# Table of Contents

**About This Manual** .................................................. 38  
**Concepts** ............................................................. 40  
**Group optim.cplex.callable** ....................................... 44  
  - CPXNETaddarcs ..................................................... 52  
  - CPXNETaddnodes ................................................... 54  
  - CPXNETbasewrite ................................................... 55  
  - CPXNETcheckcopynet ............................................. 56  
  - CPXNETchgarcname ................................................ 57  
  - CPXNETchgarcnames .............................................. 58  
  - CPXNETchgbds .................................................... 60  
  - CPXNETchgnodename .............................................. 64  
  - CPXNETchgobj ..................................................... 65  
  - CPXNETchgobjs ................................................... 66  
  - CPXNETchgsupply .................................................. 67  
  - CPXNETcopybase ................................................... 68  
  - CPXNETcopynet .................................................... 70  
  - CPXNETcreateprob ................................................ 73  
  - CPXNETdelarcs .................................................... 74  
  - CPXNETdelnodes ................................................... 75  
  - CPXNETdelset ..................................................... 76  
  - CPXNETextract .................................................... 77  
  - CPXNETfreetprob .................................................. 79  
  - CPXNETgetarcindex ............................................... 80  
  - CPXNETgetarcname ............................................... 81  
  - CPXNETgetarcnodes ............................................... 83  
  - CPXNETgetbase .................................................... 85  
  - CPXNETgetdj ....................................................... 87  
  - CPXNETgetitcnt ................................................... 89  
  - CPXNETgetlb ....................................................... 90  
  - CPXNETgetnodearcs .............................................. 91
**Table of Contents**

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXNETgetnodeindex</td>
<td>93</td>
</tr>
<tr>
<td>CPXNETgetnodename</td>
<td>94</td>
</tr>
<tr>
<td>CPXNETgetnumarcs</td>
<td>96</td>
</tr>
<tr>
<td>CPXNETgetnumnodes</td>
<td>97</td>
</tr>
<tr>
<td>CPXNETgetobj</td>
<td>98</td>
</tr>
<tr>
<td>CPXNETgetobjsen</td>
<td>99</td>
</tr>
<tr>
<td>CPXNETgetobjval</td>
<td>100</td>
</tr>
<tr>
<td>CPXNETgetphase1cnt</td>
<td>101</td>
</tr>
<tr>
<td>CPXNETgetpi</td>
<td>102</td>
</tr>
<tr>
<td>CPXNETgetprobyname</td>
<td>103</td>
</tr>
<tr>
<td>CPXNETgetslack</td>
<td>104</td>
</tr>
<tr>
<td>CPXNETgetstat</td>
<td>105</td>
</tr>
<tr>
<td>CPXNETgetsupply</td>
<td>106</td>
</tr>
<tr>
<td>CPXNETgetub</td>
<td>107</td>
</tr>
<tr>
<td>CPXNETgetx</td>
<td>108</td>
</tr>
<tr>
<td>CPXNETprimopt</td>
<td>109</td>
</tr>
<tr>
<td>CPXNETreadcopybase</td>
<td>110</td>
</tr>
<tr>
<td>CPXNETreadcopyprob</td>
<td>111</td>
</tr>
<tr>
<td>CPXNETsolninfo</td>
<td>112</td>
</tr>
<tr>
<td>CPXNETsolution</td>
<td>114</td>
</tr>
<tr>
<td>CPXNETwriteprob</td>
<td>116</td>
</tr>
<tr>
<td>CPXaddchannel</td>
<td>118</td>
</tr>
<tr>
<td>CPXaddcols</td>
<td>119</td>
</tr>
<tr>
<td>CPXadddpfdest</td>
<td>122</td>
</tr>
<tr>
<td>CPXaddfuncdest</td>
<td>123</td>
</tr>
<tr>
<td>CPXaddlazyconstraints</td>
<td>125</td>
</tr>
<tr>
<td>CPXaddqconstr</td>
<td>127</td>
</tr>
<tr>
<td>CPXaddrows</td>
<td>129</td>
</tr>
<tr>
<td>CPXaddso</td>
<td>132</td>
</tr>
<tr>
<td>CPXaddusercuts</td>
<td>134</td>
</tr>
<tr>
<td>CPXbaropt</td>
<td>137</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>CPXboundsa</td>
<td>138</td>
</tr>
<tr>
<td>CPXbranchcallbackbranchbds</td>
<td>140</td>
</tr>
<tr>
<td>CPXbranchcallbackbranchconstraints</td>
<td>142</td>
</tr>
<tr>
<td>CPXbranchcallbackbranchgeneral</td>
<td>145</td>
</tr>
<tr>
<td>CPXcheckaddcols</td>
<td>148</td>
</tr>
<tr>
<td>CPXcheckaddrows</td>
<td>149</td>
</tr>
<tr>
<td>CPXcheckchgcoeflist</td>
<td>150</td>
</tr>
<tr>
<td>CPXcheckcopyctype</td>
<td>152</td>
</tr>
<tr>
<td>CPXcheckcopylp</td>
<td>153</td>
</tr>
<tr>
<td>CPXcheckcopylpownames</td>
<td>154</td>
</tr>
<tr>
<td>CPXcheckcopyqypep</td>
<td>156</td>
</tr>
<tr>
<td>CPXcheckcopyquad</td>
<td>157</td>
</tr>
<tr>
<td>CPXcheckcopy sos</td>
<td>158</td>
</tr>
<tr>
<td>CPXcheckvals</td>
<td>159</td>
</tr>
<tr>
<td>CPXchgbds</td>
<td>161</td>
</tr>
<tr>
<td>CPXchgcoef</td>
<td>163</td>
</tr>
<tr>
<td>CPXchgcoeflist</td>
<td>165</td>
</tr>
<tr>
<td>CPXchgcolname</td>
<td>167</td>
</tr>
<tr>
<td>CPXchgctype</td>
<td>168</td>
</tr>
<tr>
<td>CPXchgname</td>
<td>170</td>
</tr>
<tr>
<td>CPXchgobj</td>
<td>172</td>
</tr>
<tr>
<td>CPXchgojbsen</td>
<td>173</td>
</tr>
<tr>
<td>CPXchgpobrname</td>
<td>174</td>
</tr>
<tr>
<td>CPXchgprobtype</td>
<td>175</td>
</tr>
<tr>
<td>CPXchgpqcof</td>
<td>177</td>
</tr>
<tr>
<td>CPXchgrhs</td>
<td>178</td>
</tr>
<tr>
<td>CPXchgrngval</td>
<td>179</td>
</tr>
<tr>
<td>CPXchgrrownname</td>
<td>180</td>
</tr>
<tr>
<td>CPXchgsense</td>
<td>181</td>
</tr>
<tr>
<td>CPXcloseprob</td>
<td>183</td>
</tr>
<tr>
<td>CPXcloseCPLEX</td>
<td>184</td>
</tr>
</tbody>
</table>
TABLE OF CONTENTS

CPXcompletingp ......................................................... 185
CPXcopybase ............................................................ 186
CPXcopyctype ............................................................ 188
CPXcopylp ............................................................... 190
CPXcopylpwnames ...................................................... 194
CPXcopymipstart ........................................................ 198
CPXcopynettolp ........................................................ 200
CPXcopyobjname ........................................................ 201
CPXcopyorder ............................................................ 202
CPXcopypartialbase ..................................................... 204
CPXcopyqsep .............................................................. 207
CPXcopyquad ............................................................. 208
CPXcopyqsos .............................................................. 210
CPXcopystart ............................................................. 212
CPXcreateprob ........................................................... 215
CPXcutcallbackadd ...................................................... 217
CPXcutcallbackaddlocal ................................................ 219
CPXdelchannel ............................................................ 221
CPXdelcols ................................................................. 222
CPXdelfpdest ............................................................. 223
CPXdefuncdest ........................................................... 224
CPXdelnames .............................................................. 226
CPXdelqconstrs .......................................................... 227
CPXdelrows ................................................................. 228
CPXdelsetcols ............................................................ 229
CPXdelsetrows ........................................................... 230
CPXdelsetsos .............................................................. 231
CPXdisconnectchannel ................................................... 233
CPXdisplayiis .............................................................. 234
CPXdperwrite ............................................................. 236
CPXdualopt ............................................................... 237
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXdualwrite</td>
<td>238</td>
</tr>
<tr>
<td>CPXembwrite</td>
<td>240</td>
</tr>
<tr>
<td>CPXfclose</td>
<td>241</td>
</tr>
<tr>
<td>CPXfeasopt</td>
<td>242</td>
</tr>
<tr>
<td>CPXfindlis</td>
<td>244</td>
</tr>
<tr>
<td>CPXflushchannel</td>
<td>245</td>
</tr>
<tr>
<td>CPXflushstdchannels</td>
<td>246</td>
</tr>
<tr>
<td>CPXopen</td>
<td>247</td>
</tr>
<tr>
<td>CPXputs</td>
<td>248</td>
</tr>
<tr>
<td>CPXfreelazyconstraints</td>
<td>249</td>
</tr>
<tr>
<td>CPXfreeprob</td>
<td>250</td>
</tr>
<tr>
<td>CPXfreeusercuts</td>
<td>251</td>
</tr>
<tr>
<td>CPXgetax</td>
<td>252</td>
</tr>
<tr>
<td>CPXgetbaritcnt</td>
<td>253</td>
</tr>
<tr>
<td>CPXgetbase</td>
<td>254</td>
</tr>
<tr>
<td>CPXgetbestobjval</td>
<td>256</td>
</tr>
<tr>
<td>CPXgetbranchcallbackfunc</td>
<td>257</td>
</tr>
<tr>
<td>CPXgetcallbacktype</td>
<td>261</td>
</tr>
<tr>
<td>CPXgetcallbackgloballb</td>
<td>263</td>
</tr>
<tr>
<td>CPXgetcallbackglobalub</td>
<td>265</td>
</tr>
<tr>
<td>CPXgetcallbackincumbent</td>
<td>267</td>
</tr>
<tr>
<td>CPXgetcallbackinfo</td>
<td>269</td>
</tr>
<tr>
<td>CPXgetcallbacklp</td>
<td>275</td>
</tr>
<tr>
<td>CPXgetcallbacknodeinfo</td>
<td>277</td>
</tr>
<tr>
<td>CPXgetcallbacknodeintfeas</td>
<td>281</td>
</tr>
<tr>
<td>CPXgetcallbacknodelb</td>
<td>283</td>
</tr>
<tr>
<td>CPXgetcallbacknodeip</td>
<td>285</td>
</tr>
<tr>
<td>CPXgetcallbacknodeoobjval</td>
<td>287</td>
</tr>
<tr>
<td>CPXgetcallbacknodestat</td>
<td>289</td>
</tr>
<tr>
<td>CPXgetcallbacknodeub</td>
<td>291</td>
</tr>
<tr>
<td>CPXgetcallbacknodeex</td>
<td>293</td>
</tr>
</tbody>
</table>
**Table of Contents**

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXgetcallbackorder</td>
<td>295</td>
</tr>
<tr>
<td>CPXgetcallbackpseudocosts</td>
<td>297</td>
</tr>
<tr>
<td>CPXgetcallbackseqinfo</td>
<td>299</td>
</tr>
<tr>
<td>CPXgetcallbackssosinfo</td>
<td>301</td>
</tr>
<tr>
<td>CPXgetchannels</td>
<td>304</td>
</tr>
<tr>
<td>CPXgetclqcnt</td>
<td>305</td>
</tr>
<tr>
<td>CPXgetcoef</td>
<td>306</td>
</tr>
<tr>
<td>CPXgetcolindex</td>
<td>307</td>
</tr>
<tr>
<td>CPXgetcolname</td>
<td>308</td>
</tr>
<tr>
<td>CPXgetcols</td>
<td>310</td>
</tr>
<tr>
<td>CPXgetcovcnt</td>
<td>312</td>
</tr>
<tr>
<td>CPXgetcrossdexchcnt</td>
<td>313</td>
</tr>
<tr>
<td>CPXgetcrossdpushcnt</td>
<td>314</td>
</tr>
<tr>
<td>CPXgetcrosspexchcnt</td>
<td>315</td>
</tr>
<tr>
<td>CPXgetcrossppushcnt</td>
<td>316</td>
</tr>
<tr>
<td>CPXgetctype</td>
<td>317</td>
</tr>
<tr>
<td>CPXgetcutcallbackfunc</td>
<td>318</td>
</tr>
<tr>
<td>CPXgetcutoff</td>
<td>319</td>
</tr>
<tr>
<td>CPXgetdbparam</td>
<td>320</td>
</tr>
<tr>
<td>CPXgetdbquality</td>
<td>321</td>
</tr>
<tr>
<td>CPXgetdeletenodecallbackfunc</td>
<td>324</td>
</tr>
<tr>
<td>CPXgetdj</td>
<td>326</td>
</tr>
<tr>
<td>CPXgetdsbcnt</td>
<td>327</td>
</tr>
<tr>
<td>CPXgeterrorstring</td>
<td>328</td>
</tr>
<tr>
<td>CPXgetgencqcnt</td>
<td>329</td>
</tr>
<tr>
<td>CPXgetgrad</td>
<td>330</td>
</tr>
<tr>
<td>CPXgetheuristiccallbackfunc</td>
<td>332</td>
</tr>
<tr>
<td>CPXgetis</td>
<td>334</td>
</tr>
<tr>
<td>CPXgetincumbentcallbackfunc</td>
<td>336</td>
</tr>
<tr>
<td>CPXgetintparam</td>
<td>338</td>
</tr>
<tr>
<td>CPXgetintquality</td>
<td>339</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------</td>
</tr>
<tr>
<td>CPXgetitcnt</td>
<td>341</td>
</tr>
<tr>
<td>CPXgetlb</td>
<td>342</td>
</tr>
<tr>
<td>CPXgetlogfile</td>
<td>343</td>
</tr>
<tr>
<td>CPXgetlpcallbackfunc</td>
<td>344</td>
</tr>
<tr>
<td>CPXgetmethod</td>
<td>346</td>
</tr>
<tr>
<td>CPXgetmipcallbackfunc</td>
<td>347</td>
</tr>
<tr>
<td>CPXgetmipitcnt</td>
<td>349</td>
</tr>
<tr>
<td>CPXgetmipobjval</td>
<td>350</td>
</tr>
<tr>
<td>CPXgetmipqconstrslack</td>
<td>351</td>
</tr>
<tr>
<td>CPXgetmipslack</td>
<td>353</td>
</tr>
<tr>
<td>CPXgetmipstart</td>
<td>355</td>
</tr>
<tr>
<td>CPXgetmipx</td>
<td>357</td>
</tr>
<tr>
<td>CPXgetnetcallbackfunc</td>
<td>359</td>
</tr>
<tr>
<td>CPXgetnodecallbackfunc</td>
<td>361</td>
</tr>
<tr>
<td>CPXgetnodecnt</td>
<td>363</td>
</tr>
<tr>
<td>CPXgetnodeint</td>
<td>364</td>
</tr>
<tr>
<td>CPXgetnodeleftcnt</td>
<td>365</td>
</tr>
<tr>
<td>CPXgetnumbin</td>
<td>366</td>
</tr>
<tr>
<td>CPXgetnumcols</td>
<td>367</td>
</tr>
<tr>
<td>CPXgetnumint</td>
<td>368</td>
</tr>
<tr>
<td>CPXgetnumnz</td>
<td>369</td>
</tr>
<tr>
<td>CPXgetnumqconstrs</td>
<td>370</td>
</tr>
<tr>
<td>CPXgetnumqpnz</td>
<td>371</td>
</tr>
<tr>
<td>CPXgetnumquad</td>
<td>372</td>
</tr>
<tr>
<td>CPXgetnumrows</td>
<td>373</td>
</tr>
<tr>
<td>CPXgetnumsemincont</td>
<td>374</td>
</tr>
<tr>
<td>CPXgetnumsemiint</td>
<td>375</td>
</tr>
<tr>
<td>CPXgetnumsos</td>
<td>376</td>
</tr>
<tr>
<td>CPXgetobj</td>
<td>377</td>
</tr>
<tr>
<td>CPXgetobjname</td>
<td>378</td>
</tr>
<tr>
<td>CPXgetobjsen</td>
<td>380</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------</td>
</tr>
<tr>
<td>CPXgetobjval</td>
<td>381</td>
</tr>
<tr>
<td>CPXgetorder</td>
<td>382</td>
</tr>
<tr>
<td>CPXgetparamname</td>
<td>384</td>
</tr>
<tr>
<td>CPXgetparamnum</td>
<td>385</td>
</tr>
<tr>
<td>CPXgetphase1cnt</td>
<td>386</td>
</tr>
<tr>
<td>CPXgetpi</td>
<td>387</td>
</tr>
<tr>
<td>CPXgetprobname</td>
<td>388</td>
</tr>
<tr>
<td>CPXgetprobtype</td>
<td>390</td>
</tr>
<tr>
<td>CPXgetpsbcnt</td>
<td>392</td>
</tr>
<tr>
<td>CPXgetqconstr</td>
<td>393</td>
</tr>
<tr>
<td>CPXgetqconstrindex</td>
<td>396</td>
</tr>
<tr>
<td>CPXgetqconstrname</td>
<td>397</td>
</tr>
<tr>
<td>CPXgetqconstrslack</td>
<td>399</td>
</tr>
<tr>
<td>CPXgetqpcoeff</td>
<td>400</td>
</tr>
<tr>
<td>CPXgetquad</td>
<td>401</td>
</tr>
<tr>
<td>CPXgetrhs</td>
<td>403</td>
</tr>
<tr>
<td>CPXgetrngval</td>
<td>404</td>
</tr>
<tr>
<td>CPXgetrowindex</td>
<td>405</td>
</tr>
<tr>
<td>CPXgetrowname</td>
<td>406</td>
</tr>
<tr>
<td>CPXgetrows</td>
<td>408</td>
</tr>
<tr>
<td>CPXgetsense</td>
<td>410</td>
</tr>
<tr>
<td>CPXgetsifticnt</td>
<td>412</td>
</tr>
<tr>
<td>CPXgetsiftphase1cnt</td>
<td>413</td>
</tr>
<tr>
<td>CPXgetslack</td>
<td>414</td>
</tr>
<tr>
<td>CPXgetsolvecallbackfunc</td>
<td>416</td>
</tr>
<tr>
<td>CPXgetssos</td>
<td>418</td>
</tr>
<tr>
<td>CPXgetstat</td>
<td>421</td>
</tr>
<tr>
<td>CPXgetstatstring</td>
<td>422</td>
</tr>
<tr>
<td>CPXgetstrparam</td>
<td>423</td>
</tr>
<tr>
<td>CPXgetsubmethod</td>
<td>424</td>
</tr>
<tr>
<td>CPXgetsubstat</td>
<td>425</td>
</tr>
</tbody>
</table>
Table of Contents

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXgetub</td>
<td>426</td>
</tr>
<tr>
<td>CPXgetx</td>
<td>427</td>
</tr>
<tr>
<td>CPXgetxqax</td>
<td>429</td>
</tr>
<tr>
<td>CPXhybbaropt</td>
<td>430</td>
</tr>
<tr>
<td>CPXhybnetopt</td>
<td>432</td>
</tr>
<tr>
<td>CPXiswrite</td>
<td>434</td>
</tr>
<tr>
<td>CPXinfodbparam</td>
<td>435</td>
</tr>
<tr>
<td>CPXinfointparam</td>
<td>437</td>
</tr>
<tr>
<td>CPXinfostrparam</td>
<td>438</td>
</tr>
<tr>
<td>CPXlpopt</td>
<td>439</td>
</tr>
<tr>
<td>CPXmbasewrite</td>
<td>440</td>
</tr>
<tr>
<td>CPXmipopt</td>
<td>441</td>
</tr>
<tr>
<td>CPXmsg</td>
<td>442</td>
</tr>
<tr>
<td>CPXmsgstr</td>
<td>444</td>
</tr>
<tr>
<td>CPXmstwrite</td>
<td>445</td>
</tr>
<tr>
<td>CPXnewcols</td>
<td>446</td>
</tr>
<tr>
<td>CPXnewrows</td>
<td>448</td>
</tr>
<tr>
<td>CPXobjsa</td>
<td>450</td>
</tr>
<tr>
<td>CPXopenCPLEX</td>
<td>452</td>
</tr>
<tr>
<td>CPXordwrite</td>
<td>453</td>
</tr>
<tr>
<td>CPXpperwrite</td>
<td>454</td>
</tr>
<tr>
<td>CPXpreslvwrite</td>
<td>455</td>
</tr>
<tr>
<td>CPXprimopt</td>
<td>457</td>
</tr>
<tr>
<td>CPXputenv</td>
<td>458</td>
</tr>
<tr>
<td>CPXpindefcertificate</td>
<td>459</td>
</tr>
<tr>
<td>CPXqpopt</td>
<td>460</td>
</tr>
<tr>
<td>CPXqpwrite</td>
<td>461</td>
</tr>
<tr>
<td>CPXreadcopybase</td>
<td>462</td>
</tr>
<tr>
<td>CPXreadcopymipstart</td>
<td>463</td>
</tr>
<tr>
<td>CPXreadcopyorder</td>
<td>464</td>
</tr>
<tr>
<td>CPXreadcopyparam</td>
<td>465</td>
</tr>
</tbody>
</table>
# Table of Contents

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXreadcopyprob</td>
<td>466</td>
</tr>
<tr>
<td>CPXreadcopyqp</td>
<td>468</td>
</tr>
<tr>
<td>CPXreadcopyqsos</td>
<td>469</td>
</tr>
<tr>
<td>CPXreadcopytree</td>
<td>470</td>
</tr>
<tr>
<td>CPXreadcopyvec</td>
<td>471</td>
</tr>
<tr>
<td>CPXrhssa</td>
<td>472</td>
</tr>
<tr>
<td>CPXsetbranchcallbackfunc</td>
<td>474</td>
</tr>
<tr>
<td>CPXsetbranchnosolncallbackfunc</td>
<td>478</td>
</tr>
<tr>
<td>CPXsetcutcallbackfunc</td>
<td>479</td>
</tr>
<tr>
<td>CPXsetdblparam</td>
<td>482</td>
</tr>
<tr>
<td>CPXsetdefaults</td>
<td>483</td>
</tr>
<tr>
<td>CPXsetdeletenodecallbackfunc</td>
<td>484</td>
</tr>
<tr>
<td>CPXsetheuristiccallbackfunc</td>
<td>487</td>
</tr>
<tr>
<td>CPXsetincumbentcallbackfunc</td>
<td>490</td>
</tr>
<tr>
<td>CPXsetintparam</td>
<td>493</td>
</tr>
<tr>
<td>CPXsetlogfile</td>
<td>494</td>
</tr>
<tr>
<td>CPXsetlpcallbackfunc</td>
<td>495</td>
</tr>
<tr>
<td>CPXsetmipcallbackfunc</td>
<td>497</td>
</tr>
<tr>
<td>CPXsetnetcallbackfunc</td>
<td>499</td>
</tr>
<tr>
<td>CPXsetnodecallbackfunc</td>
<td>501</td>
</tr>
<tr>
<td>CPXsetsolvecallbackfunc</td>
<td>504</td>
</tr>
<tr>
<td>CPXsetstrparam</td>
<td>507</td>
</tr>
<tr>
<td>CPXsolninfo</td>
<td>508</td>
</tr>
<tr>
<td>CPXsolution</td>
<td>510</td>
</tr>
<tr>
<td>CPXsoswrite</td>
<td>513</td>
</tr>
<tr>
<td>CPXstrcpy</td>
<td>514</td>
</tr>
<tr>
<td>CPXstrlen</td>
<td>515</td>
</tr>
<tr>
<td>CPXtreewrite</td>
<td>516</td>
</tr>
<tr>
<td>CPXvecwrite</td>
<td>517</td>
</tr>
<tr>
<td>CPXversion</td>
<td>518</td>
</tr>
<tr>
<td>CPXwriteparam</td>
<td>519</td>
</tr>
</tbody>
</table>
# Table of Contents

- CPXwriteprob .................................................. 520
- CPXwritesol ...................................................... 522
- **Group optim.cplex.callable.createdeletecopy** .................................................... 523
  - CPXNETaddarcs .................................................. 524
  - CPXNETaddnodes .................................................. 526
  - CPXNETcheckcopynet ........................................... 527
  - CPXNETcopybase .................................................. 528
  - CPXNETcopynet ..................................................... 530
  - CPXNETcreateprob ................................................ 533
  - CPXNETextract .................................................... 534
  - CPXNETfreeprob .................................................... 536
  - CPXcloneprob ...................................................... 537
  - CPXcompletelp ..................................................... 538
  - CPXcopybase ....................................................... 539
  - CPXcopyctype ....................................................... 541
  - CPXcopylp .......................................................... 543
  - CPXcopylpwnames .................................................. 547
  - CPXcopymipstart ................................................... 551
  - CPXcopynettolp .................................................... 553
  - CPXcopyobjname .................................................... 554
  - CPXcopyorder ....................................................... 555
  - CPXcopyqpssep ...................................................... 557
  - CPXcopyquad ......................................................... 558
  - CPXcopyystart ....................................................... 560
  - CPXcreateprob ...................................................... 563
  - CPXfreeprob ......................................................... 565
  - CPXreadcopyvos ..................................................... 566
- **Group optim.cplex.callable.optimizers** ............................................................. 566
  - CPXNETprimopt ..................................................... 568
  - CPXbaropt .......................................................... 569
  - CPXdualopt ........................................................ 570
# Table of Contents

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXhybbaropt</td>
<td>571</td>
</tr>
<tr>
<td>CPXhybnetopt</td>
<td>573</td>
</tr>
<tr>
<td>CPXlpopt</td>
<td>575</td>
</tr>
<tr>
<td>CPXmipopt</td>
<td>576</td>
</tr>
<tr>
<td>CPXprimopt</td>
<td>577</td>
</tr>
<tr>
<td>CPXqpopt</td>
<td>578</td>
</tr>
<tr>
<td><strong>Group optim.cplex.callable.accesslporqpresults</strong></td>
<td>578</td>
</tr>
<tr>
<td>CPXgetax</td>
<td>580</td>
</tr>
<tr>
<td>CPXgetbaritcnt</td>
<td>581</td>
</tr>
<tr>
<td>CPXgetbase</td>
<td>582</td>
</tr>
<tr>
<td>CPXgetcrossdexchcnt</td>
<td>584</td>
</tr>
<tr>
<td>CPXgetcrossdpushcnt</td>
<td>585</td>
</tr>
<tr>
<td>CPXgetcrossppushcnt</td>
<td>586</td>
</tr>
<tr>
<td>CPXgetdj</td>
<td>587</td>
</tr>
<tr>
<td>CPXgetdsbcnt</td>
<td>588</td>
</tr>
<tr>
<td>CPXgetitcnt</td>
<td>589</td>
</tr>
<tr>
<td>CPXgetmethod</td>
<td>590</td>
</tr>
<tr>
<td>CPXgetobjval</td>
<td>591</td>
</tr>
<tr>
<td>CPXgetphase1cnt</td>
<td>592</td>
</tr>
<tr>
<td>CPXgetpi</td>
<td>593</td>
</tr>
<tr>
<td>CPXgetpsbcnt</td>
<td>594</td>
</tr>
<tr>
<td>CPXgetsiftitcnt</td>
<td>595</td>
</tr>
<tr>
<td>CPXgetsiftphase1cnt</td>
<td>596</td>
</tr>
<tr>
<td>CPXgetsllack</td>
<td>597</td>
</tr>
<tr>
<td>CPXgetstat</td>
<td>599</td>
</tr>
<tr>
<td>CPXgetstatstring</td>
<td>600</td>
</tr>
<tr>
<td>CPXgetx</td>
<td>601</td>
</tr>
<tr>
<td>CPXsolninfo</td>
<td>603</td>
</tr>
<tr>
<td>CPXsolution</td>
<td>605</td>
</tr>
<tr>
<td><strong>Group optim.cplex.callable.accessmipresults</strong></td>
<td>607</td>
</tr>
<tr>
<td>CPXgetbestobjval</td>
<td>608</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>CPXgetcqcqcnt</td>
<td>609</td>
</tr>
<tr>
<td>CPXgetcvcnt</td>
<td>610</td>
</tr>
<tr>
<td>CPXgetcutoff</td>
<td>611</td>
</tr>
<tr>
<td>CPXgetgencqcnt</td>
<td>612</td>
</tr>
<tr>
<td>CPXgetmipcnt</td>
<td>613</td>
</tr>
<tr>
<td>CPXgetmipobjval</td>
<td>614</td>
</tr>
<tr>
<td>CPXgetmipqconstrslack</td>
<td>615</td>
</tr>
<tr>
<td>CPXgetmipslack</td>
<td>617</td>
</tr>
<tr>
<td>CPXgetmipstart</td>
<td>619</td>
</tr>
<tr>
<td>CPXgetmipx</td>
<td>621</td>
</tr>
<tr>
<td>CPXgetnodecnt</td>
<td>623</td>
</tr>
<tr>
<td>CPXgetnodeint</td>
<td>624</td>
</tr>
<tr>
<td>CPXgetnodeleftcnt</td>
<td>625</td>
</tr>
<tr>
<td>CPXgetsubmethod</td>
<td>626</td>
</tr>
<tr>
<td>CPXgetsubstat</td>
<td>627</td>
</tr>
<tr>
<td>Group optim.cplex.callable.accessnetworkresults</td>
<td>627</td>
</tr>
<tr>
<td>CPXNETgetbase</td>
<td>629</td>
</tr>
<tr>
<td>CPXNETgetdj</td>
<td>631</td>
</tr>
<tr>
<td>CPXNETgetitcnt</td>
<td>633</td>
</tr>
<tr>
<td>CPXNETgetobjval</td>
<td>634</td>
</tr>
<tr>
<td>CPXNETgetphase1cnt</td>
<td>635</td>
</tr>
<tr>
<td>CPXNETgetpi</td>
<td>636</td>
</tr>
<tr>
<td>CPXNETgetslack</td>
<td>637</td>
</tr>
<tr>
<td>CPXNETgetstat</td>
<td>638</td>
</tr>
<tr>
<td>CPXNETgetx</td>
<td>639</td>
</tr>
<tr>
<td>CPXNETsolninfo</td>
<td>640</td>
</tr>
<tr>
<td>CPXNETsolution</td>
<td>642</td>
</tr>
<tr>
<td>Group optim.cplex.callable.analyzesolution</td>
<td>643</td>
</tr>
<tr>
<td>CPXboundsa</td>
<td>645</td>
</tr>
<tr>
<td>CPXdisplayiis</td>
<td>647</td>
</tr>
<tr>
<td>CPXfeasopt</td>
<td>649</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------</td>
</tr>
<tr>
<td>CPXfindiis</td>
<td>651</td>
</tr>
<tr>
<td>CPXgetdblquality</td>
<td>652</td>
</tr>
<tr>
<td>CPXgetgrad</td>
<td>654</td>
</tr>
<tr>
<td>CPXgetiis</td>
<td>656</td>
</tr>
<tr>
<td>CPXgetintquality</td>
<td>658</td>
</tr>
<tr>
<td>CPXobjsa</td>
<td>660</td>
</tr>
<tr>
<td>CPXqipindefcertificate</td>
<td>662</td>
</tr>
<tr>
<td>CPXrhssa</td>
<td>663</td>
</tr>
<tr>
<td>CPXsolwrite</td>
<td>665</td>
</tr>
<tr>
<td>Group optim.cplex.callable.querygeneralproblem</td>
<td>667</td>
</tr>
<tr>
<td>CPXgetcoef</td>
<td>668</td>
</tr>
<tr>
<td>CPXgetcolindex</td>
<td>669</td>
</tr>
<tr>
<td>CPXgetcolname</td>
<td>670</td>
</tr>
<tr>
<td>CPXgetcols</td>
<td>672</td>
</tr>
<tr>
<td>CPXgetlb</td>
<td>674</td>
</tr>
<tr>
<td>CPXgetnumcols</td>
<td>675</td>
</tr>
<tr>
<td>CPXgetnumnz</td>
<td>676</td>
</tr>
<tr>
<td>CPXgetnumrows</td>
<td>677</td>
</tr>
<tr>
<td>CPXgetobj</td>
<td>678</td>
</tr>
<tr>
<td>CPXgetobjname</td>
<td>679</td>
</tr>
<tr>
<td>CPXgetobjsen</td>
<td>681</td>
</tr>
<tr>
<td>CPXgetprobnname</td>
<td>682</td>
</tr>
<tr>
<td>CPXgetprobytype</td>
<td>684</td>
</tr>
<tr>
<td>CPXgetrhs</td>
<td>686</td>
</tr>
<tr>
<td>CPXgetmgval</td>
<td>687</td>
</tr>
<tr>
<td>CPXgetrowindex</td>
<td>688</td>
</tr>
<tr>
<td>CPXgetrowname</td>
<td>689</td>
</tr>
<tr>
<td>CPXgetrows</td>
<td>691</td>
</tr>
<tr>
<td>CPXgetsense</td>
<td>693</td>
</tr>
<tr>
<td>CPXgetub</td>
<td>695</td>
</tr>
<tr>
<td>Group optim.cplex.callable.querymip</td>
<td>695</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------</td>
</tr>
<tr>
<td>CPXgetctype</td>
<td>697</td>
</tr>
<tr>
<td>CPXgetnumbin</td>
<td>698</td>
</tr>
<tr>
<td>CPXgetnumint</td>
<td>699</td>
</tr>
<tr>
<td>CPXgetnumsemicont</td>
<td>700</td>
</tr>
<tr>
<td>CPXgetnumsemiint</td>
<td>701</td>
</tr>
<tr>
<td>CPXgetnumso</td>
<td>702</td>
</tr>
<tr>
<td>CPXgetorder</td>
<td>703</td>
</tr>
<tr>
<td>CPXgetso</td>
<td>705</td>
</tr>
<tr>
<td>Group optim.cplex.callable.queryqp</td>
<td>707</td>
</tr>
<tr>
<td>CPXgetnumqpnz</td>
<td>708</td>
</tr>
<tr>
<td>CPXgetnumquad</td>
<td>709</td>
</tr>
<tr>
<td>CPXgetapcoef</td>
<td>710</td>
</tr>
<tr>
<td>CPXgetquad</td>
<td>711</td>
</tr>
<tr>
<td>Group optim.cplex.callable.querynetwork</td>
<td>712</td>
</tr>
<tr>
<td>CPXNETgetarcindex</td>
<td>714</td>
</tr>
<tr>
<td>CPXNETgetarcname</td>
<td>715</td>
</tr>
<tr>
<td>CPXNETgetarcnodes</td>
<td>717</td>
</tr>
<tr>
<td>CPXNETgetlb</td>
<td>719</td>
</tr>
<tr>
<td>CPXNETgetnodearcs</td>
<td>720</td>
</tr>
<tr>
<td>CPXNETgetnodeindex</td>
<td>722</td>
</tr>
<tr>
<td>CPXNETgetnodename</td>
<td>723</td>
</tr>
<tr>
<td>CPXNETgetnumarcs</td>
<td>725</td>
</tr>
<tr>
<td>CPXNETgetnumnodes</td>
<td>726</td>
</tr>
<tr>
<td>CPXNETgetobj</td>
<td>727</td>
</tr>
<tr>
<td>CPXNETgetobjsen</td>
<td>728</td>
</tr>
<tr>
<td>CPXNETgetprobnam</td>
<td>729</td>
</tr>
<tr>
<td>CPXNETgetsupply</td>
<td>730</td>
</tr>
<tr>
<td>CPXNETgetub</td>
<td>731</td>
</tr>
<tr>
<td>Group optim.cplex.callable.manageparameters</td>
<td>732</td>
</tr>
<tr>
<td>CPXgetdblparam</td>
<td>733</td>
</tr>
<tr>
<td>CPXgetintparam</td>
<td>734</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>CPXgetparamname</td>
<td>735</td>
</tr>
<tr>
<td>CPXgetparamnum</td>
<td>736</td>
</tr>
<tr>
<td>CPXsetstrparam</td>
<td>737</td>
</tr>
<tr>
<td>CPXinfodblparam</td>
<td>738</td>
</tr>
<tr>
<td>CPXinfointparam</td>
<td>740</td>
</tr>
<tr>
<td>CPXinfostrparam</td>
<td>741</td>
</tr>
<tr>
<td>CPXreadcopyparam</td>
<td>742</td>
</tr>
<tr>
<td>CPXsetdblparam</td>
<td>743</td>
</tr>
<tr>
<td>CPXsetdefaults</td>
<td>744</td>
</tr>
<tr>
<td>CPXsetintparam</td>
<td>745</td>
</tr>
<tr>
<td>CPXsetstrparam</td>
<td>746</td>
</tr>
<tr>
<td>CPXwriteparam</td>
<td>747</td>
</tr>
<tr>
<td>Group optim.cplex.callable.modifyproblem</td>
<td>747</td>
</tr>
<tr>
<td>CPXaddcols</td>
<td>749</td>
</tr>
<tr>
<td>CPXaddrows</td>
<td>752</td>
</tr>
<tr>
<td>CPXaddssos</td>
<td>755</td>
</tr>
<tr>
<td>CPXchgbds</td>
<td>757</td>
</tr>
<tr>
<td>CPXchgcoef</td>
<td>759</td>
</tr>
<tr>
<td>CPXchgcoefflist</td>
<td>761</td>
</tr>
<tr>
<td>CPXchgcolname</td>
<td>763</td>
</tr>
<tr>
<td>CPXchgname</td>
<td>764</td>
</tr>
<tr>
<td>CPXchgobjs</td>
<td>766</td>
</tr>
<tr>
<td>CPXchgprobname</td>
<td>767</td>
</tr>
<tr>
<td>CPXchgprobtype</td>
<td>768</td>
</tr>
<tr>
<td>CPXchgqpcoeff</td>
<td>770</td>
</tr>
<tr>
<td>CPXchgrhs</td>
<td>771</td>
</tr>
<tr>
<td>CPXchgrngval</td>
<td>772</td>
</tr>
<tr>
<td>CPXchgrownname</td>
<td>773</td>
</tr>
<tr>
<td>CPXchgsense</td>
<td>774</td>
</tr>
<tr>
<td>CPXdelcols</td>
<td>776</td>
</tr>
<tr>
<td>CPXdelnames</td>
<td>777</td>
</tr>
</tbody>
</table>
# Table of Contents

