NAME_LENGTH	max. empfohlene Länge eines Device-Namens (wird nur für formatierte Ausgaben benutzt)
DEVICE_PREFIX_LENGTH	max. empfohlene Länge eines Device-Präfix (wird nur für formatierte Ausgaben benutzt)

**Siehe auch** UL\_DEVICESET »Seite 205, UL\_LIBRARY »Seite 213, UL\_PART »Seite 217

Alle UL\_DEVICE-Member, mit Ausnahme von `name` und `technologies`, liefern dieselben Werte wie die zugehörigen UL\_DEVICESET-Member in dem UL\_DEVICE definiert wurde. Bitte denken Sie daran: Der `description`-Text darf Newline-Zeichen ( '\n' ) enthalten.

### Anmerkung

Der Wert des `technologies`-Member hängt vom Kontext ab aus dem es aufgerufen wurde:

- Wird das Device über UL\_DEVICESET hergeleitet, liefert `technologies` einen String, der alles über die Technologien des Devices, durch Leerzeichen getrennt, enthält
- Wird das Device über UL\_PART hergeleitet, wird nur die aktuelle Technologie, die von diesem Bauteil benutzt wird, ausgegeben.

### Beispiel

```
library(L) {
  L.devicesets(S) {
    S.devices(D) {
      printf("Device: %s, Package: %s\n", D.name, D.package.name);
      D.gates(G) {
        printf("\t%s\n", G.name);
      }
    }
  }
}
```

## UL\_DEVICESET

### Data members

area	UL_AREA »Seite 197
description	string »Seite 190
headline	string »Seite 190 (siehe Anmerkung)
library	string »Seite 190
name	string »Seite 190 (DEVICE_NAME_LENGTH)
prefix	string »Seite 190 (DEVICE_PREFIX_LENGTH)
value	string »Seite 190 ("On" oder "Off")

### Loop members

devices()	UL_DEVICE »Seite 204
gates()	UL_GATE »Seite 207

### Konstanten

DEVICE_NAME_LENGTH	max. empfohlene Länge des Device-Namen (wird nur bei formatierten Ausgaben benutzt)
DEVICE_PREFIX_LENGTH	max. empfohlene Länge des Prefix (wird nur bei formatierten Ausgaben benutzt)

**Siehe auch** UL\_DEVICE »Seite 204, UL\_LIBRARY »Seite 213, UL\_PART »Seite 217

### Anmerkung

Das `description`-Member liefert den vollständigen Beschreibungstext, der mit dem `DESCRIPTION` »Seite 60 -Befehl erzeugt wurde, während das `headline`-Member nur die erste Zeile der Beschreibung ohne Rich-Text »Seite 356-Tags ausgibt. Wenn Sie `description`-Text schreiben, denken Sie daran, dass dieser Newline-Anweisungen (`'\n'`) enthalten darf.

### Beispiel

```
library(L) {
  L.devicesets(D) {
    printf("Device set: %s, Description: %s\n", D.name, D.description);
    D.gates(G) {
      printf("\t%s\n", G.name);
    }
  }
}
```

## UL\_ELEMENT

### Data members

angle	real »Seite 189 (0.0...359.9°)
mirror	int »Seite 188
name	string »Seite 190 (ELEMENT_NAME_LENGTH)
package	UL_PACKAGE »Seite 215
value	string »Seite 190 (ELEMENT_VALUE_LENGTH)
x, y	int »Seite 188 (Ursprung, Aufhängepunkt)

### Loop members

`texts()`           UL\_TEXT »Seite 230 (siehe Anmerkung)

### Konstanten

`ELEMENT_NAME_LENGTH`   max. empfohlene Länge eines Element-Namens (wird nur für formatierte Ausgaben benutzt)

`ELEMENT_VALUE_LENGTH`   max. empfohlene Länge eines Element-Values (wird nur für formatierte Ausgaben benutzt)

**Siehe auch** `UL_BOARD` »Seite 198, `UL_CONTACTREF` »Seite 203

### Anmerkung

Das `texts()`-Member läuft nur durch die **gesmashten** »Seite 119 Texte des Elements. Ist das Element nicht gesmasht, so wird diese Schleife nicht ausgeführt. Um alle Texte eines Elements zu bearbeiten, müssen Sie eine Schleife durch das `texts()`-Member des Elements selbst und durch das `texts()`-Member des zum Element gehörenden Package »Seite 215 bilden.

### Beispiel

```
board(B) {
  B.elements(E) {
    printf("Element: %s, (%d %d), Package=%s\n",
          E.name, E.x, E.y, E.package.name);
  }
}
```

## UL\_GATE

### Data members

addlevel	int »Seite 188 (GATE_ADDLEVEL_...)
name	string »Seite 190 (GATE_NAME_LENGTH)
swaplevel	int »Seite 188
symbol	UL_SYMBOL »Seite 229
x, y	int »Seite 188 (Aufhängepunkt, siehe Anmerkung)

### Konstanten

GATE_ADDLEVEL_MUST	must
GATE_ADDLEVEL_CAN	can
GATE_ADDLEVEL_NEXT	next
GATE_ADDLEVEL_REQUEST	request
GATE_ADDLEVEL_ALWAYS	always

GATE_NAME_LENGTH	max. empfohlene Länge eines Gate-Namens (wird nur für formatierte Ausgaben benutzt)
------------------	---

**Siehe auch** UL\_DEVICE »Seite 204

### Anmerkung

Die Koordinaten des Aufhängepunktes (x, y) sind immer bezogen auf die Lage des Gates im Device, auch wenn das UL\_GATE über ein UL\_INSTANCE »Seite 210 geholt wurde.

### Beispiel

```
library(L) {
  L.devices(D) {
    printf("Device: %s, Package: %s\n", D.name, D.package.name);
    D.gates(G) {
      printf("\t%s, swaplevel=%d, symbol=%s\n",
            G.name, G.swaplevel, G.symbol.name);
    }
  }
}
```

## UL\_GRID

### Data members

distance	real »Seite 189
dots	int »Seite 188 (0=lines, 1=dots)
multiple	int »Seite 188
on	int »Seite 188 (0=off, 1=on)
unit	int »Seite 188 (GRID_UNIT_...)

### Konstanten

GRID_UNIT_MIC	Micron
GRID_UNIT_MM	Millimeter
GRID_UNIT_MIL	Mil
GRID_UNIT_INCH	Inch

**Siehe auch** UL\_BOARD »Seite 198, UL\_LIBRARY »Seite 213, UL\_SCHEMATIC »Seite 224, Unit Conversions »Seite 288

### Beispiel

```
board(B) {  
    printf("Gridsize=%f\n", B.grid.distance);  
}
```



## UL\_HOLE

### Data members

drill                   int »Seite 188  
x, y                   int »Seite 188 (Mittelpunkt)

**Siehe auch** UL\_BOARD »Seite 198, UL\_PACKAGE »Seite 215

### Beispiel

```
board(B) {  
    B.holes(H) {  
        printf("Hole: (%d %d), drill=%d\n",  
              H.x, H.y, H.drill);  
    }  
}
```

## UL\_INSTANCE

### Data members

angle	real »Seite 189 (0.0...359.9°)
gate	UL_GATE »Seite 207
mirror	int »Seite 188
name	string »Seite 190 (INSTANCE_NAME_LENGTH)
sheet	int »Seite 188 (0=unbenutzt, 1..99=Blattnummer, Sheet number)
value	string »Seite 190 (PART_VALUE_LENGTH)
x, y	int »Seite 188 (Aufhängepunkt)

### Loop members

texts()           UL\_TEXT »Seite 230 (siehe Anmerkung)

### Konstanten

INSTANCE_NAME_LENGTH	max. empfohlene Länge eines Instance-Namen (wird nur für formatierte Ausgaben benutzt)
PART_VALUE_LENGTH	max. empfohlene Länge eines Bauteil-Values (Instances haben keinen eigenen Value!)

**Siehe auch** UL\_PART »Seite 217, UL\_PINREF »Seite 220

### Anmerkung

Das `texts()`-Member läuft nur durch die **gesmashten** »Seite 119 Texte der Instance. Ist die Instance nicht gesmasht, so wird diese Schleife nicht ausgeführt. Um alle Texte einer Instance zu bearbeiten, müssen Sie eine Schleife durch das `texts()`-Member der Instance selbst und durch das `texts()`-Member des zu dem Gate der Instance gehörenden Symbols »Seite 229 bilden.

### Beispiel

```
schematic(S) {
  S.parts(P) {
    printf("Part: %s\n", P.name);
    P.instances(I) {
      if (I.sheet != 0)
        printf("\t%s used on sheet %d\n", I.name, I.sheet);
    }
  }
}
```

## UL\_JUNCTION

### Data members

diameter        int »Seite 188  
x, y            int »Seite 188 (Mittelpunkt)

**Siehe auch** UL\_SEGMENT »Seite 225

### Beispiel

```
schematic(SCH) {  
  SCH.sheets(SH) {  
    SH.nets(N) {  
      N.segments(SEG) {  
        SEG.junctions(J) {  
          printf("Junction: (%d %d)\n", J.x, J.y);  
        }  
      }  
    }  
  }  
}
```

## UL\_LAYER

### Data members

color	int »Seite 188
fill	int »Seite 188
name	string »Seite 190 (LAYER_NAME_LENGTH)
number	int »Seite 188
visible	int »Seite 188 (0=off, 1=on)

**Siehe auch** UL\_BOARD »Seite 198, UL\_LIBRARY »Seite 213, UL\_SCHEMATIC »Seite 224

### Konstanten

LAYER_NAME_LENGTH	max. empfohlene Länge eines Layer-Namens (wird nur für formatierte Ausgaben benutzt)
LAYER_TOP	Layer-Nummern
LAYER_BOTTOM	
LAYER_PADS	
LAYER_VIAS	
LAYER_UNROUTED	
LAYER_DIMENSION	
LAYER_TPLACE	
LAYER_BPLACE	
LAYER_TORIGINS	
LAYER_BORIGINS	
LAYER_TNAMES	
LAYER_BNAMES	
LAYER_TVALUES	
LAYER_BVALUES	
LAYER_TSTOP	
LAYER_BSTOP	
LAYER_TCREAM	
LAYER_BCREAM	
LAYER_TFINISH	
LAYER_BFINISH	
LAYER_TGLUE	
LAYER_BGLUE	
LAYER_TTEST	
LAYER_BTEST	
LAYER_TKEEPOUT	
LAYER_BKEEPOUT	
LAYER_TRESTRICT	
LAYER_BRESTRICT	
LAYER_VRESTRICT	
LAYER_DRILLS	
LAYER_HOLES	
LAYER_MILLING	
LAYER_MEASURES	
LAYER_DOCUMENT	
LAYER_REFERENCE	
LAYER_TDOCU	
LAYER_BDOCU	
LAYER_NETS	
LAYER_BUSSES	
LAYER_PINS	
LAYER_SYMBOLS	
LAYER_NAMES	
LAYER_VALUES	
LAYER_USER	

### Beispiel

```
board(B) {  
    B.layers(L) printf("Layer %3d %s\n", L.number, L.name);  
}
```

## UL\_LIBRARY

### Data members

<code>description</code>	string »Seite 190 (siehe Anmerkung)
<code>grid</code>	UL_GRID »Seite 208
<code>headline</code>	string »Seite 190
<code>name</code>	string »Seite 190 (LIBRARY_NAME_LENGTH, siehe Anmerkung)

### Loop members

<code>devices()</code>	UL_DEVICE »Seite 204
<code>devicesets()</code>	UL_DEVICESET »Seite 205
<code>layers()</code>	UL_LAYER »Seite 212
<code>packages()</code>	UL_PACKAGE »Seite 215
<code>symbols()</code>	UL_SYMBOL »Seite 229

### Konstanten

LIBRARY\_NAME\_LENGTH max. empfohlene Länge eines Bibliotheksnamens (wird nur für formatierte Ausgaben benutzt)

**Siehe auch** UL\_BOARD »Seite 198, UL\_SCHEMATIC »Seite 224

Das `devices()`-Member geht durch alle Package-Varianten und Technologien von UL\_DEVICESET in der Bibliothek, so dass alle möglichen Device-Variationen verfügbar werden. Das `devicesets()`-Member geht nur durch die UL\_DEVICESETs, die wiederum nach deren UL\_DEVICE-Member abgefragt werden können.

### Anmerkung

Das `description`-Member liefert den vollständigen Beschreibungstext, der mit dem DESCRIPTION »Seite 60 -Befehl erzeugt wurde, während das `headline`-Member nur die erste Zeile der Beschreibung ohne Rich-Text »Seite 356-Tags ausgibt. Wenn Sie den `description`-Text benutzen, denken Sie daran, dass dieser Newline-Anweisungen ('\n') enthalten darf. Die `description` und `headline` Texte stehen nur direkt innerhalb einer Library-Zeichnung zur Verfügung, nicht wenn die Bibliothek aus einem UL\_BOARD- oder UL\_SCHEMATIC-Kontext heraus angesprochen wird.

Wird die Bibliothek aus einem UL\_BOARD- oder UL\_SCHEMATIC-Kontext heraus angesprochen, liefert `name` den reinen Bibliotheksnamen (ohne Extension). Ansonsten wird der volle Bibliotheksname ausgegeben.

### Beispiel

```
library(L) {
    L.devices(D)      printf("Dev: %s\n", D.name);
    L.devicesets(D)   printf("Dev: %s\n", D.name);
    L.packages(P)      printf("Pac: %s\n", P.name);
    L.symbols(S)       printf("Sym: %s\n", S.name);
}
schematic(S) {
    S.libraries(L) printf("Library: %s\n", L.name);
}
```

## UL\_NET

### Data members

class           UL\_CLASS »Seite 201  
name           string »Seite 190 (NET\_NAME\_LENGTH)

### Loop members

pinrefs()       UL\_PINREF »Seite 220 (siehe Anmerkung)  
segments()      UL\_SEGMENT »Seite 225 (siehe Anmerkung)

### Konstanten

NET\_NAME\_LENGTH     max. empfohlene Länge eines Netznamens (wird nur für formatierte Ausgaben  
                          benutzt)

**Siehe auch** UL\_SHEET »Seite 226, UL\_SCHEMATIC »Seite 224

### Anmerkung

Das Loop member `pinrefs()` kann nur benutzt werden, wenn das Net innerhalb eines Schematic-Kontexts verwendet wird.

Das Loop member `segments()` kann nur benutzt werden, wenn das Net innerhalb eines Sheet-Kontexts verwendet wird.

### Beispiel

```
schematic(S) {  
  S.nets(N) {  
    printf("Net: %s\n", N.name);  
    // N.segments(SEG) will NOT work here!  
  }  
}  
schematic(S) {  
  S.sheets(SH) {  
    SH.nets(N) {  
      printf("Net: %s\n", N.name);  
      N.segments(SEG) {  
        SEG.wires(W) {  
          printf("\tWire: (%d %d) (%d %d)\n",  
                W.x1, W.y1, W.x2, W.y2);  
        }  
      }  
    }  
  }  
}
```

## UL\_PACKAGE

### Data members

area	UL_AREA »Seite 197
description	string »Seite 190
headline	string »Seite 190
library	string »Seite 190
name	string »Seite 190 (PACKAGE_NAME_LENGTH)

### Loop members

arcs()	UL_ARC »Seite 196
circles()	UL_CIRCLE »Seite 200
contacts()	UL_CONTACT »Seite 202
holes()	UL_HOLE »Seite 209
polygons()	UL_POLYGON »Seite 221
rectangles()	UL_RECTANGLE »Seite 223
texts()	UL_TEXT »Seite 230 (siehe Anmerkung)
wires()	UL_WIRE »Seite 232

### Konstanten

PACKAGE\_NAME\_LENGTH max. empfohlene Länge eines Package-Namens (wird nur für formatierte Ausgaben benutzt)

**Siehe auch** UL\_DEVICE »Seite 204, UL\_ELEMENT »Seite 206, UL\_LIBRARY »Seite 213

### Anmerkung

Das `description`-Member liefert den vollständigen Beschreibungstext, der mit dem `DESCRIPTION` »Seite 60-Befehl erzeugt wurde, während das `headline`-Member nur die erste Zeile der Beschreibung ohne Rich-Text »Seite 356-Tags ausgibt. Wenn Sie `description`-Text schreiben, denken Sie daran, dass dieser Newline-Anweisungen (`'\n'`) enthalten darf.

Stammt das `UL_PACKAGE` aus einem `UL_ELEMENT`-Kontext, so durchläuft das `texts()`-Member nur die nicht-gemashten Texte dieses Elements.

### Beispiel

```
library(L) {
    L.packages(PAC) {
        printf("Package: %s\n", PAC.name);
        PAC.contacts(C) {
            if (C.pad)
                printf("\tPad: %s, (%d %d)\n",
                    C.name, C.pad.x, C.pad.y);
            else if (C.smd)
                printf("\tSmd: %s, (%d %d)\n",
                    C.name, C.smd.x, C.smd.y);
        }
    }
}

board(B) {
    B.elements(E) {
        printf("Element: %s, Package: %s\n", E.name, E.package.name);
    }
}
```

## UL\_PAD

### Data members

```
diameter[layer]    int »Seite 188
drill              int »Seite 188
name              string »Seite 190 (PAD_NAME_LENGTH)
shape[layer]      int »Seite 188 (PAD_SHAPE_...)
signal            string »Seite 190
x, y              int »Seite 188 (Mittelpunkt, siehe Anmerkung)
```

### Konstanten

```
PAD_SHAPE_SQUARE    square
PAD_SHAPE_ROUND     round
PAD_SHAPE_OCTAGON    octagon
PAD_SHAPE_XLONGOCT   xlongoct
PAD_SHAPE_YLONGOCT   ylongoct
```

```
PAD_NAME_LENGTH      max. empfohlene Länge eines Pad-Namens (identisch mit
CONTACT_NAME_LENGTH)
```

**Siehe auch** UL\_PACKAGE »Seite 215, UL\_CONTACT »Seite 202, UL\_SMD »Seite 228

### Anmerkung

Die Koordinaten (x, y) des Pads hängen vom Kontext ab in dem sie aufgerufen werden:

- Wird das Pad aus einem UL\_LIBRARY-Kontext aufgerufen, sind die Koordinaten dieselben, wie in der Package-Zeichnung
- In allen anderen Fällen gelten die aktuellen Werte vom Board

Durchmesser und Form des Pads hängen vom Layer ab für den es erzeugt werden soll, da diese Werte, abhängig von den Design Rules »Seite 159, unterschiedlich sein können. Wird als Index für das Data-Member "diameter" oder "shape" einer der Layer »Seite 212 LAYER\_TOP...LAYER\_BOTTOM, LAYER\_TSTOP oder LAYER\_BSTOP angegeben, berechnet sich der Wert nach den Vorgaben der Design Rules. Gibt man LAYER\_PADS an, wird der in der Bibliothek definierte Wert verwendet.

### Beispiel

```
library(L) {
  L.packages(PAC) {
    PAC.contacts(C) {
      if (C.pad)
        printf("Pad: '%s', (%d %d), d=%d\n",
              C.name, C.pad.x, C.pad.y, C.pad.diameter[LAYER_BOTTOM]);
    }
  }
}
```



## UL\_PART

### Data members

device	UL_DEVICE »Seite 204
deviceset	UL_DEVICESET »Seite 205
name	string »Seite 190 (PART_NAME_LENGTH)
value	string »Seite 190 (PART_VALUE_LENGTH)

### Loop members

`instances()`    UL\_INSTANCE »Seite 210 (siehe Anmerkung)

### Konstanten

PART_NAME_LENGTH	max. empfohlene Länge eines Part-Namens (wird nur für formatierte Ausgaben benutzt)
PART_VALUE_LENGTH	max. empfohlene Länge eines Part-Values (wird nur für formatierte Ausgaben benutzt)

**Siehe auch** UL\_BOARD »Seite 198

### Anmerkung

Wenn sich Part in einem Sheet-Kontext befindet, bearbeitet Loop member `instances()` nur solche Instances, die tatsächlich auf diesem Sheet benutzt werden. Wenn sich Part in einem Schematic-Kontext befindet, geht die Schleife durch alle Instances.

### Beispiel

```
schematic(S) {  
    S.parts(P) printf("Part: %s\n", P.name);  
}
```

## UL\_PIN

### Data members

angle	real »Seite 189 (0.0...359.9°)
contact	UL_CONTACT »Seite 202 (siehe Anmerkung)
direction	int »Seite 188 (PIN_DIRECTION...)
function	int »Seite 188 (PIN_FUNCTION_FLAG...)
length	int »Seite 188 (PIN_LENGTH...)
name	string »Seite 190 (PIN_NAME_LENGTH)
swaplevel	int »Seite 188
visible	int »Seite 188 (PIN_VISIBLE_FLAG...)
x, y	int »Seite 188 (Anschlußpunkt)

### Loop members

circles()	UL_CIRCLE »Seite 200
texts()	UL_TEXT »Seite 230
wires()	UL_WIRE »Seite 232

### Konstanten

PIN_DIRECTION_NC	Not connected
PIN_DIRECTION_IN	Input
PIN_DIRECTION_OUT	Output (totem-pole)
PIN_DIRECTION_IO	In/Output (bidirectional)
PIN_DIRECTION_OC	Open Collector
PIN_DIRECTION_PWR	Power-Input-Pin
PIN_DIRECTION_PAS	Passiv
PIN_DIRECTION_HIZ	High-Impedance-Output
PIN_DIRECTION_SUP	Supply-Pin

PIN_FUNCTION_FLAG_NONE	kein Symbol
PIN_FUNCTION_FLAG_DOT	Inverter-Symbol
PIN_FUNCTION_FLAG_CLK	Taktsymbol

PIN_LENGTH_POINT	kein Wire
PIN_LENGTH_SHORT	0.1-Inch-Wire
PIN_LENGTH_MIDDLE	0.2-Inch-Wire
PIN_LENGTH_LONG	0.3-Inch-Wire

PIN_NAME_LENGTH	max. empfohlene Länge eines Pin-Namens (wird nur für formatierte Ausgaben benutzt)
-----------------	--

PIN_VISIBLE_FLAG_OFF	kein Name sichtbar
PIN_VISIBLE_FLAG_PAD	Pad-Name sichtbar
PIN_VISIBLE_FLAG_PIN	Pin-Name sichtbar

**Siehe auch** UL\_SYMBOL »Seite 229, UL\_PINREF »Seite 220, UL\_CONTACTREF »Seite 203

### Anmerkung

Das `contact` Data Member liefert den Contact »Seite 202, der dem Pin durch einen `CONNECT` »Seite 55-Befehl zugewiesen worden ist. Es kann als boolsche Function verwendet werden um zu prüfen ob dem Pin ein Contact zugewiesen wurde (siehe Beispiel unten).

Die Koordinaten (und der Layer, im Falle eines SMD) des durch das `contact` Data Member gelieferten Contacts hängen vom Kontext ab, in dem es aufgerufen wird:

- falls der Pin von einem UL\_PART stammt, welches in einem Sheet verwendet wird, und wenn es ein dazugehöriges Element im Board gibt, dann erhält der Contact die Koordinaten die er im Board hat
- in allen anderen Fällen erhält der Contact die Koordinaten wie sie in der Package-Zeichnung definiert sind

**Beispiel**

```
library(L) {
  L.symbols(S) {
    printf("Symbol: %s\n", S.name);
    S.pins(P) {
      printf("\tPin: %s, (%d %d)", P.name, P.x, P.y);
      if (P.direction == PIN_DIRECTION_IN)
        printf(" input");
      if ((P.function & PIN_FUNCTION_FLAG_DOT) != 0)
        printf(" inverted");
      printf("\n");
    }
  }
  L.devices(D) {
    D.gates(G) {
      G.symbol.pins(P) {
        if (!P.contact)
          printf("Unconnected pin: %s/%s/%s\n", D.name, G.name, P.name);
      }
    }
  }
}
```

## UL\_PINREF

### Data members

instance      UL\_INSTANCE »Seite 210  
part          UL\_PART »Seite 217  
pin          UL\_PIN »Seite 218

**Siehe auch** UL\_SEGMENT »Seite 225, UL\_CONTACTREF »Seite 203

### Beispiel

```
schematic(SCH) {  
  SCH.sheets(SH) {  
    printf("Sheet: %d\n", SH.number);  
    SH.nets(N) {  
      printf("\tNet: %s\n", N.name);  
      N.segments(SEG) {  
        SEG.pinrefs(P) {  
          printf("connected to: %s, %s, %s\n",  
                P.part.name, P.instance.name, P.pin.name);  
        }  
      }  
    }  
  }  
}
```

## UL\_POLYGON

### Data members

<code>isolate</code>	int »Seite 188
<code>layer</code>	int »Seite 188
<code>orphans</code>	int »Seite 188 (0=off, 1=on)
<code>pour</code>	int »Seite 188 (POLYGON_POUR_...)
<code>rank</code>	int »Seite 188
<code>spacing</code>	int »Seite 188
<code>thermals</code>	int »Seite 188 (0=off, 1=on)
<code>width</code>	int »Seite 188

### Loop members

<code>contours()</code>	UL_WIRE »Seite 232 (siehe Anmerkung)
<code>fillings()</code>	UL_WIRE »Seite 232
<code>wires()</code>	UL_WIRE »Seite 232

### Konstanten

<code>POLYGON_POUR_SOLID</code>	<code>solid</code>
<code>POLYGON_POUR_HATCH</code>	<code>hatch</code>

**Siehe auch** UL\_BOARD »Seite 198, UL\_PACKAGE »Seite 215, UL\_SHEET »Seite 226, UL\_SIGNAL »Seite 227, UL\_SYMBOL »Seite 229

### Anmerkung

Die Loop-Member `contours()` und `fillings()` gehen durch alle Wires, mit denen das Polygon gezeichnet wird, sofern es zu einem Signal gehört und mit dem Befehl RATSNEST »Seite 104 freigerechnet wurde. Das Loop-Member `wires()` geht immer durch die Wires, die vom Benutzer gezeichnet wurden. Für nicht freigerechnete Signal-Polygone liefert `contours()` dasselbe Ergebnis als `wires()`. `Fillings()` hat dann keine Bedeutung.