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXdelrows</td>
<td>778</td>
</tr>
<tr>
<td>CPXdelsetcols</td>
<td>779</td>
</tr>
<tr>
<td>CPXdelsetrows</td>
<td>780</td>
</tr>
<tr>
<td>CPXdelsetsos</td>
<td>781</td>
</tr>
<tr>
<td>CPXnewcols</td>
<td>783</td>
</tr>
<tr>
<td>CPXnewrows</td>
<td>785</td>
</tr>
<tr>
<td>CPXprechgobj</td>
<td>787</td>
</tr>
<tr>
<td><strong>Group optim.cplex.callable.modifynetwork</strong></td>
<td><strong>788</strong></td>
</tr>
<tr>
<td>CPXNETchgarname</td>
<td>789</td>
</tr>
<tr>
<td>CPXNETchgarcnodes</td>
<td>790</td>
</tr>
<tr>
<td>CPXNETchgbds</td>
<td>792</td>
</tr>
<tr>
<td>CPXNETchgname</td>
<td>794</td>
</tr>
<tr>
<td>CPXNETchgnodename</td>
<td>796</td>
</tr>
<tr>
<td>CPXNETchgobj</td>
<td>797</td>
</tr>
<tr>
<td>CPXNETchgobjsen</td>
<td>798</td>
</tr>
<tr>
<td>CPXNETchgsupply</td>
<td>799</td>
</tr>
<tr>
<td>CPXNETdelarcs</td>
<td>800</td>
</tr>
<tr>
<td>CPXNETdelnodes</td>
<td>801</td>
</tr>
<tr>
<td>CPXNETdelset</td>
<td>802</td>
</tr>
<tr>
<td><strong>Group optim.cplex.callable.readfiles</strong></td>
<td><strong>803</strong></td>
</tr>
<tr>
<td>CPXNETreadcopybase</td>
<td>804</td>
</tr>
<tr>
<td>CPXreadcopybase</td>
<td>805</td>
</tr>
<tr>
<td>CPXreadcopymipstart</td>
<td>806</td>
</tr>
<tr>
<td>CPXreadcopyorder</td>
<td>807</td>
</tr>
<tr>
<td>CPXreadcopyprob</td>
<td>808</td>
</tr>
<tr>
<td>CPXreadcopyqp</td>
<td>810</td>
</tr>
<tr>
<td>CPXreadcopy sos</td>
<td>811</td>
</tr>
<tr>
<td>CPXreadcopytree</td>
<td>812</td>
</tr>
<tr>
<td>CPXreadcopyvec</td>
<td>813</td>
</tr>
<tr>
<td><strong>Group optim.cplex.callable.readnetworkfiles</strong></td>
<td><strong>813</strong></td>
</tr>
<tr>
<td>CPXNETcreateprob</td>
<td>815</td>
</tr>
</tbody>
</table>
## TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXNETreadcopybase</td>
<td>816</td>
</tr>
<tr>
<td><strong>Group optim.cplex.callable.writefiles</strong></td>
<td>816</td>
</tr>
<tr>
<td>CPXNETbasewrite</td>
<td>818</td>
</tr>
<tr>
<td>CPXNETwriteprob</td>
<td>819</td>
</tr>
<tr>
<td>CPXdperwrite</td>
<td>821</td>
</tr>
<tr>
<td>CPXdualwrite</td>
<td>822</td>
</tr>
<tr>
<td>CPXembwrite</td>
<td>824</td>
</tr>
<tr>
<td>CPXiswrite</td>
<td>825</td>
</tr>
<tr>
<td>CPXmbasewrite</td>
<td>826</td>
</tr>
<tr>
<td>CPXmstwrite</td>
<td>827</td>
</tr>
<tr>
<td>CPXordwrite</td>
<td>828</td>
</tr>
<tr>
<td>CPXpperwrite</td>
<td>829</td>
</tr>
<tr>
<td>CPXpreslvwrite</td>
<td>830</td>
</tr>
<tr>
<td>CPXqwrite</td>
<td>832</td>
</tr>
<tr>
<td>CPXsoswrite</td>
<td>833</td>
</tr>
<tr>
<td>CPXtreewrite</td>
<td>834</td>
</tr>
<tr>
<td>CPXvecwrite</td>
<td>835</td>
</tr>
<tr>
<td>CPXwriteprob</td>
<td>836</td>
</tr>
<tr>
<td>CPXwritesol</td>
<td>838</td>
</tr>
<tr>
<td><strong>Group optim.cplex.callable.writenetworkfiles</strong></td>
<td>839</td>
</tr>
<tr>
<td>CPXNETbasewrite</td>
<td>840</td>
</tr>
<tr>
<td>CPXNETwriteprob</td>
<td>841</td>
</tr>
<tr>
<td><strong>Group optim.cplex.callable.network</strong></td>
<td>842</td>
</tr>
<tr>
<td>CPXNETaddarcs</td>
<td>844</td>
</tr>
<tr>
<td>CPXNETaddnodes</td>
<td>846</td>
</tr>
<tr>
<td>CPXNETbasewrite</td>
<td>847</td>
</tr>
<tr>
<td>CPXNETcheckcopynet</td>
<td>848</td>
</tr>
<tr>
<td>CPXNETchgarname</td>
<td>849</td>
</tr>
<tr>
<td>CPXNETchgarnodes</td>
<td>850</td>
</tr>
<tr>
<td>CPXNETchgbds</td>
<td>852</td>
</tr>
<tr>
<td>CPXNETchgname</td>
<td>854</td>
</tr>
</tbody>
</table>
# Table of Contents

- CPXNETchgnodename .................................................. 856
- CPXNETchgobj ........................................................... 857
- CPXNETchgobjsen ...................................................... 858
- CPXNETchgsupply ...................................................... 859
- CPXNETcopybase ....................................................... 860
- CPXNETcopynet .......................................................... 862
- CPXNETcreateprob ..................................................... 865
- CPXNETdelarcs .......................................................... 866
- CPXNETdelnodes ....................................................... 867
- CPXNETdelset .......................................................... 868
- CPXNETextract .......................................................... 869
- CPXNETfreeprob ........................................................ 871
- CPXNETgetarcindex .................................................... 872
- CPXNETgetarcname ..................................................... 873
- CPXNETgetarcnodes .................................................... 875
- CPXNETgetbase .......................................................... 877
- CPXNETgetdj ............................................................. 879
- CPXNETgetitcnt ........................................................ 881
- CPXNETgetlb ............................................................. 882
- CPXNETgetnodearcs ..................................................... 883
- CPXNETgetnodeindex ................................................... 885
- CPXNETgetnodename .................................................... 886
- CPXNETgetnumarcs ..................................................... 888
- CPXNETgetnumnodes .................................................... 889
- CPXNETgetobj ............................................................. 890
- CPXNETgetobjsen ........................................................ 891
- CPXNETgetobjval ........................................................ 892
- CPXNETgetphase1cnt ................................................... 893
- CPXNETgetpi ............................................................. 894
- CPXNETgetprobnames .................................................. 895
- CPXNETgetslack ........................................................ 896
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXNETgetstat</td>
<td>897</td>
</tr>
<tr>
<td>CPXNETgetsupply</td>
<td>898</td>
</tr>
<tr>
<td>CPXNETgetub</td>
<td>899</td>
</tr>
<tr>
<td>CPXNETgetx</td>
<td>900</td>
</tr>
<tr>
<td>CPXNETprimopt</td>
<td>901</td>
</tr>
<tr>
<td>CPXNETreadcopybase</td>
<td>902</td>
</tr>
<tr>
<td>CPXNETsolninfo</td>
<td>903</td>
</tr>
<tr>
<td>CPXNETsolution</td>
<td>905</td>
</tr>
<tr>
<td>CPXNETwriteprob</td>
<td>907</td>
</tr>
<tr>
<td><strong>Group optim.cplex.callable.util.</strong></td>
<td></td>
</tr>
<tr>
<td>CPXcloseCPLEX</td>
<td>909</td>
</tr>
<tr>
<td>CPXopenCPLEX</td>
<td>910</td>
</tr>
<tr>
<td>CPXversion</td>
<td>911</td>
</tr>
<tr>
<td><strong>Group optim.cplex.callable.portability.</strong></td>
<td>911</td>
</tr>
<tr>
<td>CPXfclose</td>
<td>912</td>
</tr>
<tr>
<td>CPXfopen</td>
<td>913</td>
</tr>
<tr>
<td>CPXfputs</td>
<td>914</td>
</tr>
<tr>
<td>CPXmsgstr</td>
<td>915</td>
</tr>
<tr>
<td>CPXputenv</td>
<td>916</td>
</tr>
<tr>
<td>CPXstrcpy</td>
<td>917</td>
</tr>
<tr>
<td>CPXstrlen</td>
<td>918</td>
</tr>
<tr>
<td><strong>Group optim.cplex.callable.debug</strong></td>
<td>918</td>
</tr>
<tr>
<td>CPXcheckaddcols</td>
<td>920</td>
</tr>
<tr>
<td>CPXcheckaddrows</td>
<td>921</td>
</tr>
<tr>
<td>CPXcheckchgcoefficient</td>
<td>922</td>
</tr>
<tr>
<td>CPXcheckcopytype</td>
<td>924</td>
</tr>
<tr>
<td>CPXcheckcopylp</td>
<td>925</td>
</tr>
<tr>
<td>CPXcheckcopylpwnames</td>
<td>926</td>
</tr>
<tr>
<td>CPXcheckcopyopsep</td>
<td>928</td>
</tr>
<tr>
<td>CPXcheckcopyquad</td>
<td>929</td>
</tr>
<tr>
<td>CPXcheckcopyços</td>
<td>930</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------</td>
</tr>
<tr>
<td>CPXcheckvals</td>
<td>931</td>
</tr>
<tr>
<td><strong>Group optim.cplex.callable.message</strong></td>
<td>932</td>
</tr>
<tr>
<td>CPXaddchannel</td>
<td>933</td>
</tr>
<tr>
<td>CPXaddfpdest</td>
<td>934</td>
</tr>
<tr>
<td>CPXaddfuncdest</td>
<td>935</td>
</tr>
<tr>
<td>CPXdelchannel</td>
<td>937</td>
</tr>
<tr>
<td>CPXdelfpdest</td>
<td>938</td>
</tr>
<tr>
<td>CPXdelfuncdest</td>
<td>939</td>
</tr>
<tr>
<td>CPXdisconnectchannel</td>
<td>941</td>
</tr>
<tr>
<td>CPXflushchannel</td>
<td>942</td>
</tr>
<tr>
<td>CPXflushstdchannels</td>
<td>943</td>
</tr>
<tr>
<td>CPXgetchannels</td>
<td>944</td>
</tr>
<tr>
<td>CPXgeterrorstring</td>
<td>945</td>
</tr>
<tr>
<td>CPXgetlogfile</td>
<td>946</td>
</tr>
<tr>
<td>CPXmsg</td>
<td>947</td>
</tr>
<tr>
<td>CPXsetlogfile</td>
<td>949</td>
</tr>
<tr>
<td><strong>Group optim.cplex.callable.callbacks</strong></td>
<td>949</td>
</tr>
<tr>
<td>CPXbranchcallbackbranchbds</td>
<td>952</td>
</tr>
<tr>
<td>CPXbranchcallbackbranchconstraints</td>
<td>954</td>
</tr>
<tr>
<td>CPXbranchcallbackbranchgeneral</td>
<td>957</td>
</tr>
<tr>
<td>CPXgetbranchcallbackfunc</td>
<td>960</td>
</tr>
<tr>
<td>CPXgetcallbackctype</td>
<td>964</td>
</tr>
<tr>
<td>CPXgetcallbackglobalb</td>
<td>966</td>
</tr>
<tr>
<td>CPXgetcallbackglobalub</td>
<td>968</td>
</tr>
<tr>
<td>CPXgetcallbackincumbent</td>
<td>970</td>
</tr>
<tr>
<td>CPXgetcallbackinfo</td>
<td>972</td>
</tr>
<tr>
<td>CPXgetcallbacklp</td>
<td>978</td>
</tr>
<tr>
<td>CPXgetcallbacknodeintfeas</td>
<td>980</td>
</tr>
<tr>
<td>CPXgetcallbacknodeobjval</td>
<td>986</td>
</tr>
<tr>
<td>CPXgetcallbacknodeobjval</td>
<td>986</td>
</tr>
</tbody>
</table>
# Table of Contents

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXgetcallbacknodestat</td>
<td>988</td>
</tr>
<tr>
<td>CPXgetcallbacknodeub</td>
<td>990</td>
</tr>
<tr>
<td>CPXgetcallbacknodex</td>
<td>992</td>
</tr>
<tr>
<td>CPXgetcallbackorder</td>
<td>994</td>
</tr>
<tr>
<td>CPXgetcallbackpseudocosts</td>
<td>996</td>
</tr>
<tr>
<td>CPXgetcutcallbackfunc</td>
<td>998</td>
</tr>
<tr>
<td>CPXgetdeletenodecallbackfunc</td>
<td>1000</td>
</tr>
<tr>
<td>CPXgetheuristiccallbackfunc</td>
<td>1002</td>
</tr>
<tr>
<td>CPXgetincumbentcallbackfunc</td>
<td>1004</td>
</tr>
<tr>
<td>CPXgetlpcallbackfunc</td>
<td>1006</td>
</tr>
<tr>
<td>CPXgetmipcallbackfunc</td>
<td>1008</td>
</tr>
<tr>
<td>CPXgetnetcallbackfunc</td>
<td>1010</td>
</tr>
<tr>
<td>CPXgetnodecallbackfunc</td>
<td>1012</td>
</tr>
<tr>
<td>CPXgetsolvecallbackfunc</td>
<td>1014</td>
</tr>
<tr>
<td>CPXsetbranchcallbackfunc</td>
<td>1016</td>
</tr>
<tr>
<td>CPXsetbranchnosolncallbackfunc</td>
<td>1020</td>
</tr>
<tr>
<td>CPXsetcutcallbackfunc</td>
<td>1021</td>
</tr>
<tr>
<td>CPXsetdeletenodecallbackfunc</td>
<td>1024</td>
</tr>
<tr>
<td>CPXsetheuristiccallbackfunc</td>
<td>1027</td>
</tr>
<tr>
<td>CPXsetincumbentcallbackfunc</td>
<td>1030</td>
</tr>
<tr>
<td>CPXsetlpcallbackfunc</td>
<td>1033</td>
</tr>
<tr>
<td>CPXsetmipcallbackfunc</td>
<td>1035</td>
</tr>
<tr>
<td>CPXsetnetcallbackfunc</td>
<td>1037</td>
</tr>
<tr>
<td>CPXsetnodecallbackfunc</td>
<td>1039</td>
</tr>
<tr>
<td>CPXsetsolvecallbackfunc</td>
<td>1042</td>
</tr>
</tbody>
</table>

**Group optim.cplex.errorcodes**                  | 1044  |
<p>| CPXERR_ABORT_CONDITION_NO                     | 1055  |
| CPXERR_ABORT_STRONGBRANCH                     | 1056  |
| CPXERR_ADJ_SIGNS                              | 1057  |
| CPXERR_ADJ_SIGN_QUAD                          | 1058  |
| CPXERR_ADJ_SIGN_SENSE                         | 1059  |</p>
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXERR_ARC_INDEX_RANGE</td>
<td>1060</td>
</tr>
<tr>
<td>CPXERR_ARRAY_BAD_SOS_TYPE</td>
<td>1061</td>
</tr>
<tr>
<td>CPXERR_ARRAY_NOT_ASCENDING</td>
<td>1062</td>
</tr>
<tr>
<td>CPXERR.ARRAY_TOO_LONG</td>
<td>1063</td>
</tr>
<tr>
<td>CPXERR_BAD_ARGUMENT</td>
<td>1064</td>
</tr>
<tr>
<td>CPXERR_BAD_BOUND_SENSE</td>
<td>1065</td>
</tr>
<tr>
<td>CPXERR_BAD_BOUND_TYPE</td>
<td>1066</td>
</tr>
<tr>
<td>CPXERR_BAD_CHAR</td>
<td>1067</td>
</tr>
<tr>
<td>CPXERR_BAD_CTYPE</td>
<td>1068</td>
</tr>
<tr>
<td>CPXERR_BAD_DIRECTION</td>
<td>1069</td>
</tr>
<tr>
<td>CPXERR_BAD_EXPONENT</td>
<td>1070</td>
</tr>
<tr>
<td>CPXERR_BAD_EXPO_RANGE</td>
<td>1071</td>
</tr>
<tr>
<td>CPXERR_BAD_FILETYPE</td>
<td>1072</td>
</tr>
<tr>
<td>CPXERR_BAD_ID</td>
<td>1073</td>
</tr>
<tr>
<td>CPXERR_BAD_INDICATOR</td>
<td>1074</td>
</tr>
<tr>
<td>CPXERR_BAD_LUB</td>
<td>1075</td>
</tr>
<tr>
<td>CPXERR_BAD_METHOD</td>
<td>1076</td>
</tr>
<tr>
<td>CPXERR_BAD_NUMBER</td>
<td>1077</td>
</tr>
<tr>
<td>CPXERR_BAD_OBJ_SENSE</td>
<td>1078</td>
</tr>
<tr>
<td>CPXERR_BAD_PARAM_NAME</td>
<td>1079</td>
</tr>
<tr>
<td>CPXERR_BAD_PARAM_NUM</td>
<td>1080</td>
</tr>
<tr>
<td>CPXERR_BAD_PIVOT</td>
<td>1081</td>
</tr>
<tr>
<td>CPXERR_BAD_PRIORITY</td>
<td>1082</td>
</tr>
<tr>
<td>CPXERR_BAD_PROB_TYPE</td>
<td>1083</td>
</tr>
<tr>
<td>CPXERR_BAD_ROW_ID</td>
<td>1084</td>
</tr>
<tr>
<td>CPXERR_BAD_SECTION_BOUNDS</td>
<td>1085</td>
</tr>
<tr>
<td>CPXERR_BAD_SECTION_ENDATA</td>
<td>1086</td>
</tr>
<tr>
<td>CPXERR_BAD_SECTION_QMATRIX</td>
<td>1087</td>
</tr>
<tr>
<td>CPXERR_BAD_SENSE</td>
<td>1088</td>
</tr>
<tr>
<td>CPXERR_BAD_SOS_TYPE</td>
<td>1089</td>
</tr>
<tr>
<td>CPXERR_BAD_STATUS</td>
<td>1090</td>
</tr>
<tr>
<td>Error Code</td>
<td>Page Number</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>CPXERR_BAS_FILE_SHORT</td>
<td>1091</td>
</tr>
<tr>
<td>CPXERR_BAS_FILE_SIZE</td>
<td>1092</td>
</tr>
<tr>
<td>CPXERR_BOUNDS_INFEAS</td>
<td>1093</td>
</tr>
<tr>
<td>CPXERR_BOUNDS_INT</td>
<td>1094</td>
</tr>
<tr>
<td>CPXERR_CALLBACK</td>
<td>1095</td>
</tr>
<tr>
<td>CPXERR_CANT_CLOSE_CHILD</td>
<td>1096</td>
</tr>
<tr>
<td>CPXERR_CHILD_OF_CHILD</td>
<td>1097</td>
</tr>
<tr>
<td>CPXERR_COL_INDEX_RANGE</td>
<td>1098</td>
</tr>
<tr>
<td>CPXERR_COL_REPEATS</td>
<td>1099</td>
</tr>
<tr>
<td>CPXERR_COL_REPEAT_PRINT</td>
<td>1100</td>
</tr>
<tr>
<td>CPXERR_COL_ROW_REPEATS</td>
<td>1101</td>
</tr>
<tr>
<td>CPXERR_COL_UNKNOWN</td>
<td>1102</td>
</tr>
<tr>
<td>CPXERR_COUNT_OVERLAP</td>
<td>1103</td>
</tr>
<tr>
<td>CPXERR_COUNT_RANGE</td>
<td>1104</td>
</tr>
<tr>
<td>CPXERR_DBL_MAX</td>
<td>1105</td>
</tr>
<tr>
<td>CPXERR_DECOMPRESSION</td>
<td>1106</td>
</tr>
<tr>
<td>CPXERR_DUP_ENTRY</td>
<td>1107</td>
</tr>
<tr>
<td>CPXERR_EXTRA_BV_BOUND</td>
<td>1108</td>
</tr>
<tr>
<td>CPXERR_EXTRA_FR_BOUND</td>
<td>1109</td>
</tr>
<tr>
<td>CPXERR_EXTRA_FX_BOUND</td>
<td>1110</td>
</tr>
<tr>
<td>CPXERR_EXTRA_INTEND</td>
<td>1111</td>
</tr>
<tr>
<td>CPXERR_EXTRA_INTORG</td>
<td>1112</td>
</tr>
<tr>
<td>CPXERR_EXTRA_SOSEND</td>
<td>1113</td>
</tr>
<tr>
<td>CPXERR_EXTRA_SOSORG</td>
<td>1114</td>
</tr>
<tr>
<td>CPXERR_FAIL_OPEN_READ</td>
<td>1115</td>
</tr>
<tr>
<td>CPXERR_FAIL_OPEN_WRITE</td>
<td>1116</td>
</tr>
<tr>
<td>CPXERR_FILE_ENTRIES</td>
<td>1117</td>
</tr>
<tr>
<td>CPXERR_FILE_FORMAT</td>
<td>1118</td>
</tr>
<tr>
<td>CPXERR_IIS_DEFAULT</td>
<td>1119</td>
</tr>
<tr>
<td>CPXERR_IIS_FEAS</td>
<td>1120</td>
</tr>
<tr>
<td>CPXERR_IIS_NOT_INFEAS</td>
<td>1121</td>
</tr>
<tr>
<td>Error Code</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>CPXERR_IIS_NO_BASIC</td>
<td>1122</td>
</tr>
<tr>
<td>CPXERR_IIS_NO_INFO</td>
<td>1123</td>
</tr>
<tr>
<td>CPXERR_IIS_NO_LOAD</td>
<td>1124</td>
</tr>
<tr>
<td>CPXERR_IIS_NO_SOLN</td>
<td>1125</td>
</tr>
<tr>
<td>CPXERR_IIS_NUM_BEST</td>
<td>1126</td>
</tr>
<tr>
<td>CPXERR_IIS_OPT_INFEAS</td>
<td>1127</td>
</tr>
<tr>
<td>CPXERR_IIS_SUB_IT_LIM</td>
<td>1128</td>
</tr>
<tr>
<td>CPXERR_IIS_SUB_OBJ_LIM</td>
<td>1129</td>
</tr>
<tr>
<td>CPXERR_IIS_SUB_TIME_LIM</td>
<td>1130</td>
</tr>
<tr>
<td>CPXERR_INDEX_NOT_BASIC</td>
<td>1131</td>
</tr>
<tr>
<td>CPXERR_INDEX_RANGE</td>
<td>1132</td>
</tr>
<tr>
<td>CPXERR_INDEX_RANGE_HIGH</td>
<td>1133</td>
</tr>
<tr>
<td>CPXERR_INDEX_RANGE_LOW</td>
<td>1134</td>
</tr>
<tr>
<td>CPXERR_INT_TOO_BIG</td>
<td>1135</td>
</tr>
<tr>
<td>CPXERR_INT_TOO_BIG_INPUT</td>
<td>1136</td>
</tr>
<tr>
<td>CPXERR_INVALID_NUMBER</td>
<td>1137</td>
</tr>
<tr>
<td>CPXERR_LIMITS_TOO_BIG</td>
<td>1138</td>
</tr>
<tr>
<td>CPXERR_LO_BOUND_REPEATS</td>
<td>1139</td>
</tr>
<tr>
<td>CPXERR_MISS_SOS_TYPE</td>
<td>1140</td>
</tr>
<tr>
<td>CPXERR_MSG_NO_CHANNEL</td>
<td>1141</td>
</tr>
<tr>
<td>CPXERR_MSG_NO_FILEPTR</td>
<td>1142</td>
</tr>
<tr>
<td>CPXERR_MSG_NO_FUNCTION</td>
<td>1143</td>
</tr>
<tr>
<td>CPXERR_NAME_CREATION</td>
<td>1144</td>
</tr>
<tr>
<td>CPXERR_NAME_NOT_FOUND</td>
<td>1145</td>
</tr>
<tr>
<td>CPXERR_NAME_TOO_LONG</td>
<td>1146</td>
</tr>
<tr>
<td>CPXERR_NAN</td>
<td>1147</td>
</tr>
<tr>
<td>CPXERR_NEED_OPT_SOLN</td>
<td>1148</td>
</tr>
<tr>
<td>CPXERR_NEGATIVE_SURPLUS</td>
<td>1149</td>
</tr>
<tr>
<td>CPXERR_NET_DATA</td>
<td>1150</td>
</tr>
<tr>
<td>CPXERR_NET_FILE_SHORT</td>
<td>1151</td>
</tr>
<tr>
<td>CPXERR_NODE_INDEX_RANGE</td>
<td>1152</td>
</tr>
<tr>
<td>Error Code</td>
<td>Page</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>CPXERR_NODE_ON_DISK</td>
<td>1153</td>
</tr>
<tr>
<td>CPXERR_NOT_DUAL_UNBOUNDED</td>
<td>1154</td>
</tr>
<tr>
<td>CPXERR_NOT_FIXED</td>
<td>1155</td>
</tr>
<tr>
<td>CPXERR_NOT_FOR_MIP</td>
<td>1156</td>
</tr>
<tr>
<td>CPXERR_NOT_FOR_QCP</td>
<td>1157</td>
</tr>
<tr>
<td>CPXERR_NOT_FOR_QP</td>
<td>1158</td>
</tr>
<tr>
<td>CPXERR_NOT_MILPCLASS</td>
<td>1159</td>
</tr>
<tr>
<td>CPXERR_NOT_MIN_COST_FLOW</td>
<td>1160</td>
</tr>
<tr>
<td>CPXERR_NOT_MIP</td>
<td>1161</td>
</tr>
<tr>
<td>CPXERR_NOT_MIQPCLASS</td>
<td>1162</td>
</tr>
<tr>
<td>CPXERR_NOT_ONE_PROBLEM</td>
<td>1163</td>
</tr>
<tr>
<td>CPXERR_NOT_QP</td>
<td>1164</td>
</tr>
<tr>
<td>CPXERR_NOT_QPCLASS</td>
<td>1165</td>
</tr>
<tr>
<td>CPXERR_NOT_SAV_FILE</td>
<td>1166</td>
</tr>
<tr>
<td>CPXERR_NOT_UNBOUNDED</td>
<td>1167</td>
</tr>
<tr>
<td>CPXERR_NO_BARRIER_SOLN</td>
<td>1168</td>
</tr>
<tr>
<td>CPXERR_NO_BASIC_SOLN</td>
<td>1169</td>
</tr>
<tr>
<td>CPXERR_NO_BASIS</td>
<td>1170</td>
</tr>
<tr>
<td>CPXERR_NO_BOUND_SENSE</td>
<td>1171</td>
</tr>
<tr>
<td>CPXERR_NO_BOUND_TYPE</td>
<td>1172</td>
</tr>
<tr>
<td>CPXERR_NO_COLUMNS_SECTION</td>
<td>1173</td>
</tr>
<tr>
<td>CPXERR_NO_DUAL_SOLN</td>
<td>1174</td>
</tr>
<tr>
<td>CPXERR_NO_ENDATA</td>
<td>1175</td>
</tr>
<tr>
<td>CPXERR_NO_ENVIRONMENT</td>
<td>1176</td>
</tr>
<tr>
<td>CPXERR_NO_FILENAME</td>
<td>1177</td>
</tr>
<tr>
<td>CPXERR_NO_ID</td>
<td>1178</td>
</tr>
<tr>
<td>CPXERR_NO_ID_FIRST</td>
<td>1179</td>
</tr>
<tr>
<td>CPXERR_NO_INT_SOLN</td>
<td>1180</td>
</tr>
<tr>
<td>CPXERR_NO_INT_X</td>
<td>1181</td>
</tr>
<tr>
<td>CPXERR_NO_LU_FACTOR</td>
<td>1182</td>
</tr>
<tr>
<td>CPXERR_NO_MEMORY</td>
<td>1183</td>
</tr>
</tbody>
</table>
## Table of Contents

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXERR_NO_MIPSTART</td>
<td>1184</td>
</tr>
<tr>
<td>CPXERR_NO_NAMES</td>
<td>1185</td>
</tr>
<tr>
<td>CPXERR_NO_NAME_SECTION</td>
<td>1186</td>
</tr>
<tr>
<td>CPXERR_NO_NORMS</td>
<td>1187</td>
</tr>
<tr>
<td>CPXERR_NO_NUMBER</td>
<td>1188</td>
</tr>
<tr>
<td>CPXERR_NO_NUMBER_BOUND</td>
<td>1189</td>
</tr>
<tr>
<td>CPXERR_NO_NUMBER_FIRST</td>
<td>1190</td>
</tr>
<tr>
<td>CPXERR_NO_OBJECTIVE</td>
<td>1191</td>
</tr>
<tr>
<td>CPXERR_NO_OBJ_Sense</td>
<td>1192</td>
</tr>
<tr>
<td>CPXERR_NO_OPERATOR</td>
<td>1193</td>
</tr>
<tr>
<td>CPXERR_NO_OP_OR_Sense</td>
<td>1194</td>
</tr>
<tr>
<td>CPXERR_NO_ORDER</td>
<td>1195</td>
</tr>
<tr>
<td>CPXERR_NO_PROBLEM</td>
<td>1196</td>
</tr>
<tr>
<td>CPXERR_NO_QMATRIX_SECTION</td>
<td>1197</td>
</tr>
<tr>
<td>CPXERR_NO_QP_OPERATOR</td>
<td>1198</td>
</tr>
<tr>
<td>CPXERR_NO_QUAD_EXP</td>
<td>1199</td>
</tr>
<tr>
<td>CPXERR_NO_RHS_COEFF</td>
<td>1200</td>
</tr>
<tr>
<td>CPXERR_NO_RHS_IN_OBJ</td>
<td>1201</td>
</tr>
<tr>
<td>CPXERR_NO_RNGVAL</td>
<td>1202</td>
</tr>
<tr>
<td>CPXERR_NO_ROWS_SECTION</td>
<td>1203</td>
</tr>
<tr>
<td>CPXERR_NO_ROW_NAME</td>
<td>1204</td>
</tr>
<tr>
<td>CPXERR_NO_ROW_Sense</td>
<td>1205</td>
</tr>
<tr>
<td>CPXERR_NO_SENSIT</td>
<td>1206</td>
</tr>
<tr>
<td>CPXERR_NO_SOLN</td>
<td>1207</td>
</tr>
<tr>
<td>CPXERR_NO_SOS</td>
<td>1208</td>
</tr>
<tr>
<td>CPXERR_NO_SOS_SEPARATOR</td>
<td>1209</td>
</tr>
<tr>
<td>CPXERR_NO_TREE</td>
<td>1210</td>
</tr>
<tr>
<td>CPXERR_NOVECTOR_SOLN</td>
<td>1211</td>
</tr>
<tr>
<td>CPXERR_NULL_NAME</td>
<td>1212</td>
</tr>
<tr>
<td>CPXERR_NULL_POINTER</td>
<td>1213</td>
</tr>
<tr>
<td>CPXERR_ORDER_BAD_DIRECTION</td>
<td>1214</td>
</tr>
<tr>
<td>Error Code</td>
<td>Page Number</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>CPXERR_PARAM_TOO_BIG</td>
<td>1215</td>
</tr>
<tr>
<td>CPXERR_PARAM_TOO_SMALL</td>
<td>1216</td>
</tr>
<tr>
<td>CPXERR_PRESLV_ABORT</td>
<td>1217</td>
</tr>
<tr>
<td>CPXERR_PRESLV_BAD_PARAM</td>
<td>1218</td>
</tr>
<tr>
<td>CPXERR_PRESLV_BASIS_MEM</td>
<td>1219</td>
</tr>
<tr>
<td>CPXERR_PRESLV_COPYORDER</td>
<td>1220</td>
</tr>
<tr>
<td>CPXERR_PRESLV_COPYSOS</td>
<td>1221</td>
</tr>
<tr>
<td>CPXERR_PRESLV_CRUSHFORM</td>
<td>1222</td>
</tr>
<tr>
<td>CPXERR_PRESLV_DUAL</td>
<td>1223</td>
</tr>
<tr>
<td>CPXERR_PRESLV_FAIL_BASIS</td>
<td>1224</td>
</tr>
<tr>
<td>CPXERR_PRESLV_INF</td>
<td>1225</td>
</tr>
<tr>
<td>CPXERR_PRESLV_INForUNBD</td>
<td>1226</td>
</tr>
<tr>
<td>CPXERR_PRESLV_NO_BASIS</td>
<td>1227</td>
</tr>
<tr>
<td>CPXERR_PRESLV_NO_PROB</td>
<td>1228</td>
</tr>
<tr>
<td>CPXERR_PRESLV_SOLN_QP</td>
<td>1229</td>
</tr>
<tr>
<td>CPXERR_QDIVISOR</td>
<td>1230</td>
</tr>
<tr>
<td>CPXERR_QDUP_ENTRY</td>
<td>1231</td>
</tr>
<tr>
<td>CPXERR_QDIVISOR</td>
<td>1232</td>
</tr>
<tr>
<td>CPXERR_QDIVISOR</td>
<td>1233</td>
</tr>
<tr>
<td>CPXERR_QDIVISOR</td>
<td>1234</td>
</tr>
<tr>
<td>CPXERR_QDIVISOR</td>
<td>1235</td>
</tr>
<tr>
<td>CPXERR_QDIVISOR</td>
<td>1236</td>
</tr>
<tr>
<td>CPXERR_QDIVISOR</td>
<td>1237</td>
</tr>
<tr>
<td>CPXERR_QDIVISOR</td>
<td>1238</td>
</tr>
<tr>
<td>CPXERR_QDIVISOR</td>
<td>1239</td>
</tr>
<tr>
<td>CPXERR_QDIVISOR</td>
<td>1240</td>
</tr>
<tr>
<td>CPXERR_QDIVISOR</td>
<td>1241</td>
</tr>
<tr>
<td>CPXERR_QDIVISOR</td>
<td>1242</td>
</tr>
<tr>
<td>CPXERR_QDIVISOR</td>
<td>1243</td>
</tr>
<tr>
<td>CPXERR_QDIVISOR</td>
<td>1244</td>
</tr>
<tr>
<td>CPXERR_QDIVISOR</td>
<td>1245</td>
</tr>
</tbody>
</table>
# Table of Contents

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXERR_Q_NOT_SYMMETRIC</td>
<td>1246</td>
</tr>
<tr>
<td>CPXERR_RANGE_SECTION_ORDER</td>
<td>1247</td>
</tr>
<tr>
<td>CPXERR_RESTRICTED_VERSION</td>
<td>1248</td>
</tr>
<tr>
<td>CPXERR_RHS_IN_OBJ</td>
<td>1249</td>
</tr>
<tr>
<td>CPXERR_RIMNZ_REPEATS</td>
<td>1250</td>
</tr>
<tr>
<td>CPXERR_RIM_REPEATS</td>
<td>1251</td>
</tr>
<tr>
<td>CPXERR_RIM_ROW_REPEATS</td>
<td>1252</td>
</tr>
<tr>
<td>CPXERR_ROW_INDEX_RANGE</td>
<td>1253</td>
</tr>
<tr>
<td>CPXERR_ROW_REPEATS</td>
<td>1254</td>
</tr>
<tr>
<td>CPXERR_ROW_REPEAT_PRINT</td>
<td>1255</td>
</tr>
<tr>
<td>CPXERR_ROW_UNKNOWN</td>
<td>1256</td>
</tr>
<tr>
<td>CPXERR_SAV_FILE_DATA</td>
<td>1257</td>
</tr>
<tr>
<td>CPXERR_SAV_FILE_WRITE</td>
<td>1258</td>
</tr>
<tr>
<td>CPXERR_SBASE_ILLEGAL</td>
<td>1259</td>
</tr>
<tr>
<td>CPXERR_SBASE_INCOMPAT</td>
<td>1260</td>
</tr>
<tr>
<td>CPXERR_SEMI_BDS</td>
<td>1261</td>
</tr>
<tr>
<td>CPXERR_SINGULAR</td>
<td>1262</td>
</tr>
<tr>
<td>CPXERR_STR_PARAM_TOO_LONG</td>
<td>1263</td>
</tr>
<tr>
<td>CPXERR_SUBPROB_SOLVE</td>
<td>1264</td>
</tr>
<tr>
<td>CPXERR_TILIM_CONDITION_NO</td>
<td>1265</td>
</tr>
<tr>
<td>CPXERR_TILIM_STRONGBRANCH</td>
<td>1266</td>
</tr>
<tr>
<td>CPXERR_TOO_MANY_COEFFS</td>
<td>1267</td>
</tr>
<tr>
<td>CPXERR_TOO_MANY_COLS</td>
<td>1268</td>
</tr>
<tr>
<td>CPXERR_TOO_MANY_RIMNZ</td>
<td>1269</td>
</tr>
<tr>
<td>CPXERR_TOO_MANY_RIMS</td>
<td>1270</td>
</tr>
<tr>
<td>CPXERR_TOO_MANY_ROWS</td>
<td>1271</td>
</tr>
<tr>
<td>CPXERR_TOO_MANY_THREADS</td>
<td>1272</td>
</tr>
<tr>
<td>CPXERR_TREE_MEMORY_LIMIT</td>
<td>1273</td>
</tr>
<tr>
<td>CPXERR_TRE_FILE_COLS</td>
<td>1274</td>
</tr>
<tr>
<td>CPXERR_TRE_FILE_DATA</td>
<td>1275</td>
</tr>
<tr>
<td>CPXERR_TRE_FILE_FORMAT</td>
<td>1276</td>
</tr>
</tbody>
</table>
## Table of Contents

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXERR_TRE_FILE_INTS</td>
<td>1277</td>
</tr>
<tr>
<td>CPXERR_TRE_FILE_NONZ</td>
<td>1278</td>
</tr>
<tr>
<td>CPXERR_TRE_FILE_OBJ</td>
<td>1279</td>
</tr>
<tr>
<td>CPXERR_TRE_FILE_OBJSN</td>
<td>1280</td>
</tr>
<tr>
<td>CPXERR_TRE_FILE_PRESOLVE</td>
<td>1281</td>
</tr>
<tr>
<td>CPXERR_TRE_FILE_ROWS</td>
<td>1282</td>
</tr>
<tr>
<td>CPXERR_TRE_FILE_TYPES</td>
<td>1283</td>
</tr>
<tr>
<td>CPXERR_TRE_FILE_VERSION</td>
<td>1284</td>
</tr>
<tr>
<td>CPXERR_TRE_FILE_WRITE</td>
<td>1285</td>
</tr>
<tr>
<td>CPXERR_UNIQUES_WEIGHTS</td>
<td>1286</td>
</tr>
<tr>
<td>CPXERR_UP_BOUND_REPEATS</td>
<td>1287</td>
</tr>
<tr>
<td>CPXERR_WORK_FILE_OPEN</td>
<td>1288</td>
</tr>
<tr>
<td>CPXERR_WORK_FILE_READ</td>
<td>1289</td>
</tr>
<tr>
<td>CPXERR_WORK_FILE_WRITE</td>
<td>1290</td>
</tr>
<tr>
<td>Group optim.cplex.solutionquality</td>
<td>1290</td>
</tr>
<tr>
<td>CPX_DUAL_OBJ</td>
<td>1293</td>
</tr>
<tr>
<td>CPX_KAPPA</td>
<td>1294</td>
</tr>
<tr>
<td>CPX_MAX_COMP_SLACK</td>
<td>1295</td>
</tr>
<tr>
<td>CPX_MAX_DUAL_INFEAS</td>
<td>1296</td>
</tr>
<tr>
<td>CPX_MAX_DUAL_RESIDUAL</td>
<td>1297</td>
</tr>
<tr>
<td>CPX_MAX_INT_INFEAS</td>
<td>1298</td>
</tr>
<tr>
<td>CPX_MAX_PI</td>
<td>1299</td>
</tr>
<tr>
<td>CPX_MAX_PRIMAL_INFEAS</td>
<td>1300</td>
</tr>
<tr>
<td>CPX_MAX_PRIMAL_RESIDUAL</td>
<td>1301</td>
</tr>
<tr>
<td>CPX_MAX_QCPRIMAL_RESIDUAL</td>
<td>1302</td>
</tr>
<tr>
<td>CPX_MAX_QCSLACK</td>
<td>1303</td>
</tr>
<tr>
<td>CPX_MAX_QCSLACK_INFEAS</td>
<td>1304</td>
</tr>
<tr>
<td>CPX_MAX_RED_COST</td>
<td>1305</td>
</tr>
<tr>
<td>CPX_MAX_SCALED_DUAL_INFEAS</td>
<td>1306</td>
</tr>
<tr>
<td>CPX_MAX_SCALED_DUAL_RESIDUAL</td>
<td>1307</td>
</tr>
<tr>
<td>CPX_MAX_SCALED_PI</td>
<td>1308</td>
</tr>
<tr>
<td>Name</td>
<td>Page</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>CPX_MAX_SCALED_PRIMAL_INFEAS</td>
<td>1309</td>
</tr>
<tr>
<td>CPX_MAX_SCALED_PRIMAL_RESIDUAL</td>
<td>1310</td>
</tr>
<tr>
<td>CPX_MAX_SCALED_RED_COST</td>
<td>1311</td>
</tr>
<tr>
<td>CPX_MAX_SCALED_SLACK</td>
<td>1312</td>
</tr>
<tr>
<td>CPX_MAX_SCALED_X</td>
<td>1313</td>
</tr>
<tr>
<td>CPX_MAX_SLACK</td>
<td>1314</td>
</tr>
<tr>
<td>CPX_MAX_X</td>
<td>1315</td>
</tr>
<tr>
<td>CPX_OBJ_GAP</td>
<td>1316</td>
</tr>
<tr>
<td>CPX_PRIMAL_OBJ</td>
<td>1317</td>
</tr>
<tr>
<td>CPX_SUM_COMP_SLACK</td>
<td>1318</td>
</tr>
<tr>
<td>CPX_SUM_DUAL_INFEAS</td>
<td>1319</td>
</tr>
<tr>
<td>CPX_SUM_DUAL_RESIDUAL</td>
<td>1320</td>
</tr>
<tr>
<td>CPX_SUM_INT_INFEAS</td>
<td>1321</td>
</tr>
<tr>
<td>CPX_SUM_PI</td>
<td>1322</td>
</tr>
<tr>
<td>CPX_SUM_PRIMAL_INFEAS</td>
<td>1323</td>
</tr>
<tr>
<td>CPX_SUM_PRIMAL_RESIDUAL</td>
<td>1324</td>
</tr>
<tr>
<td>CPX_SUM_QCPRIMAL_RESIDUAL</td>
<td>1325</td>
</tr>
<tr>
<td>CPX_SUM_QCSLACK</td>
<td>1326</td>
</tr>
<tr>
<td>CPX_SUM_QCSLACK_INFEAS</td>
<td>1327</td>
</tr>
<tr>
<td>CPX_SUM_RED_COST</td>
<td>1328</td>
</tr>
<tr>
<td>CPX_SUM_SCALED_DUAL_INFEAS</td>
<td>1329</td>
</tr>
<tr>
<td>CPX_SUM_SCALED_DUAL_RESIDUAL</td>
<td>1330</td>
</tr>
<tr>
<td>CPX_SUM_SCALED_PI</td>
<td>1331</td>
</tr>
<tr>
<td>CPX_SUM_SCALED_PRIMAL_INFEAS</td>
<td>1332</td>
</tr>
<tr>
<td>CPX_SUM_SCALED_PRIMAL_RESIDUAL</td>
<td>1333</td>
</tr>
<tr>
<td>CPX_SUM_SCALED_RED_COST</td>
<td>1334</td>
</tr>
<tr>
<td>CPX_SUM_SCALED_SLACK</td>
<td>1335</td>
</tr>
<tr>
<td>CPX_SUM_SCALED_X</td>
<td>1336</td>
</tr>
<tr>
<td>CPX_SUM_SLACK</td>
<td>1337</td>
</tr>
<tr>
<td>CPX_SUM_X</td>
<td>1338</td>
</tr>
<tr>
<td>Group optim.cplex.solutionstatus</td>
<td>1338</td>
</tr>
<tr>
<td>TABLE OF CONTENTS</td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td></td>
</tr>
<tr>
<td>CPXMIP_ABORT_FEAS .................................................. 1341</td>
<td></td>
</tr>
<tr>
<td>CPXMIP_ABORT_INFEAS .................................................. 1342</td>
<td></td>
</tr>
<tr>
<td>CPXMIP_FAIL_FEAS ..................................................... 1343</td>
<td></td>
</tr>
<tr>
<td>CPXMIP_FAIL_FEAS_NO_TREE ........................................... 1344</td>
<td></td>
</tr>
<tr>
<td>CPXMIP_FAIL_INFEAS .................................................. 1345</td>
<td></td>
</tr>
<tr>
<td>CPXMIP_FAIL_INFEAS_NO_TREE ........................................ 1346</td>
<td></td>
</tr>
<tr>
<td>CPXMIP_FEASIBLE_RELAXED ............................................. 1347</td>
<td></td>
</tr>
<tr>
<td>CPXMIP_INFEASIBLE ................................................... 1348</td>
<td></td>
</tr>
<tr>
<td>CPXMIP_INFOrUNBD .................................................... 1349</td>
<td></td>
</tr>
<tr>
<td>CPXMIP_MEM_LIM_FEAS ................................................ 1350</td>
<td></td>
</tr>
<tr>
<td>CPXMIP_MEM_LIM_INFEAS ................................................ 1351</td>
<td></td>
</tr>
<tr>
<td>CPXMIP_NODE_LIM_FEAS ............................................... 1352</td>
<td></td>
</tr>
<tr>
<td>CPXMIP_NODE_LIM_INFEAS ............................................. 1353</td>
<td></td>
</tr>
<tr>
<td>CPXMIP_OPTIMAL ....................................................... 1354</td>
<td></td>
</tr>
<tr>
<td>CPXMIP_OPTIMAL_INFEAS ............................................... 1355</td>
<td></td>
</tr>
<tr>
<td>CPXMIP_OPTIMAL_RELAXED .............................................. 1356</td>
<td></td>
</tr>
<tr>
<td>CPXMIP_OPTIMAL_TOL .................................................. 1357</td>
<td></td>
</tr>
<tr>
<td>CPXMIP_SOL_LIM ....................................................... 1358</td>
<td></td>
</tr>
<tr>
<td>CPXMIP_TIME_LIM_FEAS ............................................... 1359</td>
<td></td>
</tr>
<tr>
<td>CPXMIP_TIME_LIM_INFEAS .............................................. 1360</td>
<td></td>
</tr>
<tr>
<td>CPXMIP_UNBOUNDED .................................................... 1361</td>
<td></td>
</tr>
<tr>
<td>CPX_STAT_ABORT_DUAL_OBJ_LIM ...................................... 1362</td>
<td></td>
</tr>
<tr>
<td>CPX_STAT_ABORT_IT_LIM ................................................ 1363</td>
<td></td>
</tr>
<tr>
<td>CPX_STAT_ABORT_OBJ_LIM .............................................. 1364</td>
<td></td>
</tr>
<tr>
<td>CPX_STAT_ABORT_PRIM_OBJ_LIM ...................................... 1365</td>
<td></td>
</tr>
<tr>
<td>CPX_STAT_ABORT_TIME_LIM ............................................ 1366</td>
<td></td>
</tr>
<tr>
<td>CPX_STAT_ABORT_USER ................................................ 1367</td>
<td></td>
</tr>
<tr>
<td>CPX_STAT_FEASIBLE_RELAXED ......................................... 1368</td>
<td></td>
</tr>
<tr>
<td>CPX_STAT_INFEASIBLE ................................................ 1369</td>
<td></td>
</tr>
<tr>
<td>CPX_STAT_INFOrUNBD .................................................. 1370</td>
<td></td>
</tr>
<tr>
<td>CPX_STAT_NUM_BEST .................................................... 1371</td>
<td></td>
</tr>
<tr>
<td>Table of Contents</td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td></td>
</tr>
<tr>
<td>CPX_STAT_OPTIMAL .......................................................... 1372</td>
<td></td>
</tr>
<tr>
<td>CPX_STAT_OPTIMAL_FACE_UNBOUNDED ....................................... 1373</td>
<td></td>
</tr>
<tr>
<td>CPX_STAT_OPTIMAL_INFEAS .................................................. 1374</td>
<td></td>
</tr>
<tr>
<td>CPX_STAT_OPTIMAL_RELAXED ................................................ 1375</td>
<td></td>
</tr>
<tr>
<td>CPX_STAT_UNBOUNDED ...................................................... 1376</td>
<td></td>
</tr>
<tr>
<td>Group optim.cplex.advanced.callable .................................... 1376</td>
<td></td>
</tr>
<tr>
<td>CPXbasicpresolve .......................................................... 1379</td>
<td></td>
</tr>
<tr>
<td>CPXbinvacol ............................................................... 1381</td>
<td></td>
</tr>
<tr>
<td>CPXbinvarow ............................................................... 1382</td>
<td></td>
</tr>
<tr>
<td>CPXbinvcol ............................................................... 1383</td>
<td></td>
</tr>
<tr>
<td>CPXbinvrow ............................................................... 1384</td>
<td></td>
</tr>
<tr>
<td>CPXbranchcallbackbranchbds .............................................. 1385</td>
<td></td>
</tr>
<tr>
<td>CPXbranchcallbackbranchconstraints .................................... 1387</td>
<td></td>
</tr>
<tr>
<td>CPXbranchcallbackbranchgeneral ......................................... 1390</td>
<td></td>
</tr>
<tr>
<td>CPXbtran ................................................................. 1393</td>
<td></td>
</tr>
<tr>
<td>CPXcheckax ............................................................... 1394</td>
<td></td>
</tr>
<tr>
<td>CPXcheckpib ............................................................. 1396</td>
<td></td>
</tr>
<tr>
<td>CPXcopybasednorms ........................................................ 1398</td>
<td></td>
</tr>
<tr>
<td>CPXcopydnorms ........................................................... 1400</td>
<td></td>
</tr>
<tr>
<td>CPXcopypnorms ............................................................ 1402</td>
<td></td>
</tr>
<tr>
<td>CPXcopyprotected ........................................................ 1404</td>
<td></td>
</tr>
<tr>
<td>CPXcrushform ............................................................. 1406</td>
<td></td>
</tr>
<tr>
<td>CPXcrushpi ............................................................... 1408</td>
<td></td>
</tr>
<tr>
<td>CPXcrushx ............................................................... 1410</td>
<td></td>
</tr>
<tr>
<td>CPXdjfrompi .............................................................. 1412</td>
<td></td>
</tr>
<tr>
<td>CPXdualfarkas ............................................................ 1414</td>
<td></td>
</tr>
<tr>
<td>CPXfreenpresolve ........................................................ 1416</td>
<td></td>
</tr>
<tr>
<td>CPXftran ................................................................. 1417</td>
<td></td>
</tr>
<tr>
<td>CPXgetExactkappa ........................................................ 1418</td>
<td></td>
</tr>
<tr>
<td>CPXgetbasednorms ........................................................ 1419</td>
<td></td>
</tr>
<tr>
<td>CPXgethead ............................................................... 1421</td>
<td></td>
</tr>
<tr>
<td>Function Name</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>CPXgetbranchcallbackfunc</td>
<td>1422</td>
</tr>
<tr>
<td>CPXgetcallbacktype</td>
<td>1426</td>
</tr>
<tr>
<td>CPXgetcallbackglobalb</td>
<td>1428</td>
</tr>
<tr>
<td>CPXgetcallbackglobalub</td>
<td>1430</td>
</tr>
<tr>
<td>CPXgetcallbackincumbent</td>
<td>1432</td>
</tr>
<tr>
<td>CPXgetcallbacklp</td>
<td>1434</td>
</tr>
<tr>
<td>CPXgetcallbacknodeintfeas</td>
<td>1436</td>
</tr>
<tr>
<td>CPXgetcallbacknodelb</td>
<td>1438</td>
</tr>
<tr>
<td>CPXgetcallbacknodeip</td>
<td>1440</td>
</tr>
<tr>
<td>CPXgetcallbacknodeobjval</td>
<td>1442</td>
</tr>
<tr>
<td>CPXgetcallbacknodestat</td>
<td>1444</td>
</tr>
<tr>
<td>CPXgetcallbacknodeub</td>
<td>1446</td>
</tr>
<tr>
<td>CPXgetcallbacknodex</td>
<td>1448</td>
</tr>
<tr>
<td>CPXgetcallbackorder</td>
<td>1450</td>
</tr>
<tr>
<td>CPXgetcallbackpseudocosts</td>
<td>1452</td>
</tr>
<tr>
<td>CPXgetcutcallbackfunc</td>
<td>1454</td>
</tr>
<tr>
<td>CPXgetdeletenodecallbackfunc</td>
<td>1456</td>
</tr>
<tr>
<td>CPXgetdnorms</td>
<td>1458</td>
</tr>
<tr>
<td>CPXgetheuristiccallbackfunc</td>
<td>1460</td>
</tr>
<tr>
<td>CPXgetijdiv</td>
<td>1462</td>
</tr>
<tr>
<td>CPXgetijrow</td>
<td>1464</td>
</tr>
<tr>
<td>CPXgetincumbentcallbackfunc</td>
<td>1466</td>
</tr>
<tr>
<td>CPXgetkappa</td>
<td>1468</td>
</tr>
<tr>
<td>CPXgetnodecallbackfunc</td>
<td>1469</td>
</tr>
<tr>
<td>CPXgetobjoffset</td>
<td>1471</td>
</tr>
<tr>
<td>CPXgetpnorms</td>
<td>1472</td>
</tr>
<tr>
<td>CPXgetprestat</td>
<td>1474</td>
</tr>
<tr>
<td>CPXgetprotected</td>
<td>1477</td>
</tr>
<tr>
<td>CPXgetray</td>
<td>1479</td>
</tr>
<tr>
<td>CPXgetredlp</td>
<td>1481</td>
</tr>
<tr>
<td>CPXgetsolvecallbackfunc</td>
<td>1483</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------</td>
</tr>
<tr>
<td>CPXkilldnorms</td>
<td>1485</td>
</tr>
<tr>
<td>CPXkillpnorms</td>
<td>1486</td>
</tr>
<tr>
<td>CPXmdleave</td>
<td>1487</td>
</tr>
<tr>
<td>CPXpivot</td>
<td>1489</td>
</tr>
<tr>
<td>CPXpivotin</td>
<td>1491</td>
</tr>
<tr>
<td>CPXpivotout</td>
<td>1493</td>
</tr>
<tr>
<td>CPXpreadrows</td>
<td>1494</td>
</tr>
<tr>
<td>CPXprechgobj</td>
<td>1496</td>
</tr>
<tr>
<td>CPXpresolve</td>
<td>1497</td>
</tr>
<tr>
<td>CPXqconstrslackfromx</td>
<td>1499</td>
</tr>
<tr>
<td>CPXqpdjfrompi</td>
<td>1501</td>
</tr>
<tr>
<td>CPXqpunchrushpi</td>
<td>1503</td>
</tr>
<tr>
<td>CPXsetbranchcallbackfunc</td>
<td>1505</td>
</tr>
<tr>
<td>CPXsetbranchnolncallbackfunc</td>
<td>1509</td>
</tr>
<tr>
<td>CPXsetcutcallbackfunc</td>
<td>1510</td>
</tr>
<tr>
<td>CPXsetdeletenodecallbackfunc</td>
<td>1513</td>
</tr>
<tr>
<td>CPXsetheuristiccallbackfunc</td>
<td>1516</td>
</tr>
<tr>
<td>CPXsetincumbentcallbackfunc</td>
<td>1519</td>
</tr>
<tr>
<td>CPXsetnodecallbackfunc</td>
<td>1522</td>
</tr>
<tr>
<td>CPXsetsolvecallbackfunc</td>
<td>1525</td>
</tr>
<tr>
<td>CPXslackfromx</td>
<td>1528</td>
</tr>
<tr>
<td>CPXsolwrite</td>
<td>1530</td>
</tr>
<tr>
<td>CPXstrongbranch</td>
<td>1533</td>
</tr>
<tr>
<td>CPXtightenbds</td>
<td>1535</td>
</tr>
<tr>
<td>CPXuncrushform</td>
<td>1537</td>
</tr>
<tr>
<td>CPXuncrushpi</td>
<td>1539</td>
</tr>
<tr>
<td>CPXuncrushx</td>
<td>1541</td>
</tr>
<tr>
<td>CPXunscaleprob</td>
<td>1543</td>
</tr>
</tbody>
</table>
About This Manual

This reference manual documents the application programming interface (API) of ILOG CPLEX. There are separate reference manuals for the Callable Library, C++, Java, and C#.NET APIs.

What Are the ILOG CPLEX Component Libraries?

The ILOG CPLEX Component Libraries are designed to facilitate the development of applications to solve, modify, and interpret the results of linear, mixed integer, continuous convex quadratic, quadratically constrained, and mixed integer quadratic or quadratically constrained programming.

The ILOG CPLEX Component Libraries consist of:

◆ the CPLEX Callable Library, a C application programming interface (API), and
◆ ILOG Concert Technology, an object-oriented API for C++, Java, and C#.NET users.

ILOG Concert Technology is also part of ILOG Solver, enabling cooperative strategies using CPLEX and Solver together for solving difficult optimization problems.

What You Need to Know

This manual assumes that you are familiar with the operating system on which you are using ILOG CPLEX.

The CPLEX Callable Library is written in the C programming language. The Concert Technology libraries are written in the C++, Java, and C#.NET programming languages. If you use these products, this manual assumes you can write code in the appropriate language, and that you have a working knowledge of a supported integrated development environment (IDE) for that language.

Notation and Naming Conventions

Throughout this manual:
◆ The names of routines and parameters defined in the CPLEX Callable Library begin with CPX. The names of classes and functions defined in Concert Technology C++ API begin with Ilo. This convention helps prevent name space conflicts with user-written routines and other code libraries.

◆ The names of Component Library classes, functions, routines, and routine arguments appear in this typeface (examples: CPXprimopt, numcols, IloCplex::solveFixed).

Related Documentation

In addition to this Reference Manual documenting C and C++, ILOG CPLEX also comes with these resources:

◆ Getting Started with ILOG CPLEX introduces you to ways of specifying models and solving problems with ILOG CPLEX.

◆ The ILOG CPLEX User's Manual explores programming with ILOG CPLEX in greater depth. It provides practical ideas about how to use CPLEX in your own applications and shows how and why design and implementation decisions in the examples were made.

◆ The ILOG CPLEX Release Notes highlight the new features and important changes in this version.

◆ The ILOG CPLEX Java Reference Manual supplies detailed definitions of the Concert Technology Java interfaces and ILOG CPLEX Java classes.

◆ The ILOG CPLEX C#.NET Reference Manual supplies detailed definitions of the Concert Technology C#.NET interfaces and ILOG CPLEX C#.NET classes.

◆ Source code for examples is delivered in the standard distribution.

◆ A readme.html file is delivered in the standard distribution. This file contains the most current information about platform prerequisites for ILOG CPLEX.

All of the manuals and Release Notes are available in online versions. The online documentation, in HTML format, can be accessed through standard HTML browsers.
IloCplex, like all CPLEX technologies, uses branch & cut search when solving mixed integer programming (MIP) models. The branch & cut search procedure manages a search tree consisting of nodes. Every node represents an LP or QP subproblem to be processed; that is, to be solved, to be checked for integrality, and perhaps to be analyzed further. Nodes are called active if they have not yet been processed. After a node has been processed, it is no longer active. IloCplex processes active nodes in the tree until either no more active nodes are available or some limit has been reached.

A branch is the creation of two new nodes from a parent node. Typically, a branch occurs when the bounds on a single variable are modified, with the new bounds remaining in effect for that new node and for any of its descendants. For example, if a branch occurs on a binary variable, that is, one with a lower bound of 0 (zero) and an upper bound of 1 (one), then the result will be two new nodes, one node with a modified upper bound of 0 (the downward branch, in effect requiring this variable to take only the value 0), and the other node with a modified lower bound of 1 (the upward branch, placing the variable at 1). The two new nodes will thus have completely distinct solution domains.

A cut is a constraint added to the model. The purpose of adding any cut is to limit the size of the solution domain for the continuous LP or QP problems represented at the nodes, while not eliminating legal integer solutions. The outcome is thus to reduce the number of branches required to solve the MIP.

As an example of a cut, first consider the following constraint involving three binary (0-1) variables:

\[ 20x + 25y + 30z \leq 40 \]

That sample constraint can be strengthened by adding the following cut to the model:

\[ 1x + 1y + 1z \leq 1 \]
No feasible integer solutions are ruled out by the cut, but some fractional solutions, for example (0.0, 0.4, 1.0), can no longer be obtained in any LP or QP subproblems at the nodes, possibly reducing the amount of searching needed.

The branch & cut method, then, consists of performing branches and applying cuts at the nodes of the tree. Here is a more detailed outline of the steps involved.

First, the branch & cut tree is initialized to contain the root node as the only active node. The root node of the tree represents the entire problem, ignoring all of the explicit integrality requirements. Potential cuts are generated for the root node but, in the interest of keeping the problem size reasonable, not all such cuts are applied to the model immediately. An incumbent solution, that is, the best known solution that satisfies all the integrality requirements, is established at this point for later use in the algorithm. Ordinarily, no such solution is available at this stage, but the user can specify a starting solution using the method `setVectors`. Alternatively, when solving a sequence of modified problems, the user can set the parameter `MIPStart` to the value 1 (one) to indicate that the solution of the previous problem should be used as a starting incumbent for the present problem.

When processing a node, `IloCplex` starts by solving the continuous relaxation of its subproblem, that is, the subproblem without integrality constraints. If the solution violates any cuts, `IloCplex` may add some or all of them to the node problem and resolves it. This procedure is iterated until no more violated cuts are detected (or deemed worth adding at this time) by the algorithm. If at any point in the addition of cuts the node becomes infeasible, the node is pruned (that is, it is removed from the tree).

Otherwise, `IloCplex` checks whether the solution of the node-problem satisfies the integrality constraints. If so, and if its objective value is better than that of the current incumbent, the solution of the node-problem is used as the new incumbent. If not, branching will occur, but first a heuristic method may be tried at this point to see if a new incumbent can be inferred from the LP/QP solution at this node, and other methods of analysis may be performed on this node. The branch, when it occurs, is performed on a variable where the value of the present solution violates its integrality requirement. This practice results in two new nodes being added to the tree for later processing.

Each node, after its relaxation is solved, possesses an optimal objective function value $Z$. At any given point in the algorithm, there is a node whose $Z$ value is better (less, in the case of a minimization problem, or greater for a maximization problem) than all the others. This Best Node value can be compared to the objective function value of the incumbent solution. The resulting MIP Gap, expressed as a percentage of the incumbent solution, serves as a measure of progress toward finding and proving optimality. When active nodes no longer exist, then these two values will have converged toward each other, and the MIP Gap will thus be zero, signifying that optimality of the incumbent has been proven.
It is possible to tell IloCplex to terminate the branch & cut procedure sooner than a completed proof of optimality. For example, a user can set a time limit or a limit on the number of nodes to be processed. Indeed, with default settings, IloCplex will terminate the search when the MIP Gap has been brought lower than 0.0001 (0.01%), because it is often the case that much computation is invested in moving the Best Node value after the eventual optimal incumbent has been located. This termination criterion for the MIP Gap can be changed by the user, of course.