### Polygon-Strichstärke

Wenn Sie das Loop-Member `fillings()` verwenden um die Füll-Linien des Polygons zu erreichen, stellen Sie sicher, dass die Strichstärke `width` des Polygons nicht null ist (sie sollte etwas über null liegen, bzw. mindestens der Auflösung des Ausgabetreibers mit dem Sie die Zeichnung ausgeben wollen entsprechen). **Zeichnen Sie ein Polygon mit Strichstärke = 0, ergibt sich eine riesige Datenmenge, da das Polygon mit der kleinsten Editor-Auflösung von 1/10000mm berechnet wird.**

### Teilpolygone

Ein berechnetes Polygon kann aus verschiedenen getrennten Teilen (*positive* Polygone genannt) bestehen, wobei jedes davon Aussparungen (*negative* Polygone genannt) enthalten kann, die von anderen Objekten, die vom Polygon subtrahiert werden, herrühren. Negative Polygone können wiederum weitere positive Polygone enthalten und so weiter.

Die Wires, die mit `contours()` erreicht werden, beginnen immer in einem positiven Polygon. Um herauszufinden wo ein Teilpolygon endet und das nächste beginnt, speichern Sie einfach die Koordinate (x1,y1) des ersten Wires und prüfen diese gegenüber (x2,y2) jedes folgenden Wires. Sobald die beiden Werte identisch sind, ist der letzte Wire des Teilpolygons gefunden. Es gilt immer, dass der zweite Punkt (x2,y2) identisch mit dem ersten Punkt (x1,y1) des nächsten Wires in diesem Teilpolygon ist.

Um herauszufinden ob man innerhalb bzw. außerhalb der Polygons ist, nehmen Sie einen beliebigen Umriss-Wire und stellen sich Sie vor, von dessen Punkt (x1,y1) zum Punkt (x2,y2) zu sehen. Rechts vom Wire ist immer innerhalb des Polygons. Hinweis: Wenn Sie einfach ein Polygon zeichnen wollen, brauchen Sie all diese Details nicht.

### Beispiel

```
board(B) {  
  B.signals(S) {  
    S.polygons(P) {  
      int x0, y0, first = 1;  
      P.contours(W) {  
        if (first) {  
          // a new partial polygon is starting  
          x0 = W.x1;  
          y0 = W.y1;  
        }  
        // ...  
        // do something with the wire  
        // ...  
        if (first)  
          first = 0;  
        else if (W.x2 == x0 && W.y2 == y0) {  
          // this was the last wire of the partial polygon,  
          // so the next wire (if any) will be the first wire  
          // of the next partial polygon  
          first = 1;  
        }  
      }  
    }  
  }  
}
```

## UL\_RECTANGLE

### Data members

layer	int »Seite 188
x1, y1	int »Seite 188 (linke untere Ecke)
x2, y2	int »Seite 188 (rechte obere Ecke)

**Siehe auch** UL\_BOARD »Seite 198, UL\_PACKAGE »Seite 215, UL\_SHEET »Seite 226, UL\_SYMBOL »Seite 229

### Beispiel

```
board(B) {  
    B.rectangles(R) {  
        printf("Rectangle: (%d %d), (%d %d)\n",  
              R.x1, R.y1, R.x2, R.y2);  
    }  
}
```

## UL\_SCHEMATIC

### Data members

grid	UL_GRID »Seite 208
name	string »Seite 190

### Loop members

classes()	UL_CLASS »Seite 201
layers()	UL_LAYER »Seite 212
libraries()	UL_LIBRARY »Seite 213
nets()	UL_NET »Seite 214
parts()	UL_PART »Seite 217
sheets()	UL_SHEET »Seite 226

**Siehe auch** UL\_BOARD »Seite 198, UL\_LIBRARY »Seite 213

### Beispiel

```
schematic(S) {  
    S.parts(P) printf("Part: %s\n", P.name);  
}
```



## UL\_SEGMENT

### Loop members

`junctions()`    `UL_JUNCTION` »Seite 211 (siehe Anmerkung)  
`pinrefs()`     `UL_PINREF` »Seite 220 (siehe Anmerkung)  
`texts()`       `UL_TEXT` »Seite 230  
`wires()`       `UL_WIRE` »Seite 232

**Siehe auch** `UL_BUS` »Seite 199, `UL_NET` »Seite 214

### Anmerkung

Die Loop members `junctions()` und `pinrefs()` sind nur für Netzsegmente zugänglich.

### Beispiel

```
schematic(SCH) {  
  SCH.sheets(SH) {  
    printf("Sheet: %d\n", SH.number);  
    SH.nets(N) {  
      printf("\tNet: %s\n", N.name);  
      N.segments(SEG) {  
        SEG.pinrefs(P) {  
          printf("connected to: %s, %s, %s\n",  
                P.part.name, P.instance.name, P.pin.name);  
        }  
      }  
    }  
  }  
}
```

## UL\_SHEET

### Data members

area	UL_AREA »Seite 197
number	int »Seite 188

### Loop members

arcs()	UL_ARC »Seite 196
busses()	UL_BUS »Seite 199
circles()	UL_CIRCLE »Seite 200
nets()	UL_NET »Seite 214
parts()	UL_PART »Seite 217
polygons()	UL_POLYGON »Seite 221
rectangles()	UL_RECTANGLE »Seite 223
texts()	UL_TEXT »Seite 230
wires()	UL_WIRE »Seite 232

**Siehe auch** UL\_SCHEMATIC »Seite 224

### Beispiel

```
schematic(SCH) {  
    SCH.sheets(S) {  
        printf("Sheet: %d\n", S.number);  
    }  
}
```

## UL\_SIGNAL

### Data members

class           UL\_CLASS »Seite 201  
name           string »Seite 190 (SIGNAL\_NAME\_LENGTH)

### Loop members

contactrefs() UL\_CONTACTREF »Seite 203  
polygons()    UL\_POLYGON »Seite 221  
vias()         UL\_VIA »Seite 231  
wires()        UL\_WIRE »Seite 232

### Konstanten

SIGNAL\_NAME\_LENGTH   max. empfohlene Länge eines Signalnamens (wird nur für formatierte Ausgaben benutzt)

**Siehe auch** UL\_BOARD »Seite 198

### Beispiel

```
board(B) {  
    B.signals(S) printf("Signal: %s\n", S.name);  
}
```

## UL\_SMD

### Data members

`dx[layer], dy[layer]`      int »Seite 188 (size)  
`layer`                      int »Seite 188 (siehe Anmerkung)  
`name`                        string »Seite 190 (SMD\_NAME\_LENGTH)  
`roundness`                int »Seite 188 (siehe Anmerkung)  
`signal`                    string »Seite 190  
`x, y`                        int »Seite 188 (Mittelpunkt, siehe Anmerkung)

### Konstanten

`SMD_NAME_LENGTH`          max. empfohlenen Länge eines Smd-Namens (identisch mit  
`CONTACT_NAME_LENGTH`)

**Siehe auch** `UL_PACKAGE` »Seite 215, `UL_CONTACT` »Seite 202, `UL_PAD` »Seite 216

### Anmerkung

Die Koordinaten (`x`, `y`), `layer` und `roundness` hängen vom Kontext ab in dem sie aufgerufen werden:

- Wird das Smd aus einem `UL_LIBRARY`-Kontext aufgerufen, entsprechen die Werte der Koordinaten, und die Angabe für Layer und Roundness denen in der Package-Zeichnung
- in allen anderen Fällen erhalten Sie die aktuellen Werte aus dem Board

Ruft man die Data-Member `dx` und `dy` mit einem optionalen Layer-Index auf, werden die Werte für den zugehörigen Layer, entsprechend den Design Rules »Seite 159 ausgegeben. Gültige Layer »Seite 212 sind `LAYER_TOP`, `LAYER_TSTOP` und `LAYER_TCREAM` für ein Smd im Top-Layer, und `LAYER_BOTTOM`, `LAYER_BSTOP` und `LAYER_BCREAM` für ein Smd im Bottom-Layer.

### Beispiel

```

library(L) {
  L.packages(PAC) {
    PAC.contacts(C) {
      if (C.smd)
        printf("Smd: '%s', (%d %d), dx=%d, dy=%d\n",
              C.name, C.smd.x, C.smd.y, C.smd.dx, C.smd.dy);
    }
  }
}
  
```

## UL\_SYMBOL

### Data members

area	UL_AREA »Seite 197
library	string »Seite 190
name	string »Seite 190 (SYMBOL_NAME_LENGTH)

### Loop members

arcs()	UL_ARC »Seite 196
circles()	UL_CIRCLE »Seite 200
rectangles()	UL_RECTANGLE »Seite 223
pins()	UL_PIN »Seite 218
polygons()	UL_POLYGON »Seite 221
texts()	UL_TEXT »Seite 230 (siehe Anmerkung)
wires()	UL_WIRE »Seite 232

### Konstanten

SYMBOL\_NAME\_LENGTH    max. empfohlene Länge eines Symbol-Namens (wird nur für formatierte Ausgaben benutzt)

**Siehe auch** UL\_GATE »Seite 207, UL\_LIBRARY »Seite 213

### Anmerkung

Stammt das UL\_SYMBOL aus einem UL\_INSTANCE-Kontext, so durchläuft das `texts()`-Member nur die nicht-gemashten Texte dieser Instance.

### Beispiel

```
library(L) {  
    L.symbols(S) printf("Sym: %s\n", S.name);  
}
```

## UL\_TEXT

### Data members

angle	real »Seite 189 (0.0...359.9°)
font	int »Seite 188 (FONT_...)
layer	int »Seite 188
mirror	int »Seite 188
ratio	int »Seite 188
size	int »Seite 188
value	string »Seite 190
x, y	int »Seite 188 (Aufhängepunkt)

### Loop members

wires()           UL\_WIRE »Seite 232 (siehe Anmerkung)

### Konstanten

FONT_VECTOR	Vector-Font
FONT_PROPORTIONAL	Proportional-Font
FONT_FIXED	Fixed-Font

**Siehe auch** UL\_BOARD »Seite 198, UL\_PACKAGE »Seite 215, UL\_SHEET »Seite 226, UL\_SYMBOL »Seite 229

### Anmerkung

Das Loop-Member `wires()` greift immer auf die individuellen Wires, aus denen der Text im Vektor-Font zusammengesetzt wird, zu. Auch dann, wenn der aktuelle Font nicht `FONT_VECTOR` ist.

Wurde der `UL_TEXT` aus einem `UL_ELEMENT`- oder `UL_INSTANCE`-Kontext angesprochen, so liefern die Members die tatsächlichen Werte, so wie sie in der Board- oder Sheet-Zeichnung zu finden sind.

### Beispiel

```
board(B) {
  B.texts(T) {
    printf("Text: %s\n", T.value);
  }
}
```

## UL\_VIA

### Data members

diameter[layer]     int »Seite 188  
drill                int »Seite 188  
shape[layer]     int »Seite 188 (VIA\_SHAPE\_...)  
x, y                int »Seite 188 (Mittelpunkt)

### Konstanten

VIA\_SHAPE\_SQUARE     square  
VIA\_SHAPE\_ROUND       round  
VIA\_SHAPE\_OCTAGON    octagon

**Siehe auch** UL\_SIGNAL »Seite 227

### Anmerkung

Der Durchmesser und die Form des Vias hängen davon ab für welchen Layer es gezeichnet werden soll, denn es können in den Design Rules »Seite 159 unterschiedliche Werte definiert werden. Gibt man einen der Layer »Seite 212 LAYER\_TOP...LAYER\_BOTTOM, LAYER\_TSTOP oder LAYER\_BSTOP als Index für diameter oder shape an, wird das Via entsprechend den Vorgaben aus den Design Rules berechnet. Wird LAYER\_VIAS angegeben, wird der ursprüngliche Wert mit dem das Via definiert wurde, verwendet.

### Beispiel

```
board(B) {  
  B.signals(S) {  
    S.vias(V) {  
      printf("Via: (%d %d)\n", V.x, V.y);  
    }  
  }  
}
```

## UL\_WIRE

### Data members

layer	int »Seite 188
style	int »Seite 188 (WIRE_STYLE_...)
width	int »Seite 188
x1, y1	int »Seite 188 (Anfangspunkt)
x2, y2	int »Seite 188 (Endpunkt)

### Loop members

`pieces()`      UL\_WIRE »Seite 232 (siehe Anmerkung)

### Konstanten

WIRE_STYLE_CONTINUOUS	durchgezogen
WIRE_STYLE_LONGDASH	lang gestrichelt
WIRE_STYLE_SHORTDASH	kurz gestrichelt
WIRE_STYLE_DASHDOT	Strich-Punkt-Linie

**Siehe auch** UL\_BOARD »Seite 198, UL\_PACKAGE »Seite 215, UL\_SEGMENT »Seite 225, UL\_SHEET »Seite 226, UL\_SIGNAL »Seite 227, UL\_SYMBOL »Seite 229

### Anmerkung

Bei einem UL\_WIRE mit anderem *style* als WIRE\_STYLE\_CONTINUOUS, kann über das Loop-Member `pieces()` auf die individuellen Teile, die z. B. eine gestrichelte Linie darstellen, zugegriffen werden. Wenn `pieces()` für UL\_WIRE mit WIRE\_STYLE\_CONTINUOUS aufgerufen wird, erhält man ein Segment, das genau dem original UL\_WIRE entspricht. Das Loop-Member `pieces()` kann nicht von UL\_WIRE aus aufgerufen werden, wenn dieser selbst schon über `pieces()` aufgerufen wurde (das würde eine unendliche Schleife verursachen).

### Beispiel

```
board(B) {
    B.wires(W) {
        printf("Wire: (%d %d) (%d %d)\n",
            W.x1, W.y1, W.x2, W.y2);
    }
}
```



## Definitionen

Konstanten, Variablen und Funktionen müssen definiert werden, bevor sie in einem User-Language-Programm verwendet werden können.

Es gibt drei Arten von Definitionen:

- Konstanten-Definitionen »Seite 234
- Variablen-Definitionen »Seite 235
- Funktions-Definitionen »Seite 236

Der Gültigkeitsbereich einer *Konstanten*- oder *Variablen*-Definition reicht von der Zeile, in der sie definiert wurde, bis zum Ende des gegenwärtigen Blocks »Seite 252, oder bis zum Ende des User-Language-Programms, wenn die Definition außerhalb aller Blöcke steht.

Der Gültigkeitsbereich einer *Funktions*-Definition reicht von der schließenden geschweiften Klammer ( `}` ) des Funktionsrumpfes bis zum Ende des User-Language-Programms.

## Konstanten-Definitionen

*Konstanten* werden mit Hilfe des Schlüsselworts `enum` definiert, wie in

```
enum { a, b, c };
```

womit man den drei Konstanten `a`, `b` und `c` die Werte 0, 1 und 2 zuweisen würde.

Konstanten kann man auch mit bestimmten Werten initialisieren, wie in

```
enum { a, b = 5, c };
```

wo `a` den Wert 0, `b` den Wert 5 und `c` den Wert 6 erhält.

## Variablen-Definitionen

Die allgemeine Syntax einer *Variablen-Definition* ist

```
[numeric] type identifier [= initializer] [, ...];
```

wobei *type* ein Daten- »Seite 186 oder Objekt-Typ »Seite 193 ist, *identifier* ist der Name der Variablen, und *initializer* ist ein optionaler Initialisierungswert.

Mehrfach-Variablen-Definitionen desselben Typs werden durch Kommas ( , ) getrennt.

Wenn auf *identifier* ein Paar eckiger Klammern »Seite 179 ( [ ] ) folgt, wird ein Array von Variablen des gegebenen Typs definiert. Die Größe des Arrays wird zur Laufzeit automatisch bestimmt.

Das optionale Schlüsselwort *numeric* kann mit String »Seite 190-Arrays verwendet werden, um sie alphanumerisch mit der Funktion *sort()* »Seite 287 sortieren zu lassen.

Standardmäßig (wenn kein *Initializer* vorhanden ist), werden Daten-Variablen »Seite 186 auf 0 gesetzt (oder "", falls es sich um einen String handelt), und Objekt -Variablen »Seite 193 werden mit "invalid" initialisiert.

### Beispiele

<code>int i;</code>	definiert eine int »Seite 188-Variable mit dem Namen <i>i</i>
<code>string s = "Hello";</code>	definiert eine string »Seite 190-Variable mit dem Namen <i>s</i> und initialisiert sie mit "Hello"
<code>real a, b = 1.0, c;</code>	definiert drei real »Seite 189-Variablen mit den Namen <i>a</i> , <i>b</i> und <i>c</i> und initialisiert <i>b</i> mit dem Wert 1.0
<code>int n[] = { 1, 2, 3 };</code>	definiert ein Array of int »Seite 188 und initialisiert die ersten drei Elemente mit 1, 2 und 3
<code>numeric string names[];</code>	definiert ein string »Seite 190-Array das alphanumerisch sortiert werden kann
<code>UL_WIRE w;</code>	definiert ein UL_WIRE »Seite 232-Objekt mit dem Namen <i>w</i>

## Funktions-Definitionen

Sie können Ihre eigenen User-Language-Funktionen schreiben und sie genau so aufrufen wie Builtin-Functions »Seite 266.

Die allgemeine Syntax einer *Funktions-Definition* lautet

```
type identifier(parameters)
{
    statements
}
```

wobei *type* ein Daten- »Seite 186 oder Objekt-Typ »Seite 193 ist, *identifier* der Name einer Funktion, *parameters* eine durch Kommas getrennte Liste von Parameter-Definitionen und *statements* eine Reihe von Statements »Seite 251.

Funktionen die keinen Wert zurückgeben, haben den Typ `void`.

Eine Funktion muß definiert werden, **bevor** sie aufgerufen werden kann, und Funktionsaufrufe können nicht rekursiv sein (eine Funktion kann sich nicht selbst aufrufen).

Die Statements im Funktionsrumpf können die Werte der Parameter ändern, das hat aber keinen Einfluß auf die Argumente des Funktionsaufrufs »Seite 250.

Die Ausführung einer Funktion kann mit dem `return` »Seite 260-Statement beendet werden. Ohne `return`-Statement wird der Funktionsrumpf bis zu seiner schließenden geschweiften Klammer (`}`) ausgeführt.

Ein Aufruf der `exit()` »Seite 284-Funktion beendet das gesamte User-Language-Programm.

### Die spezielle Funktion `main()`

Wenn Ihr User-Language-Programm eine Funktion namens `main()` enthält, wird diese Funktion explizit als Hauptfunktion aufgerufen. Ihr Rückgabewert ist der Rückgabewert »Seite 163 des Programms.

Kommandozeilen-Argumente sind für das Programm über die globalen Builtin-Variablen »Seite 265 `argc` and `argv` verfügbar.

### Beispiel

```
int CountDots(string s)
{
    int dots = 0;
    for (int i = 0; s[i]; ++i)
        if (s[i] == '.')
            ++dots;
    return dots;
}
string dotted = "This.has.dots...";
output("test") {
    printf("Number of dots: %d\n",
           CountDots(dotted));
}
```

## Operatoren

Die folgende Tabelle listet alle User-Language-Operatoren in der Reihenfolge ihrer Priorität auf (*Unary* hat die höchste Priorität, *Comma* die niedrigste):

Unary	! »Seite 239 ~ »Seite 238 + - ++ -- »Seite 242
Multiplicative	* / % »Seite 242
Additive	+ - »Seite 242
Shift	<< >> »Seite 238
Relational	< <= > >= »Seite 240
Equality	== != »Seite 240
Bitwise AND	& »Seite 238
Bitwise XOR	^ »Seite 238
Bitwise OR	»Seite 238
Logical AND	&& »Seite 239
Logical OR	»Seite 239
Conditional	?: »Seite 241
Assignment	= *= /= %= += -= »Seite 242 &= ^=  = <<= >>= »Seite 238
Comma	, »Seite 241

Die Assoziativität ist **links nach rechts** für alle Operatoren außer für *Unary*, *Conditional* und *Assignment*, die **rechts-nach-links**-assoziativ sind.

Die normale Operator-Priorität kann durch den Gebrauch von runden Klammern »Seite 180 geändert werden.

## Bitweise Operatoren

Bitweise Operatoren kann man nur auf die Datentypen `char` »Seite 187 und `int` »Seite 188 anwenden.

### Unary

`~` Bitwise (1's) complement

### Binary

`<<` Shift left

`>>` Shift right

`&` Bitwise AND

`^` Bitwise XOR

`|` Bitwise OR

### Assignment

`&=` Assign bitwise AND

`^=` Assign bitwise XOR

`|=` Assign bitwise OR

`<<=` Assign left shift

`>>=` Assign right shift

## Logische Operatoren

Logische Operatoren arbeiten mit Ausdrücken »Seite 244 von jedem Datentyp.

### Unary

! Logical NOT

### Binary

&& Logical AND

|| Logical OR

Die Verwendung eines `String` »Seite 190-Ausdrucks mit einem logischen Operator prüft, ob ein String leer ist.

Die Verwendung eines Objekt-Typs »Seite 193 mit einem logischen Operator prüft, ob dieses Objekt gültige Daten enthält.

## Vergleichs-Operatoren

Vergleichs-Operatoren können mit Ausdrücken »Seite 244 von jedem Datentyp angewendet werden, ausgenommen Objekt-Typen »Seite 193.

<	Kleiner als
<=	Kleiner gleich
>	Größer als
>=	Größer gleich
==	Gleich
!=	Ungleich



## Evaluation-Operatoren

Evaluation-Operatoren werden verwendet, um Ausdrücke »Seite 244 auszuwerten, die auf einer Bedingung basieren, oder um eine Sequenz von Ausdrücken zu gruppieren und sie als einen Ausdruck auszuwerten.

?: Conditional  
, Komma

Der *Conditional*-Operator wird verwendet, um eine Entscheidung innerhalb eines Ausdrucks zu treffen, wie in

```
int a;  
// ...code that calculates 'a'  
string s = a ? "True" : "False";
```

was folgender Konstruktion entspricht

```
int a;  
string s;  
// ...code that calculates 'a'  
if (a)  
    s = "True";  
else  
    s = "False";
```

aber der Vorteil des Conditional-Operators ist, daß er innerhalb des Ausdrucks verwendet werden kann.

Der *Komma*-Operator wird verwendet, um eine Sequenz von Ausdrücken von links nach rechts auszuwerten; Typ und Wert des rechten Operanden werden als Ergebnis verwendet.

Beachten Sie, daß Argumente in einem Funktionsaufruf und Mehrfach-Variablen-Deklarationen ebenfalls Kommas als Trennzeichen verwenden. Dabei handelt es sich aber **nicht** um den Komma-Operator!

## Arithmetische Operatoren

Arithmetische Operatoren lassen sich auf die Datentypen `char` »Seite 187, `int` »Seite 188 und `real` »Seite 189 anwenden (außer `++`, `--`, `%` und `%=`).

### Unary

- `+`      Unary plus
- `-`      Unary minus
- `++`     Pre- oder postincrement
- `--`     Pre- oder postdecrement

### Binary

- `*`      Multiply
- `/`      Divide
- `%`      Remainder (modulus)
- `+`      Binary plus
- `-`      Binary minus

### Assignment

- `=`      Simple assignment
- `*=`     Assign product
- `/=`     Assign quotient
- `%=`     Assign remainder (modulus)
- `+=`     Assign sum
- `-=`     Assign difference

**Siehe auch** String-Operatoren »Seite 243

## String-Operatoren

String-Operatoren lassen sich mit den Datentypen `char` »Seite 187, `int` »Seite 188 und `string` »Seite 190 anwenden. Der linke Operand muß immer vom Typ `string` »Seite 190 sein.

### Binary

+ Concatenation

### Assignment

= Simple assignment

+= Append to string

Der `+`-Operator faßt zwei Strings zusammen oder fügt ein Zeichen am Ende eines Strings hinzu und gibt den resultierenden String zurück.

Der `+=`-Operator fügt einen String oder eine Zeichen an das Ende eines gegebenen Stings an.

**Siehe auch** Arithmetische Operatoren »Seite 242

## Ausdrücke

Es gibt folgende *Ausdrücke*:

- Arithmetischer Ausdruck »Seite 245
- Zuweisungs-Ausdruck »Seite 246
- String-Ausdruck »Seite 247
- Komma-Ausdruck »Seite 248
- Bedingter Ausdruck »Seite 249
- Funktionsaufruf »Seite 250

Ausdrücke können mit Hilfe von runden Klammern »Seite 180 gruppiert werden und dürfen rekursiv aufgerufen werden, was bedeutet, daß ein Ausdruck aus Unterausdrücken bestehen darf.

## Arithmetischer Ausdruck

Ein *arithmetischer Ausdruck* ist jede Kombination von numerischen Operanden und arithmetischem Operator »Seite 242 oder bitweisem Operator »Seite 238.