Goals

Goals can be used to control the branch and cut search in IloCplex. Goals are implemented in the class IloCplex::GoalI. The class IloCplex::Goal is the handle class. That is, it contains a pointer to an instance of IloCplex::GoalI along with accessors of objects in the implanentation class.

To implement your own goal, you need to subclass IloCplex::GoalI and implement its virtual member functions execute and duplicateGoal. The method execute controls the branch & cut search. The method duplicateGoal creates a copy of the invoking goal object to be used for parallel branch & cut search. Implementing your goal can be greatly simplified if you use one of the macros ILOCPLEXGOALn.

Every branch & cut node maintains a goal stack, possibly containing IloCplex::GoalI objects. After IloCplex solves the relaxation of a node, it pops the top goal from the goal stack and calls its method execute. There are several types of goals:

- If OrGoal is executed, IloCplex will create child nodes. Each of the child nodes will be initialized with a copy of the goal stack of the current node. Then, for each child node, the specified goal in the OrGoal is pushed onto the corresponding goal stack of the child node. Finally, the current node is deleted. (See IloCplex::GoalI::OrGoal for a more detailed discussion.)

- If a cut goal is executed, the constraint will be added to the current node relaxation. Constraint goals are provided to represent both local and global cuts. Local cut goals are conventionally used to express branching.

- If AndGoal is executed, its subgoals are simply pushed onto the stack. (See IloCplex::GoalI::AndGoal for a more detailed discussion.)

- If IloCplex::GoalI::FailGoal is executed, IloCplex will prune the current node; that is, it will discontinue the search at the current node. IloCplex will continue with another node if there is one still available in the tree.

- If IloCplex::GoalI::SolutionGoal is executed, IloCplex will attempt to inject a user-provided solution as a new incumbent. Before ILOG CPLEX
accepts the injected solution, it first tests whether the injected solution is feasible with respect to the model and goals.

♦ When ILOG CPLEX executes any other goal, the returned goal is simply pushed onto the stack.

IloCplex continues popping goals from the goal stack until OrGoal or FailGoal is executed, or until the stack becomes empty; in the case of an empty stack, it will continue with a built-in search strategy.

The predefined goals OrGoal and AndGoal allow you to combine goals. AndGoal allows you to execute different goals at one node, while OrGoal allows you to execute different goals on different, newly created nodes. A conventional use of these two goals in a return statement of a user-written goal looks like this:

```cpp
return AndGoal ( OrGoal (branch1, branch2), this); 
```

This AndGoal first pushes this (the goal currently being executed) onto the goal stack, and then it pushes the OrGoal. Thus the OrGoal is on top of the stack and will be executed next. When the OrGoal executes, it creates two new nodes and copies the remaining goal stack to both of them. Thus both new nodes will have this goal on top of the goal stack at this point. Then the OrGoal proceeds to push branch1 onto the goal stack of the first child node and branch2 onto the goal stack of the second goal child node. Conventionally, branch1 and branch2 contain cut goals, so by executing branch1 and branch2 at the respective child nodes, the child nodes will be restricted to represent smaller subproblems than their parent. After branch1 and branch2 have been executed, this is on top of the node stack of both child nodes; that is, both child nodes will continue branching according to the same rule. In summary, this example creates the branches branch1 and branch2 and continues in both branches to control the same search strategy this.

To perform a search using a goal, you need to solve the extracted model by calling the method IloCplex::solve(goal) with the goal to use as an argument instead of the standard IloCplex::solve. The method solve(goal) simply pushes the goal onto the goal stack of the root node before calling the standard solve.

See Also

IloCplex::Goal and IloCplex::GoalII
The API of the ILOG CPLEX Callable Library for users of C.

### Global Functions Summary

<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXaddchannel</td>
</tr>
<tr>
<td>CPXaddcols</td>
</tr>
<tr>
<td>CPXaddfpdest</td>
</tr>
<tr>
<td>CPXaddfuncdest</td>
</tr>
<tr>
<td>CPXaddlazyconstraints</td>
</tr>
<tr>
<td>CPXaddqconstr</td>
</tr>
<tr>
<td>CPXaddrows</td>
</tr>
<tr>
<td>CPXaddsos</td>
</tr>
<tr>
<td>CPXaddusercuts</td>
</tr>
<tr>
<td>CPXbaropt</td>
</tr>
<tr>
<td>CPXboundsa</td>
</tr>
<tr>
<td>CPXbranchcallbackbranchbdsl</td>
</tr>
<tr>
<td>CPXbranchcallbackbranchconstraints</td>
</tr>
<tr>
<td>CPXbranchcallbackbranchgeneral</td>
</tr>
<tr>
<td>CPXcheckaddcols</td>
</tr>
<tr>
<td>CPXcheckaddrows</td>
</tr>
<tr>
<td>CPXcheckchgoeflist</td>
</tr>
<tr>
<td>CPXcheckcopyctype</td>
</tr>
<tr>
<td>CPXcheckcopylp</td>
</tr>
<tr>
<td>CPXcheckcopylpwnames</td>
</tr>
<tr>
<td>CPXcheckcopyqsep</td>
</tr>
<tr>
<td>CPXcheckcopyquad</td>
</tr>
<tr>
<td>CPXcheckcopyqsos</td>
</tr>
<tr>
<td>CPXcheckvals</td>
</tr>
<tr>
<td>CPXchgbds</td>
</tr>
<tr>
<td>CPXchgoef</td>
</tr>
<tr>
<td>CPXchgoeflist</td>
</tr>
<tr>
<td>CPXchgooiname</td>
</tr>
<tr>
<td>CPXchgcttype</td>
</tr>
<tr>
<td>CPXchgname</td>
</tr>
<tr>
<td>CPXchgobj</td>
</tr>
<tr>
<td>CPXchgobjsen</td>
</tr>
<tr>
<td>CPXchgprobname</td>
</tr>
<tr>
<td>CPXchgprobtype</td>
</tr>
<tr>
<td>Function</td>
</tr>
<tr>
<td>----------------------------------</td>
</tr>
<tr>
<td>CPXchgpqcoef</td>
</tr>
<tr>
<td>CPXchgqrs</td>
</tr>
<tr>
<td>CPXchgqrgval</td>
</tr>
<tr>
<td>CPXchgrngval</td>
</tr>
<tr>
<td>CPXchgssense</td>
</tr>
<tr>
<td>CPXclonedeprob</td>
</tr>
<tr>
<td>CPXcloseCPLEX</td>
</tr>
<tr>
<td>CPXcompletelp</td>
</tr>
<tr>
<td>CPXcopybase</td>
</tr>
<tr>
<td>CPXcopyctype</td>
</tr>
<tr>
<td>CPXcopylp</td>
</tr>
<tr>
<td>CPXcopylpnames</td>
</tr>
<tr>
<td>CPXcopympstar</td>
</tr>
<tr>
<td>CPXcopyoptolp</td>
</tr>
<tr>
<td>CPXcopyobjname</td>
</tr>
<tr>
<td>CPXcopyorder</td>
</tr>
<tr>
<td>CPXcopypartialbase</td>
</tr>
<tr>
<td>CPXcopyqsep</td>
</tr>
<tr>
<td>CPXcopyquad</td>
</tr>
<tr>
<td>CPXcopyssos</td>
</tr>
<tr>
<td>CPXcopystart</td>
</tr>
<tr>
<td>CPXcreateprob</td>
</tr>
<tr>
<td>CPXcutcallbackadd</td>
</tr>
<tr>
<td>CPXcutcallbackaddlocal</td>
</tr>
<tr>
<td>CPXdelchannel</td>
</tr>
<tr>
<td>CPXdelcols</td>
</tr>
<tr>
<td>CPXdelfpdest</td>
</tr>
<tr>
<td>CPXdelfuncdest</td>
</tr>
<tr>
<td>CPXdelnames</td>
</tr>
<tr>
<td>CPXdelqconstrs</td>
</tr>
<tr>
<td>CPXdelrows</td>
</tr>
<tr>
<td>CPXdelsetcols</td>
</tr>
<tr>
<td>CPXdelsetrows</td>
</tr>
<tr>
<td>CPXdelsetsos</td>
</tr>
<tr>
<td>CPXdisconnectchannel</td>
</tr>
<tr>
<td>CPXdisplayiis</td>
</tr>
<tr>
<td>CPXdperwrite</td>
</tr>
<tr>
<td>CPXdualopt</td>
</tr>
<tr>
<td>CPXdualwrite</td>
</tr>
<tr>
<td>CPXembwrite</td>
</tr>
<tr>
<td>CPXfclose</td>
</tr>
<tr>
<td>CPXfeasopt</td>
</tr>
<tr>
<td>Function</td>
</tr>
<tr>
<td>-----------------------------------</td>
</tr>
<tr>
<td>CPXfindils</td>
</tr>
<tr>
<td>CPXflushchannel</td>
</tr>
<tr>
<td>CPXflushstdchannels</td>
</tr>
<tr>
<td>CPXfopen</td>
</tr>
<tr>
<td>CPXfputs</td>
</tr>
<tr>
<td>CPXfreelazyconstraints</td>
</tr>
<tr>
<td>CPXfreeprob</td>
</tr>
<tr>
<td>CPXfreeusercuts</td>
</tr>
<tr>
<td>CPXgetax</td>
</tr>
<tr>
<td>CPXgetbaritcnt</td>
</tr>
<tr>
<td>CPXgetbase</td>
</tr>
<tr>
<td>CPXgetbestobjval</td>
</tr>
<tr>
<td>CPXgetbranchcallbackfunc</td>
</tr>
<tr>
<td>CPXgetcallbackcallbacktype</td>
</tr>
<tr>
<td>CPXgetcallbackglobalbb</td>
</tr>
<tr>
<td>CPXgetcallbackglobalub</td>
</tr>
<tr>
<td>CPXgetcallbackincumbent</td>
</tr>
<tr>
<td>CPXgetcallbackinfo</td>
</tr>
<tr>
<td>CPXgetcallbacklp</td>
</tr>
<tr>
<td>CPXgetcallbacknodeinfo</td>
</tr>
<tr>
<td>CPXgetcallbacknodeintfeas</td>
</tr>
<tr>
<td>CPXgetcallbacknodeobvval</td>
</tr>
<tr>
<td>CPXgetcallbacknodeolb</td>
</tr>
<tr>
<td>CPXgetcallbacknodep</td>
</tr>
<tr>
<td>CPXgetcallbacknodes</td>
</tr>
<tr>
<td>CPXgetcallbacknodex</td>
</tr>
<tr>
<td>CPXgetcallbacknodeorder</td>
</tr>
<tr>
<td>CPXgetcallbackpseudocosts</td>
</tr>
<tr>
<td>CPXgetcallbackseqinfo</td>
</tr>
<tr>
<td>CPXgetcallbackssosinfo</td>
</tr>
<tr>
<td>CPXgetchannels</td>
</tr>
<tr>
<td>CPXgetclgcnt</td>
</tr>
<tr>
<td>CPXgetcoef</td>
</tr>
<tr>
<td>CPXgetcolindex</td>
</tr>
<tr>
<td>CPXgetcolname</td>
</tr>
<tr>
<td>CPXgetcoils</td>
</tr>
<tr>
<td>CPXgetcovcnt</td>
</tr>
<tr>
<td>CPXgetcrossdexchcnt</td>
</tr>
<tr>
<td>CPXgetcrossdpushcnt</td>
</tr>
<tr>
<td>CPXgetcrosspexchcnt</td>
</tr>
<tr>
<td>CPXgetcrossppushcnt</td>
</tr>
<tr>
<td>Function</td>
</tr>
<tr>
<td>----------------------------------</td>
</tr>
<tr>
<td>CPXgetctype</td>
</tr>
<tr>
<td>CPXgetcutcallbackfunc</td>
</tr>
<tr>
<td>CPXgetcutoff</td>
</tr>
<tr>
<td>CPXgetdblparam</td>
</tr>
<tr>
<td>CPXgetdblquality</td>
</tr>
<tr>
<td>CPXgetdeletenodecallbackfunc</td>
</tr>
<tr>
<td>CPXgetdj</td>
</tr>
<tr>
<td>CPXgetdsbcnt</td>
</tr>
<tr>
<td>CPXgeterrorstring</td>
</tr>
<tr>
<td>CPXgetgencllqcnt</td>
</tr>
<tr>
<td>CPXgetgrad</td>
</tr>
<tr>
<td>CPXgetheuristiccallbackfunc</td>
</tr>
<tr>
<td>CPXgetitls</td>
</tr>
<tr>
<td>CPXgetincumbentcallbackfunc</td>
</tr>
<tr>
<td>CPXgetintparam</td>
</tr>
<tr>
<td>CPXgetintquality</td>
</tr>
<tr>
<td>CPXgetitcnt</td>
</tr>
<tr>
<td>CPXgetlp</td>
</tr>
<tr>
<td>CPXgetlogfile</td>
</tr>
<tr>
<td>CPXgetlpcallbackfunc</td>
</tr>
<tr>
<td>CPXgetmethod</td>
</tr>
<tr>
<td>CPXgetmipcallbackfunc</td>
</tr>
<tr>
<td>CPXgetmipitcnt</td>
</tr>
<tr>
<td>CPXgetmipojval</td>
</tr>
<tr>
<td>CPXgetmipqconstslack</td>
</tr>
<tr>
<td>CPXgetmipslack</td>
</tr>
<tr>
<td>CPXgetmipstart</td>
</tr>
<tr>
<td>CPXgetmipx</td>
</tr>
<tr>
<td>CPXgetnetcallbackfunc</td>
</tr>
<tr>
<td>CPXgetnodecallbackfunc</td>
</tr>
<tr>
<td>CPXgetnodecnt</td>
</tr>
<tr>
<td>CPXgetnodeint</td>
</tr>
<tr>
<td>CPXgetnodeleftcnt</td>
</tr>
<tr>
<td>CPXgetnumbin</td>
</tr>
<tr>
<td>CPXgetnumcols</td>
</tr>
<tr>
<td>CPXgetnumint</td>
</tr>
<tr>
<td>CPXgetnumnz</td>
</tr>
<tr>
<td>CPXgetnumqconststrs</td>
</tr>
<tr>
<td>CPXgetnumqnpjz</td>
</tr>
<tr>
<td>CPXgetnumquad</td>
</tr>
<tr>
<td>CPXgetnumrows</td>
</tr>
<tr>
<td>CPXgetnumsemicont</td>
</tr>
<tr>
<td>Function</td>
</tr>
<tr>
<td>-------------------------</td>
</tr>
<tr>
<td>CPXgetnumsemiint</td>
</tr>
<tr>
<td>CPXgetnumdos</td>
</tr>
<tr>
<td>CPXgetobj</td>
</tr>
<tr>
<td>CPXgetobjname</td>
</tr>
<tr>
<td>CPXgetobjval</td>
</tr>
<tr>
<td>CPXgetorder</td>
</tr>
<tr>
<td>CPXgetparamname</td>
</tr>
<tr>
<td>CPXgetparamnum</td>
</tr>
<tr>
<td>CPXgetphaselcnt</td>
</tr>
<tr>
<td>CPXgetpi</td>
</tr>
<tr>
<td>CPXgetprobname</td>
</tr>
<tr>
<td>CPXgetprobtype</td>
</tr>
<tr>
<td>CPXgetpsbcnt</td>
</tr>
<tr>
<td>CPXgetqconstr</td>
</tr>
<tr>
<td>CPXgetqconstrindex</td>
</tr>
<tr>
<td>CPXgetqconstrname</td>
</tr>
<tr>
<td>CPXgetqconstrslack</td>
</tr>
<tr>
<td>CPXgetqpcocf</td>
</tr>
<tr>
<td>CPXgetquad</td>
</tr>
<tr>
<td>CPXgetrhs</td>
</tr>
<tr>
<td>CPXgetrngval</td>
</tr>
<tr>
<td>CPXgetrowindex</td>
</tr>
<tr>
<td>CPXgetrowname</td>
</tr>
<tr>
<td>CPXgetrows</td>
</tr>
<tr>
<td>CPXgetsense</td>
</tr>
<tr>
<td>CPXgetsifitcnt</td>
</tr>
<tr>
<td>CPXgetsifitphaselcnt</td>
</tr>
<tr>
<td>CPXgetslack</td>
</tr>
<tr>
<td>CPXgetsolvecallbackfunc</td>
</tr>
<tr>
<td>CPXgetzos</td>
</tr>
<tr>
<td>CPXgetstat</td>
</tr>
<tr>
<td>CPXgetstatstring</td>
</tr>
<tr>
<td>CPXgetstrparam</td>
</tr>
<tr>
<td>CPXgetsubmethod</td>
</tr>
<tr>
<td>CPXgetsubstat</td>
</tr>
<tr>
<td>CPXgetub</td>
</tr>
<tr>
<td>CPXgetx</td>
</tr>
<tr>
<td>CPXgetxqax</td>
</tr>
<tr>
<td>CPXhybbaroopt</td>
</tr>
<tr>
<td>CPXhybnetopt</td>
</tr>
<tr>
<td>CPXiiswrite</td>
</tr>
<tr>
<td>Function</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>CPXinfodblparam</td>
</tr>
<tr>
<td>CPXinfointparam</td>
</tr>
<tr>
<td>CPXinfostrparam</td>
</tr>
<tr>
<td>CPXlpopt</td>
</tr>
<tr>
<td>CPXmbswrite</td>
</tr>
<tr>
<td>CPXmipopt</td>
</tr>
<tr>
<td>CPXmsg</td>
</tr>
<tr>
<td>CPXmsgstr</td>
</tr>
<tr>
<td>CPXmsgwrite</td>
</tr>
<tr>
<td>CPXNETaddarcs</td>
</tr>
<tr>
<td>CPXNETaddnodes</td>
</tr>
<tr>
<td>CPXNETbasewrite</td>
</tr>
<tr>
<td>CPXNETcheckcopynet</td>
</tr>
<tr>
<td>CPXNETchgarcname</td>
</tr>
<tr>
<td>CPXNETchgarcnodes</td>
</tr>
<tr>
<td>CPXNETchgbd</td>
</tr>
<tr>
<td>CPXNETchggname</td>
</tr>
<tr>
<td>CPXNETchgnodename</td>
</tr>
<tr>
<td>CPXNETchgbj</td>
</tr>
<tr>
<td>CPXNETchgbjsen</td>
</tr>
<tr>
<td>CPXNETchgsupply</td>
</tr>
<tr>
<td>CPXNETcopybase</td>
</tr>
<tr>
<td>CPXNETcopycopynet</td>
</tr>
<tr>
<td>CPXNETcreateprob</td>
</tr>
<tr>
<td>CPXNETdelarcs</td>
</tr>
<tr>
<td>CPXNETdelnodes</td>
</tr>
<tr>
<td>CPXNETdelset</td>
</tr>
<tr>
<td>CPXNETextract</td>
</tr>
<tr>
<td>CPXNETfreeprob</td>
</tr>
<tr>
<td>CPXNETgetarcindex</td>
</tr>
<tr>
<td>CPXNETgetarcname</td>
</tr>
<tr>
<td>CPXNETgetarcnodes</td>
</tr>
<tr>
<td>CPXNETgetbase</td>
</tr>
<tr>
<td>CPXNETgetdj</td>
</tr>
<tr>
<td>CPXNETgetitcnt</td>
</tr>
<tr>
<td>CPXNETgetlb</td>
</tr>
<tr>
<td>CPXNETgetnodearcs</td>
</tr>
<tr>
<td>CPXNETgetnodeindex</td>
</tr>
<tr>
<td>CPXNETgetnodename</td>
</tr>
<tr>
<td>CPXNETgetnumarcs</td>
</tr>
<tr>
<td>CPXNETgetnumnodes</td>
</tr>
<tr>
<td>CPXNETgetobj</td>
</tr>
<tr>
<td>Function</td>
</tr>
<tr>
<td>----------------------------------</td>
</tr>
<tr>
<td>CPXNETgetobjsen</td>
</tr>
<tr>
<td>CPXNETgetobjval</td>
</tr>
<tr>
<td>CPXNETgetphaselcnt</td>
</tr>
<tr>
<td>CPXNETgetpi</td>
</tr>
<tr>
<td>CPXNETgetprobnme</td>
</tr>
<tr>
<td>CPXNETgetslack</td>
</tr>
<tr>
<td>CPXNETgetstat</td>
</tr>
<tr>
<td>CPXNETgetsupply</td>
</tr>
<tr>
<td>CPXNETgetub</td>
</tr>
<tr>
<td>CPXNETgetx</td>
</tr>
<tr>
<td>CPXNETprimopt</td>
</tr>
<tr>
<td>CPXNETTreadcopybase</td>
</tr>
<tr>
<td>CPXNETTreadcopyprob</td>
</tr>
<tr>
<td>CPXNETSolninfo</td>
</tr>
<tr>
<td>CPXNETsolution</td>
</tr>
<tr>
<td>CPXNETwriteprob</td>
</tr>
<tr>
<td>CPXnewcols</td>
</tr>
<tr>
<td>CPXnewrows</td>
</tr>
<tr>
<td>CPXobjlsa</td>
</tr>
<tr>
<td>CPXopenCPLEX</td>
</tr>
<tr>
<td>CPXordwrite</td>
</tr>
<tr>
<td>CPXpperwrite</td>
</tr>
<tr>
<td>CPXpreslvwrite</td>
</tr>
<tr>
<td>CPXprimopt</td>
</tr>
<tr>
<td>CPXputenv</td>
</tr>
<tr>
<td>CPXqpindefcertificate</td>
</tr>
<tr>
<td>CPXqpopt</td>
</tr>
<tr>
<td>CPXqpswrite</td>
</tr>
<tr>
<td>CPXreadcopybase</td>
</tr>
<tr>
<td>CPXreadcopymipstart</td>
</tr>
<tr>
<td>CPXreadcopyorder</td>
</tr>
<tr>
<td>CPXreadcopyparam</td>
</tr>
<tr>
<td>CPXreadcopyprob</td>
</tr>
<tr>
<td>CPXreadcopyqp</td>
</tr>
<tr>
<td>CPXreadcopyso</td>
</tr>
<tr>
<td>CPXreadcopytree</td>
</tr>
<tr>
<td>CPXreadcopyvec</td>
</tr>
<tr>
<td>CPXrhsa</td>
</tr>
<tr>
<td>CPXsetbranchcallbackfunc</td>
</tr>
<tr>
<td>CPXsetbranchnosolncallbackfunc</td>
</tr>
<tr>
<td>CPXsetcutcallbackfunc</td>
</tr>
</tbody>
</table>
### Description

For access to the routines of the Callable Library organized by their purpose, see the Overview of the API or see the groups of `optim.cplex.callable`

<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXsetdblparam</td>
</tr>
<tr>
<td>CPXsetdefaults</td>
</tr>
<tr>
<td>CPXsetdeletenodecallbackfunc</td>
</tr>
<tr>
<td>CPXsetheuristiccallbackfunc</td>
</tr>
<tr>
<td>CPXsetincumbentcallbackfunc</td>
</tr>
<tr>
<td>CPXsetintparam</td>
</tr>
<tr>
<td>CPXsetlogfile</td>
</tr>
<tr>
<td>CPXsetlpcallbackfunc</td>
</tr>
<tr>
<td>CPXsetmipcallbackfunc</td>
</tr>
<tr>
<td>CPXsetnetcallbackfunc</td>
</tr>
<tr>
<td>CPXsetnodecallbackfunc</td>
</tr>
<tr>
<td>CPXsetsolvecallbackfunc</td>
</tr>
<tr>
<td>CPXsetstrparam</td>
</tr>
<tr>
<td>CPXsolinfo</td>
</tr>
<tr>
<td>CPXsolution</td>
</tr>
<tr>
<td>CPXsoswrite</td>
</tr>
<tr>
<td>CPXstrcpy</td>
</tr>
<tr>
<td>CPXstrlen</td>
</tr>
<tr>
<td>CPXtreewrite</td>
</tr>
<tr>
<td>CPXvecwrite</td>
</tr>
<tr>
<td>CPXversion</td>
</tr>
<tr>
<td>CPXwriteparam</td>
</tr>
<tr>
<td>CPXwriteprob</td>
</tr>
<tr>
<td>CPXwritesol</td>
</tr>
</tbody>
</table>
CPXNETaddarcs

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXNETaddarcs(CPXENVptr env,
                             CPXNETptr net,
                             int narcs,
                             const int * fromnode,
                             const int * tonode,
                             const double * low,
                             const double * up,
                             const double * obj,
                             char ** anames)
```

Description:
The routine CPXNETaddarcs adds new arcs to the network stored in a network problem object.

Example:

```c
status = CPXNETaddarcs (env, net, narcs, fromnode, tonode, NULL,
                         NULL, obj, NULL);
```

See Also:
CPXNETgetnumnodes

Parameters:

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

net
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

narcs
Number of arcs to be added.

fromnode
Array of indices of the “from node” for the arcs to be added. All the indices must be greater than or equal to 0. If a node index is greater than or equal to the number of nodes currently in the network (see CPXNETgetnumnodes) new nodes are created implicitly with default supply values 0. The size of the fromnode array must be at least narcs.

tonode
Array of indices of the “to node” for the arcs to be added. All the indices must be greater than or equal to 0. If a node index is greater than or equal to the number of nodes currently in the network (see CPXNETgetnumnodes) new nodes are created implicitly with default supply values 0. The size of the tonode array must be at least narcs.

low

Pointer to an array of lower bounds on the flow through added arcs. If NULL is passed, all lower bounds default to 0 (zero). Otherwise, the size of the array must be at least narcs. Values less than or equal to $-\text{CPX\_INFBND}$ are considered as -$\infty$.

up

Pointer to an array of upper bounds on the flow of added arcs. If NULL is passed, all upper bounds default to $\text{CPX\_INFBND}$. Otherwise, the size of the array must be at least narcs. Values greater than or equal to $\text{CPX\_INFBND}$ are considered as $\infty$.

obj

Pointer to an array of objective values for the added arcs. If NULL is passed, all objective values default to 0. Otherwise, the size of the array must be at least narcs.

anames

Pointer to an array of names for added arcs. If NULL is passed and the existing arcs have names, default names are assigned to the added arcs. If NULL is passed and the existing arcs have no names, the new arcs are assigned no names. Otherwise, the size of the array must be at least narcs and every name in the array must be a string terminating in 0. If the existing arcs have no names and anames is not NULL, default names are assigned to the existing arcs.

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXNETaddnodes

Category            Global Function
Definition File     cplex.h
Include Files       cplex.h
Synopsis
int CPXPUBLIC CPXNETaddnodes(CPXENVptr env,
                                   CPXNETptr net,
                                   int nnodes,
                                   const double * supply,
                                   char ** name)
Description
The routine CPXNETaddnodes adds new nodes to the network stored in a network
problem object.
Example
status = CPXNETaddnodes (env, net, nnodes, supply, NULL);
Parameters
env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
net
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
nnodes
Number of nodes to add.
supply
Supply values for the added nodes. If NULL is passed, all supplies defaults to 0 (zero).
Otherwise, the size of the array must be at least nnodes.
name
Pointer to an array of names for added nodes. If NULL is passed and the existing nodes
have names, default names are assigned to the added nodes. If NULL is passed but the
existing nodes have no names, the new nodes are assigned no names. Otherwise, the size
of the array must be at least nnodes and every name in the array must be a string
terminating in 0. If the existing nodes have no names and nnames is not NULL, default
names are assigned to the existing nodes.
Returns
The routine returns zero on success and nonzero if an error occurs.
CPXNETbasewrite

Category          Global Function
Definition File   cplex.h
Include Files    cplex.h

Synopsis          int CPXPUBLIC CPXNETbasewrite(CPXENVptr env, 
                  CPXCNETptr net, 
                  const char * filename_str)

Description       The routine CPXNETbasewrite writes the current basis stored in a network problem object to a file in BAS format. If no arc or node names are available for the problem object, default names are used.

Example

    status = CPXNETbasewrite (env, net, "netbasis.bas");

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

net
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

filename_str
Name of the basis file to write.

Returns
The routine returns zero on success and nonzero if an error occurs.
**CPXNETcheckcopynet**

<table>
<thead>
<tr>
<th>Category</th>
<th>Global Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition File</td>
<td>cplex.h</td>
</tr>
<tr>
<td>Include Files</td>
<td>cplex.h</td>
</tr>
</tbody>
</table>

**Synopsis**

```c
int CPXPUBLIC CPXNETcheckcopynet (CPXCENVptr env,
    CPXNETptr net,
    int objsen,
    int nnodes,
    const double * supply,
    char ** nnames,
    int narcs,
    const int * fromnode,
    const int * tonode,
    const double * low,
    const double * up,
    const double * obj,
    char ** aname);
```

**Description**

The routine **CPXNETcheckcopynet** performs a consistency check on the argument passed to the routine **CPXNETcopynet**.

The **CPXNETcheckcopynet** routine has the same argument list as the **CPXNETcopynet** routine.

**Example**

```c
status = CPXNETcheckcopynet (env, net, CPX_MAX, nnodes, supply,
    nnames, narcs, fromnode, tonode,
    lb, ub, obj, anames);
```

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXNETchgarcname

Category Global Function

Definition File cplex.h

Include Files cplex.h

Synopsis

int CPXPUBLIC CPXNETchgarcname(CPXENVptr env,
CPXNETptr net,
int cnt,
const int * indices,
char ** newname)

Description

This routine CPXNETchgarcname changes the names of a set of arcs in the network stored in a network problem object.

Example

status = CPXNETchgarcname (env, net, 10, indices, newname);

Parameters

env A pointer to the CPLEX environment as returned by CPXopenCPLEX.

net A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

cnt An integer that indicates the total number of arc names to be changed. Thus cnt specifies the length of the arrays indices and anames.

indices An array of length cnt containing the numeric indices of the arcs for which the names are to be changed.

newname An array of length cnt containing the strings of the new arc names specified in indices.

Returns The routine returns zero on success and nonzero if an error occurs.
CPXNETchgarcnodes

Category Global Function

Definition File cplex.h

Include Files cplex.h

Synopsis

```c
int CPXPUBLIC CPXNETchgarcnodes(CPXCENVptr env,
    CPXNETptr net,
    int cnt,
    const int * indices,
    const int * fromnode,
    const int * tonode)
```

Description

The routine CPXNETchgarcnodes can be used to change the nodes for a set of arcs in the network stored in a network problem object.

Any solution information stored in the problem object is lost.

Example

```c
status = CPXNETchgarcs (env, net, cnt, indices, newfrom, newto);
```

Parameters

```c
env
```

A pointer to the CPLEX environment as returned by CPXopenCPLEX.

```c
net
```

A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

```c
cnt
```

Number of arcs to change.

```c
indices
```

An array of arc indices that indicate the arcs to be changed. This array must have a length of at least cnt. All indices must be in the range [0, narcs-1].

```c
fromnode
```

An array of indices of the “from node” for which the arcs are to be changed. All indices must be in the range [0, nnodes-1]. The size of the array must be at least cnt.

```c
 tonode
```

An array of indices of the “to node” for which the arcs are to be changed. All indices must be in the range [0, nnodes-1]. The size of the array must be at least cnt.
**Returns**

The routine returns zero on success and nonzero if an error occurs.
## CPXNETchgbds

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

### Synopsis

```c
int CPXPUBLIC CPXNETchgbds(CPXENVptr env,  
                           CPXNETptr net,  
                           int cnt,  
                           const int * indices,  
                           const char * lu,  
                           const double * bd)
```

### Description

The routine `CPXNETchgbds` is used to change the upper, lower, or both bounds on the flow for a set of arcs in the network stored in a network problem object. The flow value of an arc can be fixed to a value by setting both bounds to that value.

Any solution information stored in the problem object is lost.

### Example

```c
status = CPXNETchgbds (env, net, cnt, index, lu, bd);
```

### Indicators to change lower, upper bounds of flows through arcs

<table>
<thead>
<tr>
<th>Lu[i]</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'L'</td>
<td>The lower bound of arc index[i] is changed to bd[i]</td>
</tr>
<tr>
<td>'U'</td>
<td>The upper bound of arc index[i] is changed to bd[i]</td>
</tr>
<tr>
<td>'B'</td>
<td>Both bounds of arc index[i] are changed to bd[i]</td>
</tr>
</tbody>
</table>

### Parameters

- **env**  
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- **net**  
  A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.

- **cnt**  
  Number of bounds to change.

- **indices**  
  Indicators to change lower, upper bounds of flows through arcs.
An array of arc indices that indicate the bounds to be changed. This array must have a length of at least \( \text{cnt} \). All indices must be in the range \([0, \text{narcs}-1]\).

\( \text{lu} \)

An array indicating which bounds to change. This array must have a length of at least \( \text{cnt} \). The indicators appear in the table.

\( \text{bd} \)

An array of bound values. This array must have a length of at least \( \text{cnt} \). Values greater than or equal to \( \text{CPX}_\text{INFBOUND} \) and less than or equal to \(-\text{CPX}_\text{INFBOUND} \) are considered infinity or -infinity, respectively.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXNETchgname

Category          Global Function
Definition File   cplex.h
Include Files     cplex.h
Synopsis          int CPXPUBLIC CPXNETchgname(CPXENVptr env,
                                      CPXNETptr net,
                                      int key,
                                      int vindex,
                                      const char * name_str)
Description       The routine CPXNETchgname changes the name of a node or an arc in the network stored in a network problem object.

<table>
<thead>
<tr>
<th>Values of key in CPXNETchgname</th>
</tr>
</thead>
<tbody>
<tr>
<td>key == 'a'</td>
</tr>
<tr>
<td>key == 'n'</td>
</tr>
</tbody>
</table>

Example

status = CPXNETchgname (env, net, 'a', 10, "arc10");

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

net
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

key
A character to indicate whether an arc name should be changed, or a node name should be changed.

vindex
The index of the arc or node whose name is to be changed.

name_str
The new name for the arc or node.
Returns

The routine returns zero on success and nonzero if an error occurs.
CPXNETchgnodename

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXNETchgnodename (CPXCENVptr env,
CPXNETptr net,
int cnt,
const int * indices,
char ** newname)
```

**Description**  
The routine CPXNETchgnodename changes the names of a set of nodes in the network stored in a network problem object.

**Example**

```c
status = CPXNETchgnodename (env, net, 10, indices, newname);
```

**Parameters**

- **env**  
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **net**  
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

- **cnt**  
An integer that indicates the total number of node names to be changed. Thus `cnt` specifies the length of the arrays `indices` and `name`.

- **indices**  
An array of length `cnt` containing the numeric indices of the nodes for which the names are to be changed.

- **newname**  
An array of length `cnt` containing the strings of the new node names specified in `indices`.

**Returns**  
The routine returns zero on success and nonzero if an error occurs.
**CPXNETchgobj**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**

```
int CPXPUBLIC CPXNETchgobj (CPXCENVptr env,
                           CPXNETptr net,
                           int cnt,
                           const int * indices,
                           const double * obj)
```

**Description**
The routine CPXNETchgobj is used to change the objective values for a set of arcs in the network stored in a network problem object.

Any solution information stored in the problem object is lost.

**Example**

```
status = CPXNETchgobj (env, net, cnt, indices, newobj);
```

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **net**
  A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

- **cnt**
  Number of arcs for which the objective values are to be changed.

- **indices**
  An array of indices that indicate the arcs for which the objective values are to be changed. This array must have a length of at least cnt. The indices must be in the range [0, narcs-1].

- **obj**
  An array of the new objective values for the arcs. This array must have a length of at least cnt.

**Returns**
The routine returns zero on success and nonzero if an error occurs.
CPXNETchgobjsen

Category Global Function
Definition File cplex.h
Include Files cplex.h
Synopsis
int CPXPUBLIC CPXNETchgobjsen(CPXENVptr env,
                CPXNETptr net,
                int maxormin)
Description
The routine CPXNETchgobjsen is used to change the sense of the network problem
to a minimization or maximization problem.
Any solution information stored in the problem object is lost.

<table>
<thead>
<tr>
<th>Changed optimization sense in a network problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_MAX</td>
</tr>
<tr>
<td>CPX_MIN</td>
</tr>
</tbody>
</table>

Example

```c
status = CPXNETchgobjsen (env, net, CPX_MAX);
```

Parameters

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **net**
  A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
- **maxormin**
  New optimization sense for the network problem. The possible values are in the table.

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXNETchgsupply

Category    Global Function
Definition File    cplex.h
Include Files    cplex.h

Synopsis

```c
int CPXPUBLIC CPXNETchgsupply(CPXENVptr env,
    CPXNETptr net,
    int cnt,
    const int * indices,
    const double * supply)
```

Description

The routine CPXNETchgsupply is used to change supply values for a set of nodes in the network stored in a network problem object. Any solution information stored in the problem object is lost.

Example

```c
status = CPXNETchgsupply (env, net, cnt, indices, supply);
```

Parameters

```
env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

net
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

cnt
An integer indicating the number of nodes for which the objective values are to be changed.

indices
An array of indices that indicate the nodes for which the supply values are to be changed. This array must have a length of at least cnt. The indices must be in the range [0, nnodes-1].

supply
An array to contain the new supply values. This array must have a length of at least cnt.
```

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXNETcopybase

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:
```c
int CPXPUBLIC CPXNETcopybase(CPXENVptr env,
                              CPXNETptr net,
                              const int * astat,
                              const int * nstat)
```

Description:
The routine CPXNETcopybase can be used to set the network basis for a network problem object. It is not necessary to load a basis prior to optimizing a problem, but a very good starting basis may increase the speed of optimization significantly. A copied basis does not need to be feasible to be used by the network optimizer.

Any solution information stored in the problem object is lost.

Example:
```c
status = CPXNETcopybase (env, net, arc_stat, node_stat);
```

Parameters:
- `env`

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_BASIC</td>
<td>If the arc is to be basic</td>
</tr>
<tr>
<td>CPX_AT_LOWER</td>
<td>If the arc is to be nonbasic and its flow is on the lower bound</td>
</tr>
<tr>
<td>CPX_AT_UPPER</td>
<td>If the arc is to be nonbasic and its flow is on the upper bound</td>
</tr>
<tr>
<td>CPX_FREE_SUPER</td>
<td>If the arc is to be nonbasic but is free. In this case its flow is set to 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_BASIC</td>
<td>If the arc is to be basic</td>
</tr>
<tr>
<td>CPX_AT_LOWER</td>
<td>If the arc is to be nonbasic and its flow is set to 0</td>
</tr>
</tbody>
</table>

Table 1: Status of arcs in astat

Table 2: Status of artificial arcs in nstat
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

net

A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

astat

Array of status values for network arcs. Each arc needs to be assigned one of the values in Table 1.

nstat

Array of status values for artificial arcs from each node to the root node. Each artificial arc needs to be assigned one of the values in Table 2. At least one of the artificial arcs must be assigned the status CPX_BASIC for a network basis.

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXNETcopynet

Category                      Global Function
Definition File              cplex.h
Include Files                cplex.h

Synopsis
int CPXPUBLIC CPXNETcopynet(CPXENVptr env,
                             CPXNETptr net,
                             int objsen,
                             int nnodes,
                             const double * supply,
                             char ** nnames,
                             int narcs,
                             const int * fromnode,
                             const int * tonode,
                             const double * low,
                             const double * up,
                             const double * obj,
                             char ** anames)

Description
The routine CPXNETcopynet copies a network to a network object, overriding any other network saved in the object. The network to be copied is specified by providing the:

◆ the objective sense
◆ number of nodes
◆ supply values for each node
◆ names for each node
◆ number of arcs
◆ indices of the “from nodes” (or, equivalently, the tail nodes) for each arc
◆ indices of the “to nodes” (or, equivalently, the head nodes) for each arc
◆ lower and upper bounds on flow through each arc
◆ cost for flow through each arc
◆ names of each arc as parameters to the routine CPXNETcopynet.

The arcs are numbered according to the order given in the fromnode and tonode arrays. Some of the parameters are optional and replaced by default values if NULL is passed for them.

Example
status = CPXNETcopynet (env, net, CPX_MAX, nnodes, supply, NULL, 
narcs, fromnode, tonode, NULL, NULL, obj, 
NULL);

Parameters
env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
net
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
objs
Optimization sense of the network to be copied. It may take values CPX_MAX for a maximization problem or CPX_MIN for a minimization problem.
nnodes
Number of nodes to be copied to the network object.
supply
Supply values for the nodes. If NULL is passed all supply values default to 0 (zero). Otherwise, the size of the array must be at least nnodes.
nnames
Pointer to an array of names for the nodes. If NULL is passed, no names are assigned to the nodes. Otherwise, the size of the array must be at least nnodes and every name in the array must be a string terminating in 0.
narcs
Number of arcs to be copied to the network object.
fromnode
The array of indices in each arc’s “from node.” The indices must be in the range [0, nnodes-1]. The size of the array must be at least narcs.
tonode
The array of indices in each arc’s “to node.” The indices must be in the range [0, nnodes-1]. The size of the array must be at least narcs.
low
Pointer to an array of lower bounds on the flow through arcs. If NULL is passed, all lower bounds default to 0 (zero). Otherwise, the size of the array must be at least narcs. Values less than or equal to -CPX_INFBOUND are considered -infinity.
up
Pointer to an array of upper bounds on the flow through arcs. If NULL is passed, all lower bounds default to CPX_INFBOUND. Otherwise, the size of the array must be at least \texttt{narcs}. Values greater than or equal to \texttt{CPX_INFBOUND} are considered infinity.

\textbf{obj}

Pointer to an array of objective values for flow through arcs. If NULL is passed, all objective values default to 0 (zero). Otherwise, the size of the array must be at least \texttt{narcs}.

\textbf{anames}

Pointer to an array of names for the arcs. If NULL is passed, no names are assigned to the nodes. Otherwise, the size of the array must be at least \texttt{narcs}, and every name in the array must be a string terminating in 0 (zero).

\textbf{Returns}

The routine returns zero on success and nonzero if an error occurs.
**CPXNETcreateprob**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
CPXNETptr CPXPUBLIC CPXNETcreateprob(CPXENVptr env,
   int * status_p,
   const char * name_str)
```

**Description**  
The routine **CPXNETcreateprob** constructs a new network problem object. The new object contains a minimization problem for a network with 0 nodes and 0 arcs. Other network problem data can be copied to a network with one of the routines **CPXNETaddnodes**, **CPXNETaddarcs**, **CPXNETcopynet**, **CPXNETextract**, or **CPXNETreadcopyprob**.

**Example**
```c
CPXNETptr net = CPXNETcreateprob (env, &status, "mynet");
```

**See Also**  
**CPXNETaddnodes**, **CPXNETaddarcs**, **CPXNETcopynet**, **CPXNETextract**, **CPXNETreadcopyprob**

**Parameters**
- **env**  
  A pointer to the CPLEX environment as returned by **CPXopenCPLEX**.
- **status_p**  
  A pointer to an integer used to return any error code produced by this routine.
- **name_str**  
  Name of the network to be created.

**Returns**  
If the operation is successful, **CPXNETcreateprob** returns the newly constructed network problem object; if not, it returns either NULL or a nonzero value to indicate an error. In case of an error, the value pointed to by **status_p** contains an integer indicating the cause of the error.
CPXNETdelarcs

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXNETdelarcs(CPXENVptr env,
                           CPXNETptr net,
                           int begin,
                           int end)
```

Description:
The routine CPXNETdelarcs is used to remove a range of arcs from the network stored in a network problem object. The remaining arcs are renumbered starting at zero; their order is preserved. If removing arcs disconnects some nodes from the rest of the network, the disconnected nodes remain part of the network.

Any solution information stored in the problem object is lost.

Example:

```c
status = CPXNETdelarcs (env, net, 10, 20);
```

Parameters:

- **env**: A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **net**: A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
- **begin**: Index of the first arc to be deleted.
- **end**: Index of the last arc to be deleted.

Returns:
The routine returns zero on success and nonzero if an error occurs.
CPXNETdelnodes

Category                   Global Function
Definition File            cplex.h
Include Files              cplex.h

Synopsis                   int CPXPUBLIC CPXNETdelnodes(CPXENVptr env,
                          CPXNETptr net,
                          int begin,
                          int end)

Description                The routine CPXNETdelnodes is used to remove a range of nodes from the network stored in a network problem object. The remaining nodes are renumbered starting at zero; their order is preserved. All arcs incident to the nodes that are deleted are also deleted from the network.
Any solution information stored in the problem object is lost.

Example
status = CPXNETdelnodes (env, net, 10, 20);

Parameters
env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
net
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
begin
Index of the first node to be deleted.
end
Index of the last node to be deleted.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXNETdelset

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```
int CPXPUBLIC CPXNETdelset (CPXCENVptr env,
CPXNETptr net,
int * whichnodes,
int * whicharcs)
```

Description:

The routine CPXNETdelset is used to delete a set of nodes and arcs from the network stored in a network problem object. The remaining nodes and arcs are renumbered starting at zero; their order is preserved.

Any solution information stored in the problem object is lost.

Example:

```
status = CPXNETdelset (env, net, whichnodes, whicharcs);
```

Parameters:

- **env**: A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **net**: A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
- **whichnodes**: Array of size at least CPXNETgetnumnodes that indicates the nodes to be deleted. If `whichnodes[i] == 1`, the node is deleted. For every node deleted, all arcs incident to it are deleted as well. After termination, `whichnode[j]` indicates either the position to which node with index `j` before deletion has been moved or, -1 if the node has been deleted. If NULL is passed, no nodes are deleted.
- **whicharcs**: Array indicating the arc to be deleted. Every arc `i` in the network with `whicharcs[i] == 1` is deleted. After termination, `whicharc[j]` indicates either the position to which arc with index `j` before deletion has been moved or, -1 if the arc has been deleted. This array also contains the deletions due to removed nodes. If NULL is passed, the only arcs deleted are those that are incident to nodes that have been deleted.

Returns:

The routine returns zero on success and nonzero if an error occurs.
CPXNETextract

Category       Global Function
Definition File cplex.h
Include Files  cplex.h
Synopsis

    int CPXPUBLIC CPXNETextract(CPXENVptr env,
                      CPXNETptr net,
                      CPXCLPptr lp,
                      int * colmap,
                      int * rowmap)

Description

The routine CPXNETextract finds an embedded network in the LP stored in a CPLEX problem object and copies it as a network to the network problem object, net. The extraction algorithm is controlled by the environment parameter CPX_PARAM_NETFIND.

If the CPLEX problem object has a basis, an attempt is made to copy the basis to the network object. However, this may fail if the statuses corresponding to the rows and columns of the subnetworks do not form a basis. Even if the entire LP is a network, it may not be possible to load the basis to the network object if none of the slack or artificial variables are basic.

The size of colmap must be at least CPXnumcols(env, lp) + CPXnumrows(env, lp).

The size of rowmap must be at least CPXnumrows(env, lp) + 1.

Example

    status = CPXNETextract (env, net, lp, colmap, rowmap);

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

net
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

colmap
If not NULL, after completion colmap[i] contains the index of the LP column that has been mapped to arc i. If colmap[i] < 0, arc i corresponds to the slack variable for row -colmap[i]-1.

The size of colmap must be at least CPXnumcols(env, lp) + CPXnumrows(env, lp).

rowmap

If not NULL, after completion rowmap[i] contains the index of the LP row that has been mapped to node i. If colmap[i] < 0, node i is a dummy node that has no corresponding row in the LP.

The size of rowmap must be least CPXnumrows(env, lp) + 1.

Example

status = CPXNETextract (env, net, lp, colmap, rowmap);

_returns_
The routine returns zero on success and nonzero if an error occurs.
CPXNETfreeprob

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXNETfreeprob(CPXENVptr env,
   CPXNETptr * net_p)
```

Description:
The routine CPXNETfreeprob deletes the network problem object pointed to by net_p. This also deletes all network problem data and solution data stored in the network problem object.

Example:

```c
CPXNETfreeprob (env, &net);
```

Parameters:

- `env`:
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- `net_p`:
  CPLEX network problem object to be deleted.

Returns:
The routine returns zero on success and nonzero if an error occurs.
CPXNETgetarcindex

Category       Global Function
Definition File cplex.h
Include Files  cplex.h

Synopsis       int CPXPUBLIC CPXNETgetarcindex(CPXENVptr env,
                                       CPXNETptr net,
                                       const char * lname_str,
                                       int * index_p)

Description    The routine CPXNETgetarcindex returns the index of the specified arc (in the network stored in a network problem object) in the integer pointed to by index_p.

Example
    status = CPXNETgetarcindex (env, net, "from_a_to_b", &index);

Parameters

    env
    A pointer to the CPLEX environment as returned by CPXopenCPLEX.

    net
    A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

    lname_str
    Name of the arc to look for.

    index_p
    A pointer to an integer to hold the arc index. If the routine is successful, *index_p contains the index number; otherwise, *index_p is undefined.

Returns        The routine returns zero on success and nonzero if an error occurs.
CPXNETgetarcname

Category          Global Function

Definition File   cplex.h

Include Files    cplex.h

Synopsis
int CPXPUBLIC CPXNETgetarcname(CPXENVptr env,
        CPXNETptr net,
        char ** nnames,
        char * namestore,
        int namespc,
        int * surplus_p,
        int begin,
        int end)

Description
The routine CPXNETgetarcname is used to access the names of a range of arcs in a
network stored in a network problem object. The beginning and end of the range, along
with the length of the array in which the arc names are to be returned, must be specified.

Example
status = CPXNETgetarcname (env, net, nnames, namestore, namespc,
        &surplus, 0, narcs-1);

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

net
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

nnames
Where to copy pointers to arc names stored in the namestore array. The length of this
array must be at least (end-begin+1). The pointer to the name of arc i is returned in
nnames[i-begin].

namestore
Array of characters to which the specified arc names are to be copied. It may be NULL if
namespc is 0.

namespc
Length of the namestore array.

surplus_p
Pointer to an integer to which the difference between namespc and the number of characters required to store the requested names is returned. A nonnegative value indicates that namespc was sufficient. A negative value indicates that it was insufficient. In that case, CPXERR_NEGATIVE_SURPLUS is returned and the negative value of surplus_p indicates the amount of insufficient space in the array namestore.

`begin`

Index of the first arc for which a name is to be obtained.

`end`

Index of the last arc for which a name is to be obtained.

**Returns**

The routine returns zero on success and nonzero if an error occurs. The value CPXERR_NEGATIVE_SURPLUS indicates that insufficient space was available in the namestore array to hold the names.
**CPXNETgetarcnodes**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXNETgetarcnodes(CPXENVptr env,
    CPXNETptr net,
    int * fromnode,
    int * tonode,
    int begin,
    int end)
```

**Description**
The routine `CPXNETgetarcnodes` is used to access the “from nodes” and “to nodes” for a range of arcs in the network stored in a network problem object.

**Example**

```c
status = CPXNETgetarcnodes (env, net, fromnode, tonode,
    0, cur_narcs-1);
```

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- **net**
  A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.

- **fromnode**
  Array in which to write the “from node” indices of the requested arcs. If NULL is passed, no “from node” indices are retrieved. Otherwise, the size of the array must be \((end-begin+1)\).

- **tonode**
  Array in which to write the “to node” indices of the requested arcs. If NULL is passed, no “to node” indices are retrieved. Otherwise, the size of the array must be \((end-begin+1)\).

- **begin**
  Index of the first arc to get nodes for.

- **end**
  Index of the last arc to get nodes for.
Returns

The routine returns zero on success and nonzero if an error occurs.
CPXNETgetbase

Category  Global Function
Definition File  cplex.h
Include Files  cplex.h
Synopsis  

```c
int CPXPUBLIC CPXNETgetbase(CPXENVptr env,
    CPXCNETptr net,
    int * astat,
    int * nstat)
```

Description  The routine CPXNETgetbase is used to access the network basis for a network problem object. Either of the arguments astat or nstat may be NULL.

For this function to succeed, a solution must exist for the problem object.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_BASIC</td>
<td>If the arc is basic.</td>
</tr>
<tr>
<td>CPX_AT_LOWER</td>
<td>If the arc is nonbasic and its flow is on the lower bound.</td>
</tr>
<tr>
<td>CPX_AT_UPPER</td>
<td>If the arc is nonbasic and its flow is on the upper bound.</td>
</tr>
<tr>
<td>CPX_FREE_SUPER</td>
<td>If the arc is nonbasic but is free. In this case its flow is 0.</td>
</tr>
</tbody>
</table>

Table 2: Status of artificial arcs

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_BASIC</td>
<td>If the arc is basic.</td>
</tr>
<tr>
<td>CPX_AT_LOWER</td>
<td>If the arc is nonbasic and its flow is on the lower bound.</td>
</tr>
</tbody>
</table>

Example

```c
status = CPXNETgetbase (env, net, astat, nstat);
```

Parameters

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **net**
  A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
astat
An array in which the statuses for network arcs are to be written. After termination, astat[i] contains the status assigned to arc i of the network stored in net. The status may be one of the values in Table 1. If NULL is passed, no arc statuses are copied. Otherwise, astat must be an array of a size that is at least CPXNETgetnumarcs.

nstat
An array in which the statuses for artificial arcs from each node to the root node are to be written. After termination, nstat[i] contains the status assigned to the artificial arc from node i to the root node of the network stored in net. The status may be one of values in Table 2. If NULL is passed, no node statuses are copied. Otherwise, nstat must be an array of a size that is at least CPXNETgetnumnodes.

Returns
The routine returns zero on success and nonzero if an error occurs.
**CPXNETgetdj**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXNETgetdj(CPXENVptr env,  
CPXNETptr net,  
double * dj,  
int begin,  
int end)
```

**Description**  
The routine `CPXNETgetdj` is used to access reduced costs for a range of arcs of the network stored in a network problem object.

For this function to succeed, a solution must exist for the problem object. If the solution is not feasible (`CPXNETsolninfo` returns 0 in argument `pfeasind_p`), the reduced costs are computed with respect to an objective function that penalizes infeasibilities.

**Example**
```c
status = CPXNETgetdj (env, net, dj, 10, 20);
```

**Parameters**  
- `env`  
A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- `net`  
A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.

- `dj`  
Array in which to write requested reduced costs. If NULL is passed, no reduced cost values are returned. Otherwise, `dj` must point to an array of size at least `(end-begin+1)`.

- `begin`  
Index of the first arc for which a reduced cost value is to be obtained.

- `end`  
Index of the last arc for which a reduced cost value is to be obtained.

**Example**
```c
status = CPXNETgetdj (env, net, dj, 10, 20);
```
Returns

The routine returns zero on success and nonzero if an error occurs.
CPXNETgetitcnt

Category                  Global Function
Definition File          cplex.h
Include Files            cplex.h
Synopsis                  int CPXPUBLIC CPXNETgetitcnt(CPXENVptr env, CPXNETptr net)
Description               The routine CPXNETgetitcnt accesses the total number of network simplex
                           iterations for a previous call to CPXNETprimopt, for a network problem object.
Example                   itcnt = CPXNETgetitcnt (env, net);
Parameters                env
                           A pointer to the CPLEX environment as returned by CPXopenPLEX.
                           net
                           A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
Returns                   Returns the total number of network simplex iterations for the last call to
                           CPXNETprimopt, for a network problem object. If CPXNETprimopt has not been
called, zero is returned. If an error occurs, -1 is returned and an error message is issued.
CPXNETgetlb

Category Global Function
Definition File cplex.h
Include Files cplex.h
Synopsis

```c
int CPXPUBLIC CPXNETgetlb(CPXENVptr env,
                      CPXCNETptr net,
                      double * low,
                      int begin,
                      int end)
```

Description

The routine CPXNETgetlb is used to access the lower capacity bounds for a range of arcs of the network stored in a network problem object.

Example

```c
status = CPXNETgetlb (env, net, low, 0, cur_narcs-1);
```

Parameters

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **net**
  A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

- **low**
  Array in which to write the lower bound on the flow for the requested arcs. If NULL is passed, no lower bounds are retrieved. Otherwise, the size of the array must be (end-begin+1).

- **begin**
  Index of the first arc for which lower bounds are to be obtained.

- **end**
  Index of the last arc for which lower bounds are to be obtained.

Returns

The routine returns zero on success and nonzero if an error occurs.
### CPXNETgetnodearcs

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXNETgetnodearcs(CPXENVptr env,
        CPXNETptr net,
        int * arccnt_p,
        int * arcbeg,
        int * arc,
        int arcspace,
        int * surplus_p,
        int begin,
        int end)
```

**Description**  
The routine `CPXNETgetnodearcs` is used to access the arc indices incident to a range of nodes in the network stored in a network problem object.

**Example**  
```c
status = CPXNETgetnodearcs (env, net, &arccnt, arcbeg, arc,
        arcspace, &surplus, begin, end);
```

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- **net**
  A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.

- **arccnt_p**
  A pointer to an integer to contain the total number of arc indices returned in the array `arc`.

- **arcbeg**
  An array to contain indices indicating where each of the requested arc lists start in array `arc`. Specifically, the list of arcs incident to node `i` (`< end`) consists of the indices in arcs in the range from `arcbeg[i-begin]` to `arcbeg[(i+1)-begin]-1`. The list of arcs incident to node `end` consists of the entries in `arc` in the range from `arcbeg[end-begin]` to `*arccnt_p-1`. This array must have a length of at least `end-begin+1`.
arc
An array to contain the arc indices for the arcs incident to the nodes in the specified range. May be NULL if arcspace is zero.

arcspace
An integer indicating the length of the array arc. May be zero.

surplus_p
A pointer to an integer to contain the difference between arcspace and the number of arcs incident to the nodes in the specified range. A nonnegative value indicates that arcspace was sufficient. A negative value indicates that it was insufficient and that the routine could not complete its task. In that case, CPXERR_NEGATIVE_SURPLUS is returned and the negative value of surplus_p indicates the amount of insufficient space in the array arc.

begin
Index of the first node for which arcs are to be obtained.

end
Index of the last node for which arcs are to be obtained.

**Returns**
The routine returns zero on success and nonzero if an error occurs.
CPXNETgetnodeindex

Category               Global Function
Definition File        cplex.h
Include Files          cplex.h

Synopsis               int CPXPUBLIC CPXNETgetnodeindex(CPXENVptr env,
                             CPXNETptr net,
                             const char * lname_str,
                             int * index_p)

Description            The routine CPXNETgetnodeindex returns the index of the
                        specified node (in the network stored in a network problem object) in the
                        integer pointed to by index_p.

Example                 status = CPXNETgetnodeindex (env, net, "root", &index);

Parameters             env
                       A pointer to the CPLEX environment as returned by CPXopenCPLEX.
net
                       A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
lname_str
                       Name of the node to look for.
index_p
                       A pointer to an integer to hold the node index. If the routine is successful, *index_p
                       contains the index number; otherwise, *index_p is undefined.

Returns                 The routine returns zero on success and nonzero if an error occurs.
CPXNETgetnodename

Category       Global Function
Definition File  cplex.h
Include Files  cplex.h

Synopsis

```c
int CPXPUBLIC CPXNETgetnodename(CPXENVptr env,
                          CPXNETptr net,
                          char ** nnames,
                          char * namestore,
                          int namespc,
                          int * surplus_p,
                          int begin,
                          int end)
```

Description

The routine CPXNETgetnodename is used to obtain the names of a range of nodes in a network stored in a network problem object. The beginning and end of the range, along with the length of the array in which the node names are to be returned, must be specified.

Example

```c
status = CPXNETgetnodename (env, net, nnames, namestore, namespc,
                          &surplus, 0, nnodes-1);
```

Parameters

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **net**
  A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
- **nnames**
  Where to copy pointers to node names stored in the namestore array. The length of this array must be at least (end-begin+1). The pointer to the name of node i is returned in nnames[i-begin].
- **namestore**
  Array of characters to which the specified node names are to be copied. It may be NULL if namespc is 0.
- **namespc**
  Length of the namestore array.
- **surplus_p**
CPXNETgetnodename

Pointer to an integer in which the difference between namespc and the number of characters required to store the requested names is returned. A nonnegative value indicates that namespc was sufficient. A negative value indicates that it was insufficient. In that case, CPXERR_NEGATIVE_SURPLUS is returned and the negative value of surplus_p indicates the amount of insufficient space in the array namestore.

begin
Index of the first node for which a name is to be obtained.

end
Index of the last node for which a name is to be obtained.

Returns
The routine returns zero on success and nonzero if an error occurs. The value CPXERR_NEGATIVE_SURPLUS indicates that there was not enough space in the namestore array to hold the names.
CPXNETgetnumarcs

Category: Global Function
Definition File: cplex.h
Include Files: cplex.h
Synopsis: int CPXPUBLIC CPXNETgetnumarcs(CPXENVptr env, CPXCNETptr net)
Description: The routine CPXNETgetnumarcs is used to access the number of arcs in a network stored in a network problem object.
Example:

cur_narcs = CPXNETgetnumarcs (env, net);

Parameters:
- env: A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- net: A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
Returns: The routine returns the number of network arcs stored in a network problem object. If an error occurs, 0 is returned and an error message is issued.
**CPXNETgetnumnodes**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXNETgetnumnodes(CPXENVptr env, CPXCNETptr net)
```

**Description**  
The routine `CPXNETgetnumnodes` is used to access the number of nodes in a network stored in a network problem object.

**Example**  
```c
cur_nnodes = CPXNETgetnumnodes (env, net);
```

**Parameters**  
- **env**  
A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
- **net**  
A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.

**Returns**  
The routine returns the number of network nodes stored in a network problem object. If an error occurs, 0 is returned and an error message is issued.
CPXNETgetobj

Category       Global Function
Definition File  cplex.h
Include Files   cplex.h
Synopsis
int CPXPUBLIC CPXNETgetobj(CPXCENVptr env,
                        CPXCNETptr net,
                        double * obj,
                        int begin,
                        int end)

Description
The routine CPXNETgetobj is used to access the objective function values for a range of arcs in the network stored in a network problem object.

Example
status = CPXNETgetobj (env, net, obj, 0, cur_narcs-1);

Parameters
    env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

    net
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

    obj
Array in which to write the objective values for the requested range of arcs. If NULL is passed, no objective values are retrieved. Otherwise, obj must point to an array of size at least (end-begin+1).

    begin
Index of the first arc for which the objective value is to be obtained.

    end
Index of the last arc for which the objective value is to be obtained.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXNETgetobjsen

Category       Global Function

Definition File cplex.h

Include Files  cplex.h

Synopsis       int CPXPUBLIC CPXNETgetobjsen(CPXENVptr env, CPXNETptr net)

Description    The routine CPXNETgetobjsen returns the sense of the objective function, that is, maximization or minimization, of a network problem object.

Example        objsen = CPXNETgetobjsen (env, net);

Parameters

   env
   A pointer to the CPLEX environment as returned by CPXopenCPLEX.

   net
   A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

Returns        The value CPX_MAX (-1) is returned for a maximization problem; the value CPX_MIN (1) is returned for a minimization problem. In case of an error, the value zero is returned.
CPXNETgetobjval

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
#include <cplex.h>

int CPXPUBLIC CPXNETgetobjval(CPXCENVptr env,
                               CPXCNETptr net,
                               double * objval_p);
```

Description:

The routine CPXNETgetobjval returns the objective value of the solution stored in a network problem object. If the current solution is not feasible, the value returned depends on the setting of the parameter CPX_PARAM_NETDISPLAY. If this parameter is set to CPXNET_PENALIZED_OBJECTIVE (2), an objective function value is reported that includes penalty contributions for arcs on which the flow at termination violated the flow bounds on that arc.

Example:

```c
status = CPXNETgetobjval (env, net, &objval);
```

Parameters:

- env
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- net
  A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
- objval_p
  Pointer to where the objective value is written. If NULL is passed, no objective value is returned.

Returns:

The routine returns zero on success and nonzero if an error occurs.
CPXNETgetphase1cnt

Category                Global Function
Definition File         cplex.h
Include Files           cplex.h
Synopsis                int CPXPUBLIC CPXNETgetphase1cnt(CPXCENVptr env,
                                CPXNETptr net)
Description             The routine CPXNETgetphase1cnt returns the number of phase 1 network simplex iterations for a previous call to CPXNETprimopt.
Example                 phase1cnt = CPXNETgetphase1cnt (env, net);

Parameters              env
                        A pointer to the CPLEX environment as returned by CFXopenCPLEX.
                        net
                        A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

Returns                 Returns the total number of phase 1 network simplex iterations for the last call to CPXNETprimopt, for a CPXNETptr object. If CPXNETprimopt has not been called, zero is returned. In an error occurs, -1 is returned and an error message is issued.
CPXNETgetpi

Category                  Global Function
Definition File           cplex.h
Include Files            cplex.h
Synopsis                  int CPXPUBLIC CPXNETgetpi (CPXCENVptr env,
                          CPXCNETptr net,
                          double * pi,
                          int begin,
                          int end)

Description               The routine CPXNETgetpi is used to access dual values for a range of nodes in the
                           network stored in a network problem object.
                           For this function to succeed, a solution must exist for the problem object.

Example                   status = CPXNETgetpi (env, net, pi, 10, 20);

Parameters                env
                          A pointer to the CPLEX environment as returned by CPXopenCPLEX.

                          net
                          A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

                          pi
                          Array in which to write solution dual values for requested nodes. If NULL is passed, no
                          data is returned. Otherwise, pi must point to an array of size at least (end-begin+1).

                          begin
                          Index of the first node for which the dual value is to be obtained.

                          end
                          Index of the last node for which the dual value is to be obtained.

Returns                   The routine returns zero on success and nonzero if an error occurs.
CPXNETgetprobname

Category          Global Function
Definition File   cplex.h
Include Files     cplex.h
Synopsis          int CPXPUBLIC CPXNETgetprobname(CPXENVptr env,  
                  CPXNETptr net,  
                  char * buf_str,  
                  int bufspace,  
                  int * surplus_p)

Description       The routine CPXNETgetprobname is used to access the name of the problem stored in a network problem object.

Example

    status = CPXNETgetprobname (env, net, name, namesize, &surplus);

Parameters

    env  
    A pointer to the CPLEX environment as returned by CPXopenCPLEX.

    net  
    A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

    buf_str  
    Buffer into which the problem name is copied.

    bufspace  
    Size of the array buf_str in bytes.

    surplus_p  
    Pointer to an integer in which the difference between bufspace and the number of characters required to store the problem name is returned. A nonnegative value indicates that bufspace was sufficient. A negative value indicates that it was insufficient. In that case, CPXERR_NEGATIVE_SURPLUS is returned and the negative value of surplus_p indicates the amount of insufficient space in the array buf.

Returns

    The routine returns zero on success and nonzero if an error occurs. The value CPXERR_NEGATIVE_SURPLUS indicates that there was not enough space in the buf array to hold the name.
CPXNETgetslack

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**
```c
int CPXPUBLIC CPXNETgetslack (CPXCENVptr env, 
        CPXNETptr net, 
        double * slack, 
        int begin, 
        int end)
```

**Description**
The routine CPXNETgetslack is used to access slack values or, equivalently, violations of supplies/demands for a range of nodes in the network stored in a network problem object.

For this function to succeed, a solution must exist for the problem object.

**Example**
```c
status = CPXNETgetslack (env, net, slack, 10, 20);
```

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **net**
  A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

- **slack**
  Array in which to write solution slack variables for requested nodes. If NULL is passed, no data is returned. Otherwise, slack must point to an array of size at least (end–begin+1).

- **begin**
  Index of the first node for which a slack value is to be obtained.

- **end**
  Index of the last node for which a slack value is to be obtained.

**Returns**
The routine returns zero on success and nonzero if an error occurs.
CPXNETgetstat

Category  Global Function
Definition File  cplex.h
Include Files  cplex.h
Synopsis  int CPXPUBLIC CPXNETgetstat(CPXCENVptr env, CPXCNETptr net)
Description  The routine CPXNETgetstat returns the solution status for a network problem object.
Example  

    netstatus = CPXNETgetstat (env, net);

Parameters

env  
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

net  
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

Returns  If no solution is available for the network problem object, CPXNETgetstat returns 0 (zero). When a solution exists, the possible return values are:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_STAT_OPTIMAL</td>
<td>Optimal solution found.</td>
</tr>
<tr>
<td>CPX_STAT_UNBOUNDED</td>
<td>Problem has an unbounded ray.</td>
</tr>
<tr>
<td>CPX_STAT_INFEASIBLE</td>
<td>Problem is infeasible.</td>
</tr>
<tr>
<td>CPX_STAT_INFOrUNB</td>
<td>Problem is infeasible or unbounded.</td>
</tr>
<tr>
<td>CPX_STAT_ABORT_IT_LIM</td>
<td>Aborted due to iteration limit.</td>
</tr>
<tr>
<td>CPX_STAT_ABORT_TIME_LIM</td>
<td>Aborted due to time limit.</td>
</tr>
<tr>
<td>CPX_STAT_ABORT_USER</td>
<td>Aborted on user request.</td>
</tr>
</tbody>
</table>
CPXNETgetsupply

Category          Global Function
Definition File   cplex.h
Include Files     cplex.h
Synopsis
int CPXPUBLIC CPXNETgetsupply(CPXENVptr env,
                                   CPXCNETptr net,
                                   double * supply,
                                   int begin,
                                   int end)

Description
The routine CPXNETgetsupply is used to obtain supply values for a range of nodes in the network stored in a CPLEX network problem object.

Example
status = CPXNETgetsupply (env, net, supply,
                          0, CPXNETgetnumnodes (env, net) - 1);

Parameters
env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
net
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
supply
Place where requested supply values are copied. If NULL is passed, no supply values are copied. Otherwise, the array must be of length at least (end-begin+1).
begin
Index of the first node for which a supply value is to be obtained.
end
Index of the last node for which a supply value is to be obtained.

Returns
The routine returns zero on success and nonzero if an error occurs.
**CPXNETgetub**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXNETgetub (CPXCENVptr env,  
CPXCNETptr net,  
double * up,  
int begin,  
int end)
```

**Description**  
The routine CPXNETgetub is used to access the upper capacity bounds for a range of arcs in the network stored in a network problem object.