### Beispiele

```
a + b  
C++  
m << 1
```

## Zuweisungs-Ausdruck

Ein *Zuweisungs-Ausdruck* besteht aus einer Variablen auf der linken Seite eines Zuweisungsoperators »Seite 242 und einem Ausdruck auf der rechten Seite.

### Beispiele

```
a = x + 42  
b += c  
s = "Hello"
```

## String-Ausdruck

Ein *String-Ausdruck* ist jede Kombination von string- »Seite 190 und char- »Seite 187 Operanden und einem String-Operator »Seite 243.

### Beispiele

```
s + ".brd"  
t + 'x'
```

## Komma-Ausdruck

Ein *Komma-Ausdruck* ist eine Sequenz von Ausdrücken, die mit dem Komma-Operator »Seite 241 abgegrenzt werden.

Komma-Ausdrücke werden von links nach rechts ausgewertet, und das Ergebnis eines Komma-Ausdrucks ist der Typ und der Wert des am weitesten rechts stehenden Ausdrucks.

### Beispiel

`i++, j++, k++`



## Bedingter Ausdruck

Ein *bedingter Ausdruck* verwendet den Conditional-Operator »Seite 241, um eine Entscheidung innerhalb eines Ausdrucks zu treffen.

### Beispiel

```
int a;  
// ...code that calculates 'a'  
string s = a ? "True" : "False";
```

## Funktionsaufruf

Ein *Funktionsaufruf* transferiert den Programmfluß zu einer benutzerdefinierten Funktion »Seite 236 oder einer Builtin-Function »Seite 266. Die formalen Parameter, die in der Funktions-Definition »Seite 236 definiert sind, werden ersetzt durch die Werte der Ausdrücke, die als aktuelle Argumente des Funktionsaufrufs dienen.

### Beispiel

```
int p = strchr(s, 'b');
```

## Statements

Ein *Statement* kann folgendes sein:

- Compound-Statement (Verbundanweisung) »Seite 252
- Control-Statement (Steueranweisung) »Seite 254
- Expression-Statement (Ausdrucksanweisung) »Seite 253
- Builtin-Statement »Seite 310
- Konstanten-Definition »Seite 234
- Variablen-Definition »Seite 235

Statements spezifizieren die Programmausführung. Wenn keine Control-Statements vorhanden sind, werden Statements der Reihe nach in der Reihenfolge ihres Auftretens in der ULP-Datei ausgeführt.

## Compound-Statement (Verbundanweisung)

Ein *Compound-Statement* (auch bekannt als *Block*) ist eine Liste (kann auch leer sein) von Statements in geschweiften Klammern (`{ }`). Syntaktisch kann ein Block als einzelnes Statement angesehen werden, aber er steuert auch den Gültigkeitsbereich von Identifiern. Ein Identifier »Seite 171«, der innerhalb eines Blocks deklariert wird, gilt ab der Stelle, an der er definiert wurde, bis zur schließenden geschweiften Klammer.

Compound-Statements können beliebig verschachtelt werden.

## Expression-Statement (Ausdrucksanweisung)

Ein *Expression-Statement* ist jeder beliebige Ausdruck »Seite 244, gefolgt von einem Semicolon »Seite 183.

Ein Expression-Statement wird ausgeführt, indem der Ausdruck ausgewertet wird. Alle Nebeneffekte dieser Auswertung sind vollständig abgearbeitet, bevor das nächste Statement »Seite 251 ausgeführt wird. Die meisten Expression-Statements sind Zuweisungen »Seite 246 oder Funktionsaufrufe »Seite 250.

Ein Spezialfall ist das *leere Statement*, das nur aus einem Semicolon »Seite 183 besteht. Ein leeres Statement tut nichts, aber es ist nützlich in den Fällen, in denen die ULP-Syntax ein Statement erwartet, aber Ihr Programm keines benötigt.

## Control-Statements (Steueranweisungen)

*Control-Statements* werden verwendet, um den Programmfluß zu steuern.

Iteration-Statements sind

```
do...while »Seite 257  
for »Seite 258  
while »Seite 262
```

Selection-Statements sind

```
if...else »Seite 259  
switch »Seite 261
```

Jump-Statements sind

```
break »Seite 255  
continue »Seite 256  
return »Seite 260
```

## **break**

Das *break*-Statement hat die allgemeine Syntax

```
break;
```

und bricht sofort das **nächste** einschließende do...while- »Seite 257, for- »Seite 258, switch- »Seite 261 oder while- »Seite 262Statement ab.

Da all diese Statements gemischt und verschachtelt werden können, stellen Sie bitte sicher, daß *break* vom korrekten Statement aus ausgeführt wird.

## **continue**

Das *continue*-Statement hat die allgemeine Syntax

```
continue;
```

und transferiert die Steuerung direkt zur Testbedingung des **nächsten** einschließenden `do...while`- »Seite 257, `while`- »Seite 262, oder `for`- »Seite 258 Statements oder zum Increment-Ausdruck des **nächsten** einschließenden `for` »Seite 262-Statements.

Da all diese Statements gemischt und verschachtelt werden können, stellen Sie bitte sicher, daß `continue` das richtige Statement betrifft.



## do...while

Das *do...while*-Statement hat die allgemeine Syntax

```
do statement while (condition);
```

und führt das statement aus, bis der condition-Ausdruck null wird.

condition wird **nach** der ersten Ausführung von statement getestet, was bedeutet, daß das Statement wenigstens einmal ausgeführt wird.

Wenn kein break »Seite 255 oder return »Seite 260 im statement vorkommt, muß das statement den Wert der condition verändern, oder condition selbst muß sich während der Auswertung ändern, um eine Endlosschleife zu vermeiden.

### Beispiel

```
string s = "Trust no one!";  
int i = -1;  
do {  
    ++i;  
} while (s[i]);
```

## for

Das *for*-Statement hat die allgemeine Syntax

`for ([init]; [test]; [inc])`-Statement

und führt folgende Schritte aus:

1. Wenn es einen Initialisierungs-Ausdruck `init` gibt, wird er ausgeführt.
2. Wenn es einen `test`-Ausdruck gibt, wird er ausgeführt. Wenn das Ergebnis ungleich null ist (oder wenn es keinen `test`-Ausdruck gibt), wird das `statement` ausgeführt.
3. Wenn es einen `inc`-Ausdruck gibt, wird er ausgeführt.
4. Schließlich wird die Programmsteuerung wieder an Schritt 2 übergeben.

Wenn es kein `break »Seite 255` oder `return »Seite 260` im `statement` gibt, muß der `inc`-Ausdruck (oder das `statement`) den Wert des `test`-Ausdrucks beeinflussen, oder `test` selbst muß sich während der Auswertung ändern, um eine Endlosschleife zu vermeiden.

Der Initialisierungs-Ausdruck `init` initialisiert normalerweise einen oder mehrere Schleifenzähler. Er kann auch eine neue Variable als Schleifenzähler definieren. Eine solche Variable ist bis zum Ende des aktiven Blocks gültig.

### Beispiel

```
string s = "Trust no one!";
int sum = 0;
for (int i = 0; s[i]; ++i)
    sum += s[i]; // sums up the characters in s
```

## if...else

Das *if...else*-Statement hat die allgemeine Syntax

```
if (expression)
    t_statement
[else
    f_statement]
```

Der bedingte Ausdruck wird ausgewertet und, wenn der Wert ungleich null ist, wird `t_statement` ausgeführt. Anderenfalls wird `f_statement` ausgeführt, sofern der `else`-Teil vorhanden ist.

Der `else`-Teil bezieht sich immer auf das letzte `if` ohne `else`. Wenn Sie etwas anderes wollen, müssen Sie geschweifte Klammern »Seite 181 verwenden, um die Statements zu gruppieren, wie in

```
if (a == 1) {
    if (b == 1)
        printf("a == 1 and b == 1\n");
    }
else
    printf("a != 1\n");
```

## **return**

Eine Funktion »Seite 236 mit einem Return-Typ ungleich `void` muß mindestens ein *return*-Statement mit der Syntax

```
return expression;
```

enthalten, wobei die Auswertung von `expression` einen Wert ergeben muß, der kompatibel ist mit dem Return-Typ der Funktion.

Wenn die Funktion vom Typ `void` ist, kann ein *return*-Statement ohne `expression` verwendet werden, um vom Funktionsaufruf zurückzukehren.

## switch

Das *switch*-Statement hat die allgemeine Syntax

```
switch (sw_exp) {  
    case case_exp: case_statement  
    ...  
    [default: def_statement]  
}
```

und erlaubt die Übergabe der Steuerung an eines von mehreren *case*-Statements (mit "case" als Label), abhängig vom Wert des Ausdrucks *sw\_exp* (der vom Integral-Typ sein muß).

Jedes *case\_statement* kann mit einem oder mehreren *case*-Labels versehen sein. Die Auswertung des Ausdrucks *case\_exp* jedes *case*-Labels muß einen konstanten Integer-Wert ergeben, der innerhalb des umschließenden *switch*-Statements nur einmal vorkommt.

Es darf höchstens ein *default*-Label vorkommen.

Nach der Auswertung von *sw\_exp* werden die *case\_exp*-Ausdrücke auf Übereinstimmung geprüft. Wenn eine Übereinstimmung gefunden wurde, wird die Steuerung zum *case\_statement* mit dem entsprechenden *case*-Label transferiert.

Wird keine Übereinstimmung gefunden und gibt es ein *default*-Label, dann erhält *def\_statement* die Steuerung. Anderenfalls wird kein Statement innerhalb der *switch*-Anweisung ausgeführt.

Die Programmausführung wird nicht beeinflusst, wenn *case*- und *default*-Labels auftauchen. Die Steuerung wird einfach an das folgende Statement übergeben.

Um die Programmausführung am Ende einer Gruppe von Statements für ein bestimmtes *case* zu stoppen, verwenden Sie das *break* »Seite 255-Statement.

### Beispiel

```
string s = "Hello World";  
int vowels = 0, others = 0;  
for (int i = 0; s[i]; ++i)  
    switch (toupper(s[i])) {  
        case 'A':  
        case 'E':  
        case 'I':  
        case 'O':  
        case 'U': ++vowels;  
                break;  
        default: ++others;  
    }  
printf("There are %d vowels in '%s'\n", vowels, s);
```

## while

Das *while*-Statement hat die allgemeine Syntax

```
while (condition) statement
```

und führt *statement* so lange aus, bis der *condition*-Ausdruck ungleich null ist.

*condition* wird **vor** der erstmöglichen Ausführung von *statement* getestet, was bedeutet, daß das Statement überhaupt nicht ausgeführt wird, wenn *condition* von Anfang an null ist.

Wenn kein `break` »Seite 255 oder `return` »Seite 260 im *statement* vorkommt, muß das *statement* den Wert der *condition* verändern, oder *condition* selbst muß sich während der Auswertung ändern, um eine Endlosschleife zu vermeiden.

### Beispiel

```
string s = "Trust no one!";  
int i = 0;  
while (s[i])  
    ++i;
```

## Builtins

Builtins sind *Konstanten*, *Variablen*, *Funktionen* und *Statements*, die zusätzliche Informationen liefern und die Manipulation der Daten erlauben.

- Builtin-Constants »Seite 264
- Builtin Variables »Seite 265
- Builtin-Functions »Seite 266
- Builtin-Statements »Seite 310

## Builtin-Constants

*Builtin-Constants* liefern Informationen über Objekt-Parameter, wie die maximale empfohlene Namenslänge, Flags und so weiter.

Viele Objekt-Typen »Seite 193 haben ihren eigenen **Konstanten**-Bereich, in dem die Builtin-Constants für das betreffende Objekt aufgelistet sind (siehe z.B. UL\_PIN »Seite 218).

Die folgenden Builtin-Constants sind zusätzlich zu denen definiert, die für die einzelnen Objekt-Typen aufgeführt sind:

EAGLE_VERSION	EAGLE-Programm-Versionsnummer (int »Seite 188)
EAGLE_RELEASE	EAGLE-Programm-Release-Nummer (int »Seite 188)
EAGLE_SIGNATURE	ein String »Seite 190 der EAGLE-Programmnamen, -Version und -Copyright-Information enthält
REAL_EPSILON	die minimale positive real »Seite 189 Zahl so dass $r + \text{REAL\_EPSILON} \neq r$
REAL_MAX	der größte mögliche real »Seite 189 Wert
REAL_MIN	der kleinste mögliche (positive!) real »Seite 189 Wert
	die kleinste darstellbare Zahl ist $-\text{REAL\_MAX}$
INT_MAX	der größte mögliche int »Seite 188 Wert
INT_MIN	der kleinste mögliche int »Seite 188 Wert
PI	der Wert von "pi" (3.14..., real »Seite 189)
usage	ein string »Seite 190 der den Text der #usage »Seite 169-Direktive enthält



## Builtin Variablen

*Builtin-Variablen* werden verwendet, um zur Laufzeit Informationen zu erhalten.

<code>int argc</code>	Anzahl der Argumente, die an den RUN »Seite 113 Befehl übergeben wurden
<code>string argv[]</code>	Argumente, die an den RUN-Befehl übergeben wurden ( <code>argv[0]</code> ist der volle ULP-Datei-Name)

## Builtin-Functions

*Builtin-Functions* werden für spezielle Aufgaben benötigt, z.B. formatierte Strings drucken, Daten-Arrays sortieren o.ä.

Sie können auch eigene Funktionen »Seite 236 definieren und sie dazu verwenden, um Ihre User-Language-Programme zu strukturieren.

Builtin-Functions sind in folgende Kategorien eingeteilt:

- Character-Funktionen »Seite 268
- File-Handling-Funktionen »Seite 271
- Mathematische Funktionen »Seite 278
- Verschiedene Funktionen »Seite 283
- Printing-Funktionen »Seite 289
- String-Funktionen »Seite 294
- Zeit-Funktionen »Seite 307

Alphabetische Auflistung aller Builtin-Functions:

- abs() »Seite 279
- acos() »Seite 281
- asin() »Seite 281
- atan() »Seite 281
- ceil() »Seite 280
- cos() »Seite 281
- exit() »Seite 284
- exp() »Seite 282
- ferror() »Seite 272
- fileerror() »Seite 272
- fileext() »Seite 274
- fileglob() »Seite 273
- filename() »Seite 274
- fileread() »Seite 277
- filesSetText() »Seite 274
- filesize() »Seite 275
- filetime() »Seite 275
- floor() »Seite 280
- frac() »Seite 280
- isalnum() »Seite 269
- isalpha() »Seite 269
- iscntrl() »Seite 269
- isdigit() »Seite 269
- isgraph() »Seite 269
- islower() »Seite 269
- isprint() »Seite 269
- ispunct() »Seite 269
- isspace() »Seite 269
- isupper() »Seite 269
- isxdigit() »Seite 269
- log() »Seite 282
- log10() »Seite 282
- lookup() »Seite 285
- max() »Seite 279
- min() »Seite 279
- pow() »Seite 282
- printf() »Seite 290
- round() »Seite 280
- sin() »Seite 281
- sort() »Seite 287
- sprintf() »Seite 293
- sqrt() »Seite 282
- strchr() »Seite 295
- strjoin() »Seite 296

- `strlen()` »Seite 297
- `strlwr()` »Seite 298
- `strrchr()` »Seite 299
- `strrstr()` »Seite 300
- `strsplit()` »Seite 301
- `strstr()` »Seite 302
- `strsub()` »Seite 303
- `strtod()` »Seite 304
- `strtol()` »Seite 305
- `strupr()` »Seite 306
- `t2day()` »Seite 309
- `t2dayofweek()` »Seite 309
- `t2hour()` »Seite 309
- `t2minute()` »Seite 309
- `t2month()` »Seite 309
- `t2second()` »Seite 309
- `t2string()` »Seite 309
- `t2year()` »Seite 309
- `tan()` »Seite 281
- `time()` »Seite 308
- `tolower()` »Seite 270
- `toupper()` »Seite 270
- `trunc()` »Seite 280
- `u2inch()` »Seite 288
- `u2mic()` »Seite 288
- `u2mil()` »Seite 288
- `u2mm()` »Seite 288

## Character-Funktionen

Mit *Character-Funktionen* manipuliert man einzelne Zeichen.

Die folgenden Character-Funktionen sind verfügbar:

- `isalnum()` »Seite 269
- `isalpha()` »Seite 269
- `iscntrl()` »Seite 269
- `isdigit()` »Seite 269
- `isgraph()` »Seite 269
- `islower()` »Seite 269
- `isprint()` »Seite 269
- `ispunct()` »Seite 269
- `isspace()` »Seite 269
- `isupper()` »Seite 269
- `isxdigit()` »Seite 269
- `tolower()` »Seite 270
- `toupper()` »Seite 270

## is...()

### Funktion

Prüfen, ob ein Zeichen in eine bestimmte Kategorie fällt.

### Syntax

```
int isalnum(char c);
int isalpha(char c);
int iscntrl(char c);
int isdigit(char c);
int isgraph(char c);
int islower(char c);
int isprint(char c);
int ispunct(char c);
int isspace(char c);
int isupper(char c);
int isxdigit(char c);
```

### Rückgabewert

Die is...-Funktionen liefern einen Wert ungleich null, wenn das Zeichen in die Kategorie fällt, sonst null.

### Character-Kategorien

isalnum	Buchstaben (A bis Z oder a bis z) oder Digits (0 bis 9)
isalpha	Buchstaben (A bis Z oder a bis z)
iscntrl	Delete-Zeichen oder normale Steuerzeichen (0x7F oder 0x00 bis 0x1F)
isdigit	Digits (0 bis 9)
isgraph	Druckbare Zeichen (außer Leerzeichen)
islower	Kleinbuchstaben (a bis z)
isprint	Druckbare Zeichen (0x20 bis 0x7E)
ispunct	Punctuation-Zeichen (iscntrl oder isspace)
isspace	Space, Tab, Carriage Return, New Line, Vertical Tab oder Formfeed (0x09 bis 0x0D, 0x20)
isupper	Großbuchstaben (A bis Z)
isxdigit	Hex-Digits (0 bis 9, A bis F, a bis f)

### Beispiel

```
char c = 'A';
if (isxdigit(c))
    printf("%c is hex\n", c);
else
    printf("%c is not hex\n", c);
```

## **to...()**

### **Funktion**

Buchstaben in Groß- oder Kleinbuchstaben umwandeln.

### **Syntax**

```
char tolower(char c);  
char toupper(char c);
```

### **Rückgabewert**

Die `tolower`-Funktion gibt den konvertierten Buchstaben zurück, wenn `c` ein Großbuchstabe ist. Alle anderen Zeichen werden unverändert zurückgegeben.

Die `toupper`-Funktion gibt den konvertierten Buchstaben zurück, wenn `c` ein Kleinbuchstabe ist. Alle anderen Zeichen werden unverändert zurückgegeben.

**Siehe auch** `strupr` »Seite 306, `strlwr` »Seite 298

## File-Handling-Funktionen

*File-Handling-Funktionen* behandeln File-Namen, -Größen und -Timestamps.

Folgende File-Handling-Funktionen sind verfügbar:

- `fileerror()` »Seite 272
- `fileglob()` »Seite 273
- `filedir()` »Seite 274
- `fileext()` »Seite 274
- `filename()` »Seite 274
- `fileread()` »Seite 277
- `filesetext()` »Seite 274
- `filesize()` »Seite 275
- `filetime()` »Seite 275

Weitere Informationen über Ausgaben in eine Datei, finden Sie unter `output()` »Seite 314.

## fileerror()

### Funktion

Zeigt den Status von I/O-Operationen.

### Syntax

```
int fileerror();
```

### Rückgabewert

Gibt die fileerror-Funktion 0 zurück, ist alles in Ordnung.

### Beschreibung

fileerror prüft den Status beliebiger I/O-Operation, die seit dem letzten Aufruf dieser Funktion ausgeführt wurden und gibt 0 zurück, wenn alles in Ordnung war. Verursachte eine der I/O-Operationen einen Fehler, wird ein Wert ungleich 0 ausgegeben.

Vor der Ausführung von I/O-Operationen sollten Sie mit fileerror den Fehlerstatus zurücksetzen. Nach der Ausführung der I/O-Operationen rufen Sie fileerrorerneut auf, um zu prüfen ob alles in Ordnung war.

Wenn fileerror einen Wert ungleich 0 ausgibt (und so einen Fehler anzeigt), wird dem Benutzer eine Fehlermeldung angezeigt.

**Siehe auch** output »Seite 314, printf »Seite 290, fileread »Seite 277

### Beispiel

```
fileerror();
output("file.txt", "wt") {
    printf("Test\n");
}
if (fileerror())
    exit(1);
```



## fileglob()

### Funktion

Sucht in einem Verzeichnis.

### Syntax

```
int fileglob(string &array[], string pattern);
```

### Rückgabewert

Die Funktion fileglob liefert die Anzahl der Einträge, die in array kopiert wurden.

### Beschreibung

fileglob sucht in einem Verzeichnis nach pattern.

pattern kann '\*' und '?' als Platzhalter enthalten. Endet pattern mit einem '/', wird der Inhalt des angegebenen Verzeichnis zurückgegeben.

Namen die im resultierenden array mit einem '/' enden, sind Verzeichnisnamen.

Das array ist alphabetisch sortiert, die Verzeichnisse kommen zuerst.

Die Sondereinträge '.' und '..' (für das aktuelle und das übergeordnete Verzeichnis) werden nie in array geschrieben.

Wenn pattern nicht gefunden wird, oder wenn Sie kein Recht haben, das angegebene Verzeichnis zu durchsuchen, ist das array leer.

**Siehe auch** dlgFileOpen() »Seite 322, dlgFileSave() »Seite 322

### Hinweis für Windows-Anwender

Das Pfad-Trennzeichen in array ist immer ein **Forward-Slash** (Schrägstrich). So ist sichergestellt, dass User-Language-Programme betriebsystemunabhängig arbeiten. In pattern wird der **backslash** ('\\') auch als Pfad-Trennzeichen behandelt.

Die Sortierreihenfolge unter Windows unterscheidet nicht zwischen Groß- und Kleinschreibung.

### Beispiel

```
string a[];  
int n = fileglob(a, "*.brd");
```

## Filename-Funktionen

### Funktion

File-Namen in seine Einzelteile aufspalten.

### Syntax

```
string filedir(string file);  
string fileext(string file);  
string filename(string file);  
string filesetext(string file, string newext);
```

### Rückgabewert

filedir liefert das Directory von file (einschließlich Drive).  
fileext liefert die Extension von file.  
filename liefert den File-Namen von file (einschließlich Extension).  
filesetext liefert file mit Extension auf newext gesetzt.

**Siehe auch** Filedata-Funktionen »Seite 275

Wenn ein File-Name in seine drei Bestandteile aufgeteilt worden ist, können die drei Teile mit dem + Operator »Seite 243 wieder zum ursprünglichen Namen zusammengesetzt werden.

### Beispiel

```
if (board) board(B) {  
    output(filesetext(B.name, ".OUT")) {  
        ...  
    }  
}
```

## Filedata-Funktionen

### Funktion

Holt den Timestamp und die Größe eines Files.

### Syntax

```
int filesize(string filename);  
int filetime(string filename);
```

### Rückgabewert

filesize liefert die Größe (in Byte) des Files.

filetime liefert den Timestamp des Files in einem Format, das mit den Zeit-Funktionen »Seite 307 benutzt wird.

**Siehe auch** time »Seite 308, Filename-Funktionen »Seite 274

### Beispiel

```
board(B)  
printf("Board: %s\nSize: %d\nTime: %s\n",  
       B.name, filesize(B.name),  
       t2string(filetime(B.name)));
```

## File-Input-Funktionen

*File-Input-Funktionen* werden verwendet um Daten von Dateien auszulesen.

Folgender File-Input ist möglich:

- `fileread()` »Seite 277

Siehe `output()` »Seite 314 für Informationen zum Thema 'In eine Datei schreiben'.

## fileread()

### Funktion

Liest Daten aus einer Datei aus.

### Syntax

```
int fileread(dest, string file);
```

### Rückgabewert

`fileread` liefert die Anzahl der Objekte, die aus einer Datei ausgelesen wurden. Die tatsächliche Bedeutung des Rückgabewerts hängt vom `dest`-Typ ab.

**Siehe auch** [lookup](#) »Seite 285, [strsplit](#) »Seite 301, [fileerror](#) »Seite 272

Wenn `dest` ein Character-Array ist, werden Binär-Daten aus der Datei ausgelesen. Der Rückgabewert entspricht dann der Anzahl der Bytes, die in das Character-Array eingelesen wurden (das entspricht der Dateigröße).

Wenn `dest` ein String-Array ist, wird die Datei als Textdatei gelesen (eine Zeile pro Array-Member). Der Rückgabewert zeigt die Anzahl der Zeilen, die in das Array eingelesen wurden. Newline-Zeichen werden nicht berücksichtigt.

Wenn `dest` ein String ist, wird die ganze Datei in diesen String eingelesen. Der Rückgabewert ist die Länge des Strings (die nicht unbedingt der Dateigröße entsprechen muss, wenn das Betriebssystem Textdateien mit "cr/lf" anstatt "newline" am Zeilenende speichert).

### Beispiel

```
char b[];  
int nBytes = fileread(b, "data.bin");  
string lines[];  
int nLines = fileread(lines, "data.txt");  
string text;  
int nChars = fileread(text, "data.txt");
```

## Mathematische Funktionen

*Mathematische Funktionen* werden dazu verwendet, matematische Operationen auszuführen.