**Example**  
```c
status = CPXNETgetub (env, net, up, 0, cur_narcs-1);
```

**Parameters**  
- **env**  
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **net**  
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
- **up**  
Array in which to write the upper bound on the flow for the requested arcs. If NULL is passed, no upper bounds are retrieved. Otherwise, the array must be of size \((end-begin+1)\).
- **begin**  
Index of the first arc for which upper bounds are to be obtained.
- **end**  
Index of the last arc for which upper bounds are to be obtained.

**Returns**  
The routine returns zero on success and nonzero if an error occurs.
### CPXNETgetx

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXNETgetx(CPXENVptr env,  
CPXNETptr net,  
double * x,  
int begin,  
int end)
```

**Description**  
The routine CPXNETgetx is used to access solution values or, equivalently, flow values for a range of arcs stored in a network problem object. For this routine to succeed, a solution must exist for the network problem object.

**Example**  
```c
status = CPXNETgetx (env, net, x, 10, 20);
```

**Parameters**

- **env**  
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **net**  
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

- **x**  
Array in which to write solution (or flow) values for requested arcs. If NULL is passed, no solution vector is returned. Otherwise, x must point to an array of size at least (end-begin+1).

- **begin**  
Index of the first arc for which a solution (or flow) value is to be obtained.

- **end**  
Index of the last arc for which a solution (or flow) value is to be obtained.

**Returns**  
The routine returns zero on success and nonzero if an error occurs.
CPXNETprimopt

Category          Global Function
Definition File   cplex.h
Include Files    cplex.h
Synopsis          int CPXPUBLIC CPXNETprimopt(CPXCENVptr env, CPXNETptr net)

Description       The routine CPXNETprimopt can be called after a network problem has been copied to a network problem object, to find a solution to that problem using the primal network simplex method. When this function is called, the CPLEX primal network algorithm attempts to optimize the problem. The results of the optimization are recorded in the problem object and can be retrieved by calling the appropriate solution functions for that object.

Example           status = CPXNETprimopt (env, net);

See also the examples netex1.c and netex2.c in the standard distribution of the product.

Parameters        env
                   A pointer to the CPLEX environment as returned by CPXopenCPLEX.
net
                   A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

Returns           The routine returns zero unless an error occurred during the optimization. Examples of errors include exhausting available memory (CPXERR_NO_MEMORY) or encountering invalid data in the CPLEX problem object (CPXERR_NO_PROBLEM). Exceeding a user-specified CPLEX limit, or proving the model infeasible or unbounded, are not considered errors. Note that a zero return value does not necessarily mean that a solution exists. Use query routines CPXNETsolninfo, CPXNETgetstat, and CPXNETsolution to obtain further information about the status of the optimization.
**CPXNETreadcopybase**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXNETreadcopybase(CPXENVptr env,
                                CPXNETptr net,
                                const char * filename_str)
```

**Description**  
The routine CPXNETreadcopybase reads a basis file in BAS format and copies the basis to a network problem object. If no arc or node names are available for the problem object when reading the basis file, default names are assumed. Any basis that may have been created or saved in the problem object is replaced.

**Example**
```c
status = CPXNETreadcopybase (env, net, "netbasis.bas");
```

**Parameters**
- **env**  
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **net**  
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
- **filename_str**  
Name of the basis file to read.

**Returns**  
The routine returns zero on success and nonzero if an error occurs.
CPXNETreadcopyprob

Category       Global Function
Definition File cplex.h
Include Files  cplex.h

Synopsis       int CPXPUBLIC CPXNETreadcopyprob(CPXENVptr env,
                      CPXNETptr net,
                      const char * filename_str)

Description    The routine CPXNETreadcopyprob reads a network, in the CPLEX .net or
                DIMACS .min format, from a file and copies it to a network problem object. Any
                existing network or solution data in the problem object is replaced.

Example
    status = CPXNETreadcopyprob (env, net, "network.net");

Parameters
    env
    A pointer to the CPLEX environment as returned by CPXopenCPLEX.
    net
    A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
    filename_str
    Name of the network file to read.

Returns        The routine returns zero on success and nonzero if an error occurs.
CPXNETsolninfo

Category       Global Function
Definition File cplex.h
Include Files  cplex.h
Synopsis       int CPXPUBLIC CPXNETsolninfo(CPXENVptr env,
                             CPXNETptr net,
                             int * pfeasind_p,
                             int * dfeasind_p)

Description    The routine CPXNETsolninfo is used to access solution information computed by a previous call to CPXNETprimopt. The solution values are maintained in the object as long as no changes are applied to it with one of the CPXNETchg..., CPXNETcopy... or CPXNETadd... functions.

The arguments to CPXNETsolninfo are pointers to locations where data are to be written. The returned values indicate what is known about the primal and dual feasibility of the current solution. If either piece of information represented by an argument to CPXNETsolninfo is not required, a NULL pointer can be passed for that argument.

Example

    status = CPXNETsolninfo (env, lp, &pfeasind, &dfeasind);

Parameters

    env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

    net
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

    pfeasind_p
A pointer to an integer variables indicating whether the current solution is known to be primal feasible. Note that a false return value does not necessarily mean that the solution is not feasible. It simply means that the relevant algorithm was not able to conclude that it was feasible when it terminated.

    dfeasind_p
A pointer to an integer variables indicating whether the current solution is known to be dual feasible. Note that a false return value does not necessarily mean that the solution is not feasible. It simply means that the relevant algorithm was not able to conclude that it was feasible when it terminated.
**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXNETsolution

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXNETsolution(CPXENVptr env,
                              CPXCNETptr net,
                              int * netstat_p,
                              double * objval_p,
                              double * x,
                              double * pi,
                              double * slack,
                              double * dj)
```

Description:

The routine CPXNETsolution accesses solution values for a network problem object computed by a previous call to CPXNETprimopt for that object. The solution values are maintained in the object as long as no changes are applied to it with one of the CPXNETchg..., CPXNETcopy..., or CPXNETadd... functions. Whether or not a solution exists can be determined by CPXNETsolninfo.

The arguments to CPXNETsolution are pointers to locations where data is to be written. Such data includes the solution status, the value of the objective function, primal, dual and slack values and the reduced costs.

Although all the above data exists after a successful call to CPXNETprimopt, it is possible that the user only needs a subset of the available data. Thus, if any part of the solution represented by an argument to CPXNETsolution is not required, a NULL pointer can be passed for that argument.

Example:

```c
status = CPXNETsolution (env, net, &netstatus, &objval, x, pi,
                        slack, dj);
```

Parameters:

- `env`:
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- `net`:
  A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
- `netstat_p`:
  A pointer to an integer where the solution status will be written.
- `objval_p`:
  A pointer to a double where the objective function value will be written.
- `x`:
  A pointer to a double array where the primal values will be written.
- `pi`:
  A pointer to a double array where the dual values will be written.
- `slack`:
  A pointer to a double array where the slack values will be written.
- `dj`:
  A pointer to a double array where the reduced costs will be written.
Pointer to which the solution status is to be written. The specific values that *netstat_p can take and their meanings are the same as the return values documented for CPXNETgetstat.

**objval_p**

Pointer to which the objective value is to be written. If NULL is passed, no objective value is returned. If the solution status is one of the CPX_STAT_ABORT codes, the value returned depends on the setting of parameter CPX_PARAM_NETDISPLAY. If this parameter is set to 2, objective function values that are penalized for infeasible flows are used to compute the objective value of the solution. Otherwise, the true objective function values are used.

**x**

Array to which the solution (flow) vector is to be written. If NULL is passed, no solution vector is returned. Otherwise, x must point to an array of size at least that returned by CPXNETgetnumarcs.

**pi**

Array to which the dual values are to be written. If NULL is passed, no dual values are returned. Otherwise, pi must point to an array of size at least that returned by CPXNETgetnumnodes.

**slack**

Array to which the slack values (violations of supplies/demands) are to be written. If NULL is passed, no slack values are returned. Otherwise, slack must point to an array of size at least that returned by CPXNETgetnumnodes.

**dj**

Array to which the reduced cost values are to be written. If NULL is passed, no reduced cost values are returned. Otherwise, dj must point to an array of size at least that returned by CPXNETgetnumarcs.

**Returns**

If a solution exists, it returns zero; if not, it returns nonzero to indicate an error.
**CPXNETwriteprob**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXNETwriteprob (CPXCENVptr env,
    CPXCNETptr net,
    const char * filename_str,
    const char * format_str)
```

**Description**  
The routine `CPXNETwriteprob` writes the network stored in a network problem object to a file. This can be done in CPLEX (.net) or DIMACS (.min) network file format (MIN) or as the LP representation of the network in any of the LP formats (.lp, .mps, or .sav).

If the file name ends with .gz, a compressed file is written.

**File extensions for network files**

<table>
<thead>
<tr>
<th>Extension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>net</td>
<td>for CPLEX network format</td>
</tr>
<tr>
<td>min</td>
<td>for DIMACS network format</td>
</tr>
<tr>
<td>lp</td>
<td>for LP format of LP formulation</td>
</tr>
<tr>
<td>mps</td>
<td>for MPS format of LP formulation</td>
</tr>
<tr>
<td>sav</td>
<td>for SAV format of LP formulation</td>
</tr>
</tbody>
</table>

**Example**

```c
status = CPXNETwriteprob (env, net, "network.net", NULL);
```

**Parameters**

**env**
A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

**net**
A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.

**filename_str**
Name of the network file to write, where the file extension specifies the file format unless overridden by the `format_str` argument. If the file name ends with .gz a compressed file is written in accordance with the selected file type.

**format_str**

File format to generate. Possible values appear in the table. If NULL is passed, the format is inferred from the file name.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXaddchannel

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**
CPXCHANNELptr CPXPUBLIC CPXaddchannel(CPXENVptr env)

**Description**
The routine `CPXaddchannel` instantiates a new channel object.

**Example**
mychannel = CPXaddchannel (env);

See also lpex5.c in the *CPLEX User's Manual*.

**Parameters**
env
A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

**Returns**
If successful, `CPXaddchannel` returns a pointer to the new channel object; otherwise, it returns NULL.
CPXaddcols

Category                      Global Function
Definition File               cplex.h
Include Files                 cplex.h

Synopsis

int CPXPUBLIC CPXaddcols(CPXENVptr env,
    CPXLPptr lp,
    int ccnt,
    int nzcnt,
    const double * obj,
    const int * cmatbeg,
    const int * cmatind,
    const double * cmatval,
    const double * lb,
    const double * ub,
    char ** colname)

Description

The routine CPXaddcols adds columns to a specified CPLEX problem object. This routine may be called any time after a problem object is created via CPXcreateprob.

The routine CPXaddcols is very similar to the routine CPXaddrows. The primary difference is that CPXaddcols cannot add coefficients in rows that do not already exist (that is, in rows with index greater than the number returned by CPXgetnumrows); whereas CPXaddrows can add coefficients in columns with index greater than the value returned by CPXgetnumcols, by the use of the ccnt argument. (See the discussion of the ccnt argument for CPXaddrows.) Thus, CPXaddcols has no variable rcnt and no array rowname.

The routine CPXnewrows can be used to add empty rows before adding new columns via CPXaddcols.

The nonzero elements of every column must be stored in sequential locations in the array cmatval from position cmatbeg[i] to cmatbeg[i+1] (or from cmatbeg[i] to nzcnt-1 if i=ccnt-1). Each entry, cmatind[i], indicates the row number of the corresponding coefficient, cmatval[i]. Unlike CPXcopylp, all columns must be contiguous, and cmatbeg[0] must be 0.

Example

status = CPXaddcols (env, lp, ccnt, nzcnt, obj, cmatbeg,
    cmatind, cmatval, lb, ub, newcolname);

Parameters

env
A pointer to the CPLEX environment as returned by the CPXopenCPLEX routine.

lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

cnt
An integer that indicates the number of new columns being added to the constraint matrix.

nzcnt
An integer that indicates the number of nonzero constraint coefficients to be added to the constraint matrix.

obj
An array of length cnt containing the objective function coefficients of the new variables. May be NULL, in which case, the objective coefficients of the new columns are set to 0.0.

cmatbeg
Array that specifies the nonzero elements of the columns being added.

cmatind
Array that specifies the nonzero elements of the columns being added.

cmatval
Array that specifies the nonzero elements of the columns being added. The format is similar to the format used to specify the constraint matrix in the routine CPXcopylp. (See description of matbeg, matcnt, matind, and matval in that routine).

lb
An array of length cnt containing the lower bound on each of the new variables. Any lower bound that is set to a value less than or equal to that of the constant – CPX_INFBOUND is treated as negative infinity. CPX_INFBOUND is defined in the header file cplex.h. May be NULL, in which case the lower bounds of the new columns are set to 0.0.

ub
An array of length cnt containing the upper bound on each of the new variables. Any upper bound that is set to a value greater than or equal to that of the constant CPX_INFBOUND is treated as infinity. CPX_INFBOUND is defined in the header file cplex.h. May be NULL, in which case the upper bounds of the new columns are set to CPX_INFBOUND (positive infinity).

colname
An array of length \( ccnt \) containing pointers to character strings that specify the names of the new variables added to the problem object. May be NULL, in which case the new columns are assigned default names if the columns already resident in the CPLEX problem object have names; otherwise, no names are associated with the variables. If column names are passed to \texttt{CPXaddcols} but existing variables have no names assigned, default names are created for them.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXaddfpdest

Category                 Global Function
Definition File          cplex.h
Include Files            cplex.h
Synopsis                  int CPXPUBLIC CPXaddfpdest(CPXENVptr env,
                          CPXCHANNELptr channel,
                          CPXFILEptr fileptr)
Description               The routine CPXaddfpdest adds a file to the list of message destinations for a channel. The destination list for all CPLEX-defined channels is initially empty.
Example                    CPXaddfpdest (env, mychannel, fileptr);


Parameters

  env
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

  channel
  A pointer to the channel for which destinations are to be added.

  fileptr
  A pointer to the file to be added to the destination list. Before calling this routine, obtain this pointer with a call to CPXfopen.

Returns                  The routine returns zero on success and nonzero if an error occurs.
CPXaddfuncdest

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```
#include <cplex.h>

int CPXPUBLIC CPXaddfuncdest (CPXCENVptr env,
                               CPXCHANNELptr channel,
                               void * handle,
                               void(CPXPUBLIC *msgfunction)(void *, const char *) );
```

Description:

The routine CPXaddfuncdest adds a function msgfunction to the message destination list for a channel. This routine allows users to “trap” messages instead of printing them. That is, when a message is sent to the channel, each destination that was added to the message destination list by CPXaddfuncdest calls its associated message.

To illustrate, consider an application in which a developer wishes to trap CPLEX error messages and display them in a dialog box that prompts the user for an action. Use CPXaddfuncdest to add the address of a function to the list of message destinations associated with the cpxerror channel. Then write the msgfunction routine. It must contain the code that controls the dialog box. When CFXmsg is called with cpxerror as its first argument, it calls the msgfunction routine, which can then display the error message.

Note: The argument handle is a generic pointer that can be used to hold information needed by the msgfunction routine to avoid making such information global to all routines.

Example:

```c
void msgfunction (void *handle, char *msg_string)
{
    FILE *fp;
    fp = (FILE *)handle;
    fprintf (fp, "%s", msg_string);
}

status = CPXaddfuncdest (env, mychannel, fileptr, msgfunction);
```

Parameters:
env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

channel
A pointer to the channel to which the function destination is to be added.

handle
A void pointer that can be used to pass arbitrary information into msgfunction.

msgfunction
A pointer to the function to be called when a message is sent to a channel.

See Also
CPXdelfuncdest

Returns
This routine returns zero on success and nonzero if an error occurs. Failure occurs when msgfunction is not in the message destination list or the channel does not exist.
CPXaddlazyconstraints

Category                  Global Function

Definition File           cplex.h

Include Files            cplex.h

Synopsis                  int CPXPUBLIC CPXaddlazyconstraints(CPXCENVptr env,
                                CPXLPptr lp,
                                int rcnt,
                                int nzcnt,
                                const double * rhs,
                                const char * sense,
                                const int * rmatbeg,
                                const int * rmatind,
                                const double * rmatval)

Description               The routine CPXaddlazyconstraints is used to add constraints to the list of
                           constraints that should be added to the LP subproblem of a MIP optimization if they are
                           violated. CPLEX handles addition of the constraints and makes sure that all integer
                           solutions satisfy all the constraints. The constraints are added to those specified in prior
                           calls to CPXaddlazyconstraints.

                           Lazy constraints are constraints not specified in the constraint matrix of the MIP
                           problem, but that must be not be violated in a solution. Using lazy constraints makes
                           sense when there are a large number of constraints that must be satisfied at a solution,
                           but are unlikely to be violated if they are left out.

                           The CPLEX parameter CPX_PARAM_REDUCE should be set to
                           CPX_PREREDUCE_NOPRIMALORDUAL (0) or to
                           CPX_PREREDUCE_PRIMALONLY (1) in order to turn off dual reductions.

                           Use CPXfreelazyconstraints to clear the list of lazy constraints.

                           The arguments of CPXaddlazyconstraints are the same as those of
                           CPXaddrows, with the exception that new columns may not be specified, so there are
                           no ccnt and colname arguments, and row names may not be specified, so there is no
                           rowname argument. Furthermore, unlike CPXaddrows,
                           CPXaddlazyconstraints does not accept a NULL pointer for the array of right-hand
                           side values.

Example

                           status = CPXaddlazyconstraints(env, lp, cnt, nzcnt, rhs, sense,
                                          beg, ind, val);
Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.
rcnt
An integer that indicates the number of new lazy constraints to be added.
nzcnt
An integer that indicates the number of nonzero constraint coefficients to be added to the constraint matrix. This specifies the length of the arrays rmatind and rmatval.
rhs
An array of length rcnt containing the right-hand side (RHS) term for each constraint to be added to the CPLEX problem object. May be NULL, in which case the new right-hand side values are set to 0.0.
sense
An array of length rcnt containing the sense of each constraint to be added to the CPLEX problem object. May be NULL, in which case the new constraints are created as equality constraints. Possible values of this argument appear in the table.
rmatbeg
An array used with rmatind and rmatval to define the lazy constraints to be added.
rmatind
An array used with rmatbeg and rmatval to define the lazy constraints to be added.
rmatval
An array used with rmatbeg and rmatind to define the lazy constraints to be added. The format is similar to the format used to describe the constraint matrix in the routine CPXcopylp (see description of matbeg, matcnt, matind, and matval in that routine), but the nonzero coefficients are grouped by row instead of column in the array rmatval. The nonzero elements of every lazy constraint must be stored in sequential locations in this array from position rmatbeg[i] to rmatbeg[i+1]-1 (or from rmatbeg[i] to nzcnt -1 if i=rcnt-1). Each entry, rmatind[i], indicates the column index of the corresponding coefficient, rmatval[i]. Unlike CPXcopylp, all rows must be contiguous, and rmatbeg[0] must be 0 (zero).

Returns
The routine returns zero on success and nonzero if an error occurs.
**CPXaddqconstr**

**Category**    
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXaddqconstr(CPXENVptr env,
CPXLPptr lp,
int linnzcnt,
int quadnzcnt,
double rhs,
int sense,
const int * linind,
const double * linval,
const int * quadrow,
const int * quadcol,
const double * quadval,
const char * constrname_str)
```

**Description**  
The routine `CPXaddqconstr` adds a quadratic constraint to a specified CPLEX problem object. This routine may be called any time after a call to `CPXcreateprob`.

**Codes for sense of constraints in QCPs**

<table>
<thead>
<tr>
<th>sense[i]</th>
<th>= 'L'</th>
<th>≤ constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>sense[i]</td>
<td>= 'G'</td>
<td>≥ constraint</td>
</tr>
</tbody>
</table>

**Example**

```c
status = CPXaddqconstr (env, lp, linnzcnt, quadnzcnt, rhsval,
sense, linind, linval,
quadrow, quadcol, quadval, NULL);
```

See also the example `qcpex1.c` in the *ILOG CPLEX User's Manual* and in the standard distribution.

**Parameters**

env
A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

lp
A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

linnzcnt
An integer that indicates the number of nonzero constraint coefficients in the linear part of the constraint. This specifies the length of the arrays linind and linval.

quadnzcnt

An integer that indicates the number of nonzero constraint coefficients in the quadratic part of the constraint. This specifies the length of the arrays quadrow, quadcol and quadval.

rhs
The right-hand side term for the constraint to be added.

sense
The sense of the constraint to be added. Note that quadratic constraints may only be less than or equal or greater than or equal constraints. See the discussion of QCP in the ILOG CPLEX User's Manual.

linind
An array that with linval defines the linear part of the quadratic constraint to be added.

linval
An array that with linind defines the linear part of the constraint to be added. The nonzero coefficients of the linear terms must be stored in sequential locations in the arrays linind and linval from positions 0 to linnzcnt-1. Each entry, linind[i], indicates the variable index of the corresponding coefficient, linval[i]. May be NULL; then the constraint will have no linear terms.

quadrow
An array that with quadcol and quadval defines the quadratic part of the quadratic constraint to be added.

quadcol
An array that with quadrow and quadval defines the quadratic part of the quadratic constraint to be added.

quadval
An array that with quadrow and quadcol define the quadratic part of the constraint to be added. The nonzero coefficients of the quadratic terms must be stored in sequential locations in the arrays quadrow, quadcol and quadval from positions 0 to quadnzcnt-1. Each pair, quadrow[i], quadcol[i], indicates the variable indices of the quadratic term, and quadval[i] the corresponding coefficient.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXaddrows

Category          Global Function
Definition File   cplex.h
Include Files     cplex.h

Synopsis          int CPXPUBLIC CPXaddrows(CPXENVptr env,
                         int ccnt,                        
                         int rcnt,                       
                         int nzcnt,                      
                         const double * rhs,           
                         const char * sense,           
                         const int * rmatbeg,          
                         const int * rmatind,          
                         const double * rmatval,       
                         char ** colname,              
                         char ** rowname)

Description       The routine CPXaddrows adds constraints to a specified CPLEX problem object. This routine may be called any time after a call to CPXcreateprob.

When you add a ranged row, CPXaddrows sets the corresponding range value to 0 (zero). Use the routine CPXchgrngval to change the range value.

Values of sense

<table>
<thead>
<tr>
<th>sense[i]</th>
<th>= 'L'</th>
<th>≤ constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>sense[i]</td>
<td>= 'E'</td>
<td>= constraint</td>
</tr>
<tr>
<td>sense[i]</td>
<td>= 'G'</td>
<td>≥ constraint</td>
</tr>
<tr>
<td>sense[i]</td>
<td>= 'R'</td>
<td>ranged constraint</td>
</tr>
</tbody>
</table>

Note: The use of CPXaddrows as a way to add new columns is discouraged in favor of a direct call to CPXnewcols before calling CPXaddrows.

Example

status = CPXaddrows (env, lp, ccnt, rcnt, nzcnt, rhs,
                     sense, rmatbeg, rmatind, rmatval,
newcolname, newrowname);

See also the example lpex3.c in the *ILOG CPLEX User's Manual* and in the standard distribution.

### Parameters

- **env**
  
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- **lp**
  
  A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

- **ccnt**
  
  An integer that indicates the number of new columns in the constraints being added to the constraint matrix. When new columns are added, they are given an objective coefficient of zero, a lower bound of zero, and an upper bound of `CPX_INFBOUND`.

- **rcnt**
  
  An integer that indicates the number of new rows to be added to the constraint matrix.

- **nzcnt**
  
  An integer that indicates the number of nonzero constraint coefficients to be added to the constraint matrix. This specifies the length of the arrays `rmatind` and `rmatval`.

- **rhs**
  
  An array of length `rcnt` containing the right-hand side term for each constraint to be added to the CPLEX problem object. May be NULL, in which case the new right-hand side values are set to 0.0.

- **sense**
  
  An array of length `rcnt` containing the sense of each constraint to be added to the CPLEX problem object. May be NULL, in which case the new constraints are created as equality constraints. Possible values of this argument appear in the table.

- **rmatbeg**
  
  An array used with `rmatind` and `rmatval` to define the rows to be added.

- **rmatind**
  
  An array used with `rmatbeg` and `rmatval` to define the rows to be added.

- **rmatval**
  
  An array used with `rmatbeg` and `rmatind` to define the rows to be added. The format is similar to the format used to describe the constraint matrix in the routine `CPXcopylp` (see description of `matbeg`, `matcnt`, `matind`, and `matval` in that routine), but the nonzero coefficients are grouped by row instead of column in the array `rmatval`. The
nonzero elements of every row must be stored in sequential locations in this array from position \( \text{rmatbeg}[i] \) to \( \text{rmatbeg}[i+1]-1 \) (or from \( \text{rmatbeg}[i] \) to \( \text{nzcnt} -1 \) if \( i=\text{rcnt}-1 \)). Each entry, \( \text{rmatind}[i] \), indicates the column index of the corresponding coefficient, \( \text{rmatval}[i] \). Unlike \texttt{CPXcopylp}, all rows must be contiguous, and \( \text{rmatbeg}[0] \) must be 0.

\texttt{colname}

An array of length \( \text{ccnt} \) containing pointers to character strings that represent the names of the new columns added to the CPLEX problem object, or equivalently, the new variable names. May be NULL, in which case the new columns are assigned default names if the columns already resident in the CPLEX problem object have names; otherwise, no names are associated with the variables. If column names are passed to \texttt{CPXaddrows} but existing variables have no names assigned, default names are created for them.

\texttt{rowname}

An array containing pointers to character strings that represent the names of the new rows, or equivalently, the constraint names. May be NULL, in which case the new rows are assigned default names if the rows already resident in the CPLEX problem object have names; otherwise, no names are associated with the constraints. If row names are passed to \texttt{CPXaddrows} but existing constraints have no names assigned, default names are created for them.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXaddsos

Category  
Global Function

Definition File  
cplex.h

Include Files  
cplex.h

Synopsis  
int CPXPUBLIC CPXaddsos(CPXENVptr env,  
CPXLPptr lp,  
int numsos,  
int numsosnz,  
const char * sostype,  
const int * sospri,  
const int * sosbeg,  
const int * sosind,  
const double * soswt)

Description  
The routine CPXaddsos adds information about a Special Ordered Set (SOS) to a problem object of type CPXPROB_MILP, CPXPROB_MIQP, or CPXPROB_MIQCP. The problem may already contain SOS information.

<table>
<thead>
<tr>
<th>sostype</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>'1'</td>
<td>Type 1</td>
</tr>
<tr>
<td>'2'</td>
<td>Type 2</td>
</tr>
</tbody>
</table>

Example

status = CPXaddsos (env, lp, numsos, numsosnz, sostype, sospri,  
sosbeg, sosind, soswt);

Parameters

env

A pointer to the CPLEX environment as returned by CPXopenCPLEX.
A pointer to a CPLEX problem object as returned by CPXcreateprob.

The number of sets to be added to existing SOS sets, if any.

The total number of members in all of the sets to be added to existing SOS sets, if any.

An array containing SOS type information for the sets to be added. According to Table 1, sostype[i] specifies the SOS type of set i. The length of this array must be at least numsos.

An array containing priority values for each set to be added. sospri[i] specifies the priority for set i, and may take any nonnegative value. This array may be NULL; otherwise, its length must be at least numsos.

An array that with sosind and soswt defines the weights for the sets to be added.

An array that with sosbeg and soswt defines the weights of the sets to be added.

An array that with sosbeg and sosind defines the indices and weights for the sets to be added. The indices of each set must be stored in sequential locations in sosind. The weights of each set must be stored in sequential locations in soswt. The array sosbeg[j] containing the index of the beginning of set j. The weights must be unique within each set.

The routine returns zero on success and nonzero if an error occurs.
CPXaddusercuts

Category                Global Function
Definition File         cplex.h
Include Files           cplex.h

Synopsis

int CPXPUBLIC CPXaddusercuts (CPXCENVptr env,
    CPXLPptr lp,
    int rcnt,
    int nzcnt,
    const double * rhs,
    const char * sense,
    const int * rmatbeg,
    const int * rmatind,
    const double * rmatval);

Description

The routine CPXaddusercuts is used to add constraints to the list of constraints that should be added to the LP subproblem of a MIP optimization if they are violated. CPLEX handles addition of the constraints and makes sure that all integer solutions satisfy all the constraints. The constraints are added to those specified in prior calls to CPXaddusercuts.

The constraints must be cuts, which are implied by the constraint matrix. The CPLEX parameter CPX_PARAM_PRELIM should be set to CPX_OFF (0).

Use CPXfreeusercuts to clear the list of cuts.

The arguments of CPXaddusercuts are the same as those of CPXaddrows, with the exception that new columns may not be specified, so there are no ccnt and colname arguments, and row names may not be specified, so there is no rowname argument. Furthermore, unlike CPXaddrows, CPXaddusercuts does not accept a NULL pointer for the array of right-hand side values.

Example

status = CPXaddusercuts (env, lp, cutcnt,
    cutnzcnt, cutrhs,
    cutsense, cutbeg,
    cutind, cutval);

See also admipex4.c in the standard distribution.
Parameters

**env**
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

**lp**
A pointer to a CPLEX problem object as returned by CPXcreateprob.

**rcnt**
An integer that indicates the number of new rows to be added to the constraint matrix.

**nzcnt**
An integer that indicates the number of nonzero constraint coefficients to be added to the constraint matrix. This specifies the length of the arrays `rmatind` and `rmatval`.

**rhs**
An array of length `rcnt` containing the right-hand side term for each constraint to be added to the CPLEX problem object.

**sense**
An array of length `rcnt` containing the sense of each constraint to be added to the CPLEX problem object. May be NULL, in which case the new constraints are created as equality constraints. Possible values of this argument appear in the table.

**rmatbeg**
An array used with `rmatind` and `rmatval` to define the rows to be added.

**rmatind**
An array used with `rmatbeg` and `rmatval` to define the rows to be added.

**rmatval**
An array used with `rmatbeg` and `rmatind` to define the rows to be added. The format is similar to the format used to describe the constraint matrix in the routine CPXcopylp (see description of `matbeg`, `matcnt`, `matind`, and `matval` in that routine), but the nonzero coefficients are grouped by row instead of column in the array `rmatval`. The nonzero elements of every row must be stored in sequential locations in this array from position `rmatbeg[i]` to `rmatbeg[i+1]-1` (or from `rmatbeg[i]` to `nzcnt - 1` if `i=rcnt-1`). Each entry, `rmatind[i]`, indicates the column index of the

### Values of sense

<table>
<thead>
<tr>
<th>sense[i]</th>
<th>Sense</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘L’</td>
<td>≤ constraint</td>
</tr>
<tr>
<td>‘E’</td>
<td>= constraint</td>
</tr>
<tr>
<td>‘G’</td>
<td>≥ constraint</td>
</tr>
<tr>
<td>‘R’</td>
<td>ranged constraint</td>
</tr>
</tbody>
</table>


corresponding coefficient, \texttt{rmatval[i]}. Unlike \texttt{CPXcopylp}, all rows must be contiguous, and \texttt{rmatbeg[0]} must be 0.

\textbf{Returns} 

The routine returns zero on success and nonzero if an error occurs.
CPXbaropt

Category   Global Function
Definition File   cplex.h
Include Files   cplex.h
Synopsis

    int CPXPUBLIC CPXbaropt (CPXCENVptr env,
                             CPXLPptr lp)

Description

The routine CPXbaropt may be used to find a solution to a linear program (LP),
quadratic program (QP), or quadratically constrained program (QCP) using the barrier
algorithm, at any time after the problem is created via a call to CPXcreateprob. The
optimization results are recorded in the CPLEX problem object.

Example

    status = CPXbaropt (env, lp);

Parameters

    env
    A pointer to the CPLEX environment as returned by CPXopenCPLEX.

    lp
    A pointer to a CPLEX problem object as returned by CPXcreateprob.

Returns

The routine returns zero unless an error occurred during the optimization. Examples of
errors include exhausting available memory (CPXERR_NO_MEMORY) or
encountering invalid data in the CPLEX problem object (CPXERR_NO_PROBLEM).
Exceeding a user-specified CPLEX limit, or proving the model infeasible or unbounded,
are not considered errors. Note that a zero return value does not necessarily mean that a
solution exists. Use query routines CPXsolninfo, CPXgetstat, and CPXsolution to obtain
further information about the status of the optimization.
CPXboundsa

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXboundsa(CPXENVptr env, CPXCLPptr lp, int begin, int end, double * lblower, double * lbupper, double * ublower, double * ubupper);
```

Description:

The routine CPXboundsa is used to access ranges for lower and/or upper bound values. The beginning and end of the range must be specified. Information for variable j, where begin ≤ j ≤ end, is returned in position (j-begin) of the arrays lblower, lbupper, ublower, and ubupper.

**Note:** If only lower bound ranges are desired, then both lblower and lbupper should be non-NULL, and ublower and ubupper can be NULL.

Example:

```c
status = CPXboundsa (env, lp, 0, CPXgetnumcols(env,lp)-1, lblower, lbupper, ublower, ubupper);
```

Parameters:

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCplex.

- **lp**
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

- **begin**
  An integer indicating the beginning of the range of ranges to be returned.

- **end**
  An integer indicating the end of the range of ranges to be returned.
```c
status = CPXboundsa (env, lp, 0, CPXgetnumcols(env,lp)-1,
                     lblower, lbupper, ublower, ubupper);
```

### Returns

The routine returns zero on success and nonzero if an error occurs. This routine fails if no basis exists.
CPXbranchcallbackbranchbds

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXbranchcallbackbranchbds (CPXCENVptr env,
    void * cbdata,
    int wherefrom,
    double nodeest,
    int cnt,
    const int * indices,
    const char * lu,
    const int * bd,
    void * userhandle,
    int * seqnum_p)
```

Description:

Note: This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXbranchcallbackbranchbds specifies the branches to be taken from the current node. It may be called only from within a user-written branch callback function.

Branch variables are in terms of the original problem if the parameter CPX_PARAM_MIPCBREDLP is set to CPX_OFF before the call to CPXmipopt that calls the callback. Otherwise, branch variables are in terms of the presolved problem.

Parameters:

- **env**
  A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

- **cbdata**
  A pointer passed to the user-written callback. This argument must be the value of cbdata passed to the user-written callback.

- **wherefrom**
  An integer value that indicates where the user-written callback was called from. This argument must be the value of wherefrom passed to the user-written callback.
nodeest
A double that indicates the value of the node estimate for the node to be created with this
branch. The node estimate is used to select nodes from the branch & cut tree with certain
values of the NodeSel parameter.

cnt
An integer. The integer indicates the number of bound changes that are specified in the
arrays indices, lu, and bd.

indices
An array. Together with lu and bd, this array defines the bound changes for the branch.
The entry indices[i] is the index for the variable.

lu
An array. Together with indices and bd, this array defines the bound changes for each
of the created nodes. The entry lu[i] is one of the three possible values indicating
which bound to change: L for lower bound, U for upper bound, or B for both bounds.

bd
An array. Together with indices and lu, this array defines the bound changes for each
of the created nodes. The entry bd[i] indicates the new value of the bound.

userhandle
A pointer to user private data that should be associated with the node created by this
branch. May be NULL.

seqnum_p
A pointer to an integer. On return, that integer will contain the sequence number that
CPLEX has assigned to the node created from this branch. The sequence number may be
used to select this node in later calls to the node callback.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXbranchcallbackbranchconstraints

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:
```c
int CPXPUBLIC CPXbranchcallbackbranchconstraints(CPXCENVptr env,
    void * cbdata,
    int wherefrom,
    double nodeest,
    int rcnt,
    int nzcnt,
    const double * rhs,
    const char * sense,
    const int * rmatbeg,
    const int * rmatind,
    const double * rmatval,
    void * userhandle,
    int * seqnum_p)
```

Description:

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXbranchcallbackbranchconstraints` specifies the branches to be taken from the current node when the branch is specified by adding one or more constraints to the node problem. It may be called only from within a user-written branch callback function.

Constraints are in terms of the original problem if the parameter `CPX_PARAM_MIPCBREDLP` is set to `CPX_OFF` before the call to `CPXmipopt` that calls the callback. Otherwise, constraints are in terms of the presolved problem.

**Table 1: Values of sense[i]**

<table>
<thead>
<tr>
<th>L</th>
<th>less than or equal to constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>equal to constraint</td>
</tr>
<tr>
<td>G</td>
<td>greater than or equal to constraint</td>
</tr>
</tbody>
</table>
Parameters

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbdata
A pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.

wherefrom
An integer value that indicates where the user-written callback was called from. This parameter must be the value of wherefrom passed to the user-written callback.

nodeest
A double that indicates the value of the node estimate for the node to be created with this branch. The node estimate is used to select nodes from the branch & cut tree with certain values of the NodeSel parameter.

cpynt
An integer that indicates the number of constraints for the branch.

nzcnt
An integer that indicates the number of nonzero constraint coefficients for the branch. This specifies the length of the arrays rmatind and rmatval.

rhs
An array of length rcnt containing the right-hand side term for each constraint for the branch.

sense
An array of length rcnt containing the sense of each constraint to be added for the branch. Values of the sense appear in Table 1.

rmatbeg
An array that with rmatind and rmatval defines the constraints for the branch.

rmatind
An array that with rmatbeg and rmatval defines the constraints for the branch.

rmatval
An array that with rmatbeg and rmatind defines the constraints for the branch. The format is similar to the format used to describe the constraint matrix in the routine CPXaddrows. Every row must be stored in sequential locations in this array from position rmatbeg[i] to rmatbeg[i+1]-1 (or from rmatbeg[i] to nzcnt -1 if i=rcnt-1). Each entry, rmatind[i], indicates the column index of the corresponding coefficient, rmatval[i]. All rows must be contiguous, and rmatbeg[0] must be 0.
userhandle

A pointer to user private data that should be associated with the node created by this branch. May be NULL.

seqnum_p

A pointer to an integer that, on return, will contain the sequence number that CPLEX has assigned to the node created from this branch. The sequence number may be used to select this node in later calls to the node callback.

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXbranchcallbackbranchgeneral

Category       Global Function
Definition File cplex.h
Include Files  cplex.h

Synopsis
int CPXPUBLIC CPXbranchcallbackbranchgeneral(CPXENVptr env,
    void * cbdata,
    int wherefrom,
    double nodeest,
    int varcnt,
    const int * varind,
    const char * varlu,
    const int * varbd,
    int rcnt,
    int nzcnt,
    const double * rhs,
    const char * sense,
    const int * rmatbeg,
    const int * rmatind,
    const double * rmatval,
    void * userhandle,
    int * seqnum_p)

Description

Note: This is an advanced routine. Advanced routines typically demand a profound
understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher
risk of incorrect behavior in your application, behavior that can be difficult to
dbg. Therefore, ILOG encourages you to consider carefully whether you can
accomplish the same task by means of other Callable Library routines instead.

The routine CPXbranchcallbackbranchgeneral specifies the branches to be
taken from the current node when the branch includes variable bound changes and
additional constraints. It may be called only from within a user-written branch callback
function.

Branch variables are in terms of the original problem if the parameter
CPX_PARAM_MIPCBREDLP is set to CPX_OFF before the call to CPXmipopt that
calls the callback. Otherwise, branch variables are in terms of the presolved problem.
Parameters

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbdata
A pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.

wherefrom
An integer value that indicates where the user-written callback was called from. This parameter must be the value of wherefrom passed to the user-written callback.

nodeest
A double that indicates the value of the node estimate for the node to be created with this branch. The node estimate is used to select nodes from the branch & cut tree with certain values of the NodeSel parameter.

varcnt
An integer that indicates the number of bound changes that are specified in the arrays varind, varlu, and varbd.

varind
Together with varlu and varbd, this array defines the bound changes for the branch. The entry varind[i] is the index for the variable.

varlu
Together with varind and varbd, this array defines the bound changes for the branch. The entry varlu[i] is one of three possible values indicating which bound to change. Those values appear in Table 1.

varbd

### Table 1: Values of varlu[i]

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>change the lower bound</td>
</tr>
<tr>
<td>U</td>
<td>change the upper bound</td>
</tr>
<tr>
<td>B</td>
<td>change both upper and lower bounds</td>
</tr>
</tbody>
</table>

### Table 2: Values of sense[i]

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>less than or equal to constraint</td>
</tr>
<tr>
<td>E</td>
<td>equal to constraint</td>
</tr>
<tr>
<td>G</td>
<td>greater than or equal to constraint</td>
</tr>
</tbody>
</table>
Together with `varind` and `varlu`, this array defines the bound changes for the branch. The entry `varbd[i]` indicates the new value of the bound.

- `rcnt`
  An integer that indicates the number of constraints for the branch.

- `nzcnt`
  An integer that indicates the number of nonzero constraint coefficients for the branch. This specifies the length of the arrays `rmatind` and `rmatval`.

- `rhs`
  An array of length `rcnt` containing the right-hand side term for each constraint for the branch.

- `sense`
  An array of length `rcnt` containing the sense of each constraint to be added for the branch. Possible values appear in Table 2.

- `rmatbeg`
  An array that with `rmatbeg` and `rmatind` defines the constraints for the branch.

- `rmatind`
  An array that with `rmatbeg` and `rmatind` defines the constraints for the branch.

- `rmatval`
  An array that with `rmatbeg` and `rmatind` defines the constraints for the branch. The format is similar to the format used to describe the constraint matrix in the routine `CPXaddrows`. Every row must be stored in sequential locations in this array from position `rmatbeg[i]` to `rmatbeg[i+1]-1` (or from `rmatbeg[i]` to `nzcnt -1` if `i=rcnt-1`). Each entry, `rmatind[i]`, indicates the column index of the corresponding coefficient, `rmatval[i]`. All rows must be contiguous, and `rmatbeg[0]` must be 0.

- `userhandle`
  A pointer to user private data that should be associated with the node created by this branch. May be NULL.

- `seqnum_p`
  A pointer to an integer that, on return, will contain the sequence number that CPLEX has assigned to the node created from this branch. The sequence number may be used to select this node in later calls to the node callback.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXcheckaddcols

Category        Global Function

Definition File  cplex.h

Include Files   cplex.h

Synopsis        int CPXPUBLIC CPXcheckaddcols(CPXENVptr env,
                           CPXCLPptr lp,
                           int ccnt,
                           int nzcnt,
                           const double * obj,
                           const int * cmatbeg,
                           const int * cmatind,
                           const double * cmatval,
                           const double * lb,
                           const double * ub,
                           char ** colname)

Description     The routine CPXcheckaddcols validates the arguments of the corresponding CPXaddcols routine. This data checking routine is found in source format in the file check.c which is provided with the standard CPLEX distribution. To call this routine, you must compile and link check.c with your program as well as the CPLEX Callable Library.

The CPXcheckaddcols routine has the same argument list as the CPXaddcols routine. The second argument, lp, is technically a pointer to a constant LP object of type CPXCLPptr rather than type CPXLPptr, as this routine will not modify the model. For most user applications, this distinction is unimportant.

Example

    status = CPXcheckaddcols (env, lp, ccnt, nzcnt, obj, cmatbeg,
                               cmatind, cmatval, lb, ub, newcolname);

Returns         The routine returns nonzero if it detects an error in the data; it returns zero if it does not detect any data errors.
CPXcheckaddrows

Category  Global Function
Definition File  cplex.h
Include Files  cplex.h
Synopsis  

int CPXPUBLIC CPXcheckaddrows (CPXCENVptr env,
   CPXCLPptr lp,
   int ccnt,
   int rcnt,
   int nzcnt,
   const double * rhs,
   const char * sense,
   const int * rmatbeg,
   const int * rmatind,
   const double * rmatval,
   char ** colname,
   char ** rowname)

Description  The routine CPXcheckaddrows validates the arguments of the corresponding CPXaddrows routine. This data checking routine is found in source format in the file check.c which is provided with the standard CPLEX distribution. To call this routine, you must compile and link check.c with your program as well as the CPLEX Callable Library.

The CPXcheckaddrows routine has the same argument list as the CPXaddrows routine. The second argument, lp, is technically a pointer to a constant LP object of type CPXCLPptr rather than type CPXLPptr, as this routine will not modify the model. For most user applications, this distinction is unimportant.

Example  

status = CPXcheckaddrows (env, lp, ccnt, rcnt, nzcnt, rhs,
   sense, rmatbeg, rmatind, rmatval,
   newcolname, newrowname);

Returns  The routine returns nonzero if it detects an error in the data; it returns zero if it does not detect any data errors.
CPXcheckchgcoeflist

Category          Global Function
Definition File    cplex.h
Include Files     cplex.h
Synopsis          


description The routine CPXcheckchgcoeflist validates the arguments of the corresponding
CPXchgcoeflist routine. This data checking routine is found in source format in the
file check.c which is provided with the standard CPLEX distribution. To call this
routine, you must compile and link check.c with your program as well as the CPLEX
Callable Library.

The CPXcheckchgcoeflist routine has the same argument list as the
CPXchgcoeflist routine. The second argument, lp, is technically a pointer to a
constant LP object of type CPXCLPptr rather than type CPXLPptr, as this routine
will not modify the model. For most user applications, this distinction is unimportant.

Example

status = CPXcheckchgcoeflist (env, lp, numcoefs, rowlist,
collist, vallist);

Parameters

env

A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp

A pointer to a CPLEX problem object as returned by CPXcreateprob.

numcoefs

The number of coefficients to check, or, equivalently, the length of the arrays rowlist,
collist, and vallist.

rowlist

An array of length numcoefs that with collist and vallist indicates the
coefficients to check.

collist

vallist

An array of length numcoefs that with collist and rowlist indicates the
coefficients to check.

An array of length numcoefs that with rowlist and vallist indicates the coefficients to check.

vallist

An array of length numcoefs that with rowlist and collist indicates the coefficients to change. The entries rowlist[k], collist[k], and vallist[k] indicate that the matrix coefficient in row rowlist[k] and column collist[k] should be checked with respect to the value vallist[k].

**Returns**

The routine returns nonzero if it detects an error in the data; it returns zero if it does not detect any data errors.
CPXcheckcopyctype

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXcheckcopyctype(CPXENVptr env, 
    CPXCLPptr lp, 
    const char * xtype)
```

**Description**  
The routine CPXcheckcopyctype validates the arguments of the corresponding CPXcopyctype routine. This data checking routine is found in source format in the file check.c which is provided with the standard CPLEX distribution. To call this routine, you must compile and link check.c with your program as well as the CPLEX Callable Library.

The CPXcheckcopyctype routine has the same argument list as the CPXcopyctype routine. The second argument, lp, is technically a pointer to a constant LP object of type CPXCLPptr rather than type CPXLPptr, as this routine will not modify the model. For most user applications, this distinction is unimportant.

**Example**
```c
status = CPXcheckcopyctype (env, lp, xtype);
```

**Returns**  
The routine returns nonzero if it detects an error in the data; it returns zero if it does not detect any data errors.
**CPXcheckcopylp**

**Category**  Global Function

**Definition File**  cplex.h

**Include Files**  cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXcheckcopylp(CPXENVptr env,
   CPXCLPptr lp,
   int numcols,
   int numrows,
   int objsen,
   const double * obj,
   const double * rhs,
   const char * sense,
   const int * matbeg,
   const int * matcnt,
   const int * matind,
   const double * matval,
   const double * lb,
   const double * ub,
   const double * rngval)
```

**Description**

The routine CPXcheckcopylp validates the arguments of the corresponding CPXcopylp routine. This data checking routine is found in source format in the file check.c which is provided with the standard CPLEX distribution. To call this routine, you must compile and link check.c with your program as well as the CPLEX Callable Library.

The CPXcheckcopylp routine has the same argument list as the CPXcopylp routine. The second argument, lp, is technically a pointer to a constant LP object of type CPXCLPptr rather than type CPXPtr, as this routine will not modify the model. For most user applications, this distinction is unimportant.

**Example**

```c
status = CPXcheckcopylp (env, lp, numcols, numrows, objsen, obj,
   rhs, sense, matbeg, matcnt, matind,
   matval, lb, ub, rngval);
```

**Returns**

The routine returns nonzero if it detects an error in the data; it returns zero if it does not detect any data errors.
CPXcheckcopylpwnames

Category       Global Function
Definition File cplex.h
Include Files  cplex.h

Synopsis

```
int CPXPUBLIC CPXcheckcopylpwnames (CPXCENVptr env,
    CPXCLPptr lp,
    int numcols,
    int numrows,
    int objsen,
    const double * obj,
    const double * rhs,
    const char * sense,
    const int * matbeg,
    const int * matcnt,
    const int * matind,
    const double * matval,
    const double * lb,
    const double * ub,
    const double * rngval,
    char ** colname,
    char ** rowname)
```

Description

The routine CPXcheckcopylpwnames validates the arguments of the corresponding CPXcopylpwnames routine. This data checking routine is found in source format in the file check.c which is provided with the standard CPLEX distribution. To call this routine, you must compile and link check.c with your application as well as the CPLEX Callable Library.

The routine CPXcheckcopylpwnames has the same argument list as the routine CPXcopylpwnames. The second argument, lp, is technically a pointer to a constant LP object of type CPXCLPptr rather than type CPXLPtr, as this routine will not modify the model. For most user applications, this distinction is unimportant.

Example

```
status = CPXcheckcopylpwnames (env,
    lp,
    numcols,
    numrows,
    objsen,
    obj,
    rhs,
    sense,
    matbeg,
    matcnt,
    ...
```

CPXcheckCOPYlpwnames

matind, matval, lb, ub, rngval, colname, cowname);

**Returns**

The routine returns nonzero if it detects an error in the data; it returns zero if it does not detect any data errors.
**CPXcheckcopyqpsep**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXcheckcopyqpsep(CPXENVptr env,
                        CPXCLPptr lp,
                        const double * qsepvec)
```

**Description**  
The routine `CPXcheckcopyqpsep` validates the argument of the corresponding `CPXcopyqpsep` routine. This data checking routine is found in source format in the file `check.c` which is provided with the standard CPLEX distribution. To call this routine, you must compile and link `check.c` with your program as well as the CPLEX Callable Library.

The `CPXcheckcopyqpsep` routine has the same argument list as the `CPXcopyqpsep` routine. The second argument, `lp`, is technically a pointer to a constant LP object of type `CPXCLPptr` rather than type `CPXLPptr`, as this routine will not modify the model. For most user applications, this distinction is unimportant.

**Example**  
```c
status = CPXcheckcopyqpsep (env, lp, qsepvec);
```

**Returns**  
The routine returns nonzero if it detects an error in the data; it returns zero if it does not detect any data errors.
CPXcheckcopyquad

Category          Global Function
Definition File   cplex.h
Include Files     cplex.h
Synopsis          int CPXPUBLIC CPXcheckcopyquad (CPXCENVptr env,
                        CPXCLPptr lp,
                        const int * qmatbeg,
                        const int * qmatcnt,
                        const int * qmatind,
                        const double * qmatval)

Description       The routine CPXcheckcopyquad validates the arguments of the corresponding
                   CPXcopyquad routine. This data checking routine is found in source format in the file
                   check.c which is provided with the standard CPLEX distribution. To call this routine,
                   you must compile and link check.c with your program as well as the CPLEX
                   Callable Library.

                   The CPXcheckcopyquad routine has the same argument list as the CPXcopyquad
                   routine. The second argument, lp, is technically a pointer to a constant LP object of
                   type CPXCLPptr rather than type CPXLPtr, as this routine will not modify the
                   model. For most user applications, this distinction is unimportant.

Example

                   status = CPXcheckcopyquad (env, lp, qmatbeg, qmatcnt,
                                                qmatind, qmatval);

Returns          The routine returns nonzero if it detects an error in the data; it returns zero if it does not
                   detect any data errors.
CPXcheckcopy sos

Category  Global Function
Definition File  cplex.h
Include Files  cplex.h

Synopsis  

```c
int CPXPUBLIC CPXcheckcopy sos(CPXENVptr env,
   CPXCLPptr lp,
   int num sos,
   int num sos nz,
   const char * sostype,
   const int * sospri,
   const int * sos beg,
   const int * sosind,
   const double * soswt)
```

Description  The routine CPXcheckcopy sos validates the arguments of the corresponding CPXcopy sos routine. This data checking routine is found in source format in the file check.c which is provided with the standard CPLEX distribution. To call this routine, you must compile and link check.c with your program as well as the CPLEX Callable Library.

The CPXcheckcopy sos routine has the same argument list as the CPXcopy sos routine. The second argument, lp, is technically a pointer to a constant LP object of type CPXCLPptr rather than type CPXLPptr, as this routine will not modify the model. For most user applications, this distinction is unimportant.

Example  

```c
status = CPXcheckcopy sos (env, lp, num sos, num sos nz, sostype, sospri,
   sos beg, sosind, soswt);
```

Returns  The routine returns nonzero if it detects an error in the data; it returns zero if it does not detect any data errors.
CPXcheckvals

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:
```c
int CPXPUBLIC CPXcheckvals(CPXENVptr env,
    CPXCLPtr lp,
    int cnt,
    const int * rowind,
    const int * colind,
    const double * values)
```

Description:
The routine CPXcheckvals checks an array of indices and a corresponding array of values for input errors. The routine is useful for validating the arguments of problem modification routines such as CPXchgcoeflist, CPXchgbds, CPXchgobj, and CPXchgrhs. This data checking routine is found in source format in the file check.c which is provided with the standard CPLEX distribution. To call this routine, you must compile and link check.c with your program as well as the CPLEX Callable Library.

Example:
Consider the following call to CPXchgobj:
```c
status = CPXchgobj (env, lp, cnt, indices, values);
```
The parameters to this routine could be checked with a call to CPXcheckvals like this:
```c
status = CPXcheckvals (env, lp, cnt, NULL, indices, values);
```

Parameters:

- **env**: A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **lp**: A pointer to a CPLEX problem object as returned by CPXcreateprob.
- **cnt**: The length of the indices and values arrays to be examined.
- **rowind**: An array containing row indices. May be NULL.
colind
An array containing column indices. May be NULL.

values
An array of values. May be NULL.

Returns
The routine returns zero on success and nonzero if an error occurs.
**CPXchgbds**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXchgbds (CPXCENVptr env,
                           CPXLPptr lp,
                           int cnt,
                           const int * indices,
                           const char * lu,
                           const double * bd)
```

**Description**
The routine CPXchgbds is used to change the upper or lower bounds on a set of variables of a problem. Several bounds can be changed at once, with each bound specified by the index of the variable with which it is associated. The value of a variable can be fixed at one value by setting the upper and lower bounds to the same value.

**Example**

```c
status = CPXchgbds (env, lp, cnt, indices, lu, bd);
```

**Values of lu indicating lower or upper bound in indices[j]**

<table>
<thead>
<tr>
<th>lu[j]</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'U'</td>
<td>bd[j] is an upper bound</td>
</tr>
<tr>
<td>'L'</td>
<td>bd[j] is a lower bound</td>
</tr>
<tr>
<td>'B'</td>
<td>bd[j] is the lower and upper bound</td>
</tr>
</tbody>
</table>

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **lp**
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

- **cnt**
  An integer that indicates the total number of bounds to be changed, and thus specifies the length of the arrays indices, lu, and bd.

- **indices**
  Values of lu indicating lower or upper bound in indices[j]
An array of length cnt containing the numeric indices of the columns corresponding to the variables for which bounds are to be changed.

lu

An array of length cnt containing characters that indicate whether the corresponding entry in the array bd specifies the lower or upper bound on column indices[j]. Possible values appear in the table.

bd

An array of length cnt containing the new values of the upper or lower bounds of the variables present in indices.

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXchgcoef

Category               Global Function
Definition File        cplex.h
Include Files          cplex.h
Synopsis               int CPXPUBLIC CPXchgcoef(CPXENVptr env,
                             CPXLPtr lp,
                             int i,
                             int j,
                             double newvalue)

Description            The routine CPXchgcoef is used to change a single coefficient in the constraint
                        matrix, linear objective coefficients, right-hand side or ranges of a CPLEX problem
                        object. The coefficient is specified using its coordinates in the constraint matrix. When
                        changing matrix coefficients from zero to nonzero values, be sure that the corresponding
                        row and column indices exist in the problem, so that -1 ≤ i < CPXgetnumrows(env, lp) and
                        -2 ≤ j < CPXgetnumcols(env, lp)

Example                status = CPXchgcoef (env, lp, 10, 15, 23.2);

See Also               CPXchgobj, CPXchgrhs, CPXchgrngval

Parameters             env
                        A pointer to the CPLEX environment as returned by CPXopenCPLEX.
                        lp
                        A pointer to a CPLEX problem object as returned by CPXcreateprob.
                        i
                        An integer that indicates the numeric index of the row in which the coefficient is located.
                        The linear objective row is referenced with i = -1.
                        j
                        An integer that indicates the numeric index of the column in which the coefficient is
                        located. The RHS column is referenced with j = -1. The range value column is
                        referenced with j = -2. If j = -2 is specified and row i is not a ranged row, an error
                        status is returned.
                        newvalue
The new value for the coefficient being changed.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXchgcoeflist

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXchgcoeflist (CPXCENVptr env,
  CPXLPptr lp,
  int numcoefs,
  const int * rowlist,
  const int * collist,
  const double * vallist)
```

**Description**

The routine `CPXchgcoeflist` is used to change a list of matrix coefficients of a CPLEX problem object. The list is prepared as a set of triples \((i, j, \text{value})\), where \(i\) is the row index, \(j\) is the column index, and \(\text{value}\) is the new value. The list may be in any order.

**Example**

```c
status = CPXchgcoeflist (env, lp, numcoefs, rowlist, collist, vallist);
```

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- **lp**
  A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

- **numcoefs**
  The number of coefficients to change, or, equivalently, the length of the arrays `rowlist`, `collist`, and `vallist`.

**Note:**
- The corresponding rows and columns must already exist in the CPLEX problem object.
- This routine cannot be used to change objective, right-hand side, range, or bound coefficients.
- Duplicate entries, that is, two triplets with identical \(i\) and \(j\), are not allowed.
rowlist
An array of length numcoefs that with collist and vallist indicates the coefficients to change.

collist
An array of length numcoefs that with rowlist and vallist indicates the coefficients to change.

vallist
An array of length numcoefs that with rowlist and collist indicates the coefficients to change. The entries rowlist[k], collist[k], and vallist[k] indicate that the matrix coefficient in row rowlist[k] and column collist[k] should be changed to the value vallist[k].

Returns
The routine returns zero on success and nonzero if an error occurs.
**CPXchgcolname**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**
```c
int CPXPUBLIC CPXchgcolname(CPXENVptr env,
    CPXLPptr lp,
    int cnt,
    const int * indices,
    char ** newname)
```

**Description**
The routine **CPXchgcolname** changes the names of variables in a CPLEX problem object. If this routine is performed on a problem object with no variable names, default names are created before the change is made.

**See Also**
CPXdelnames

**Parameters**
- **env**  
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **lp**  
  A pointer to a CPLEX problem object as returned by CPXcreateprob.
- **cnt**  
  An integer that indicates the total number of variable names to be changed. Thus cnt specifies the length of the arrays indices and newname.
- **indices**  
  An array of length cnt containing the numeric indices of the variables for which the names are to be changed.
- **newname**  
  An array of length cnt containing the strings of the new variable names for the columns specified in indices.

**Returns**
The routine returns zero on success and nonzero if an error occurs.
### CPXchgctype

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**
```c
int CPXPUBLIC CPXchgctype(CPXENVptr env, CPXLPptr lp, int cnt, const int * indices, const char * xctype)
```

**Description**
The routine `CPXchgctype` is used to change the types of a set of variables of a CPLEX problem object. Several types can be changed at once, with each type specified by the index of the variable with which it is associated.

**Example**
```c
status = CPXchgctype (env, lp, cnt, indices, ctype);
```

**Parameters**
- `env`
  A pointer to the CPLEX environment as returned by the CPXopenCPLEX routine.

**Note:** If a variable is to be changed to binary, a call to `CPXchgbds` should also be made to change the bounds to 0 and 1.

**Table 1: Values of elements of ctype**

<table>
<thead>
<tr>
<th>Type</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_CONTINUOUS</td>
<td>'C'</td>
<td>make column indices[j] continuous</td>
</tr>
<tr>
<td>CPX_BINARY</td>
<td>'B'</td>
<td>make column indices[j] binary</td>
</tr>
<tr>
<td>CPX_INTEGER</td>
<td>'I'</td>
<td>make column indices[j] general integer</td>
</tr>
<tr>
<td>CPX_SEMICONT</td>
<td>'S'</td>
<td>make column indices[j] semi-continuous</td>
</tr>
<tr>
<td>CPX_SEMIINT</td>
<td>'N'</td>
<td>make column indices[j] semi-integer</td>
</tr>
</tbody>
</table>
lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

cnt
An integer that indicates the total number of types to be changed, and thus specifies the length of the arrays indices and ctype.

indices
An array containing the numeric indices of the columns corresponding to the variables the types of which are to be changed.

xctype
An array containing characters that represent the new types for the columns specified in indices. Possible values for ctype[j] appear in Table 1.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXchgname

Category: Global Function
Definition File: cplex.h
Include Files: cplex.h

Synopsis

```c
int CPXPUBLIC CPXchgname(CPXENVptr env, CPXLPptr lp, int key, int ij, const char * newname_str);
```

Description

The routine CPXchgname changes the name of a constraint or the name of a variable in a CPLEX problem object. If this routine is performed on a problem object with no row or column names, default names are created before the change is made.

Example

```c
status = CPXchgname (env, lp, 'c', 10, "name10");
```

Values of key

<table>
<thead>
<tr>
<th>key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'r'</td>
<td>change row name</td>
</tr>
<tr>
<td>'c'</td>
<td>change column name</td>
</tr>
</tbody>
</table>

See Also

CPXdelnames

Parameters

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **lp**
  A pointer to a CPLEX problem object as returned by CPXcreateprob.
- **key**
  A character to indicate whether a row name or a column name should be changed. Possible values appear in the table.
- **ij**
  An integer that indicates the numeric index of the column or row for which the name is to be changed.
newname_str
A pointer to a character string containing the new name.

**Returns**
The routine returns zero on success and nonzero if an error occurs.
CPXchgobj

Category        Global Function
Definition File cplex.h
Include Files   cplex.h
Synopsis

int CPXPUBLIC CPXchgobj (CPXCENVptr env,
                          CPXLPptr lp,
                          int cnt,
                          const int * indices,
                          const double * values)

Description     The routine CPXchgobj is used to change the linear objective coefficients of a set of variables in a CPLEX problem object.

Example

status = CPXchgobj (env, lp, cnt, indices, values);

Parameters

env
   A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
   A pointer to a CPLEX problem object as returned by CPXcreateprob.
cnt
   An integer that indicates the total number of objective coefficients to be changed, and thus specifies the length of the arrays indices and values.
indices
   An array of length cnt containing the numeric indices of the columns corresponding to the variables for which objective coefficients are to be changed.
values
   An array of length cnt containing the new values of the objective coefficients of the variables specified in indices.

Returns     The routine returns zero on success and nonzero if an error occurs.
CPXchgobjsen

Category                  Global Function
Definition File          cplex.h
Include Files            cplex.h
Synopsis                 void CPXPUBLIC CPXchgobjsen(CPXENVptr env,
                         CPXLPptr lp,
                         int maxormin)
Description              The routine CPXchgobjsen is used to change the sense of the optimization for a problem, to maximization or minimization.

Note: For problems with a quadratic objective function, changing the objective sense may make the problem unsolvable. Further changes to the quadratic coefficients may then be required to restore the convexity (concavity) of a minimization (maximization) problem.

Values of maxormin

| CPX_MIN  | (1)     | new sense is minimize |
| CPX_MAX  | (-1)    | new sense is maximize |

Example

CPXchgobjsen (env, lp, CPX_MAX);

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

maxormin
An integer that indicates the new sense of the problem.

Returns
This routine does not return a result.
CPXchgprobname

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```
int CPXPUBLIC CPXchgprobname(CPXENVptr env, 
   CPXLPptr lp, 
   const char * probname_str)
```

Description:

The routine CPXchgprobname is used to change the name of the current problem.

Example:

```
status = CPXchgprobname (env, lp, probname);
```

Parameters:

- env
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- lp
  A pointer to a CPLEX problem object as returned by CPXcreateprob.
- probname_str
  The new name of the problem.