Die folgenden mathematischen Funktionen sind verfügbar:

- `abs()` »Seite 279
- `acos()` »Seite 281
- `asin()` »Seite 281
- `atan()` »Seite 281
- `ceil()` »Seite 280
- `cos()` »Seite 281
- `exp()` »Seite 282
- `floor()` »Seite 280
- `frac()` »Seite 280
- `log()` »Seite 282
- `log10()` »Seite 282
- `max()` »Seite 279
- `min()` »Seite 279
- `pow()` »Seite 282
- `round()` »Seite 280
- `sin()` »Seite 281
- `sqrt()` »Seite 282
- `trunc()` »Seite 280
- `tan()` »Seite 281

### Fehlermeldungen

Wenn die Argumente eines mathematischen Funktionsaufrufs zu einem Fehler führen, zeigen die Fehlermeldungen die aktuellen Werte der Argumente. Deshalb führen die Statements

```
real x = -1.0;  
real r = sqrt(2 * x);
```

zur Fehlermeldung

```
Invalid argument in call to 'sqrt(-2)'
```

## Absolutwert-, Maximum- und Minimum-Funktion

### Funktion

Absolutwert-, Maximum- und Minimum-Funktion.

### Syntax

```
type abs(type x);  
type max(type x, type y);  
type min(type x, type y);
```

### Rückgabewert

abs liefert den absoluten Wert von x.  
max liefert das Maximum von x und y.  
min liefert das Minimum von x und y.

Der Return-Typ dieser Funktionen ist identisch mit dem größeren Typ der Argumente. type muß char »Seite 187, int »Seite 188 oder real »Seite 189 sein.

### Beispiel

```
real x = 2.567, y = 3.14;  
printf("The maximum is %f\n", max(x, y));
```

## Rundungs-Funktionen

### Funktion

Rundungs-Funktionen.

### Syntax

```
real ceil(real x);  
real floor(real x);  
real frac(real x);  
real round(real x);  
real trunc(real x);
```

### Rückgabewert

`ceil` liefert den kleinsten Integer-Wert nicht kleiner als  $x$ .  
`floor` liefert den größten Integer-Wert nicht größer als  $x$ .  
`frac` liefert den Dezimalbruch von  $x$ .  
`round` liefert  $x$  gerundet auf den nächsten Integer-Wert.  
`trunc` liefert den ganzzahligen Teil von  $x$ .

### Beispiel

```
real x = 2.567;  
printf("The rounded value of %f is %f\n", x, round(x));
```



## Trigonometrische Funktionen

### Funktion

Trigonometrische Funktionen.

### Syntax

```
real acos(real x);  
real asin(real x);  
real atan(real x);  
real cos(real x);  
real sin(real x);  
real tan(real x);
```

### Rückgabewert

acos liefert den arc-cosinus von x.  
asin liefert den arc-sinus von x.  
atan liefert den arc-tangens von x.  
cos liefert den cosinus von x.  
sin liefert den sinus von x.  
tan liefert den tangens von x.

### Konstanten

PI                    der Wert von "pi" (3.14...)

### Beispiel

```
real x = 2.0;  
printf("The sine of %f is %f\n", x, sin(x));
```

## Exponential-Funktionen

### Funktion

Exponential-Funktionen.

### Syntax

```
real exp(real x);  
real log(real x);  
real log10(real x);  
real pow(real x, real y);  
real sqrt(real x);
```

### Rückgabewert

exp liefert  $e$  hoch  $x$ .  
log liefert den natürlichen Logarithmus von  $x$ .  
log10 liefert den Zehnerlogarithmus von  $x$ .  
pow liefert den Wert von  $x$  hoch  $y$ .  
sqrt liefert die Quadratwurzel von  $x$ .

### Anmerkung

Die "n-te" Wurzel kann mit Hilfe der pow-Funktion und einem negativen Exponenten berechnet werden.

### Beispiel

```
real x = 2.1;  
printf("The square root of %f is %f\n", x, sqrt(x));
```

## Sonstige Funktionen

*Sonstige Funktionen* werden für weitere Aufgaben benötigt.

Die folgenden verschiedenen Funktionen sind verfügbar:

- `exit()` »Seite 284
- `lookup()` »Seite 285
- `sort()` »Seite 287
- Einheiten-Konvertierung »Seite 288

## exit()

### Funktion

Beendet ein User-Language-Programm.

### Syntax

```
void exit(int result);  
void exit(string command);
```

### Konstanten

EXIT_SUCCESS	Rückgabewert für erfolgreiche Programmausführung (Wert 0)
EXIT_FAILURE	Rückgabewert für fehlerhafte Programmausführung (Wert -1)

**Siehe auch** RUN »Seite 113

### Beschreibung

Die `exit`-Funktion beendet die Ausführung des User-Language-Programms.

Wird `result` (integer) angegeben, wird es als Rückgabewert »Seite 163 des Programms benutzt.

Wird ein `command`-String angegeben, wird dieser Befehl genauso ausgeführt, als wäre über die Kommandozeile direkt nach dem RUN-Befehl eingegeben worden. In diesem Fall wird der Rückgabewert des ULPs auf `EXIT_SUCCESS` gesetzt.

## lookup()

### Funktion

Sucht Daten in einem String-Array.

### Syntax

```
string lookup(string array[], string key, int field_index[, char  
separator]);  
string lookup(string array[], string key, string field_name[, char  
separator]);
```

### Rückgabewert

lookup liefert den Wert des Feldes, das durch `field_index` oder `field_name` markiert wird. Existiert dieses Feld nicht oder wird kein passender String für `key` gefunden, kommt ein leerer String zurück.

**Siehe auch** `fileread` »Seite 277, `strsplit` »Seite 301

Ein array das mit `lookup()` benutzt werden kann, besteht aus Text-Strings, wobei jeder String einen Daten-Record darstellt.

Jeder Daten-Record enthält eine beliebige Anzahl von Feldern, die durch das Zeichen `separator` (default ist `'\t'`, der Tabulator) getrennt sind. Das erste Feld in einem Record wird als `key` benutzt und hat die Nummer 0.

Alle Records müssen eindeutige `key`-Felder haben. Keines der `key`-Felder darf leer sein - ansonsten ist nicht definiert welcher Record gefunden wird.

Enthält der erste String in `array` einen "Header"-Record (der Record, in dem der Inhalt der Felder beschrieben wird), bestimmt `lookup` mit einem String `field_name` automatisch den Index des Feldes. Das erlaubt es, die `lookup`-Funktion zu benutzen, ohne genau zu wissen, welcher Feld-Index die gewünschten Daten enthält. Es bleibt dem Benutzer überlassen, sicherzustellen, dass der erste Record tatsächlich Header-Informationen enthält.

Ist der `key`-Parameter beim Aufruf von `lookup()` ein leerer String, wird der erste String von `array` verwendet. Das erlaubt dem Programm zu bestimmen, ob ein Header-Record mit den gewünschten Feld-Namen existiert.

Enthält ein Feld das `separator`-Zeichen, muss es in Anführungszeichen eingeschlossen werden (wie in `"abc;def"`, wobei hier das Semikolon (`' ; '`) das Trennzeichen ist). Das gilt auch für Felder, die Anführungszeichen (`"`) enthalten, wobei die Anführungszeichen im Feld verdoppelt werden müssen (wie hier: `"abc;""def"";ghi"` ergibt also `abc; "def" ;ghi`).

**Es wird empfohlen den "tab"-Separator (default) zu verwenden, der diese Probleme nicht kennt (kein Feld kann einen Tabulator enthalten).**

Hier folgt eine Beispiel-Daten-Datei (zur besseren Lesbarkeit wurde der Separator `' ; '` verwendet):

```
Name;Manufacturer;Code;Price  
7400;Intel;I-01-234-97;$0.10  
68HC12;Motorola;M68HC1201234;$3.50
```

### Beispiel

```
string OrderCodes[];
if (fileread(OrderCodes, "ordercodes") > 0) {
    if (lookup(OrderCodes, "", "Code", ';')) {
        schematic(SCH) {
            SCH.parts(P) {
                string OrderCode;
                // both following statements do exactly the same:
                OrderCode = lookup(OrderCodes, P.device.name, "Code", ';');
                OrderCode = lookup(OrderCodes, P.device.name, 2, ';');
            }
        }
    }
}
else
    dlgMessageBox("Missing 'Code' field in file 'ordercodes'");
}
```

## sort()

### Funktion

Sortiert ein Array oder einen Satz von Arrays.

### Syntax

```
void sort(int number, array1[, array2,...]);
```

### Beschreibung

Die sort-Funktion sortiert entweder direkt ein array1, oder sie sortiert einen Satz von Arrays (beginnend mit array2), wobei array1 ein **int**-Array ist, das als Pointer-Array verwendet wird.

In jedem Fall definiert das Argument number die Zahl der Einträge im Array oder in den Arrays.

### Einzelnes Array sortieren

Wenn die sort-Funktion mit einem einzelnen Array aufgerufen wird, wird dieses Array direkt sortiert, wie im folgenden Beispiel:

```
string A[];
int n = 0;
A[n++] = "World";
A[n++] = "Hello";
A[n++] = "The truth is out there...";
sort(n, a);
for (int i = 0; i < n; ++i)
    printf(A[i]);
```

### Einen Satz von Arrays sortieren

Wenn die sort-Funktion mit mehr als einem Array aufgerufen wird, muß das erste Array ein **int**-Array sein, während alle anderen Arrays von jedem Typ sein können. Sie enthalten die zu sortierenden Daten. Das folgende Beispiel zeigt, wie das erste Array als Pointer verwendet wird:

```
numeric string Nets[], Parts[], Instances[], Pins[];
int n = 0;
int index[];
schematic(S) {
    S.nets(N) N.pinrefs(P) {
        Nets[n] = N.name;
        Parts[n] = P.part.name;
        Instances[n] = P.instance.name;
        Pins[n] = P.pin.name;
        ++n;
    }
    sort(n, index, Nets, Parts, Instances, Pins);
    for (int i = 0; i < n; ++i)
        printf("%-8s %-8s %-8s %-8s\n",
            Nets[index[i]], Parts[index[i]],
            Instances[index[i]], Pins[index[i]]);
}
```

Die Idee dahinter ist, daß an ein Netz mehrere Pins angeschlossen sein können. In einer Netzliste wollen Sie unter Umständen die Netznamen sortieren und innerhalb eines Netzes die Bauteilenamen, und so weiter.

Beachten Sie die Verwendung des Schlüsselworts **numeric** in den String-Arrays. Das führt dazu, daß die String-Sortierung einen numerischen Teil am Ende des Namens berücksichtigt (IC1, IC2,... IC9, IC10 anstelle von IC1, IC10, IC2,...IC9).

Wenn man einen Satz von Arrays sortiert, muß das erste (Index-)Array vom Typ **int** »Seite 188 sein und braucht nicht initialisiert zu werden. Jeder vor dem Aufruf der sort-Funktion vorhandene Inhalt wird mit den resultierenden Indexwerten überschrieben.

## Einheiten-Konvertierung

### Funktion

Konvertiert interne Einheiten.

### Syntax

```
real u2inch(int n);  
real u2mic(int n);  
real u2mil(int n);  
real u2mm(int n);
```

### Rückgabewert

u2inch liefert den Wert von *n* in *Inch*.  
u2mic liefert den Wert von *n* in *Micron* (1/1000mm).  
u2mil liefert den Wert von *n* in *Mil* (1/1000inch).  
u2mm liefert den Wert von *n* in *Millimeter*.

**Siehe auch** UL\_GRID »Seite 208

EAGLE speichert alle Koordinaten und Größen als *int* »Seite 188-Werte mit einer Auflösung von 1/10000mm (0.1μ). Die oben angegebenen Einheiten-Konvertier-Funktionen können dazu verwendet werden, die internen Einheiten in die gewünschten Maßeinheiten umzuwandeln.

### Beispiel

```
board(B) {  
  B.elements(E) {  
    printf("%s at (%f, %f)\n", E.name,  
          u2mm(E.x), u2mm(E.y));  
  }  
}
```



## Print-Funktionen

*Print-Funktionen* dienen zur Ausgabe formatierter Strings.

Die folgenden Print-Funktionen sind verfügbar:

- `printf()` »Seite 290
- `sprintf()` »Seite 293

## printf()

### Funktion

Schreibt formatierte Ausgaben in ein File.

### Syntax

```
int printf(string format[, argument, ...]);
```

### Rückgabewert

Die printf-Funktion liefert die Zahl der Zeichen, die in das vom letzten output »Seite 314-Statement geöffnete File geschrieben wurden.

Wenn ein Fehler auftritt, liefert printf -1.

**Siehe auch** sprintf »Seite 293, output »Seite 314, fileerror »Seite 272

### Format-String

Der Format-String steuert, wie die Argumente konvertiert, formatiert und ausgegeben werden. Es müssen genau so viele Argumente vorhanden sein, wie für das Format erforderlich sind. Die Zahl und der Typ der Argumente werden für jedes Format geprüft, und wenn sie nicht den Anforderungen entsprechen, wird eine Fehlermeldung ausgegeben.

Der Format-String enthält zwei Objekt-Typen - *einfache Zeichen* und *Format-Specifier*.

- Einfache Zeichen werden direkt ausgegeben
- Format-Specifier holen Argumente von der Argument-Liste und formatieren sie

### Format-Specifier

Ein Format-Specifier hat folgende Form:

```
% [flags] [width] [.prec] type
```

Jede Format-Spezifizierung beginnt mit dem Prozentzeichen (%). Nach dem % kommt folgendes, in dieser Reihenfolge:

- optional eine Folge von Flag-Zeichen, [flags]
- optional ein Breiten-Specifier, [width]
- optional ein Präzisions-Specifier, [.prec]
- das Konvertiertyp-Zeichen, type

### Konvertiertyp-Zeichen

d	<b>signed decimal int</b>
o	<b>unsigned octal int</b>
u	<b>unsigned decimal int</b>
x	<b>unsigned hexadecimal int</b> (with <b>a</b> , <b>b</b> ,...)
X	<b>unsigned hexadecimal int</b> (with <b>A</b> , <b>B</b> ,...)
f	<b>signed real</b> value von der Form [-] dddd.ddd
e	<b>signed real</b> value von der Form [-] d.dddde [±] ddd
E	wie e, aber mit <b>E</b> für Exponent
g	<b>signed real</b> value entweder wie e oder f, abhängig vom gegebenen Wert und Präzision
G	wie g, aber mit <b>E</b> für Exponent, wenn e-Format verwendet wird
c	einzelnes Zeichen
s	Character-String
%	das %-Zeichen wird ausgegeben

### Flag-Zeichen

Die folgenden Flag-Zeichen können in jeder Kombination und Reihenfolge auftreten.

- " - " das formatierte Argument wird innerhalb des Feldes linksbündig ausgegeben; normalerweise ist die Ausgabe rechtsbündig
- " + " ein positiver Wert mit Vorzeichen wird mit Pluszeichen (+) ausgegeben; normalerweise werden nur negative Werte mit Vorzeichen ausgegeben

" " ein positiver Wert mit Vorzeichen wird mit Leerzeichen am Anfang ausgegeben; wenn "+" und " " angegeben sind, überschreibt "+" die Angabe " "

### Width-Specifier

Der Width-Specifier setzt die minimale Feldbreite für einen Ausgabewert.

Die Breite wird entweder direkt mit einem Dezimalstellen-String oder indirekt mit einem Stern (\*) gesetzt. Wenn Sie \* verwenden, legt das nächste Argument im Aufruf (das vom Typ `int` sein muß) die minimale Feldbreite fest.

Auf keinen Fall führt ein nicht existierendes oder zu ein kleines Feld dazu, daß ein Wert abgeschnitten wird. Wenn das Ergebnis der Konvertierung breiter ist als das Feld, wird das Feld einfach so vergrößert, daß das Ergebnis platz hat.

- `n` Mindestens  $n$  Zeichen werden ausgegeben. Wenn der Ausgabewert weniger als  $n$  Zeichen hat, wird er mit Leerzeichen aufgefüllt (rechts wenn das "-"-Flag gesetzt ist, sonst links).
- `0n` Mindestens  $n$  Zeichen werden ausgegeben. Wenn der Ausgabewert weniger als  $n$  Zeichen hat, wird links mit Nullen aufgefüllt.
- `*` Die Argument-Liste liefert den Width-Specifier, der dem eigentlichen (zu formatierenden) Argument vorausgehen muß.

### Präzisions-Specifier

Ein Präzisions-Specifier beginnt immer mit einem Punkt (.), um ihn von einem vorangehenden Width-Specifier zu trennen. Dann wird, wie bei "Width", die Präzision entweder direkt mit einem Dezimalstellen-String oder indirekt mit einem Stern (\*) angegeben. Wenn Sie \* verwenden, legt das nächste Argument im Aufruf (das vom Typ `int` sein muß) die Präzision fest.

- keiner Präzision auf Defaultwert gesetzt.
- `.0` Für `int`-Typen, Präzision wird auf Default gesetzt; für `real`-Typen, kein Dezimalpunkt wird ausgegeben.
- `.n`  $n$  Zeichen oder  $n$  Dezimalstellen werden ausgegeben. Wenn der Ausgabewert mehr als  $n$  Zeichen hat, kann er abgeschnitten oder gerundet werden (abhängig vom Typ-Zeichen).
- `*` Die Argument-Liste liefert den Präzisions-Specifier, der dem eigentlichen (zu formatierenden) Argument vorausgehen muß.

### Default-Präzisionswerte

<code>douxX</code>	1
<code>eEf</code>	6
<code>gG</code>	alle signifikanten Stellen
<code>c</code>	keine Auswirkung
<code>s</code>	gesamten String ausgeben

### Wie die Präzisionsangabe (.n) die Konvertierung beeinflusst

<code>douxX</code>	$.n$ spezifiziert daß mindestens $n$ Zeichen ausgegeben werden. Wenn das Eingangs-Argument weniger als $n$ Stellen hat, wird der Ausgangswert links mit Nullen aufgefüllt. Wenn das Eingangs-Argument mehr als $n$ Stellen hat, wird die Ausgabe <b>nicht</b> abgeschnitten.
<code>eEf</code>	$.n$ spezifiziert daß $n$ Zeichen nach dem Dezimalpunkt ausgegeben werden, und die letzte ausgegebene Stelle wird gerundet.
<code>gG</code>	$.n$ spezifiziert daß höchstens $n$ signifikante Stellen ausgegeben werden.
<code>c</code>	$.n$ hat keinen Einfluß auf die Ausgabe.
<code>s</code>	$.n$ spezifiziert daß nicht mehr als $n$ Zeichen gedruckt werden.

### Der binäre Wert 0

Im Gegensatz zu `printf` »Seite 293 kann die `printf`-Funktion den binären Wert 0 (0x00) ausgeben.

```
char c = 0x00;
printf("%c", c);
```

### Beispiel

```
int i = 42;
real r = 3.14;
char c = 'A';
string s = "Hello";
printf("Integer: %8d\n", i);
printf("Hex:      %8X\n", i);
printf("Real:     %8f\n", r);
printf("Char:     %-8c\n", c);
printf("String:   %-8s\n", s);
```

## sprintf()

### Funktion

Schreibt eine formatierte Ausgabe in einen String.

### Syntax

```
int sprintf(string result, string format[, argument, ...]);
```

### Rückgabewert

Die `sprintf`-Funktion liefert die Zahl der Zeichen, die in den `result`-String geschrieben wurden.

Im Falle eines Fehlers liefert `sprintf` den Wert `-1`.

**Siehe auch** `printf` »Seite 290

### Format-String

Siehe `printf` »Seite 290.

### Der binäre Wert 0

Bitte beachten Sie, dass `sprintf` den binären Wert 0 (0x00) nicht verarbeiten kann. Wenn der Ergebnis-String 0x00 enthält, werden die folgenden Zeichen ignoriert. Verwenden Sie `printf` »Seite 290 um binäre Daten auszugeben.

### Beispiel

```
string result;  
int number = 42;  
sprintf(result, "The number is %d", number);
```

## String-Funktionen

*String-Funktionen* werden dazu verwendet, Character-Strings zu manipulieren.

Die folgenden String-Funktionen sind verfügbar:

- `strchr()` »Seite 295
- `strjoin()` »Seite 296
- `strlen()` »Seite 297
- `strlwr()` »Seite 298
- `strrchr()` »Seite 299
- `strrstr()` »Seite 300
- `strsplit()` »Seite 301
- `strstr()` »Seite 302
- `strsub()` »Seite 303
- `strtod()` »Seite 304
- `strtol()` »Seite 305
- `strupr()` »Seite 306

## strchr()

### Funktion

Durchsucht einen String nach dem ersten Vorkommen eines gegebenen Zeichens.

### Syntax

```
int strchr(string s, char c);
```

### Rückgabewert

Die `strchr`-Funktion liefert den Integer-Offset des Zeichen im String oder -1, wenn das Zeichen nicht vorkommt.

**Siehe auch** `strchr` »Seite 299, `strstr` »Seite 302

### Beispiel

```
string s = "This is a string";  
char c = 'a';  
int pos = strchr(s, c);  
if (pos >= 0)  
    printf("The character %c is at position %d\n", c, pos);  
else  
    printf("The character was not found\n");
```

## strjoin()

### Funktion

Erzeugt aus einem String-Array einen einzelnen String.

### Syntax

```
string strjoin(string array[], char separator);
```

### Rückgabewert

Die `strjoin`-Funktion liefert die kombinierten Einträge von `array`.

### Beschreibung

`strjoin` fügt alle Einträge aus `array`, getrennt durch den angegebenen `separator` zusammen, und liefert den Ergebnis-String.

Wenn `separator` ein Newline-Zeichen ("`\n`") ist, wird der Ergebnis-String mit einem Newline-Zeichen abgeschlossen. So erhält man eine Textdatei mit N Zeilen (jede davon ist mit einem Newline-Zeichen abgeschlossen). Die Datei wird mit den Funktionen `fileread()` »Seite 277 eingelesen und mit `split` »Seite 301 in ein Array mit N Strings aufgeteilt und zu dem ursprünglichen String, der aus der Datei eingelesen wurde, hinzugefügt.

**Siehe auch** `strsplit` »Seite 301, `lookup` »Seite 285, `fileread` »Seite 277

### Beispiel

```
string a[] = { "Field 1", "Field 2", "Field 3" };  
string s = strjoin(a, ':');
```



## **strlen()**

### **Funktion**

Berechnet die Länge eines Strings.

### **Syntax**

```
int strlen(string s);
```

### **Rückgabewert**

Die strlen-Funktion liefert die Zahl der Zeichen im String.

### **Beispiel**

```
string s = "This is a string";  
int l = strlen(s);  
printf("The string is %d characters long\n", l);
```

## **strlwr()**

### **Funktion**

Wandelt Großbuchstaben in einem String in Kleinbuchstaben um.

### **Syntax**

```
string strlwr(string s);
```

### **Rückgabewert**

Die `strlwr`-Funktion liefert den modifizierten String. Der Original-String (als Parameter übergeben) wird nicht geändert.

**Siehe auch** `strupr` »Seite 306, `tolower` »Seite 270

### **Beispiel**

```
string s = "This Is A String";  
string r = strlwr(s);  
printf("Prior to strlwr: %s - after strlwr: %s\n", s, r);
```

## strrchr()

### Funktion

Durchsucht einen String nach dem letzten Vorkommen eines gegebenen Zeichens.

### Syntax

```
int strrchr(string s, char c);
```

### Rückgabewert

Die `strrchr`-Funktion liefert den Integer-Offset des Zeichens im String oder `-1`, wenn das Zeichen nicht vorkommt.

**Siehe auch** `strchr` »Seite 295, `strrstr` »Seite 300

### Beispiel

```
string s = "This is a string";
char c = 'a';
int pos = strrchr(s, c);
if (pos >= 0)
    printf("The character %c is at position %d\n", c, pos);
else
    printf("The character was not found\n");
```

## strrstr()

### Funktion

Durchsucht einen String nach dem letzten Vorkommen eines gegebenen Substrings.

### Syntax

```
int strrstr(string s1, string s2);
```

### Rückgabewert

Die `strrstr`-Funktion liefert den Integer-Offset des ersten Zeichens von `s2` in `s1`, oder `-1`, wenn der Substring nicht vorkommt.

**Siehe auch** `strstr` »Seite 302, `strchr` »Seite 299

### Beispiel

```
string s1 = "This is a string", s2 = "is a";
int pos = strrstr(s1, s2);
if (pos >= 0)
    printf("The substring starts at %d\n", pos);
else
    printf("The substring was not found\n");
```

## strsplit()

### Funktion

Teilt einen String in einzelne Felder.

### Syntax

```
int strsplit(string &array[], string s, char separator);
```

### Rückgabewert

Die `strsplit`-Funktion liefert die Anzahl der Einträge die nach `array` kopiert wurden.

### Beschreibung

`strsplit` teilt den String `s` am angegebenen `separator` und speichert die so erzeugten Felder in `array`.

Wenn `separator` ein Newline-Zeichen ist ("`\n`"), wird das letzte Feld einfach ignoriert, sofern es leer ist. So erhält man eine Textdatei, die aus `N` Zeilen besteht (jede durch Newline beendet). Diese wird durch die Funktion `fileread()` »Seite 277 eingelesen und in ein Array von `N` Strings aufgeteilt. Mit jedem anderen `separator` ist ein leeres Feld am Ende des Strings gültig. So entstehen aus "`a:b:c:`" 4 Felder, das letzte davon ist leer.

**Siehe auch** `strjoin` »Seite 296, `lookup` »Seite 285, `fileread` »Seite 277

### Beispiel

```
string a[];  
int n = strsplit(a, "Field 1:Field 2:Field 3", ':');
```

## strstr()

### Funktion

Durchsucht einen String nach dem ersten Vorkommen eines gegebenen Substrings.

### Syntax

```
int strstr(string s1, string s2);
```

### Rückgabewert

Die `strstr`-Funktion liefert den Integer-Offset des ersten Zeichens von `s2` in `s1`, oder `-1`, wenn der Substring nicht vorkommt.

**Siehe auch** `strstr` »Seite 300, `strchr` »Seite 295

### Beispiel

```
string s1 = "This is a string", s2 = "is a";
int pos = strstr(s1, s2);
if (pos >= 0)
    printf("The substring starts at %d\n", pos);
else
    printf("The substring was not found\n");
```

## strsub()

### Funktion

Extrahiert einen Substring aus einem String.

### Syntax

```
string strsub(string s, int start[, int length]);
```

### Rückgabewert

Die `strsub`-Funktion liefert den Substring, der durch `start` und `length` definiert ist.

Der Wert für `length` muß positiv sein, anderenfalls wird ein leerer String zurückgegeben. Wenn `length` nicht angegeben ist, wird der Reststring (beginnend bei `start`) zurückgegeben.

Wenn `start` auf eine Position außerhalb des Strings deutet, wird ein leerer String zurückgegeben.

### Beispiel

```
string s = "This is a string";  
string t = strsub(s, 4, 7);  
printf("The extracted substring is: %s\n", t);
```

## **strtod()**

### **Funktion**

Konvertiert einen String in einen Real-Wert.

### **Syntax**

```
real strtod(string s);
```

### **Rückgabewert**

Die `strtod`-Funktion liefert die numerische Repräsentation eines gegebenen Strings als `real`-Wert. Die Konvertierung wird beim ersten Zeichen beendet, das nicht dem Format einer Real-Konstanten »Seite 175 entspricht. Wenn ein Fehler während der Konvertierung auftritt, wird der Wert `0.0` zurückgegeben.

**Siehe auch** `strtod` »Seite 305

### **Beispiel**

```
string s = "3.1415";  
real r = strtod(s);  
printf("The value is %f\n", r);
```



## strtol()

### Funktion

Konvertiert einen String in einen Integer-Wert.

### Syntax

```
int strtol(string s);
```

### Rückgabewert

Die `strtol`-Funktion liefert die numerische Representation eines gegebenen Strings als `int`-Wert. Die Konvertierung wird beim ersten Zeichen beendet, das nicht dem Format einer Integer-Konstanten »Seite 174 entspricht. Wenn ein Fehler während der Konvertierung auftritt, wird der Wert 0 zurückgegeben.

**Siehe auch** `strtod` »Seite 304

### Beispiel

```
string s = "1234";  
int i = strtol(s);  
printf("The value is %d\n", i);
```

## **strupr()**

### **Funktion**

Konvertiert Kleinbuchstaben in einem String in Großbuchstaben.

### **Syntax**

```
string strupr(string s);
```

### **Rückgabewert**

Die `strupr`-Funktion liefert den modifizierten String. Der Original-String (als Parameter übergeben) wird nicht geändert.

**Siehe auch** `strlwr` »Seite 298, `toupper` »Seite 270

### **Beispiel**

```
string s = "This Is A String";  
string r = strupr(s);  
printf("Prior to strupr: %s - after strupr: %s\n", s, r);
```

## Zeit-Funktionen

*Zeit-Funktionen* werden dazu verwendet, die Zeit- und Datums- Informationen zu erhalten und weiterzuverarbeiten.

Die folgenden Zeit-Funktionen sind verfügbar:

- `t2day()` »Seite 309
- `t2dayofweek()` »Seite 309
- `t2hour()` »Seite 309
- `t2minute()` »Seite 309
- `t2month()` »Seite 309
- `t2second()` »Seite 309
- `t2string()` »Seite 309
- `t2year()` »Seite 309
- `time()` »Seite 308

## **time()**

### **Funktion**

Holt die gegenwärtige Systemzeit.

### **Syntax**

```
int time(void);
```

### **Rückgabewert**

Die `time`-Funktion liefert die gegenwärtige Systemzeit als Zahl von Sekunden, die seit einem systemabhängigen Referenzzeitpunkt vergangen sind.

**Siehe auch** Zeit-Konvertierungen »Seite 309, `filetime` »Seite 275

### **Beispiel**

```
int CurrentTime = time();
```

## Zeit-Konvertierungen

### Funktion

Zeit-Wert in Tag, Monat, Jahr etc. konvertieren.

### Syntax

```
int t2day(int t);  
int t2dayofweek(int t);  
int t2hour(int t);  
int t2minute(int t);  
int t2month(int t);  
int t2second(int t);  
int t2year(int t);  
  
string t2string(int t);
```

### Rückgabewert

t2day	liefert den Tag des Monats (1..31)
t2dayofweek	liefert den Tag der Woche (0=sunday..6)
t2hour	liefert die Stunde (0..23)
t2minute	liefert die Minute (0..59)
t2month	liefert den Monat (0..11)
t2second	liefert die Sekunde (0..59)
t2year	liefert das Jahr (einschließlich Jahrhundert!)
t2string	liefert einen formatierten String, der Datum und Zeit enthält

**Siehe auch** time »Seite 308

### Beispiel

```
int t = time();  
printf("It is now %02d:%02d:%02d\n",  
       t2hour(t), t2minute(t), t2second(t));
```

## Builtin-Statements

*Builtin-Statements* werden im allgemeinen dazu verwendet, einen Kontext zu eröffnen, der den Zugriff auf Datenstrukturen und Dateien erlaubt.

Die allgemeine Syntax von Builtin-Statements ist

```
name(parameters) statement
```

wobei *name* der Name des Builtin-Statement ist, *parameters* steht für einen oder mehrere Parameter, und *statement* ist der Code, der innerhalb des vom Builtin-Statement geöffneten Kontexts ausgeführt wird.

Beachten Sie, daß es sich bei *statement* um eine Compound-Statement handeln kann, wie in

```
board(B) {  
    B.elements(E) printf("Element: %s\n", E.name);  
    B.Signals(S)   printf("Signal: %s\n", S.name);  
}
```

Die folgenden Builtin-Statements sind verfügbar:

- `board()` »Seite 311
- `deviceset()` »Seite 312
- `library()` »Seite 313
- `output()` »Seite 314
- `package()` »Seite 315
- `schematic()` »Seite 316
- `sheet()` »Seite 317
- `symbol()` »Seite 318

## board()

### Funktion

Öffnet einen Board-Kontext.

### Syntax

```
board(identifizier) statement
```

### Beschreibung

Das `board`-Statement öffnet einen Board-Kontext wenn das gegenwärtige Editor-Fenster ein Board enthält. Eine Variable vom Typ `UL_BOARD` »Seite 198 wird angelegt und erhält den Namen, den `identifizier` angibt.

Sobald der Board-Kontext erfolgreich geöffnet wurde und eine Board-Variable angelegt ist, wird `statement` ausgeführt. Innerhalb des Gültigkeitsbereichs von `statement` kann man auf die Board-Variable zugreifen, um weitere Daten aus dem Board zu erhalten.

Wenn das gegenwärtige Editor-Fenster kein Board enthält, wird eine Fehlermeldung ausgegeben, und das ULP wird beendet.

**Siehe auch** `schematic` »Seite 316, `library` »Seite 313

### Prüfen, ob ein Board geladen ist

Mit dem `board`-Statement ohne Angabe eines Arguments können Sie prüfen, ob das gegenwärtige Editor-Fenster ein Board enthält. In diesem Fall verhält sich `board` wie eine Integer-Konstante, die den Wert 1 zurückgibt, sofern ein Board geladen ist. Andernfalls wird der Wert 0 zurückgegeben.

### Zugriff auf ein Board von einem Schaltplan aus

Wenn das gegenwärtige Editor-Fenster einen Schaltplan enthält, können Sie trotzdem auf das zugehörige Board zugreifen, indem Sie dem `board`-Statement den Präfix `project` voranstellen, wie in

```
project.board(B) { ... }
```

Das öffnet einen Board-Kontext, unabhängig davon, ob das gegenwärtige Editor-Fenster ein Board oder eine Schaltung enthält. Allerdings muß es auf dem Desktop ein Fenster geben, das dieses Board enthält!

### Beispiel

```
if (board)
  board(B) {
    B.elements(E)
    printf("Element: %s\n", E.name);
  }
```

## device()

### Funktion

Öffnet einen Device-Set-Kontext.

### Syntax

```
deviceset(identifizier) statement
```

### Beschreibung

Das `deviceset`-Statement öffnet einen Device-Set-Kontext wenn das gegenwärtige Editor-Fenster ein Device-Set enthält. Eine Variable vom Typ `UL_DEVICESET` »Seite 205 wird angelegt und erhält den Namen, den `identifizier` angibt.

Sobald der Device-Set-Kontext erfolgreich geöffnet wurde und eine Device-Set-Variable angelegt ist, wird `statement` ausgeführt. Innerhalb des Gültigkeitsbereichs von `statement` kann man auf die Device-Set-Variable zugreifen, um weitere Daten aus dem Device-Set zu erhalten.

Wenn das gegenwärtige Editor-Fenster kein Device-Set enthält, wird eine Fehlermeldung ausgegeben, und das ULP wird beendet.

**Siehe auch** package »Seite 315, symbol »Seite 318, library »Seite 313

### Prüfen, ob ein Device-Set geladen ist

Mit dem `deviceset`-Statement ohne Angabe eines Arguments können Sie prüfen, ob das gegenwärtige Editor-Fenster ein Device-Set enthält. In diesem Fall verhält sich `deviceset` wie eine Integer-Konstante, die den Wert 1 zurückgibt, sofern ein Device-Set geladen ist. Andernfalls wird der Wert 0 zurückgegeben.

### Beispiel

```
if (deviceset)
    deviceset(D) {
        D.gates(G)
        printf("Gate: %s\n", G.name);
    }
```



## library()

### Funktion

Öffnet einen Library-Kontext.

### Syntax

```
library(identifizier) statement
```

### Beschreibung

Das `library`-Statement öffnet einen Library-Kontext wenn das gegenwärtige Editor-Fenster eine Library enthält. Eine Variable vom Typ `UL_LIBRARY` »Seite 213 wird angelegt und erhält den Namen, den `identifizier` angibt.

Sobald der Library-Kontext erfolgreich geöffnet wurde und eine Board-Variable angelegt ist, wird `statement` ausgeführt. Innerhalb des Gültigkeitsbereichs von `statement` kann man auf die Library-Variable zugreifen, um weitere Daten aus der Bibliothek zu erhalten.

Wenn das gegenwärtige Editor-Fenster keine Bibliothek enthält, wird eine Fehlermeldung ausgegeben, und das ULP wird beendet.

**Siehe auch** `board` »Seite 311, `schematic` »Seite 316, `deviceset` »Seite 312, `package` »Seite 315, `symbol` »Seite 318

### Prüfen, ob eine Bibliothek geladen ist

Mit dem `library`-Statement ohne Angabe eines Arguments können Sie prüfen, ob das gegenwärtige Editor-Fenster eine Bibliothek enthält. In diesem Fall verhält sich `library` wie eine Integer-Konstante, die den Wert 1 zurückgibt, sofern eine Bibliothek geladen ist. Andernfalls wird der Wert 0 zurückgegeben.

### Beispiel

```
if (library)
  library(L) {
    L.devices(D)
    printf("Device: %s\n", D.name);
  }
```

## output()

### Funktion

Öffnet ein Ausgabe-File für nachfolgende printf()-Aufrufe.

### Syntax

```
output(string filename[, string mode]) statement
```

### Beschreibung

Das output-Statement öffnet eine Datei mit dem Namen `filename` und dem Parameter `mode` für die Ausgabe mit nachfolgenden printf()-Aufrufen. Sobald die Datei erfolgreich geöffnet wurde, wird `statement` ausgeführt, und danach wird die Datei geschlossen.

Wenn die Datei nicht geöffnet werden kann, wird eine Fehlermeldung ausgegeben, und das ULP wird beendet.

Standardmäßig wird die erzeugte Datei in das **Projekt** Verzeichnis geschrieben.

**Siehe auch** printf »Seite 290, fileerror »Seite 272

### File Modes

Der `mode`-Parameter definiert, wie das File geöffnet werden soll. Wenn kein `mode`-Parameter angegeben ist, gilt der Default-Wert "wt".

"a"	an existierendes File anhängen oder neues File anlegen, falls das File nicht existiert
"w"	neues File anlegen (existierendes überschreiben)
"t"	File im Textmodus öffnen
"b"	File im Binärmodus öffnen

Mode-Parameter können in beliebiger Kombination und Reihenfolge angegeben werden. Allerdings ist nur der letzte aus a und w bzw. t und b signifikant. Die Angabe "abtw" würde zum Beispiel ein Text-File öffnen (entsprechend "wt").

### Verschachtelte Output-Statements

output-Statements können verschachtelt werden, solange genügend File-Handles verfügbar sind - vorausgesetzt, es greifen nicht mehrere aktive output-Statements auf dasselbe File zu.

### Beispiel

```
void PrintText(string s)
{
    printf("This also goes into the file: %s\n", s);
}
output("file.txt", "wt") {
    printf("Directly printed\n");
    PrintText("via function call");
}
```

## package()

### Funktion

Öffnet einen Package-Kontext.

### Syntax

```
package(identifizier) statement
```

### Beschreibung

Das `package`-Statement öffnet einen Package-Kontext wenn das gegenwärtige Editor-Fenster ein Package enthält. Eine Variable vom Typ `UL_PACKAGE` »Seite 215 wird angelegt und erhält den Namen, den `identifizier` angibt.

Sobald der Package-Kontext erfolgreich geöffnet wurde und eine Package-Variable angelegt ist, wird `statement` ausgeführt. Innerhalb des Gültigkeitsbereichs von `statement` kann man auf die Package-Variable zugreifen, um weitere Daten aus dem Package zu erhalten.

Wenn das gegenwärtige Editor-Fenster kein Package enthält, wird eine Fehlermeldung ausgegeben, und das ULP wird beendet.

**Siehe auch** `library` »Seite 313, `deviceset` »Seite 312, `symbol` »Seite 318

### Prüfen, ob ein Package geladen ist

Mit dem `package`-Statement ohne Angabe eines Arguments können Sie prüfen, ob das gegenwärtige Editor-Fenster ein Package enthält. In diesem Fall verhält sich `package` wie eine Integer-Konstante, die den Wert 1 zurückgibt, sofern ein Package geladen ist. Andernfalls wird der Wert 0 zurückgegeben.

### Beispiel

```
if (package)
    package(P) {
        P.contacts(C)
        printf("Contact: %s\n", C.name);
    }
```

## schematic()

### Funktion

Öffnet einen Schematic-Kontext.

### Syntax

```
schematic(identifizier) statement
```

### Beschreibung

Das `schematic`-Statement öffnet einen Schematic-Kontext wenn das gegenwärtige Editor-Fenster eine Schaltung enthält. Eine Variable vom Typ `UL_SCHEMATIC` »Seite 224 wird angelegt und erhält den Namen, den `identifizier` angibt.

Sobald der Schematic-Kontext erfolgreich geöffnet wurde und eine Schematic-Variable angelegt ist, wird `statement` ausgeführt. Innerhalb des Gültigkeitsbereichs von `statement` kann man auf die Schematic-Variable zugreifen, um weitere Daten aus der Schaltung zu erhalten.

Wenn das gegenwärtige Editor-Fenster keine Schaltung enthält, wird eine Fehlermeldung ausgegeben, und das ULP wird beendet.

**Siehe auch** `board` »Seite 311, `library` »Seite 313, `sheet` »Seite 317

### Prüfen, ob eine Schaltung geladen ist

Mit dem `schematic`-Statement ohne Angabe eines Arguments können Sie prüfen, ob das gegenwärtige Editor-Fenster eine Schaltung enthält. In diesem Fall verhält sich `schematic` wie eine Integer-Konstante, die den Wert 1 zurückgibt, sofern eine Schaltung geladen ist. Andernfalls wird der Wert 0 zurückgegeben.

### Zugriff auf einen Schaltplan vom Board aus

Wenn das gegenwärtige Editor-Fenster ein Board enthält, können Sie trotzdem auf den zugehörigen Schaltplan zugreifen, indem Sie dem `schematic`-Statement den Präfix `project` voranstellen, wie in

```
project.schematic(S) { ... }
```

Das öffnet einen Schematic-Kontext, unabhängig davon, ob das gegenwärtige Editor-Fenster ein Board oder eine Schaltung enthält. Allerdings muß es auf dem Desktop ein Fenster geben, das diesen Schaltplan enthält!

### Zugriff auf das gegenwärtige Blatt eines Schaltplans (Sheet)

Verwenden Sie das `sheet` »Seite 317-Statement, um direkt auf das gegenwärtig geladen Sheet zuzugreifen.

### Beispiel

```
if (schematic)
    schematic(S) {
        S.parts(P)
        printf("Part: %s\n", P.name);
    }
```

## sheet()

### Funktion

Öffnet einen Sheet-Kontext.

### Syntax

```
sheet(identifier) statement
```

### Beschreibung

Das `sheet`-Statement öffnet einen Sheet-Kontext wenn das gegenwärtige Editor-Fenster ein Sheet enthält. Eine Variable vom Typ `UL_SHEET` »Seite 226 wird angelegt und erhält den Namen, den `identifier` angibt.

Sobald der Sheet-Kontext erfolgreich geöffnet wurde und eine Sheet-Variable angelegt ist, wird `statement` ausgeführt. Innerhalb des Gültigkeitsbereichs von `statement` kann man auf die Sheet-Variable zugreifen, um weitere Daten aus dem Sheet zu erhalten.

Wenn das gegenwärtige Editor-Fenster kein Sheet enthält, wird eine Fehlermeldung ausgegeben, und das ULP wird beendet.

**Siehe auch** `schematic` »Seite 316

### Prüfen, ob ein Sheet geladen ist

Mit dem `Sheet`-Statement ohne Angabe eines Arguments können Sie prüfen, ob das gegenwärtige Editor-Fenster ein Sheet enthält. In diesem Fall verhält sich `sheet` wie eine Integer-Konstante, die den Wert 1 zurückgibt, sofern ein Sheet geladen ist. Andernfalls wird der Wert 0 zurückgegeben.

### Beispiel

```
if (sheet)
    sheet(S) {
        S.parts(P)
        printf("Part: %s\n", P.name);
    }
```

## symbol()

### Funktion

Öffnet einen Symbol-Kontext.

### Syntax

```
symbol(identifizier) statement
```

### Beschreibung

Das `symbol`-Statement öffnet einen Symbol-Kontext wenn das gegenwärtige Editor-Fenster ein Symbol enthält. Eine Variable vom Typ `UL_SYMBOL` »Seite 229 wird angelegt und erhält den Namen, den `identifizier` angibt.

Sobald der Symbol-Kontext erfolgreich geöffnet wurde und eine Symbol-Variable angelegt ist, wird `statement` ausgeführt. Innerhalb des Gültigkeitsbereichs von `statement` kann man auf die Symbol-Variable zugreifen, um weitere Daten aus dem Symbol zu erhalten.

Wenn das gegenwärtige Editor-Fenster kein Symbol enthält, wird eine Fehlermeldung ausgegeben, und das ULP wird beendet.

**Siehe auch** `library` »Seite 313, `deviceset` »Seite 312, `package` »Seite 315

### Prüfen, ob ein Symbol geladen ist

Mit dem `symbol`-Statement ohne Angabe eines Arguments können Sie prüfen, ob das gegenwärtige Editor-Fenster ein Symbol enthält. In diesem Fall verhält sich `symbol` wie eine Integer-Konstante, die den Wert 1 zurückgibt, sofern ein Symbol geladen ist. Andernfalls wird der Wert 0 zurückgegeben.

### Beispiel

```
if (symbol)
    symbol(S) {
        S.pins(P)
        printf("Pin: %s\n", P.name);
    }
```

## Dialoge

User-Language-Dialoge ermöglichen es, ein eigenes Frontend für ein User-Language-Programm zu definieren.

In den folgenden Abschnitten werden die User-Language-Dialoge detailliert beschrieben:

Vordefinierte Dialoge »Seite 320 beschreibt vordefinierte Standard-Dialoge

Dialog-Objekte »Seite 324 beschreibt die Objekte aus denen ein Dialog bestehen kann

Layout-Information »Seite 348 erklärt wie man die Position von Objekten in einem Dialog bestimmt

Dialog-Funktionen »Seite 349 beschreibt besondere Funktionen, die mit Dialogen verwendet werden können

Ein vollständiges Beispiel »Seite 355 zeigt ein vollständiges ULP mit einem Dialog zur Daten-Eingabe

## Vordefinierte Dialoge

*Vordefinierte Dialoge* sind die typischen Dialoge, die häufig zur Dateiauswahl oder bei Fehlermeldungen verwendet werden.

Es gibt folgende vordefinierte Dialoge:

- `dlgDirectory()` »Seite 321
- `dlgFileOpen()` »Seite 322
- `dlgFileSave()` »Seite 322
- `dlgMessageBox()` »Seite 323

Siehe Dialog-Objekte »Seite 324 für Informationen über das Definieren eigener, komplexer Benutzer-Dialoge.



## dlgDirectory()

### Funktion

Zeigt den Verzeichnis-Dialog.

### Syntax

```
string dlgDirectory(string Title[, string Start])
```

### Rückgabewert

Die dlgDirectory-Funktion liefert den vollen Pfadnamen des gewählten Verzeichnisses.

Hat der Benutzer den Dialog abgebrochen, ist das Resultat ein leerer String.

### Beschreibung

Die dlgDirectory-Funktion zeigt einen Verzeichnis-Dialog in dem der Benutzer ein Verzeichnis selektieren kann.

Title zeigt den Titel des Dialogs.

Wenn Start nicht leer ist, wird diese Angabe als Startpunkt für dlgDirectory verwendet.

**Siehe auch** dlgFileOpen »Seite 322

### Beispiel

```
string dirName;  
dirName = dlgDirectory("Select a directory", "");
```

## dlgFileOpen(), dlgFileSave()

### Funktion

Zeigt einen Datei-Dialog.

### Syntax

```
string dlgFileOpen(string Title[, string Start[, string Filter]])  
string dlgFileSave(string Title[, string Start[, string Filter]])
```

### Rückgabewert

Die Funktionen `dlgFileOpen` und `dlgFileSave` liefern die volle Pfadangabe der gewählten Datei. Bricht der Benutzer den Dialog ab, ist das Ergebnis ein leerer String.

### Beschreibung

Die Funktionen `dlgFileOpen` und `dlgFileSave` zeigen einen File-Dialog, aus dem der Benutzer eine Datei selektieren kann.

`Title` wird als Titel des Dialogs verwendet.

Ist `Start` nicht leer, wird diese Angabe als Startpunkt für den Dialog verwendet. Ansonsten wird das aktuelle Verzeichnis verwendet.

Nur Dateien, die der Angabe von `Filter` entsprechen, werden angezeigt. Wird kein `Filter` angegeben, werden alle Dateien angezeigt.

**Siehe auch** `dlgDirectory` »Seite 321

### Beispiel

```
string fileName;  
fileName = dlgFileOpen("Select a file", "", "*.brd");
```

## dlgMessageBox()

### Funktion

Zeigt eine Message-Box.

### Syntax

```
int dlgMessageBox(string Message[, button_list])
```

### Rückgabewert

Die dlgMessageBox-Funktion liefert den Index der Schaltfläche, die der Benutzer selektiert hat. Die erste Schaltfläche in `button_list` hat den Index 0.

### Beschreibung

Die dlgMessageBox-Funktion zeigt die angegebene Message in einem modalen Dialog-Fenster und wartet darauf, dass der Benutzer eine der Schaltflächen, die über `button_list` definiert wurden, selektiert..

`button_list` ist eine optionale Liste durch Komma getrennter Strings, die einen Satz von Schaltflächen, die unten im Dialog-Fenster angezeigt werden, definiert.

Es können maximal drei Schaltflächen definiert werden. Wird keine `button_list` angegeben, erscheint automatisch "OK".

Die erste Schaltfläche in `button_list` wird die Default-Schaltfläche (sie wird gedrückt, wenn der Benutzer ENTER drückt), und der letzte Eintrag in der Liste wird der "Cancel-Button", der gewählt wird, wenn der Benutzer ESC drückt oder das Dialog-Fenster einfach schließt. Sie können eine andere Schaltfläche als Default-Button definieren, indem Sie den String mit einem '+' beginnen. Wollen Sie eine andere Schaltfläche als Cancel-Button definieren, stellen Sie dem String ein '-' voran. Um einen Schaltflächen-Text mit einem '+' oder '-' zu beginnen, muss das Zeichen mit einem Escape-Zeichen »Seite 354 markiert werden..

Enthält der Text ein '&', wird das folgende Zeichen zum Hotkey. Wenn der Benutzer die entsprechende Taste drückt, wird diese Schaltfläche gewählt. Um das Zeichen '&' im Schaltflächen-Text zu verwenden, muss es mit einem Escape-Zeichen »Seite 354 markiert werden.

### Beispiel

```
if (dlgMessageBox("Are you sure?", "&Yes", "&No") == 0) {  
    // let's do it!  
}
```

## Dialog-Objekte

Ein User-Language-Dialog kann aus folgenden *Dialog-Objekten* bestehen (die einzelnen Begriffe wurden in diesem Fall nicht ins Deutsche übersetzt, da sonst der Zusammenhang zu den ULP-Objekten verloren ginge):

dlgCell »Seite 325 ein Grid-Cell-Kontext  
dlgCheckBox »Seite 326 eine Checkbox  
dlgComboBox »Seite 327 ein Combo-Box-Auswahl-Feld  
dlgDialog »Seite 328 die Grundlage eines jeden Dialogs  
dlgGridLayout »Seite 329 ein Grid-basierter-Layout-Kontext  
dlgGroup »Seite 330 ein Group-Feld  
dlgHBoxLayout »Seite 331 ein Horizontal-Box-Layout-Kontext  
dlgIntEdit »Seite 332 ein Integer-Eingabe-Feld  
dlgLabel »Seite 333 ein Text-Label  
dlgListBox »Seite 334 eine List-Box  
dlgListView »Seite 335 eine List-View  
dlgPushButton »Seite 336 ein Push-Button  
dlgRadioButton »Seite 337 ein Radio-Button  
dlgRealEdit »Seite 338 ein Real-Eingabe-Feld  
dlgSpacing »Seite 339 ein Layout-Spacing-Objekt  
dlgSpinBox »Seite 340 ein Spin-Box-Auswahl-Feld  
dlgStretch »Seite 341 ein Layout-Stretch-Objekt  
dlgStringEdit »Seite 342 ein String-Eingabe-Feld  
dlgTabPage »Seite 343 eine Tab-Page  
dlgTabWidget »Seite 344 ein Tab-Page-Container  
dlgTextEdit »Seite 345 ein Text-Eingabe-Feld  
dlgTextView »Seite 346 ein Text-Viewer-Feld  
dlgVBoxLayout »Seite 347 ein Vertical-Box-Layout-Kontext

## dlgCell

### Funktion

Definiert die Position einer Cell (Zelle) in einem Grid-Layout-Kontext.

### Syntax

```
dlgCell(int row, int column[, int row2, int column2]) statement
```

### Beschreibung

Das `dlgCell`-Statement definiert die Lage einer Cell in einem Grid-Layout-Kontext »Seite 329.

Der Index für Reihe (`row`) und Spalte (`column`) beginnt mit 0, so dass die obere linke Cell den Index (0, 0) hat.

Mit zwei Parametern wird das Dialog-Objekt, das in `statement` angegeben wurde, in einer Cell an der Stelle `row` und `column` platziert. Mit vier Parametern erstreckt sich das Objekt über alle Cells von `row/column` bis zu `row2/column2`.

Standardmäßig enthält `dlgCell` ein `dlgHBoxLayout` »Seite 331. Enthält eine Cell mehr als ein Dialog-Objekt, werden diese nebeneinander horizontal angeordnet.

**Siehe auch** `dlgGridLayout` »Seite 329, `dlgHBoxLayout` »Seite 331, `dlgVBoxLayout` »Seite 347, Layout-Information »Seite 348, Ein vollständiges Beispiel »Seite 355

### Beispiel

```
string Text;
dlgGridLayout {
    dlgCell(0, 0) dlgLabel("Cell 0,0");
    dlgCell(1, 2, 4, 7) dlgTextEdit(Text);
}
```

## dlgCheckBox

### Funktion

Definiert eine Checkbox.

### Syntax

```
dlgCheckBox(string Text, int &Checked) [ statement ]
```

### Beschreibung

Das `dlgCheckBox`-Statement definiert eine Checkbox mit dem angegebenen Text.

Wenn Text ein ' & ' enthält, wird das folgende Zeichen als Hotkey markiert. Wenn der Benutzer `Alt+hotkey` drückt, wird die Checkbox selektiert/deselektiert. Um ein ' & ' -Zeichen im Text zu verwenden, muss er mit einem Escape-Zeichen »Seite 354 markiert werden.

`dlgCheckBox` wird hauptsächlich in `dlgGroup` »Seite 330 benutzt, kann aber auch anders verwendet werden.

Alle Check-Boxen innerhalb eines gemeinsamen Dialogs müssen **unterschiedliche** `Checked`-Variablen haben!

Wenn ein Benutzer eine `dlgCheckBox` wählt, wird die entsprechende `Checked`-Variable auf 1 gesetzt, andernfalls ist sie auf 0 gesetzt. Der ursprüngliche Wert von `Checked` definiert, ob eine Checkbox anfänglich selektiert ist oder nicht. Wenn `Checked` ungleich 0 ist, ist die Checkbox defaultmäßig selektiert.

Das optionale `statement` wird jedesmal ausgeführt, wenn Sie die `dlgCheckBox` selektieren/deselektieren.

**Siehe auch** `dlgRadioButton` »Seite 337, `dlgGroup` »Seite 330, Layout-Information »Seite 348, Ein vollständiges Beispiel »Seite 355

### Beispiel

```
int mirror = 0;
int rotate = 1;
int flip   = 0;
dlgGroup("Orientation") {
    dlgCheckBox("&Mirror", mirror);
    dlgCheckBox("&Rotate", rotate);
    dlgCheckBox("&Flip", flip);
}
```

## dlgComboBox

### Funktion

Definiert ein Combo-Box-Auswahl-Feld.

### Syntax

```
dlgComboBox(string array[], int &Selected) [ statement ]
```

### Beschreibung

Das `dlgComboBox`-Statement definiert ein Combo-Box-Auswahlfeld mit dem Inhalt von `array`.

`Selected` reflektiert den Index des selektieren Combo-Box-Eintrags. Der erste Eintrag hat den Index 0.

Jedes Element von `array` legt den Inhalt eines Eintrags in der Combo-Box fest. Keiner der Strings in `array` darf leer sein (sollte ein leerer String existieren, werden alle folgenden, inklusive des leeren, ignoriert).

Das optionale `statement` wird jedesmal ausgeführt, wenn die Auswahl in der `dlgComboBox` verändert wird.

Bevor `statement` ausgeführt wird, werden alle Variablen, die in den Dialog-Objekten verwendet werden neu eingelesen und jede Veränderung innerhalb von `statement` wird im Dialog angezeigt.

Ist der Ausgangswert von `Selected` ausserhalb des Bereichs der Indices von `array`, wird dieser auf 0 gesetzt..

**Siehe auch** `dlgListBox` »Seite 334, `dlgLabel` »Seite 333, Layout-Information »Seite 348, Ein vollständiges Beispiel »Seite 355

### Beispiel

```
string Colors[] = { "red", "green", "blue", "yellow" };
int Selected = 2; // initially selects "blue"
dlgComboBox(Colors, Selected) dlgMessageBox("You have selected " +
Colors[Selected]);
```

## dlgDialog

### Funktion

Führt einen User-Language-Dialog aus.

### Syntax

```
int dlgDialog(string Title) block ;
```

### Rückgabewert

Die `dlgDialog`-Funktion liefert einen Integer-Wert, dem durch den Aufruf der `dlgAccept()` »Seite 350-Funktion eine benutzerspezifische Bedeutung zugeordnet werden kann.

Wird der Dialog einfach geschlossen, ist der Rückgabewert 0.

### Beschreibung

Die `dlgDialog`-Funktion, die durch `block` »Seite 252 definiert wird. Das ist das einzige Dialog-Objekt das tatsächlich eine User-Language-Builtin-Funktion ist. Sie kann überall wo ein Funktionsaufruf erlaubt ist, verwendet werden.

`block` enthält normalerweise andere Dialog-Objekte »Seite 324. Man kann aber auch andere User-Language-Statements verwenden, zum Beispiel, um bedingungsabhängig dem Dialog Objekte hinzuzufügen (siehe das zweite der folgenden Beispiele).

Standardmäßig enthält `dlgDialog` ein `dlgVBoxLayout` »Seite 347, so dass man sich bei einem einfachen Dialog um das Layout kein Gedanken machen muss.

Ein `dlgDialog` sollte an einer Stelle den Aufruf der `dlgAccept()` »Seite 350-Funktion enthalten, um dem Benutzer zu erlauben den Dialog zu schliessen und dessen Inhalt zu akzeptieren.

Wenn Sie nur eine einfache Message-Box oder einen einfachen Dialog brauchen, können Sie statt dessen auch einen der Vordefinierten Dialoge »Seite 320 verwenden.

**Siehe auch** `dlgGridLayout` »Seite 329, `dlgHBoxLayout` »Seite 331, `dlgVBoxLayout` »Seite 347, `dlgAccept` »Seite 350, `dlgReset` »Seite 352, `dlgReject` »Seite 353, Ein vollständiges Beispiel »Seite 355

### Beispiele

```
int Result = dlgDialog("Hello") {  
    dlgLabel("Hello world");  
    dlgPushButton("+OK") dlgAccept();  
};  
  
int haveButton = 1;  
dlgDialog("Test") {  
    dlgLabel("Start");  
    if (haveButton)  
        dlgPushButton("Here") dlgAccept();  
};
```



## dlgGridLayout

### Funktion

Öffnet einen Grid-Layout-Kontext.

### Syntax

`dlgGridLayout statement`

### Beschreibung

Das `dlgGridLayout`-Statement öffnet einen Grid-Layout-Kontext.

Das einzige Dialog-Objekt, das direkt in `statement` verwendet werden kann, ist `dlgCell` »Seite 325, das die Position eines Dialog-Objekts im Grid-Layout festlegt.

Die Indices für `row` und `column` beginnen mit 0, so dass die obere linke Cell den Index (0, 0) hat.

Die Anzahl der Reihen und Spalten wird automatisch an die Position von Dialog-Objekten, die innerhalb des Grid-Layout-Kontexts definiert werden, angepasst. Die Anzahl der Reihen und Spalten muss nicht explizit definiert werden.

**Siehe auch** `dlgCell` »Seite 325, `dlgHBoxLayout` »Seite 331, `dlgVBoxLayout` »Seite 347, Layout-Information »Seite 348, Ein vollständiges Beispiel »Seite 355

### Beispiel

```
dlgGridLayout {  
    dlgCell(0, 0) dlgLabel("Row 0/Col 0");  
    dlgCell(1, 0) dlgLabel("Row 1/Col 0");  
    dlgCell(0, 1) dlgLabel("Row 0/Col 1");  
    dlgCell(1, 1) dlgLabel("Row 1/Col 1");  
}
```

## dlgGroup

### Funktion

Definiert ein Group-Feld.

### Syntax

```
dlgGroup(string Title) statement
```

### Beschreibung

Das `dlgGroup`-Statement definiert eine Gruppe mit dem gegebenen `Title`.

Standardmäßig enthält `dlgGroup` ein `dlgVBoxLayout` »Seite 347, so braucht man sich bei einer einfachen Group keine Gedanken zum Layout machen.

`dlgGroup` wird hauptsächlich für einen Satz von Radio-Buttons »Seite 337 oder Check-Boxes »Seite 326 verwendet, kann aber auch jedes andere beliebige Objekt in `statement` enthalten.  
Radio-Buttons in einer `dlgGroup` werden mit 0 beginnend numeriert.

**Siehe auch** `dlgCheckBox` »Seite 326, `dlgRadioButton` »Seite 337, Layout-Information »Seite 348, Ein vollständiges Beispiel »Seite 355

### Beispiel

```
int align = 1;
dlgGroup("Alignment") {
    dlgRadioButton("&Top", align);
    dlgRadioButton("&Center", align);
    dlgRadioButton("&Bottom", align);
}
```

## dlgHBoxLayout

### Funktion

Öffnet einen Horizontal-Box-Layout-Kontext.

### Syntax

`dlgHBoxLayout statement`

### Beschreibung

Das `dlgHBoxLayout`-Statement öffnet einen Horizontal-Box-Layout-Kontext für das angegebene `statement`.

**Siehe auch** `dlgGridLayout` »Seite 329, `dlgVBoxLayout` »Seite 347, `Layout-Information` »Seite 348, `Ein vollständige Beispiel` »Seite 355

### Beispiel

```
dlgHBoxLayout {  
    dlgLabel ("Box 1");  
    dlgLabel ("Box 2");  
    dlgLabel ("Box 3");  
}
```

## dlgIntEdit

### Funktion

Definiert ein Integer-Eingabe-Feld.

### Syntax

```
dlgIntEdit(int &Value, int Min, int Max)
```

### Beschreibung

Das `dlgIntEdit`-Statement definiert ein Integer-Eingabe-Feld mit einem in `Value` angegebenen Wert..

Ist `Value` ursprünglich ausserhalb des Bereichs `Min` und `Max`, wird er auf diesen Bereich limitiert.

**Siehe auch** `dlgRealEdit` »Seite 338, `dlgStringEdit` »Seite 342, `dlgLabel` »Seite 333, Layout-Information »Seite 348  
, Ein vollständiges Beispiel »Seite 355

### Beispiel

```
int Value = 42;
dlgHBoxLayout {
    dlgLabel("Enter a &Number between 0 and 99");
    dlgIntEdit(Value, 0, 99);
}
```

## dlgLabel

### Funktion

Definiert ein Text-Label.

### Syntax

```
dlgLabel(string Text [, int Update])
```

### Beschreibung

Das `dlgLabel`-Statement definiert ein Label mit dem angegebenen Text.

Text kann entweder ein fester String wie "Hello" sein, oder eine String-Variable.

Wenn der Update-Parameter nicht 0 ist und Text eine String-Variable, kann dessen Inhalt mit statement modifiziert werden, z. B. wird ein `dlgPushButton` »Seite 336, und sein Label automatisch aktualisiert. Das ist natürlich nur sinnvoll wenn Text eine eindeutig bestimmte String-Variable ist (und beispielsweise keine Loop-Variable eines `for`-Statements).

Enthält Text ein '&'-Zeichen, wird das folgende Zeichen zum Hot-Key. Drückt der Benutzer `Alt+hotkey`, wird das Objekt, das direkt nach `dlgLabel` definiert wurde, aktiv. Um ein '&'-Zeichen direkt im Text zu verwenden, muss man es mit einem Escape-Zeichen »Seite 354 markieren.

**Siehe auch** Layout-Information »Seite 348, Ein vollständiges Beispiel »Seite 355

### Beispiel

```
string Name = "Linus";
dlgHBoxLayout {
    dlgLabel("Enter &Name");
    dlgStringEdit(Name, 1);
}
```

## dlgListBox

### Funktion

Definiert ein List-Box-Auswahl-Feld.

### Syntax

```
dlgListBox(string array[], int &Selected) [ statement ]
```

### Beschreibung

Das `dlgListBox`-Statement definiert ein List-Box-Auswahl-Feld mit dem Inhalt von `array`.

`Selected` gibt den Index des selektierten List-Box-Eintrags wieder. Der erste Eintrag hat den Index 0.

Jedes Element von `array` legt den Inhalt einer Zeile in der List-Box fest. Keiner der Strings in `array` darf leer sein (sollte ein leerer String existieren, werden alle folgenden, inklusive des leeren, ignoriert).

Das optionale `statement` wird immer dann ausgeführt, wenn der Benutzer einen Doppelklick auf einen Eintrag der `dlgListBox` ausführt.

Bevor `statement` ausgeführt wird, werden alle Variablen, die von Dialog-Objekten benutzt werden, aktualisiert. Alle Änderungen, die in `statement` gemacht wurden, wirken sich auf den Dialog aus, sobald das Statement zurückgegeben wird.

Ist der Ausgangswert von `Selected` ausserhalb des Index-Bereichs von `array`, wird kein Eintrag selektiert.

**Siehe auch** `dlgComboBox` »Seite 327, `dlgListView` »Seite 335, `dlgLabel` »Seite 333, Layout-Information »Seite 348, Ein vollständiges Beispiel »Seite 355

### Beispiel

```
string Colors[] = { "red", "green", "blue", "yellow" };
int Selected = 2; // initially selects "blue"
dlgListBox(Colors, Selected) dlgMessageBox("You have selected " +
Colors[Selected]);
```

## dlgListView

### Funktion

Definiert ein mehrspaltiges List-View-Auswahl-Feld.

### Syntax

```
dlgListView(string Headers, string array[], int &Selected) [ statement ]
```

### Beschreibung

Das `dlgListView`-Statement definiert ein mehrspaltiges List-View-Auswahl-Feld mit dem Inhalt, der in `array` angegeben ist.

`Headers` definiert die durch Tabulatoren getrennte Liste der Spalten-Überschriften.

`Selected` gibt den Index des selektierten List-View-Eintrags von `array` wieder (die Reihenfolge in der die Einträge tatsächlich angezeigt werden, kann unterschiedliche sein, da der Inhalt einer `dlgListView` in den verschiedenen Spalten sortiert werden kann). Der erste Eintrag hat den Index 0. Wenn kein spezieller Eintrag selektiert werden soll, wählt man für `Selected` den Wert -1.

Jedes Element von `array` legt den Inhalt einer Zeile in der List-View fest und muss durch Tabulatoren getrennte Werte enthalten. Sind weniger Werte eines Elements in `array` definiert als im `Headers`-String vorgegeben, bleiben die restlichen Felder leer. Sind mehr Werte eines Element in `array` angegeben als im `Headers`-String, werden die überzähligen stillschweigend ignoriert. Keiner der Strings in `array` darf leer sein (sollte ein leerer String vorhanden sein, werden alle nachfolgenden, inklusive dem Leerstring ignoriert).

Das optionale `statement` wird ausgeführt, wann immer der Benutzer auf einen Eintrag in `dlgListView` doppelklickt.

Bevor `statement` ausgeführt wird, werden alle Variablen, die mit den Dialog-Objekten benutzt wurden, aktualisiert. Alle Änderungen, die in `statement` gemacht wurden, wirken sich auf den Dialog aus, sobald das Statement zurückgegeben wird.

Ist der Ausgangswert von `Selected` ausserhalb des Index-Bereichs von `array`, wird kein Eintrag selektiert.

Ist `Headers` ein leerer String, wird das erste Element von `array` als Header-String benutzt. Folglich ist der Index des ersten Eintrags dann 1.

Der Inhalt von `dlgListView` kann in einer beliebigen Spalte sortiert werden, indem man auf dessen Spalten-Header klickt. Die Spalten-Reihenfolge kann man durch Anklicken&Ziehen des Spalten-Headers verändern. Beachten Sie, dass keine dieser Änderungen eine Auswirkung auf den Inhalt von `array` hat. Soll der Inhalt alphanumerisch sortiert werden, kann ein `numeric string[]`-Array verwendet werden.

**Siehe auch** `dlgListBox` »Seite 334, `dlgLabel` »Seite 333, Layout-Information »Seite 348, Ein vollständiges Beispiel »Seite 355

### Beispiel

```
string Colors[] = { "red\tThe color RED", "green\tThe color GREEN",  
"blue\tThe color BLUE" };  
int Selected = 0; // initially selects "red"  
dlgListView("Name\tDescription", Data, Selected) dlgMessageBox("You have  
selected " + Colors[Selected]);
```

## dlgPushButton

### Funktion

Definiert einen Push-Button.

### Syntax

```
dlgPushButton(string Text) statement
```

### Beschreibung

Das `dlgPushButton`-Statement definiert einen Push-Button mit dem angegebenen Text.

Enthält Text ein ' & ' -Zeichen, wird das folgende Zeichen zum Hot-Key. Wenn der Benutzer dann `Alt+hotkey` drückt, wird dieser Button selektiert. Soll ein ' & ' -Zeichen im ZText verwendet werden, muss es mit einem Escape-Zeichen »Seite 354 markiert werden.

Beginnt Text mit einem ' + ' -Zeichen, wird dieser Button der Default-Button. Dieser wird betätigt, wenn der Benutzer ENTER drückt.

Wenn Text mit einem ' - ' -Zeichen beginnt, wird dieser Button der Cancel-Button. Dieser wird gewählt wenn der Benutzer den Dialog schließt.

**Achtung: Stellen Sie sicher, dass das `statement` eines so markierten Buttons einen Aufruf von `dlgReject()` »Seite 353 enthält! Ansonsten ist es dem Benutzer nicht möglich den Dialog überhaupt zu schließen!**

Um ein ' + ' oder ' - ' -Zeichen als erstes Zeichen des Textes zu verwenden, muss es mit einem Escape-Zeichen »Seite 354 markiert werden.

Wenn der Benutzer einen `dlgPushButton` selektiert, wird das angegebene `statement` ausgeführt. Bevor `statement` ausgeführt wird, werden alle Variablen, die mit den Dialog-Objekten benutzt wurden, aktualisiert. Alle Änderungen, die in `statement` gemacht wurden, wirken sich auf den Dialog aus, sobald das Statement zurückgegeben wird.

**Siehe auch** Layout-Information »Seite 348, Dialog-Funktionen »Seite 349, Ein vollständige Beispiel »Seite 355

### Beispiel

```
int defaultWidth = 10;
int defaultHeight = 20;
int width = 5;
int height = 7;
dlgPushButton("&Reset defaults") {
    width = defaultWidth;
    height = defaultHeight;
}
dlgPushButton("+&Accept") dlgAccept();
dlgPushButton("-Cancel") { if (dlgMessageBox("Are you sure?", "Yes", "No")
== 0) dlgReject(); }
```



## dlgRadioButton

### Funktion

Definiert einen Radio-Button.

### Syntax

```
dlgRadioButton(string Text, int &Selected) [ statement ]
```

### Beschreibung

Das `dlgRadioButton`-Statement definiert einen Radio-Button mit dem angegebenen Text.

Enthält Text ein `'&'`-Zeichen, wird das folgende Zeichen zum Hot-Key. Wenn der Benutzer dann `Alt+hotkey` drückt, wird dieser Button selektiert. Soll ein `'&'`-Zeichen im Text verwendet werden, muss es mit einem Escape-Zeichen »Seite 354 markiert werden.

`dlgRadioButton` kann nur innerhalb einer `dlgGroup` »Seite 330 verwendet werden.

Alle Radio-Buttons innerhalb derselben Group müssen **dieselbe** `Selected`-Variable haben!

Wenn der Benutzer einen `dlgRadioButton` selektiert, wird der Index dieses Buttons innerhalb der `dlgGroup` in der `Selected`-Variablen gespeichert.

Der Ausgangswert von `Selected` definiert, welcher Radio-button per default selektiert ist. Liegt `Selected` ausserhalb des gültigen Bereichs dieser Group, ist kein Radio-Button selektiert. Um die richtige Radio-Button-Selektion zu erhalten, muss `Selected` schon **vor** der Definition des ersten `dlgRadioButton` festgelegt werden, und darf nicht verändert werden, während man weitere Radio-Buttons hinzufügt. Ansonsten ist es ungewiss welcher Radio-Button (wenn überhaupt einer) selektiert wird.

Das optionale `statement` wird ausgeführt, wenn ein `dlgRadioButton` selektiert wird.

**Siehe auch** `dlgCheckBox` »Seite 326, `dlgGroup` »Seite 330, Layout-Information »Seite 348, Ein vollständiges Beispiel »Seite 355

### Beispiel

```
int align = 1;
dlgGroup("Alignment") {
    dlgRadioButton("&Top", align);
    dlgRadioButton("&Center", align);
    dlgRadioButton("&Bottom", align);
}
```

## dlgRealEdit

### Funktion

Definiert ein Real-Eingabe-Feld.

### Syntax

```
dlgRealEdit(real &Value, real Min, real Max)
```

### Beschreibung

Das `dlgRealEdit`-Statement definiert ein Real-Eingabe-Feld mit dem angegebenen `Value` (Wert).

Wenn `Value` ursprünglich ausserhalb des Bereiches von `Min` und `Max` liegt, wird dieser auf diese Werte begrenzt.

**Siehe auch** `dlgIntEdit` »Seite 332, `dlgStringEdit` »Seite 342, `dlgLabel` »Seite 333, `Layout-Information` »Seite 348, `Ein vollständiges Beispiel` »Seite 355

### Beispiel

```
real Value = 1.4142;
dlgHBoxLayout {
    dlgLabel("Enter a &Number between 0 and 99");
    dlgRealEdit(Value, 0.0, 99.0);
}
```

## dlgSpacing

### Funktion

Definiert zusätzlichen Abstand in einem Box-Layout-Kontext.

### Syntax

```
dlgSpacing(int Size)
```

### Beschreibung

Das `dlgSpacing`-Statement definiert zusätzlichen Abstand in einem Vertical- bzw. Horizontal-Box-Layout-Kontext.

`Size` definiert die Anzahl der Pixel des zusätzlichen Abstands.

**Siehe auch** `dlgHBoxLayout` »Seite 331, `dlgVBoxLayout` »Seite 347, `dlgStretch` »Seite 341, Layout-Information »Seite 348, Ein vollständiges Beispiel »Seite 355

### Beispiel

```
dlgVBoxLayout {  
    dlgLabel("Label 1");  
    dlgSpacing(40);  
    dlgLabel("Label 2");  
}
```

## dlgSpinBox

### Funktion

Definiert ein Spin-Box-Auwahl-Feld.

### Syntax

```
dlgSpinBox(int &Value, int Min, int Max)
```

### Beschreibung

Das `dlgSpinBox`-Statement definiert ein Spin-Box-Auswahl-Feld mit dem angegebenen `Value`.

Wenn `Value` ursprünglich ausserhalb des Bereiches von `Min` und `Max` liegt, wird dieser auf diese Werte begrenzt.

**Siehe auch** `dlgIntEdit` »Seite 332, `dlgLabel` »Seite 333, Layout-Information »Seite 348, Ein vollständiges Beispiel »Seite 355

### Beispiel

```
int Value = 42;
dlgHBoxLayout {
    dlgLabel("&Select value");
    dlgSpinBox(Value, 0, 99);
}
```

## dlgStretch

### Funktion

Definiert einen leeren, dehnbaren Abstand in einem Box-Layout-Kontext.

### Syntax

```
dlgStretch(int Factor)
```

### Beschreibung

Das `dlgStretch`-Statement definiert einen leeren dehnbaren Abstand in einem Vertical- oder einem Horizontal-Box-Layout-Kontext.

`Factor` definiert den Dehnungsfaktor des Abstands.

**Siehe auch** `dlgHBoxLayout` »Seite 331, `dlgVBoxLayout` »Seite 347, `dlgSpacing` »Seite 339, Layout-Information »Seite 348, Ein vollständiges Beispiel »Seite 355

### Beispiel

```
dlgHBoxLayout {  
    dlgStretch(1);  
    dlgPushButton("+OK")    { dlgAccept(); };  
    dlgPushButton("Cancel") { dlgReject(); };  
}
```

## dlgStringEdit

### Funktion

Definiert ein String-Eingabe-Feld.

### Syntax

```
dlgStringEdit(string &Text)
```

### Beschreibung

Das `dlgStringEdit`-Statement definiert ein Text-Eingabe-Feld mit dem angegebenen Text.

**Siehe auch** `dlgRealEdit` »Seite 338, `dlgIntEdit` »Seite 332, `dlgTextEdit` »Seite 345, `dlgLabel` »Seite 333, `Layout-Information` »Seite 348, `Ein vollständiges Beispiel` »Seite 355

### Beispiel

```
string Name = "Linus";
dlgHBoxLayout {
    dlgLabel("Enter &Name");
    dlgStringEdit(Name);
}
```

## dlgTabPage

### Funktion

Definiert eine Tab-Page.

### Syntax

```
dlgTabPage(string Title) statement
```

### Beschreibung

Das `dlgTabPage`-Statement definiert eine Tab-Page mit dem angegebenen `Title`, die `statement` enthält.

Enthält `Title` ein `'&'`-Zeichen, wird das folgende Zeichen zum Hot-Key. Drückt der Benutzer `Alt+hotkey`, wird diese Tab-Page geöffnet. Soll im Text ein `'&'`-Zeichen verwendet werden, muss es mit einem Escape-Zeichen »Seite 354 markiert werden.

Tab-Pages können nur innerhalb eines `dlgTabWidget` »Seite 344 verwendet werden.

Standardmäßig enthält `dlgTabPage` ein `dlgVBoxLayout` »Seite 347, so dass man sich bei einer einfachen Tab-Page nicht um das Layout kümmern muss.

**Siehe auch** `dlgTabWidget` »Seite 344, Layout-Information »Seite 348, Ein vollständiges Beispiel »Seite 355

### Beispiel

```
dlgTabWidget {  
    dlgTabPage("Tab &1") {  
        dlgLabel("This is page 1");  
    }  
    dlgTabPage("Tab &2") {  
        dlgLabel("This is page 2");  
    }  
}
```

## dlgTabWidget

### Funktion

Definiert einen Container für Tab-Pages.

### Syntax

`dlgTabWidget statement`

### Beschreibung

Das `dlgTabWidget`-Statement definiert einen Platzhalter für einen Satz von Tab-Pages.

`statement` ist eine Liste eines oder mehrerer `dlgTabPage` »Seite 343-Objekte. Es dürfen keine anderen Dialog-Objekte in dieser Liste enthalten sein.

**Siehe auch** `dlgTabPage` »Seite 343, Layout-Information »Seite 348, Ein vollständiges Beispiel »Seite 355

### Beispiel

```
dlgTabWidget {  
    dlgTabPage("Tab &1") {  
        dlgLabel("This is page 1");  
    }  
    dlgTabPage("Tab &2") {  
        dlgLabel("This is page 2");  
    }  
}
```



## dlgTextEdit

### Funktion

Definiert ein mehrzeiliges Text-Eingabe-Feld.

### Syntax

```
dlgTextEdit(string &Text)
```

### Beschreibung

Das `dlgTextEdit`-Statement definiert ein mehrzeiliges text-Eingabe-Feld mit dem angegebenen `Text`.

Die einzelnen Zeilen in `Text` müssen mit einem Newline-Zeichen ( ' \n ' ) getrennt werden. Beliebige Leerzeichen am Ende der `Text`-Zeilen werden gelöscht. Leere Zeilen am Endes des Textes werden vollständig entfernt.

**Siehe auch** `dlgStringEdit` »Seite 342, `dlgTextView` »Seite 346, `dlgLabel` »Seite 333, Layout-Information »Seite 348, Ein vollständiges Beispiel »Seite 355

### Beispiel

```
string Text = "This is some text.\nLine 2\nLine 3";
dlgVBoxLayout {
    dlgLabel("&Edit the text");
    dlgTextEdit(Text);
}
```

## dlgTextView

### Funktion

Definiert ein mehrzeiliges Text-Viewer-Feld.

### Syntax

```
dlgTextView(string Text)
```

### Beschreibung

Das `dlgTextView`-Statement definiert ein mehrzeiliges Text-Viewer-Feld dmit dem angegebenen `Text`.

Der Text darf Rich-Text »Seite 356-Tags enthalten.

**Siehe auch** `dlgTextEdit` »Seite 345, `dlgLabel` »Seite 333, Layout-Information »Seite 348, Ein vollständiges Beispiel »Seite 355

### Beispiel

```
string Text = "This is some text.\nLine 2\nLine 3";
dlgVBoxLayout {
    dlgLabel("&View the text");
    dlgTextView(Text);
}
```

## dlgVBoxLayout

### Funktion

Öffnet einen Vertical-Box-Layout-Kontext.

### Syntax

`dlgVBoxLayout statement`

### Beschreibung

Das `dlgVBoxLayout`-Statement öffnet einen Vertical-Box-Layout-Kontext für das angegebene `statement`.

Standardmäßig enthält `dlgDialog` »Seite 328 ein `dlgVBoxLayout`, so dass man sich bei einfachen Dialogen keine Gedanken zum Layout machen muss.

**Siehe auch** `dlgGridLayout` »Seite 329, `dlgHBoxLayout` »Seite 331, `Layout-Information` »Seite 348, Ein vollständiges Beispiel »Seite 355

### Beispiel

```
dlgVBoxLayout {  
  dlgLabel ("Box 1");  
  dlgLabel ("Box 2");  
  dlgLabel ("Box 3");  
}
```

## Layout-Information

Alle Objekte eines User-Language-Didialogs werden in einem *Layout-Kontext* verwendet.

Es gibt verschiedene Layout-Kontexte, wie grid »Seite 329, horizontal »Seite 331 oder vertical »Seite 347.

### Grid-Layout-Kontext

Objekte in einem Grid-Layout-Kontext müssen die Raster-Koordinaten der Zelle (Cell) oder der Zellen angeben, in der/denen sie platziert werden sollen. Um einen Text in Reihe (row) 5, Spalte (column) 2 zu platzieren, schreiben Sie

```
dlgGridLayout {
    dlgCell(5, 2) dlgLabel("Text");
}
```

Soll das Objekt über mehr als eine Zelle gehen, müssen Sie die Koordinaten der Start-Zelle und der End-Zelle angeben. Um eine Group zu platzieren, die sich von Reihe 1, Spalte 2 über Reihe 3, Spalte 5 erstreckt, schreiben Sie

```
dlgGridLayout {
    dlgCell(1, 2, 3, 5) dlgGroup("Title") {
        //...
    }
}
```

### Horizontal-Layout-Kontext

Objekte in einem Horizontal-Layout-Kontext werden von rechts nach links platziert.

Die Sonder-Objekte `dlgStretch` »Seite 341 und `dlgSpacing` »Seite 339 können verwendet werden, um die Verteilung der Abstände zu verfeinern.

Um zwei Buttons zu definieren, die sich bis an den rechten Rand des Dialogs erstrecken, schreiben Sie

```
dlgHBoxLayout {
    dlgStretch(1);
    dlgPushButton("+OK")    dlgAccept();
    dlgPushButton("Cancel") dlgReject();
}
```

### Vertical-Layout-Kontext

Objekte in einem Vertical-Layout-Kontext folgen denselben Vorschriften wie die in einem Horizontal-Layout-Kontext mit dem Unterschied, dass sie von oben nach unten angeordnet werden.

### Gemischter Layout-Kontext

Vertical-, Horizontal- und Grid-Layout-Kontexte können gemischt werden, um die gewünschte Dialog-Struktur zu erzeugen. Siehe Ein vollständiges Beispiel »Seite 355.

## Dialog-Funktionen

Folgende Funktionen können mit User-Language-Dialogen verwendet werden:

dlgAccept() »Seite 350	schließt den Dialog und akzeptiert dessen Inhalt
dlgRedisplay() »Seite 351	aktualisiert den Dialog nachdem beliebige Werte verändert wurden
dlgReset() »Seite 352	setzt alle Dialog-Objekte auf den Ursprungswert zurück
dlgReject() »Seite 353	schließt den Dialog und verwirft dessen Inhalt

## dlgAccept()

### Funktion

Schließt den Dialog und akzeptiert dessen Inhalt.

### Syntax

```
void dlgAccept([ int Result ]);
```

### Beschreibung

Die `dlgAccept`-Funktion schließt `dlgDialog` »Seite 328, und kehrt zurück nachdem das aktuelle Statement beendet wurde.

Jede Änderung, die der Benutzer im Dialog macht, wird übernommen und an die Variablen, die bei der Definition der Dialog-Objekte »Seite 324 angegeben wurden, kopiert.

Die optionale Angabe von `Result` ist der Wert der vom Dialog geliefert wird. Das sollte typischerweise ein positiver Integer-Wert sein. Wird kein Wert angegeben, ist der Default-Wert gleich 1.

Bitte beachten Sie, dass `dlgAccept()` wieder in die normale Programm- Routine zurückkehrt, so wie in dieser Sequenz:

```
dlgPushButton("OK") {  
    dlgAccept();  
    dlgMessageBox("Accepting!");  
}
```

Das Statement nach `dlgAccept()` wird noch ausgeführt!

**Siehe auch** `dlgReject` »Seite 353, `dlgDialog` »Seite 328, Ein vollständiges Beispiel »Seite 355

### Beispiel

```
int Result = dlgDialog("Test") {  
    dlgPushButton("+OK")    dlgAccept(42);  
    dlgPushButton("Cancel") dlgReject();  
};
```

## dlgRedisplay()

### Funktion

Aktualisiert den Dialog-Inhalt nachdem Werte verändert wurden.

### Syntax

```
void dlgRedisplay(void);
```

### Beschreibung

Die `dlgRedisplay`-Funktion wird aufgerufen, um den `dlgDialog` »Seite 328, nach dem Verändern von Variablen, die in den Dialog-Objekten »Seite 324 definiert wurden, zu aktualisieren.

Sie brauchen nur `dlgRedisplay()` aufrufen, wenn der Dialog während der Ausführung des Programmcodes aktualisiert werden soll. Im folgenden Beispiel wird der Status auf "Running..." gesetzt und `dlgRedisplay()` muss aufgerufen werden, um die Änderungen für die Ausführung des Programms wirksam zu machen. Nach dem Ändern des Status auf "Finished.", braucht man `dlgRedisplay()` nicht mehr aufrufen, da alle Dialog-Objekte nach dem Verlassen des Statements aktualisiert werden.

**Siehe auch** `dlgReset` »Seite 352, `dlgDialog` »Seite 328, Ein vollständiges Beispiel »Seite 355

### Beispiel

```
string Status = "Idle";
int Result = dlgDialog("Test") {
    dlgLabel(Status, 1); // note the '1' to tell the label to be
updated!
    dlgPushButton("+OK")    dlgAccept(42);
    dlgPushButton("Cancel") dlgReject();
    dlgPushButton("Run") {
        Status = "Running...";
        dlgRedisplay();
        // some program action here...
        Status = "Finished.";
    }
};
```

## dlgReset()

### Funktion

Setzt alle Dialog-Objekte auf ihren ursprünglichen Wert.

### Syntax

```
void dlgReset(void);
```

### Beschreibung

Die `dlgReset`-Funktion kopiert die ursprünglichen Werte in alle Dialog-Objekte »Seite 324 des aktuellen `dlgDialog` »Seite 328 zurück.

Alle Änderungen, die der Benutzer im Dialog machte, werden verworfen.

Ein Aufruf von `dlgReject()` »Seite 353 impliziert einen Aufruf von `dlgReset()`.

**Siehe auch** `dlgReject` »Seite 353, `dlgDialog` »Seite 328, Ein vollständiges Beispiel »Seite 355

### Beispiel

```
int Number = 1;
int Result = dlgDialog("Test") {
    dlgIntEdit(Number);
    dlgPushButton("+OK")    dlgAccept(42);
    dlgPushButton("Cancel") dlgReject();
    dlgPushButton("Reset")  dlgReset();
};
```



## dlgReject()

### Funktion

Schließt den Dialog und verwirft seinen Inhalt.

### Syntax

```
void dlgReject([ int Result ]);
```

### Beschreibung

Die `dlgReject`-Funktion veranlasst, dass `dlgDialog` »Seite 328 geschlossen wird und nach dem Beenden der aktuellen Statement-Sequenz zurückkehrt.

Jede Änderung, die der Benutzer im Dialog machte, wird verworfen. Die Variablen, die während der Definition der Dialog-Objekte »Seite 324 übergeben wurden, werden auf Ihren ursprünglichen Wert zurückgesetzt.

Der optionale Wert für `Result` wird vom Dialog zurückgegeben. Typischerweise ist das 0 oder ein negativer Integer-Wert. Wird kein Wert angegeben, ist er standardmäßig 0.

Beachten Sie, dass `dlgReject()` wieder in die normale Programm-Routine zurückkehrt, wie in dieser Sequenz:

```
dlgPushButton("Cancel") {  
    dlgReject();  
    dlgMessageBox("Rejecting!");  
}
```

Das Statement nach `dlgReject()` wird auch noch ausgeführt!

Der Aufruf von `dlgReject()` impliziert den Aufruf von `dlgReset()` »Seite 352.

**Siehe auch** `dlgAccept` »Seite 350, `dlgReset` »Seite 352, `dlgDialog` »Seite 328, Ein vollständiges Beispiel »Seite 355

### Beispiel

```
int Result = dlgDialog("Test") {  
    dlgPushButton("+OK")    dlgAccept(42);  
    dlgPushButton("Cancel") dlgReject();  
};
```

## Escape-Zeichen

Die Zeichen '&', '+' und '-' haben in Schaltflächen- oder Label-Texten eine besondere Bedeutung, so dass sie mit *Escape-Zeichen* markiert werden müssen, wenn sie tatsächlich im Text erscheinen sollen.

Dazu müssen Sie dem Zeichen einen *Backslash* voranstellen, so wie in

```
dlgLabel("Miller \\& Co.");
```

Das Ergebnis wird im Dialog so aussehen: "Miller & Co."

Beachten Sie, dass hier in Wirklichkeit **zwei** Backslash-Zeichen verwendet wurden, da diese Zeile erst durch den User-Language-Parser geht, der den ersten Backslash abzieht.

## Ein vollständiges Beispiel

Hier folgt ein vollständiges Beispiel eines User-Language-Diagrams:

```
int hor = 1;
int ver = 1;
string fileName;
int Result = dlgDialog("Enter Parameters") {
    dlgHBoxLayout {
        dlgStretch(1);
        dlgLabel("This is a simple dialog");
        dlgStretch(1);
    }
    dlgHBoxLayout {
        dlgGroup("Horizontal") {
            dlgRadioButton("&Top", hor);
            dlgRadioButton("&Center", hor);
            dlgRadioButton("&Bottom", hor);
        }
        dlgGroup("Vertical") {
            dlgRadioButton("&Left", ver);
            dlgRadioButton("C&enter", ver);
            dlgRadioButton("&Right", ver);
        }
    }
    dlgHBoxLayout {
        dlgLabel("File &name:");
        dlgStringEdit(fileName);
        dlgPushButton("Bro&wse") {
            fileName = dlgFileOpen("Select a file", fileName);
        }
    }
    dlgGridLayout {
        dlgCell(0, 0) dlgLabel("Row 0/Col 0");
        dlgCell(1, 0) dlgLabel("Row 1/Col 0");
        dlgCell(0, 1) dlgLabel("Row 0/Col 1");
        dlgCell(1, 1) dlgLabel("Row 1/Col 1");
    }
    dlgSpacing(10);
    dlgHBoxLayout {
        dlgStretch(1);
        dlgPushButton("+OK")    dlgAccept();
        dlgPushButton("Cancel") dlgReject();
    }
};
```

## Rich Text

*Rich Text* enthält eine Teilmenge von Tags (Steuerzeichen), die zum Formatieren von HTML-Seiten verwendet werden. Damit kann man Texte von einigen Objekten im User-Language-Dialog »Seite 319, in der #usage »Seite 169-Directive oder in der Description »Seite 60 von Bibliotheks-Objekten formatieren.

Text wird zu Rich Text, wenn die erste Zeile ein Tag enthält. Wenn das nicht der Fall ist und Sie den Text formatieren wollen, schließen Sie den ganzen Text in das `<qt>...</qt>` Tag.

Die folgende Tabelle listet alle unterstützten Rich-Text-Tags mit ihren verfügbaren Attributen auf:

Tag	Beschreibung
<code>&lt;qt&gt;...&lt;/qt&gt;</code>	Ein Rich-Text-Dokument. Es versteht folgende Attribute
<ul style="list-style-type: none"> <li><code>bgcolor</code> - Die Hintergrundfarbe, z. B. <code>bgcolor="yellow"</code> or <code>bgcolor="#0000FF"</code>.</li> <li><code>background</code> - Das Hintergrundbild, zum Beispiel <code>background="granit.xpm"</code>.</li> <li><code>text</code> - Die default Textfarbe, z. B. <code>text="red"</code>.</li> <li><code>link</code> - Die Farbe eines Links, z. B. <code>link="green"</code>.</li> </ul>	
<code>&lt;h1&gt;...&lt;/h1&gt;</code>	Eine Haupt-Überschrift.
<code>&lt;h2&gt;...&lt;/h2&gt;</code>	Eine untergeordnete Überschrift.
<code>&lt;h3&gt;...&lt;/h3&gt;</code>	Eine weiter untergeordnete Überschrift.
<code>&lt;p&gt;...&lt;/p&gt;</code>	Ein links-bündiger Abschnitt. Bestimmen Sie die Anordnung mit dem <code>align</code> Attribut. Mögliche Werte sind <code>left</code> , <code>right</code> und <code>center</code> .
<code>&lt;center&gt;...&lt;/center&gt;</code>	Ein zentrierter Abschnitt.
<code>&lt;blockquote&gt;...&lt;/blockquote&gt;</code>	Ein eingerückter Abschnitt, sinnvoll für Zitate.
<code>&lt;ul&gt;...&lt;/ul&gt;</code>	Eine ungeordnete Liste. Sie können auch ein <code>type</code> -Argument angeben um einen Bullet-Style zu definieren. Default ist <code>type=disc</code> , andere Typen sind <code>circle</code> und <code>square</code> .
<code>&lt;ol&gt;...&lt;/ol&gt;</code>	Eine geordnete Liste. Sie können auch ein <code>type</code> -Argument angeben um die Art der Nummerierung zu definieren. Default ist <code>type="1"</code> , andere Typen sind <code>"a"</code> und <code>"A"</code> .
<code>&lt;li&gt;...&lt;/li&gt;</code>	Ein Punkt in einer Liste. Dieses Tag kann nur innerhalb eines <code>ol</code> oder <code>ul</code> Kontext verwendet werden.
<code>&lt;pre&gt;...&lt;/pre&gt;</code>	Für größere Mengen von Code. Leerzeichen im Inhalt bleiben erhalten. Für kleinere Mengen Code, benutzen Sie den Inline-style <code>code</code> .
<code>&lt;a&gt;...&lt;/a&gt;</code>	Ein Anker oder Link. Folgende Attribute sind erlaubt
<ul style="list-style-type: none"> <li><code>href</code> - Das Referenz-Ziel wie in <code>&lt;a href="target.qml"&gt;...&lt;/a&gt;</code>. Sie dürfen auch einen zusätzlichen Anker innerhalb des angegebenen Ziel-Dokuments angeben, z. B. <code>&lt;a href="target.qml#123"&gt;...&lt;/a&gt;</code>.</li> <li><code>name</code> - Der Anker-Name, wie in <code>&lt;a name="target"&gt;...&lt;/a&gt;</code>.</li> </ul>	
<code>&lt;em&gt;...&lt;/em&gt;</code>	Emphasized (kursiv)(genauso wie <code>&lt;i&gt;...&lt;/i&gt;</code> ).
<code>&lt;strong&gt;...&lt;/strong&gt;</code>	Stark (genauso wie <code>&lt;b&gt;...&lt;/b&gt;</code> ).
<code>&lt;i&gt;...&lt;/i&gt;</code>	Kursiver Text.
<code>&lt;b&gt;...&lt;/b&gt;</code>	Fetter Text.
<code>&lt;u&gt;...&lt;/u&gt;</code>	Unterstrichener Text.
<code>&lt;big&gt;...&lt;/big&gt;</code>	Eine größere Texthöhe.
<code>&lt;small&gt;...&lt;/small&gt;</code>	Eine kleinere Texthöhe.
<code>&lt;code&gt;...&lt;/code&gt;</code>	Kennzeichnet Code. (wie auch <code>&lt;tt&gt;...&lt;/tt&gt;</code> ). Für größere Mengen an Code, verwenden Sie das Block-Tag <code>pre</code> .
<code>&lt;tt&gt;...&lt;/tt&gt;</code>	Typewriter Schriftart.
<code>&lt;font&gt;...&lt;/font&gt;</code>	Zur Bestimmung von Texthöhe, Schrift-Familie und Textfarbe. Das Tag versteht folgende Attribute:
<ul style="list-style-type: none"> <li><code>color</code> - Die Textfarbe, z. B. <code>color="red"</code> or <code>color="#FF0000"</code>.</li> <li><code>size</code> - Die logische Größe der Schrift. Logische Größen von 1 bis 7 werden unterstützt. Der Wert darf entweder absolut, z. B. <code>size=3</code>, oder relativ, wie <code>size=-2</code> sein. Im letzten Fall werden die Größen einfach addiert.</li> <li><code>face</code> - Die Schriftart-Familie, z. B. <code>face=times</code>.</li> </ul>	

`<img...>` Ein Bild. Dieses Tag versteht die folgenden Attribute:

- `src` - Den Namen des Bildes, z. B. ``.
- `width` - Die Breite des Bildes. Passt das Bild nicht in die angegebene Größe, wird es automatisch skaliert.
- `height` - Die Höhe des Bildes.
- `align` - Bestimmt wo das Bild platziert wird. Defaultmäßig wird ein Bild "inline" platziert, genauso wie ein

Buchstabe. Legen Sie `left` oder `right` fest, um das Bild an der entsprechenden Stelle zu platzieren.

`<hr>` Eine waagrechte Linie.

`<br>` Ein Zeilenumbruch.

`<nobr>...</nobr>` Kein Zeilenumbruch. Erhält "Word Wrap".

`<table>...</table>` Eine Tabellen-Definition. Die Standardtabelle ist ohne Rahmen. Geben Sie das boolsche Attribut `border` an um einen Rahmen zu erhalten. Andere Attribute sind:

- `bgcolor` - Die Hintergrundfarbe.
- `width` - Die Tabellenbreite. Wird entweder in Pixel oder in Prozent der Spaltenbreite angegeben, z. B. `width=80%`.
- `border` - Die Breite des Tabellenrandes. Default ist 0 (= kein Rand).
- `cellspacing` - Zusätzlicher Leerraum um die Tabellenzelle. Default ist 2.
- `cellpadding` - Zusätzlicher Leerraum um den Inhalt einer Tabellenzelle. Default ist 1.

`<tr>...</tr>` Eine Tabellen-Reihe. Kann nur mit `table` verwendet werden. Versteht die Attribute:

- `bgcolor` - Die Hintergrundfarbe.

`<td>...</td>` Eine Zelle in einer Tabelle. Kann nur innerhalb `tr` verwendet werden. Versteht die Attribute:

- `bgcolor` - Die Hintergrundfarbe.
- `width` - Die Zellenbreite. Wird entweder in Pixel oder in Prozent der gesamten Tabellenbreite angegeben, z. B. `width=50%`.
- `colspan` - Legt fest wieviele Spalten diese Zelle belegt. Default ist 1.
- `rowspan` - Legt fest wieviele Reihen diese Zelle belegt. Default ist 1.
- `align` - Positionierung, mögliche Angaben sind `left`, `right` und `center`. Default ist links-bündig.

`<th>...</th>` Eine "Header"-Zelle in der Tabelle. Wie `td` aber als default mit zentrierter Ausrichtung und fetter Schriftart.

`<author>...</author>` Markiert den Autor des Texts.

`<dl>...</dl>` Eine Definitions-Liste.

`<dt>...</dt>` Ein Definitions-Tag. Kann nur innerhalb `dl` verwendet werden.

`<dd>...</dd>` Definitions-Daten. Kann nur innerhalb `dl` verwendet werden.

Tag	Bedeutung
<code>&amp;lt;</code>	<
<code>&amp;gt;</code>	>
<code>&amp;amp;</code>	&
<code>&amp;nbsp;</code>	Leerzeichen ohne Umbruch
<code>&amp;auml;</code>	ä
<code>&amp;ouml;</code>	ö
<code>&amp;uuml;</code>	ü
<code>&amp;Auml;</code>	Ä
<code>&amp;Ouml;</code>	Ö
<code>&amp;Uuml;</code>	Ü
<code>&amp;szlig;</code>	ß
<code>&amp;copy;</code>	©
<code>&amp;deg;</code>	°
<code>&amp;micro;</code>	μ
<code>&amp;plusmn;</code>	±

## Automatischer Backup

### Maximum backup level

Der Write-Befehl erzeugt Backup-Kopien der gesicherten Dateien. Sie haben denselben Namen wie die Originaldateien mit einer modifizierten Extension, nach dem Muster

.x#n

Dabei steht für 'x' der Buchstabe

b in Board-Dateien  
s in Schaltplan-Dateien  
l in Bibliotheks-Dateien

n steht für eine einstellige Zahl von 1..9. Höhere Ziffern zeigen ältere Dateien an.

Die feste Position des Zeichens '#' ermöglicht das einfache Löschen aller Backup-Dateien mit dem Betriebssystembefehl

DEL \*.?#?

Bitte beachten Sie, daß Backup-Dateien mit derselben Ziffer 'n' nicht notwendigerweise konsistente Paare von Platinen- und Schaltplan-Dateien repräsentieren.

Die maximale Zahl von Backup-Kopien kann im Backup-Dialog »Seite 28 gesetzt werden.

### Auto-Backup-Interval

Wurde eine Zeichnung modifiziert, wird automatisch nach der unter *Auto backup interval* eingestellten Zeit eine Sicherungskopie erstellt.

Diese Sicherungskopie erhält den Namen nach folgendem Schema:

.x##

Dabei steht für 'x' der Buchstabe

b in Board-Dateien  
s in Schaltplan-Dateien  
l in Bibliotheks-Dateien

Die Sicherheits-Bakup-Datei wird nach einem erfolgreichen Abspeichern der Zeichnung wieder gelöscht. Kann die Zeichnung nicht mit dem WRITE-Befehl gespeichert werden (z. B. aufgrund eines Stromausfalls), benennen Sie die Datei einfach um. So kann sie als normale Schaltplan-, Board- bzw. Bibliotheksdatei wieder geladen werden.

Die Sicherungsintervall kann im Backup-Dialog »Seite 28 gesetzt werden.

## Forward&Back-Annotation

Eine Schaltungsdatei und die zugehörigen Platinendatei sind durch automatische Forward&Back-Annotation logisch verknüpft. Der Benutzer muß sich darum normalerweise nicht kümmern. Dennoch wird in diesem Abschnitt beschrieben, was genau bei der Forward&Back-Annotation geschieht:

- Holt man ein neues Bauteil in den Schaltplan, so wird das zugehörige Package in der linken unteren Ecke der Platinenzeichnung platziert. Enthält das Bauteil Pins mit der Direction "Pwr", dann werden die zugehörigen Pads automatisch mit dem entsprechenden Versorgungssignal verbunden.
- Wird ein Bauteil aus einer Schaltung gelöscht, so wird das zugehörige Package auch aus der Platine gelöscht. Wires, die mit dem Package verbunden waren, bleiben unberührt. Unter Umständen sind zusätzliche Vias erforderlich, um diese Signale an den Stellen zu verbinden, an denen sich die Pads des gelöschten Bauteils befunden haben.
- Löscht man ein Bauteil aus einer Platine, werden alle "Gates", die sich in diesem Bauteil befinden, aus der Schaltung gelöscht. Dies kann unterschiedliche Sheets betreffen, falls die "Gates" auf mehrere Sheets verteilt waren.
- Nach einer Operation, die ein Pad von einem Signal entfernt, das als Supply-Layer realisiert ist, kann die Anzeige der Thermal-/Annulus-Symbole fehlerhaft sein. Ein Window-Refresh korrigiert die Anzeige. Derselbe Effekt kann bei Undo-/Redo-Operationen auftreten, die sich auf mit einem Supply-Layer verbundene Pads beziehen.
- Die Befehle Pinswap und Gateswap im Schaltplan sorgen dafür, daß alle notwendigen Änderungen auch im Board durchgeführt werden. Allerdings können die Wires anschließend die Design Rules verletzen. Der Benutzer sollte deshalb nach diesen Befehlen die Platine entsprechend editieren.
- Um sicherzustellen, daß eine Platine und ein Schaltplan zusammengehören (und über Forward&Back-Annotation verbunden sind), müssen die beiden Dateien denselben Namen (mit Extensions .brd und .sch) haben und im gleichen Verzeichnis gespeichert sein.
- Der Replace-Befehl prüft, ob alle Pads des alten Package, die einem Pin zugewiesen waren, auch im neuen Package vorhanden sind - unabhängig davon, ob sie mit einem Signal verbunden sind oder nicht.
- Liegen die Pins zweier Schaltplan-Symbole übereinander (Verbindung ohne sichtbare Netz-Linie), dann wird eine Netz-Linie erzeugt, wenn eines der Bauteile wegbewegt wird. Damit wird eine unnötige Ripup-Operation im Board vermieden.

## Konsistenzprüfung

Damit die Forward&Back-Annotation wirksam werden kann, müssen Platine und Schaltung konsistent sein. Das heißt, sie müssen äquivalente Bauteile und Netze bzw. Signale enthalten.

Unter normalen Umständen sind Platine und Schaltung immer konsistent, sofern sie nicht separat editiert worden sind (in diesem Fall würden Sie mit der Meldung *"No Forward&Back-Annotation will be performed!"* gewarnt worden sein).

Wenn ein Platinen/Schaltungs-Paar geladen wird, überprüft das Programm Konsistenzmarkierungen in den Dateien, um zu sehen, ob sie noch konsistent sind. Weisen diese Markierungen auf eine Inkonsistenz hin, dann bietet Ihnen das Programm an, einen Electrical Rule Check »Seite 65 (ERC) auszuführen, der beide Dateien überprüft.

Fällt die Prüfung positiv aus, werden die Dateien als konsistent markiert, und die Forward&Back-Annotation wird aktiviert.

Werden die Dateien als inkonsistent erkannt, erscheint das ERC-Protokoll in einem Text-Editor-Fenster, und die Forward&Back-Annotation wird **nicht** aktiviert.

Das ERC-Protokoll enthält mehrere Abschnitte, in denen die gefundenen Inkonsistenzen aufgelistet werden. Jeder Abschnitt kann entfallen, falls keine entsprechende Fehlermeldung vorliegt. **Bitte geraten Sie nicht in Panik, wenn zahlreiche Fehlermeldungen erscheinen. In den meisten Fällen reduziert schon eine einzige Korrektur (wie die Umbenennung eines Netzes) die Zahl der Meldungen für den nächsten Durchlauf erheblich.**

```
Parts not found in board:
  IC1
  R7
```

Dieser Abschnitt gibt die Namen der Parts aus, die in der Schaltung vorhanden sind, aber nicht im Board.

```
Elements not found in schematic:
  C33
  D2
```

Dieser Abschnitt gibt die Namen der Elemente aus, die in der Platine vorhanden sind, aber nicht in der Schaltung.

Die folgenden Abschnitte sind nur dann vorhanden, wenn die Part- und Elementnamen konsistent sind:

```
Parts/Elements with inconsistent packages:
  IC12
  R1
```

Dieser Abschnitt gibt die Namen der Parts/Elemente aus, die in der Schaltung und im Board vorhanden sind, die aber inkonsistente Packages haben. Packages werden als konsistent angesehen, wenn sie einen Satz von Pads/Smds mit gleichen Namen haben.

```
Parts/Elements with inconsistent values:
  R55      100k      47k
  C99      10n       10p
```

Dieser Abschnitt gibt die Namen der Parts/Elemente aus, die in der Schaltung und im Board vorhanden sind, die aber verschiedene Values haben. Die zweite Spalte gibt den Value des Parts im Schaltplan aus, in der dritten Spalte erscheint der Name des Elements im Board.

Die folgenden Abschnitte erscheinen nur, wenn die Packages der Parts und der Elemente konsistent sind:

```
Pins/Pads with different connections:
  Part  Gate  Pin  Net  Pad  Signal
  IC5   A     2    GND  2    S$42
  R3     R     1    D1   1    D2
```

Dieser Abschnitt gibt die Namen der *Pins* und *Pads* aus, die mit unterschiedlichen Netzen/Signalen im Schaltplan und im Board verbunden sind. Die Spalte *Net* enthält den Netznamen der Schaltung, während die Spalte *Signal* den Signalnamen des Boards enthält. Ist jeder der beiden Einträge leer, sind die Pins/Pads nicht an ein Netz/Signal angeschlossen.



**Platine und Schaltung konsistent machen**

Um ein inkonsistentes Schaltungs-/Platinen-Paar konsistent zu machen, müssen Sie alle im ERC-Protokoll aufgeführten Inkonsistenzen manuell beseitigen. Das kann mit Hilfe folgender Editor-Befehle erreicht werden: NAME »Seite 87, VALUE »Seite 128, PINSWAP »Seite 98, REPLACE »Seite 109 etc. Nach der Korrektur müssen Sie den ERC »Seite 65-Befehl nochmals verwenden, um die Dateien zu überprüfen und um die Forward&Back-Annotation aktivieren zu können.

## Einschränkungen

Folgende Aktionen sind in einer Platine nicht erlaubt, wenn die Back-Annotation aktiv ist, wenn also die Schaltung ebenfalls geladen ist:

- Bauteil hinzufügen (ADD) oder kopieren, das Pads oder Smads enthält
- Luftlinie löschen
- Verbindungen mit dem Signal-Befehl definieren
- Schaltungsteile mit Paste von einem Board in ein Board kopieren, wenn darin Pads, Smads oder verbundene Signale enthalten sind

Sollten Sie eine dieser Operationen auszuführen versuchen, dann erhalten Sie eine Meldung, daß dies unter Kontrolle der Back-Annotation nicht möglich ist. Bitte führen Sie die Operation dann im Schaltplan aus, sie wird dann automatisch in die Platine übernommen. Sollten Sie die Operation dennoch im Board ausführen wollen, müssen Sie das Schaltplan-Fenster schließen. In diesem Fall sind Schaltung und Board aber nicht mehr konsistent!

## Technische Unterstützung

Als registrierter EAGLE-Benutzer erhalten Sie von CadSoft kostenlose technische Unterstützung. Es gibt folgende Möglichkeiten, uns zu erreichen oder die neuesten Programmversionen, Bibliotheken und Treiber zu erhalten:

CadSoft Computer GmbH  
Hofmark 2  
84568 Pleiskirchen  
Deutschland

Vertrieb	08635-6989-10
Hotline	08635-6989-30
Fax	08635-6989-40
Email	<a href="mailto:Support@CadSoft.DE">Support@CadSoft.DE</a>
URL	<a href="http://www.CadSoft.DE">www.CadSoft.DE</a>

## Lizenz

Als legaler EAGLE-Benutzer müssen Sie im Besitz einer registrierten Benutzer-Lizenz sein. Bitte überprüfen Sie, ob Ihr "User License Certificate" einen Hologrammaufkleber mit dem EAGLE-Logo enthält und ein Label, auf dem Ihr Name, Ihre Adresse, eine Seriennummer und ein Installationscode aufgedruckt sind. Sollten Sie Zweifel an der Echtheit des Zertifikats haben, setzen Sie sich bitte mit unserem Support-Personal »Seite 363 in Verbindung.

Es gibt verschiedene Lizenz-Typen, die sich darin unterscheiden, wie viele Benutzer erlaubt sind und wo das Programm verwendet werden darf.

### Single-User License

Nur EIN Benutzer darf das Programm zu einer bestimmten Zeit benutzen. Der Benutzer darf das Programm allerdings auf unterschiedlichen Computern installieren, solange er sicherstellt, daß nur eine Programmkopie gleichzeitig verwendet wird.

Ein typischer Anwender einer "Single-User License" ist ein Benutzer, der einen fest installierten PC und zusätzlich einen Notebook-Computer hat, den er unterwegs benutzt. Da er immer nur einen Computer benutzt, reicht die "Single-User License" aus.

### 3-User License

Bis zu DREI Benutzer dürfen auf drei unterschiedlichen Computern gleichzeitig mit dem Programm arbeiten. Die einzige Einschränkung besteht darin, daß alle drei Computer im Besitz des Lizenznehmers sein müssen und daß sich alle drei Computer im selben Gebäude befinden müssen.

### 5-User License

Wie "3-User License", aber es sind bis zu FÜNF Benutzer erlaubt.

### Network License

Das Programm darf auf EINEM LAN-Server installiert werden, und eine UNBESCHRÄNKTE Zahl von Benutzern, die in diesen LAN-Server eingeloggt sind, darf das Programm gleichzeitig benutzen.

### Commercial License

Das Programm darf für jeden Zweck verwendet werden, kommerziell oder privat.

### Educational License

Das Programm darf ausschließlich in Ausbildungsstätten wie Schulen, Universitäten oder Lehrwerkstätten zu Ausbildungszwecken verwendet werden.

### Student License

Das Programm darf ausschließlich für private Zwecke verwendet werden. Jede kommerzielle Anwendung ist untersagt. Es stellt eine Verletzung der Lizenzbedingungen dar, wenn Sie durch Gebrauch einer Studentenversion Geld verdienen.

## Produkt-Registrierung

Bevor Sie mit EAGLE arbeiten können, müssen Sie dem Programm Ihre persönlichen Lizenzdaten "mitteilen".

Bitte stellen Sie sicher, daß Ihre EAGLE-Lizenzdiskette in das Laufwerk eingelegt ist.

Dann müssen Sie Ihren `Installation Code` eingeben, der auf dem Aufkleber des `User License Certificate` aufgedruckt ist. Dieser Code besteht aus zehn Kleinbuchstaben und muß genauso eingegeben werden, wie er auf dem Aufkleber steht.

Nach dem Betätigen der Enter-Taste oder dem Anklicken des **OK**-Buttons ist EAGLE mit Ihren persönlichen Lizenzdaten installiert.

Wenn Sie Probleme mit der Installation haben oder über die Gültigkeit Ihrer Lizenz im Zweifel sind, setzen Sie sich bitte mit unserem Technischen Support »Seite 363 in Verbindung.

### Nachträgliches Installieren von Modulen

Soll die Lizenz um das Schaltplan-/Autorouter-Modul erweitert werden, erhalten Sie von uns ein neues `User License Certificate` mit einem neuen Installations-Code. Diesen müssen Sie dem Programm mitteilen. Dazu starten Sie das EAGLE-Programm und wählen im Control Panel »Seite 23 im Help-Menü den Eintrag Product Registration.

## EAGLE-Editionen

EAGLE ist in drei verschiedenen Editionen verfügbar, um den Ansprüchen verschiedener Benutzergruppen gerecht zu werden.

### Professional

Die *Professional* Edition hat keinerlei Einschränkungen.

### Standard

Die *Standard* Edition besitzt folgende Einschränkungen:

- die Platinenfläche ist auf 160x100mm (6.3x4inch) beschränkt, was einer ganzen Europakarte entspricht
- es stehen nur vier Signallayer (Top, Route2, Route15 und Bottom) zur Verfügung

### Light

Die *Light* Edition besitzt folgende Einschränkungen:

- die Platinenfläche ist auf 100x80mm (4x3.2inch) beschränkt, was einer halben Europakarte entspricht
- es stehen nur zwei Signal-Layer (Top und Bottom) zur Verfügung
- ein Schaltplan kann nur aus einem einzelnen Sheet bestehen

Falls Sie eine Fehlermeldung der Form

*The Light edition of EAGLE can't perform the requested action!*

erhalten, so bedeutet dies, daß Sie versucht haben, etwas zu tun, das im Widerspruch zu den Einschränkungen der benutzten EAGLE-Edition steht, wie etwa ein Bauteil außerhalb der zulässigen Fläche zu plazieren.

Sowohl die *Standard* als auch die *Light* Edition von EAGLE ist in der Lage, Dateien zu laden, die mit der *Professional* Edition erzeugt wurden, selbst wenn diese Zeichnungen die Editier-Möglichkeiten der verwendeten Edition übersteigen.

Um zu sehen, welcher Edition Ihre Lizenz entspricht, wählen Sie "Help/Product information" aus dem Menü des Control Panels.







# Stichwortverzeichnis

---

## —#—

#include 168  
#usage 169

---

## —A—

Absolutwert-, Maximum- und Minimum-Funktion 279  
ADD 42  
ARC 45  
Arithmetische Operatoren 242  
Arithmetischer Ausdruck 245  
ASSIGN 46  
Ausdrücke 244  
Ausgabedaten 153  
Ausgabedaten erzeugen 134  
Ausgabetreiber (Output Device) 142  
AUTO 48  
Automatischer Backup 358  
Autorouter 157

---

## —B—

Backup 28  
Bauelement in Bibliothek definieren 22  
Bedingter Ausdruck 249  
Bedruckbarer Bereich 150  
Befehlseingabe 40  
Benachbarte Objekte selektieren 31  
Bibliotheks-Editier-Modus 34  
Bibliotheks-Editor 33  
Bitweise Operatoren 238  
Blenden-Emulation 145  
Blenden-Konfigurationsdatei 144  
Blenden-Toleranzen 146  
BOARD 49  
board() 311  
Bohrer-Konfigurationsdatei 147  
Bohrer-Toleranzen 148  
break 255  
Builtin Variablen 265  
Builtin-Constants 264  
Builtin-Functions 266  
Builtins 263  
Builtin-Statements 310  
BUS 50

---

## —C—

CAM-Prozessor 139  
CAM-Prozessor-Hauptmenü 140  
CAM-Prozessor-Job 141  
CHANGE 51  
char 187  
Character-Funktionen 268  
Character-Konstanten 173  
CIRCLE 52

CLASS 53  
CLOSE 54  
Compound-Statement (Verbundanweisung) 252  
CONNECT 55  
continue 256  
Control Panel 23  
Control Panel und Editor-Fenster 16  
Control-Statements (Steueranweisungen) 254  
COPY 56  
CUT 57

---

## —D—

Datentypen 186  
Definitionen 233  
DELETE 58  
DESCRIPTION 60  
Design Rules 159  
device() 312  
Device-Parameter 143  
Dialoge 319  
Dialog-Funktionen 349  
Dialog-Objekte 324  
Direktiven 167  
DISPLAY 61  
dlgAccept() 350  
dlgCell 325  
dlgCheckBox 326  
dlgComboBox 327  
dlgDialog 328  
dlgDirectory() 321  
dlgFileOpen(), dlgFileSave() 322  
dlgGridLayout 329  
dlgGroup 330  
dlgHBoxLayout 331  
dlgIntEdit 332  
dlgLabel 333  
dlgListBox 334  
dlgListView 335  
dlgMessageBox() 323  
dlgPushButton 336  
dlgRadioButton 337  
dlgRealEdit 338  
dlgRedisplay() 351  
dlgReject() 353  
dlgReset() 352  
dlgSpacing 339  
dlgSpinBox 340  
dlgStretch 341  
dlgStringEdit 342  
dlgTabPage 343  
dlgTabWidget 344  
dlgTextEdit 345  
dlgTextView 346  
dlgVBoxLayout 347  
do...while 257  
Doppelpunkt 184  
DRC 63  
Drucken 135  
Drucken einer Zeichnung 136  
Drucken eines Textes 137

---

**—E—**

EAGLE konfigurieren 13  
EAGLE-Editionen 366  
Eckige Klammern 179  
EDIT 64  
Editor-Befehle 38  
Editor-Fenster 32  
Eigenen Device-Treiber definieren 152  
Ein vollständiges Beispiel 355  
Eingabe von Parametern und Werten 17  
Einheiten-Konvertierung 288  
Einschränkungen 362  
ERC 65  
Erklärung der Help-Funktion 12  
ERRORS 67  
Escape-Sequenzen 177  
Escape-Zeichen 354  
Evaluation-Operatoren 241  
exit() 284  
Exponential-Funktionen 282  
EXPORT 68  
Expression-Statement (Ausdrucksanweisung) 253

---

**—F—**

Filedata-Funktionen 275  
fileerror() 272  
fileglob() 273  
File-Handling-Funktionen 271  
File-Input-Funktionen 276  
Filename-Funktionen 274  
fileread() 277  
Flag-Options 154  
for 258  
Forward&Back-Annotation 359  
Funktionsaufruf 250  
Funktions-Definitionen 236

---

**—G—**

GATESWAP 70  
Geschweifte Klammern 181  
Gleichheitszeichen 185  
GRID 71  
GROUP 73

---

**—H—**

HELP 74  
HOLE 75

---

**—I—**

Identifier 171  
if...else 259  
INFO 76  
int 188  
Integer-Konstanten 174  
INVOKE 77  
is...() 269

---

**—J—**

JUNCTION 78

---

**—K—**

Komma 182  
Komma-Ausdruck 248  
Kommentare 166  
Konsistenzprüfung 360  
Konstanten 172  
Konstanten-Definitionen 234  
Kontext Menüs 25  
Konturdaten 156

---

**—L—**

LABEL 79  
LAYER 80  
Layer und Farben 155  
Layout überprüfen 21  
Layout-Editor 35  
Layout-Information 348  
library() 313  
Lizenz 364  
Logische Operatoren 239  
lookup() 285

---

**—M—**

MARK 82  
Mathematische Funktionen 278  
MENU 83  
MIRROR 85  
MOVE 86

---

**—N—**

NAME 87  
NET 88

---

**—O—**

Objekt-Typen 193  
Offset 149  
OPEN 89  
Operatoren 237  
OPTIMIZE 90  
output() 314

---

**—P—**

PACKAGE 91  
package() 315  
PAD 92  
PASTE 94  
PIN 95  
PINSWAP 98  
Platine aus Schaltplan erzeugen 20

POLYGON 99  
PREFIX 101  
PRINT 102  
printf() 290  
Print-Funktionen 289  
Produkt-Registrierung 365  
Punctuator-Zeichen 178

---

## —Q—

QUIT 103

---

## —R—

RATSNEST 104  
real 189  
Real-Konstanten 175  
RECT 105  
REDO 106  
REMOVE 107  
RENAME 108  
REPLACE 109  
return 260  
Rich Text 356  
RIPUP 110  
ROTATE 111  
ROUTE 112  
RUN 113  
Runde Klammern 180  
Rundungs-Funktionen 280

---

## —S—

Schaltplan entwerfen 18  
Schaltplan-Editor 36  
Schaltung überprüfen 19  
schematic() 316  
Schlüsselwörter (Keywords) 170  
Schnelle Einführung 15  
Schreiben eines ULP 162  
SCRIPT 114  
Seiteneinrichtung 138  
Semicolon 183  
SET 115  
sheet() 317  
SHOW 117  
SIGNAL 118  
SMASH 119  
SMD 120  
Sonstige Funktionen 283  
sort() 287  
SPLIT 121  
sprintf() 293  
Statements 251  
Stiftdaten 151  
strchr() 295  
string 190  
String-Ausdruck 247  
String-Funktionen 294  
String-Konstanten 176  
String-Operatoren 243  
strjoin() 296  
strlen() 297  
strlwr() 298

strrchr() 299  
strrstr() 300  
strsplit() 301  
strstr() 302  
strsub() 303  
strtod() 304  
strtol() 305  
strupr() 306  
switch 261  
symbol() 318  
Syntax 164

---

## —T—

Tastatur und Maus 30  
Technische Unterstützung 363  
TECHNOLOGY 122  
TEXT 123  
Text-Editor 37  
time() 308  
to...() 270  
Trigonometrische Funktionen 281  
Typecast 192  
Typ-Umwandlung 191

---

## —Ü—

Überprüfen des Designs 158  
UL\_ARC 196  
UL\_AREA 197  
UL\_BOARD 198  
UL\_BUS 199  
UL\_CIRCLE 200  
UL\_CLASS 201  
UL\_CONTACT 202  
UL\_CONTACTREF 203  
UL\_DEVICE 204  
UL\_DEVICESET 205  
UL\_ELEMENT 206  
UL\_GATE 207  
UL\_GRID 208  
UL\_HOLE 209  
UL\_INSTANCE 210  
UL\_JUNCTION 211  
UL\_LAYER 212  
UL\_LIBRARY 213  
UL\_NET 214  
UL\_PACKAGE 215  
UL\_PAD 216  
UL\_PART 217  
UL\_PIN 218  
UL\_PINREF 220  
UL\_POLYGON 221  
UL\_RECTANGLE 223  
UL\_SCHEMATIC 224  
UL\_SEGMENT 225  
UL\_SHEET 226  
UL\_SIGNAL 227  
UL\_SMD 228  
UL\_SYMBOL 229  
UL\_TEXT 230  
UL\_VIA 231  
UL\_WIRE 232  
ULP ausführen 163  
UNDO 125  
UPDATE 126

USE 127  
User Interface 29  
User Language 161

---

## —V—

VALUE 128  
Variablen-Definitionen 235  
Vergleichs-Operatoren 240  
Verzeichnisse 27  
VIA 129  
Vordefinierte Dialoge 320

---

## —W—

while 262  
Whitespace 165  
WINDOW 130  
WIRE 131  
WRITE 133

---

## —Z—

Zeit-Funktionen 307  
Zeit-Konvertierungen 309  
Zuweisungs-Ausdruck 246