Returns:

The routine returns zero on success and nonzero if an error occurs.
CPXchgprobtype

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXchgprobtype(CPXCENVptr env, CPXLPptr lp, int type)
```

Description:
The routine CPXchgprobtype is used to change the current problem to a related problem. The problem types that can be used appear in the table.

### Table 1: Problem Types

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CPXPROB_LP</td>
<td>Linear program, no ctype or quadratic data stored.</td>
</tr>
<tr>
<td>1</td>
<td>CPXPROB_MILP</td>
<td>Problem with ctype information.</td>
</tr>
<tr>
<td>3</td>
<td>CPXPROB_FIXEDMILP</td>
<td>Problem with ctype information, integer variables fixed.</td>
</tr>
<tr>
<td>5</td>
<td>CPXPROB_QP</td>
<td>Problem with quadratic data stored.</td>
</tr>
<tr>
<td>6</td>
<td>CPXPROB_ZEROEDQP</td>
<td>Problem with quadratic data stored, ignoring Q.</td>
</tr>
<tr>
<td>7</td>
<td>CPXPROB_MIQP</td>
<td>Problem with quadratic data and ctype information.</td>
</tr>
<tr>
<td>8</td>
<td>CPXPROB_FIXEDMIQP</td>
<td>Problem with quadratic data and ctype information, integer variables fixed.</td>
</tr>
<tr>
<td>10</td>
<td>CPXPROB_QCP</td>
<td>Problem with quadratic constraints.</td>
</tr>
<tr>
<td>11</td>
<td>CPXPROB_MIQCP</td>
<td>Problem with quadratic constraints and ctype information.</td>
</tr>
</tbody>
</table>

A mixed integer problem (CPXPROB_MILP or CPXPROB_MIQP) can be changed to a problem (CPXPROB_FIXEDMILP or CPXPROB_FIXEDMIQP) where bounds on integer variables are fixed to the values attained in the integer solution. A mixed integer
A quadratic programming problem (CPXPROB_QP) can be changed to a linear program (CPXPROB_LP), causing any existing quadratic information to be permanently discarded from the problem object. Changing a problem from type CPXPROB_LP to CPXPROB_MILP or from type CPXPROB_QP to CPXPROB_MIQP causes a ctype array to be created such that all variables are considered continuous. Changing a problem from type CPXPROB_MILP to CPXPROB_MIQP causes an empty quadratic matrix to be created such that the objective is quadratic with $Q = 0$. A problem of type CPXPROB_MILP or CPXPROB_MIQP can be solved only by the routine CPXmipopt.

A quadratic programming problem (CPXPROB_QP) can be changed to a linear program (CPXPROB_LP), causing any existing quadratic information to be permanently discarded from the problem object. Changing a problem from type CPXPROB_LP to CPXPROB_QP causes an empty quadratic matrix to be created such that the problem is quadratic with the matrix $Q = 0$.

**Example**

```c
status = CPXchgprobtype (env, lp, CPXPROB_MILP);
```

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **lp**
  A pointer to a CPLEX LP problem object as returned by CPXcreateprob.
- **type**
  An integer indicating the desired problem type. See the previous discussion for possible values.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXchgqpcoef

Category  Global Function
Definition File  cplex.h
Include Files  cplex.h
Synopsis  
```c
int CPXPUBLIC CPXchgqpcoef(CPXCENVptr env,
                          CPXLPptr lp,
                          int i,
                          int j,
                          double newvalue)
```
Description  The routine `CPXchgqpcoef` changes the coefficient in the quadratic objective of a quadratic problem (QP) corresponding to the variable pair \( (i, j) \) to the value `newvalue`. If \( i \) is not equal to \( j \), both \( Q(i, j) \) and \( Q(j, i) \) are changed to `newvalue`.

Example  
```c
status = CPXchgqpcoef (env, lp, 10, 12, 82.5);
```
Parameters  
- `env`: A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
- `lp`: A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
- `i`: An integer that indicates the first variable number (row number in \( Q \)).
- `j`: An integer that indicates the second variable number (column number in \( Q \)).
- `newvalue`: The new coefficient value.
Returns  The routine returns zero on success and nonzero if an error occurs.
**CPXchgrhs**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXchgrhs(CPXCENVptr env,
    CPXLPptr lp,
    int cnt,
    const int * indices,
    const double * values)
```

**Description**  
The routine CPXchgrhs is used to change the right-hand side coefficients of a set of linear constraints in the CPLEX problem object.

**Example**

```c
status = CPXchgrhs (env, lp, cnt, indices, values);
```

**Parameters**

- **env**  
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **lp**  
A pointer to a CPLEX problem object as returned by CPXcreateprob.

- **cnt**  
An integer that indicates the total number of right-hand side coefficients to be changed, and thus specifies the length of the arrays indices and values.

- **indices**  
An array of length cnt containing the numeric indices of the rows corresponding to the linear constraints for which right-hand side coefficients are to be changed.

- **values**  
An array of length cnt containing the new values of the right-hand side coefficients of the linear constraints present in indices.

**Returns**  
The routine returns zero on success and nonzero if an error occurs.
CPXchgrngval

Category       Global Function
Definition File cplex.h
Include Files  cplex.h

Synopsis     int CPXPUBLIC CPXchgrngval (CPXCENVptr env,
               CPXLPptr lp,
               int cnt,
               const int * indices,
               const double * values)

Description  The routine CPXchgrngval is used to change the range coefficients of a set of linear constraints in the CPLEX problem object.

Example       status = CPXchgrngval (env, lp, cnt, indices, values);

Parameters   env
             A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
             A pointer to a CPLEX problem object as returned by CPXcreateprob.
ct
             An integer that indicates the total number of range coefficients to be changed, and thus specifies the length of the arrays indices and values.
indices
             An array of length cnt containing the numeric indices of the rows corresponding to the linear constraints for which range coefficients are to be changed.
values
             An array of length cnt containing the new values of the range coefficients of the linear constraints present in indices.

Returns      The routine returns zero on success and nonzero if an error occurs.
CPXchgrowname

Category           Global Function
Definition File    cplex.h
Include Files      cplex.h
Synopsis
int CPXPUBLIC CPXchgrowname(CPXENVptr env,
               CPXLPptr lp,
               int cnt,
               const int * indices,
               char ** newname)

Description
This routine changes the names of linear constraints in a CPLEX problem object. If this routine is performed on a problem object with no constraint names, default names are created before the change is made.

Example
status = CPXchgrowname (env, lp, cnt, indices, values);

See Also
CPXdelnames

Parameters
env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

cnt
An integer that indicates the total number of linear constraint names to be changed, and thus specifies the length of the arrays indices and newname.

indices
An array of length cnt containing the numeric indices of the linear constraints for which the names are to be changed.

newname
An array of length cnt containing the strings of the new names for the linear constraints specified in indices.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXchgsense

Category       Global Function
Definition File cplex.h
Include Files  cplex.h
Synopsis       int CPXPUBLIC CPXchgsense(CPXENVptr env,
            CPXLPptr lp,
            int cnt,
            const int * indices,
            const char * sense)

Description    The routine CPXchgsense is used to change the sense of a set of linear constraints of
                a CPLEX problem object. When changing the sense of a row to ranged,
                CPXchgsense sets the corresponding range value to 0 (zero). The routine
                CPXchgrngval can then be used to change the range value.
Example         status = CPXchgsense (env, lp, cnt, indices, sense);

Values of sense

| sense[i] | = 'L' | The new sense is ≤ |
| sense[i] | = 'E' | The new sense is = |
| sense[i] | = 'G' | The new sense is ≥ |
| sense[i] | = 'R' | The constraint is ranged |

Parameters      env
                A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
                A pointer to a CPLEX problem object as returned by CPXcreateprob.
cnt
                An integer that indicates the total number of linear constraints to be changed, and thus
                represents the length of the arrays indices and sense.
indices
                An array of length cnt containing the numeric indices of the rows corresponding to
                the linear constraints which are to have their senses changed.
sense

An array of length cnt containing characters that indicate the new sense of the linear constraints specified in indices. Possible values appear in the table.

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXcloneprob

Category: Global Function
Definition File: cplex.h
Include Files: cplex.h

Synopsis:
CPXLPptr CPXPUBLIC CPXcloneprob(CPXENVptr env,
   CPXCLPptr lp,
   int * status_p)

Description:
The routine CPXcloneprob can be used to create a new CPLEX problem object and copy all the problem data from an existing problem object to it. Solution and starting information is not copied.

Example:
copy = CPXcloneprob (env, lp, &status);

Parameters:
env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
A pointer to a CPLEX problem object of which a copy is to be created.
status_p
A pointer to an integer used to return any error code produced by this routine.

Example:
copy = CPXcloneprob (env, lp, &status);

Returns:
If successful, CPXcloneprob returns a pointer that can be passed to other CPLEX routines to identify the problem object that is created. If not successful, a NULL pointer (NULL) is returned, and an error status is returned in the variable *status_p. If the routine is successful, *status_p is zero.
CPXcloseCPLEX

Category Global Function

Definition File cplex.h

Include Files cplex.h

Synopsis int CPXPUBLIC CPXcloseCPLEX(CPXENVptr * env_p)

Description This routine frees all of the data structures associated with CPLEX and releases the license. It should be the last CPLEX routine called in any Callable Library application.

Example

status = CPXcloseCPLEX (&env);

See also lpex1.c in the CPLEX User’s Manual.

Parameters env_p

A pointer to a variable holding the pointer to the CPLEX environment as returned by CPXopenCPLEX.

Returns The routine returns zero on success and nonzero if an error occurs.
**CPXcompletelp**

**Category** | Global Function
---|---

**Definition File** | cplex.h

**Include Files** | cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXcompletelp(CPXENVptr env,
                           CPXLPptr lp)
```

**Description**

The routine `CPXcompletelp` is provided to allow users to handle those rare cases where modification steps need to be closely managed; for example, when careful timings are desired for the individual steps in a user's solution process, or more control of memory allocations for problem modifications is needed.

**Example**

```c
status = CPXcompletelp(env, lp);
```

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
- **lp**
  A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXcopybase

Category: Global Function
Definition File: cplex.h
Include Files: cplex.h
Synopsis:
```c
int CPXPUBLIC CPXcopybase(CPXCENVptr env,
   CPXLPptr lp,
   const int * cstat,
   const int * rstat)
```
Description:
The routine CPXcopybase is used to copy a basis into a CPLEX problem object. It is not necessary to copy a basis prior to optimizing an LP problem, but a good initial basis can increase the speed of optimization significantly. A basis does not need to be primal or dual feasible to be used by the optimizer.

**Note:** The basis is ignored by the optimizer if CPX_PARAM_ADVIND is set to CPX_OFF.

| Table 1: Values of basis status for columns in cstat[j] |
|-----------------|-----------------|
| CPX_AT_LOWER    | 0               | variable at lower bound |
| CPX_BASIC       | 1               | variable is basic |
| CPX_AT_UPPER    | 2               | variable at upper bound |
| CPX_FREE_SUPER  | 3               | variable free and nonbasic |

| Table 2: Values of basis status for rows other than ranged rows in rstat[j] |
|-----------------|-----------------|
| CPX_AT_LOWER    | 0               | associated slack/surplus/artificial variable nonbasic at value 0.0 |
| CPX_BASIC       | 1               | associated slack/surplus/artificial variable basic |
Example

```c
status = CPXcopybase (env, lp, cstat, rstat);
```

See Also

- `CPXreadcopybase`

Parameters

- `env`
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
- `lp`
  A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
- `cstat`
  An array containing the basis status of the columns in the constraint matrix. The length
  of the array is equal to the number of columns in the problem object. Possible values of
  the basis status of columns appear in Table 1.
- `rstat`
  An array containing the basis status of the slack/surplus/artificial variable associated
  with each row in the constraint matrix. The array's length is equal to the number of rows
  in the CPLEX problem object. For rows other than ranged rows, the array element
  `rstat[i]` has the meaning in Table 2. For ranged rows, the array element `rstat[i]`
  has the meaning in Table 3.

Returns

The routine returns zero on success and nonzero if an error occurs.

<table>
<thead>
<tr>
<th>Basis Status</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_AT_LOWER</td>
<td>0</td>
<td>associated slack/surplus/artificial variable nonbasic at its lower bound</td>
</tr>
<tr>
<td>CPX_BASIC</td>
<td>1</td>
<td>associated slack/surplus/artificial variable basic</td>
</tr>
<tr>
<td>CPX_AT_UPPER</td>
<td>2</td>
<td>associated slack/surplus/artificial variable nonbasic at its upper bound</td>
</tr>
</tbody>
</table>

Table 3: Values of basis status for ranged rows in `rstat[j]`
**CPXcopyctype**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**
```c
int CPXPUBLIC CPXcopyctype(CPXENVptr env, 
CPXLPptr lp, 
const char * xctype)
```

**Description**
The routine `CPXcopyctype` can be used to copy variable type information to a given problem. Variable types indicate whether a variable is continuous, integer, binary, semi-continuous, or semi-integer. If the type of the problem object is `CPXPROB_LP`, it is changed to `CPXPROB_MILP`. If the type of the problem object is `CPXPROB_QP`, it is changed to `CPXPROB_MIQP`. If the type of the problem object is `CPXPROB_QCP`, it is changed to `CPXPROB_MIQCP`.

This routine allows the types of all the variables to be set in one function call. When `CPXcopyctype` is called, any current solution information is freed.

**Note:**
Defining a variable \( j \) to be binary by setting the corresponding \( ctype[j] = 'B' \) does not change the bounds associated with that variable. Because a call to `CPXmipopt` fails if the value of the corresponding variable is not between 0 and 1 in the solution of the LP relaxation at the root of the search tree, explicitly setting the bounds of binary variables to 0 and 1 is recommended.

**Table 1: Possible values for elements of xctype**

<table>
<thead>
<tr>
<th>xctype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_CONTINUOUS</td>
<td>'C' continuous variable</td>
</tr>
<tr>
<td>CPX_BINARY</td>
<td>'B' binary variable</td>
</tr>
<tr>
<td>CPX_INTEGER</td>
<td>'I' general integer variable</td>
</tr>
<tr>
<td>CPX_SEMICONT</td>
<td>'S' semi-continuous variable</td>
</tr>
<tr>
<td>CPX_SEMIINT</td>
<td>'N' semi-integer variable</td>
</tr>
</tbody>
</table>

**Example**
```c
status = CPXcopyctype (env, lp, xtype);
```
See also the example `mipex1.c` in the standard distribution.

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- **lp**
  A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

- **xctype**
  An array of length `CPXgetnumcols(env, lp)` containing the type of each column in the constraint matrix. Possible values appear in Table 1.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CSXcopylp

Category          Global Function
Definition File   cplex.h
Include Files     cplex.h

Synopsis          int CPXPUBLIC CPXcopylp(CPXENVptr env,
                   CPXLPptr lp,
                   int numcols,
                   int numrows,
                   int objsen,
                   const double * obj,
                   const double * rhs,
                   const char * sense,
                   const int * matbeg,
                   const int * matcnt,
                   const int * matind,
                   const double * matval,
                   const double * lb,
                   const double * ub,
                   const double * rngval)

Description       The routine CPXcopylp copies data that defines an LP problem to a CPLEX problem object. The arguments to CPXcopylp define an objective function, the constraint matrix, the right-hand side, and the bounds on the variables. The routine CPXcopylp does not copy names. Calling CPXcopylp destroys any existing data associated with the problem object.

The more comprehensive routine CPXcopylpwnames can be used in place of CPXcopylp to copy linear programs with associated names.

The arguments passed to CPXcopylp define a linear program. Since these arguments are copied into local arrays maintained by CPLEX, the LP problem data passed via CPXcopylp may be modified or freed after the call to CPXcopylp without affecting the state of the CPLEX problem object.

Table 1: Values of objsen

<table>
<thead>
<tr>
<th>objsen</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>(CPX_MIN) minimize</td>
</tr>
<tr>
<td>-1</td>
<td></td>
<td>(CPX_MAX) maximize</td>
</tr>
</tbody>
</table>
The arrays matbeg, matcnt, matind, and matval are accessed as follows. Suppose that CPLEX wants to access the entries in some column \( j \). These are assumed to be given by the array entries:

\[
\text{matval[matbeg}[j]\text{,..., matval[matbeg}[j]+matcnt[j]-1]}
\]

The corresponding row indices are:

\[
\text{matind[matbeg}[j]\text{,..., matind[matbeg}[j]+matcnt[j]-1]}
\]

Entries in matind are not required to be in row order. Duplicate entries in matind within a single column are not allowed. The length of the arrays matbeg and matind should be of at least numcols. The length of arrays matind and matval should be of at least matbeg[numcols-1]+matcnt[numcols-1].

**Example**

```c
status = CPXcopylp (env, lp, numcols, numrows, objsen, obj, rhs, 
sense, matbeg, matcnt, matind, matval, lb, 
ub, rngval);
```

See also the example lpex1.c in the *ILOG CPLEX User's Manual* and in the standard distribution.

**Parameters**

- **env**
  
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **lp**
  
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

- **numcols**
  
  An integer that indicates the number of columns in the constraint matrix, or equivalently, the number of variables in the problem object.

---

**Table 2: Values of sense**

<table>
<thead>
<tr>
<th>sense[i]</th>
<th>= ‘L’</th>
<th>≤ constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>sense[i]</td>
<td>= ‘E’</td>
<td>= constraint</td>
</tr>
<tr>
<td>sense[i]</td>
<td>= ‘G’</td>
<td>≥ constraint</td>
</tr>
<tr>
<td>sense[i]</td>
<td>= ‘R’</td>
<td>ranged constraint</td>
</tr>
</tbody>
</table>
numrows
An integer that indicates the number of rows in the constraint matrix, not including the objective function or bounds on the variables.

objs
An integer that indicates whether the problem is a minimization or maximization problem.

objen
An integer that indicates whether the problem is a minimization or maximization problem.

rhs
An array of length at least numrows containing the right-hand side value for each constraint in the constraint matrix.

sense
An array of at least length at least numrows containing the sense of each constraint in the constraint matrix.

matbeg
An array that with matval, matcnt, and matind defines the constraint matrix.

matcnt
An array that with matbeg, matval, and matind defines the constraint matrix.

matind
An array that with matbeg, matcnt, and matval defines the constraint matrix.

matval
An array that with matbeg, matcnt, and matind defines the constraint matrix. CPLEX needs to know only the nonzero coefficients. These are grouped by column in the array matval. The nonzero elements of every column must be stored in sequential locations in this array with matbeg[j] containing the index of the beginning of column j and matcnt[j] containing the number of entries in column j. The components of matbeg must be in ascending order. For each k, matind[k] indicates the row number of the corresponding coefficient, matval[k].

lb
An array of at least length numcols containing the lower bound on each of the variables. Any lower bound that is set to a value less than or equal to that of the constant -CPX_INFBOUND is treated as negative ?. CPX_INFBOUND is defined in the header file cplex.h.
ub
An array of at least length numcols containing the upper bound on each of the variables. Any upper bound that is set to a value greater than or equal to that of the constant CPX_INFBOUND is treated as ?. CPX_INFBOUND is defined in the header file cplex.h.

rngval
An array of at least length numrows containing the range value of each ranged constraint. Ranged rows are those designated by 'R' in the sense array. If the row is not ranged, the rngval array entry is ignored. If rngval[i] > 0, then row i activity is in [rhs[i], rhs[i]+rngval[i]], and if rngval[i] ≤ 0, then row i activity is in [rhs[i]+rngval[i], rhs[i]]. This parameter may be NULL.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXcopylpwnames

Category       Global Function
Definition File cplex.h
Include Files  cplex.h

Synopsis

```c
int CPXPUBLIC CPXcopylpwnames(CPXENVptr env,
    CPXLPptr lp,
    int numcols,
    int numrows,
    int objsen,
    const double * obj,
    const double * rhs,
    const char * sense,
    const int * matbeg,
    const int * matcnt,
    const int * matind,
    const double * matval,
    const double * lb,
    const double * ub,
    const double * rngval,
    char ** colname,
    char ** rowname)
```

Description

The routine CPXcopylpwnames copies LP data into a CPLEX problem object in the same way as the routine CPXcopylp, but using some additional arguments to specify the names of constraints and variables in the CPLEX problem object. The arguments to CPXcopylpwnames define an objective function, constraint matrix, variable bounds, right-hand side constraint senses, and range values. Unlike the routine CPXcopylp, CPXcopylpwnames also copies names. This routine is used in the same way as CPXcopylp.

Table 1: Settings for objsen

<table>
<thead>
<tr>
<th>objsen</th>
<th>setting</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(CPX_MIN) minimize</td>
<td>(CPX_MIN) minimize</td>
</tr>
<tr>
<td>-1</td>
<td>(CPX_MAX) maximize</td>
<td>(CPX_MAX) maximize</td>
</tr>
</tbody>
</table>

Table 2: Settings for sense

<table>
<thead>
<tr>
<th>sense[i]</th>
<th>setting</th>
<th>constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>'L'</td>
<td>≤ constraint</td>
<td>≤ constraint</td>
</tr>
<tr>
<td>'E'</td>
<td>= constraint</td>
<td>= constraint</td>
</tr>
<tr>
<td>'G'</td>
<td>≥ constraint</td>
<td>≥ constraint</td>
</tr>
</tbody>
</table>
With respect to the parameters \texttt{matbeg} (beginning of the matrix), \texttt{matcnt} (count of the matrix), \texttt{matind} (indices of the matrix), and \texttt{matval} (values of the matrix), CPLEX needs to know only the nonzero coefficients. These are grouped by column in the array \texttt{matval}. The nonzero elements of every column must be stored in sequential locations in this array with \texttt{matbeg[j]} containing the index of the beginning of column \( j \) and \texttt{matcnt[j]} containing the number of entries in column \( j \). The components of \texttt{matbeg} must be in ascending order. For each \( k \), \texttt{matind[k]} indicates the row number of the corresponding coefficient, \texttt{matval[k]}.

These arrays are accessed as follows. Suppose that CPLEX wants to access the entries in some column \( j \). These are assumed to be given by the array entries:

\[
\texttt{matval[matbeg[j]]}, \ldots, \texttt{matval[matbeg[j]+matcnt[j]-1]}
\]

The corresponding row indices are:

\[
\texttt{matind[matbeg[j]]}, \ldots, \texttt{matind[matbeg[j]+matcnt[j]-1]}
\]

Entries in \texttt{matind} are not required to be in row order. Duplicate entries in \texttt{matind} and \texttt{matval} within a single column are not allowed. The length of the arrays \texttt{matbeg} and \texttt{matind} should be of at least \texttt{numcols}. The length of arrays \texttt{matind} and \texttt{matval} should be of at least \texttt{matbeg[numcols]-1}+\texttt{matcnt[numcols]-1}.

**Example**

```c
status = CPXcopylpwnames (env,
    lp,
    numcols,
    numrows,
    objsen,
    obj,
    rhs,
    sense,
    matbeg,
    matcnt,
    matind,
    matval,
    lb,
    ub,
    rngval,
    colname,
    rowname);
```

### Table 2: Settings for sense

<table>
<thead>
<tr>
<th>sense[i]</th>
<th>= 'R'</th>
<th>ranged constraint</th>
</tr>
</thead>
</table>

**sense[i]** is an input parameter indicating whether the constraint represented by the \( i \)-th row is a range constraint or not. The value of **sense[i]** is set to **'R'** for range constraints, and any other value for non-range constraints.
Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

numcols
An integer that indicates the number of columns in the constraint matrix, or equivalently, the number of variables in the problem object.

numrows
An integer that indicates the number of rows in the constraint matrix, not including the objective function or bounds on the variables.

objsen
An integer that indicates whether the problem is a minimization or maximization problem. Table 1 shows its possible settings.

objen
An integer that indicates whether the problem is a minimization or maximization problem. Table 1 shows its possible settings.

rhs
An array of length at least numrows containing the right-hand side value for each constraint in the constraint matrix.

sense
An array of at least length at least numrows containing the sense of each constraint in the constraint matrix. Table 2 shows the possible settings.

matbeg
An array that defines the constraint matrix.

matcnt
An array that defines the constraint matrix.

matind
An array that defines the constraint matrix.

matval
An array that defines the constraint matrix.
lb
An array of at least length numcols containing the lower bound on each of the
variables. Any lower bound that is set to a value less than or equal to that of the constant
-CPX_INFBOUND is treated as negative infinity. CPX_INFBOUND is defined in the
header file cplex.h.

ub
An array of at least length numcols containing the upper bound on each of the
variables. Any upper bound that is set to a value greater than or equal to that of the
constant CPX_INFBOUND is treated as infinity. CPX_INFBOUND is defined in the
header file cplex.h.

rngval
An array of at least length numrows containing the range value of each ranged
constraint. Ranged rows are those designated by R in the sense array. If the row is not
ranged, the rngval array entry is ignored. If rngval[i] > 0, then row i activity is in
[rhs[i], rhs[i]+rngval[i]], and if rngval[i] ≤ 0, then row i activity is in
[rhs[i]+rngval[i], rhs[i]]. This parameter may be NULL.

colname
An array of length at least numcols containing pointers to character strings. Each string
is terminated with the NULL character. These strings represent the names of the matrix
columns or, equivalently, the variable names. May be NULL if no names are associated
with the variables. If colname is not NULL, every variable must be given a name. The
addresses in colname do not have to be in ascending order.

rowname
An array of length at least numrows containing pointers to character strings. Each string
is terminated with the NULL character. These strings represent the names of the matrix
rows or, equivalently, the constraint names. May be NULL if no names are associated
with the constraints. If rowname is not NULL, every constraint must be given a name.
The addresses in rowname do not have to be in ascending order.

Returns
zero on success and nonzero if an error occurs.
CPXcopymipstart

Category          Global Function
Definition File   cplex.h
Include Files    cplex.h

Synopsis          int CPXPUBLIC CPXcopymipstart (CPXCENVptr env,
                   CPXLPptr lp,
                   int cnt,
                   const int * indices,
                   const double * value)

Description       The routine CPXcopymipstart is used to copy MIP starting values to a CPLEX
                   problem object of type CPXPROB_MILP, CPXPROB_MIQP, or CPXPROB_MIQCP.
                   Values must be specified for all integer and SOS member variables. Optionally, values may be
                   specified for continuous variables. If the provided starting values are compatible with
                   an integer feasible solution, that solution becomes the incumbent at the start of the
                   optimizations. Otherwise, the starting values are ignored. The values for member
                   variables of Special Ordered Sets are used to indicate which members of the set take
                   nonzero values.
                   The parameter CPX_PARAM_MIPSTART must be set to CPX_ON (1) for the values to
                   be used. If it is set to zero, which is the default, the values are not used.

Example            status = CPXcopymipstart (env, lp, cnt, indices, value);

See Also           CPXreadcopyorder, CPXreadcopymipstart

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

cnt
An integer giving the number of entries in the list.

indices
An array of length cnt containing the numeric indices of the columns corresponding to
the variables which are assigned starting values.
value

An array of length cnt containing the values to be used for the starting integer solution. The entry value[j] is the value assigned to variable indices[j]. An entry value[j] greater than or equal to CPX_INFBOUND indicates no value is set for variable indices[j].

Example

    status = CPXcopymipstart (env, lp, cnt, indices, value);

Returns

The routine returns zero on success and nonzero if an error occurs.
**CPXcopynettolp**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXcopynettolp(CPXENVptr env,
                              CPXLPtr lp,
                              CPXCNETptr net)
```

**Description**  
The routine `CPXcopynettolp` is used to copy a network problem stored in a network problem object to a CPLEX problem object (as an LP). Any problem data previously stored in the CPLEX problem object is overridden.

**Example**  
```c
status = CPXcopynettolp(env, lp, net);
```

**Parameters**  
- `env`  
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
- `lp`  
  A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
- `net`  
  A pointer to a CPLEX network problem object containing the network problem to be copied.

**Example**  
```c
status = CPXcopynettolp(env, lp, net);
```

**Returns**  
The routine returns zero on success and nonzero if an error occurs.
CPXcopyobjname

Category          Global Function
Definition File   cplex.h
Include Files     cplex.h

Synopsis          int CPXPUBLIC CPXcopyobjname(CPXENVptr env,  
                  CPXLPptr lp,  
                  const char * objname_str)

Description       The routine CPXcopyobjname copies a name for the objective function into a CPLEX problem object. An argument to CPXcopyobjname defines the objective name.

Example

        status = CPXcopyobjname (env, ip, "Cost");

Parameters        

        env
        A pointer to the CPLEX environment as returned by CPXopenCPLEX.

        ip
        A pointer to a CPLEX problem object as returned by CPXcreateprob.

        objname_str
        A pointer to a string representing the objective name.

Returns           The routine returns zero on success and nonzero if an error occurs.
**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**

```c
#include <cplex.h>

int CPXPUBLIC CPXcopyorder(CPXENVptr env, CPXLPptr lp, int cnt, const int * indices, const int * priority, const int * direction);
```

**Description**

The routine `CPXcopyorder` is used to copy a priority order to a CPLEX problem object of type `CPXPROB_MILP`, `CPXPROB_MIQP`, or `CPXPROB_MIQCP`. During branching, integer variables with higher priorities are given preference over integer variables with lower priorities. Priorities must be positive integers. A preferred branching direction may also be specified for each variable.

The CPLEX parameter `CPX_PARAM_MIPORDIND` must be set to `CPX_ON`, its default value, for the priority order to be used in a subsequent optimization.

**Example**

```c
status = CPXcopyorder (env, lp, cnt, indices, priority, direction);
```

**See Also**

`CPXreadcopyorder`

**Parameters**

- `env`
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
- `lp`
  A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

**Table 1: Settings for direction**

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_BRANCH_GLOBAL</td>
<td>Use global branching direction when setting the parameter <code>CPX_PARAM_BRDIR</code></td>
</tr>
<tr>
<td>CPX_BRANCH_DOWN</td>
<td>Branch down first on variable <code>indices[i]</code></td>
</tr>
<tr>
<td>CPX_BRANCH_UP</td>
<td>Branch up first on variable <code>indices[i]</code></td>
</tr>
</tbody>
</table>
cnt
An integer giving the number of entries in the list.

indices
An array of length cnt containing the numeric indices of the columns corresponding to the integer variables that are assigned priorities.

priority
An array of length cnt containing the priorities assigned to the integer variables. The entry priority[j] is the priority assigned to variable indices[j]. May be NULL.

direction
An array of type integer containing the branching direction assigned to the integer variables. The entry direction[j] is the direction assigned to variable indices[j]. May be NULL. Possible settings for direction[j] appear in Table 1.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXcopypartialbase

Category: Global Function
Definition File: cplex.h
Include Files: cplex.h

Synopsis:
```c
int CPXPUBLIC CPXcopypartialbase(CPXENVptr env,
                                  CPXLPptr lp,
                                  int ccnt,
                                  const int * cindices,
                                  const int * cstat,
                                  int rcnt,
                                  const int * rindices,
                                  const int * rstat)
```

Description:
The routine `CPXcopypartialbase` is used to copy a partial basis into an LP problem object. Basis statuses do not need to be specified for every variable or slack/surplus/artificial variable. If the status of a variable is not specified, it is made nonbasic at its lower bound if the lower bound is finite; otherwise, it is made nonbasic at its upper bound if the upper bound is finite; otherwise, it is made nonbasic at 0.0. If the status of a slack/surplus/artificial variable is not specified, it is made basic.

<table>
<thead>
<tr>
<th>Table 1: Values of cstat[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_AT_LOWER 0</td>
</tr>
<tr>
<td>CPX_BASIC 1</td>
</tr>
<tr>
<td>CPX_AT_UPPER 2</td>
</tr>
<tr>
<td>CPX_FREE_SUPER 3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2: Status of rows other than ranged rows in rstat[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_AT_LOWER 0 associated slack variable nonbasic at value 0.0</td>
</tr>
<tr>
<td>CPX_BASIC 1 associated slack/surplus/artificial variable basic</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 3: Status of ranged rows in rstat[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_AT_LOWER 0 associated slack variable nonbasic at its lower bound</td>
</tr>
</tbody>
</table>
Example

```c
status = CPXcopypartialbase (env, lp, ccnt, colind, colstat,
                               rcnt, rowind, rowstat);
```

### Parameters

- **env**
  A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

- **lp**
  A pointer to a CPLEX LP problem object, as returned by CPXcreateprob.

- **ccnt**
  An integer that indicates the number of variable/column statuses specified, and is the length of the `cindices` and `cstat` arrays.

- **cindices**
  An array of length `ccnt` that contains the indices of the variables for which statuses are being specified.

- **cstat**
  An array of length `ccnt` where the `ith` entry contains the status for variable `cindices[i]`.

- **rcnt**
  An integer that indicates the number of slack/surplus/artificial statuses specified, and is the length of the `rindices` and `rstat` arrays.

- **rindices**
  An array of length `rcnt` that contains the indices of the slack/surplus/artificials for which statuses are being specified.

- **rstat**
  An array of length `rcnt` where the `ith` entry contains the status for slack/surplus/artificial `rindices[i]`. For rows other than ranged rows, the array element `rstat[i]` has the meaning summarized in Table 2. For ranged rows, the array element `rstat[i]` has the meaning summarized in Table 3.

### Table 3: Status of ranged rows in rstat[i]

<table>
<thead>
<tr>
<th>Status</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_BASIC</td>
<td>1</td>
<td>associated slack variable basic</td>
</tr>
<tr>
<td>CPX_AT_UPPER</td>
<td>2</td>
<td>associated slack variable nonbasic at its upper bound</td>
</tr>
</tbody>
</table>

---

**Example**

```
status = CPXcopypartialbase (env, lp, ccnt, colind, colstat,
                               rcnt, rowind, rowstat);
```
Returns

This routine returns zero on success and nonzero if an error occurs.
**CPXcopyqpsep**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXcopyqpsep(CPXENVptr env,
    CPXLPptr lp,
    const double * qsepvec)
```

**Description**  
The routine CPXcopyqpsep is used to copy the quadratic objective matrix Q for a separable QP problem. A separable QP problem is one where the coefficients of Q have no nonzero off-diagonal elements.

**Note:** CPLEX evaluates the corresponding objective with a factor of 0.5 in front of the quadratic objective term.

**Example**
```c
status = CPXcopyqpsep (env, lp, qsepvec);
```

**Parameters**

- **env**  
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **lp**  
A pointer to a CPLEX problem object as returned by CPXcreateprob.

- **qsepvec**  
An array of length CPXgetnumcols(env,lp).qsepvec[0], qsepvec[1],... qsepvec[numcols-1] should contain the quadratic coefficients of the separable quadratic objective.

**Returns**  
The routine returns zero on success and nonzero if an error occurs.
**CPXcopyquad**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXcopyquad(CPXENVptr env, CPXLPptr lp, const int * qmatbeg, const int * qmatcnt, const int * qmatind, const double * qmatval)
```

**Description**  
The routine CPXcopyquad is used to copy quadratic objective matrix Q when Q is not diagonal. The arguments qmatbeg, qmatcnt, qmatind, and qmatval are used to specify the nonzero coefficients of the matrix Q. The meaning of these vectors is identical to the meaning of the corresponding vectors matbeg, matcnt, matind and matval, which are used to specify the structure of A in a call to CPXcopylp.

Q must be symmetric when copied by this function. Therefore, if the quadratic coefficient in algebraic form is $2x_1x_2$, then $x_2$ should be in the list for $x_1$, and $x_1$ should be in the list for $x_2$, and the coefficient would be 1.0 in each of those entries. See the corresponding example C program to review how the symmetry requirement is implemented.

**Note:** CPLEX evaluates the corresponding objective with a factor of 0.5 in front of the quadratic objective term.

**How the arrays are accessed**

Suppose that CPLEX wants to access the entries in a column $j$. These are assumed to be given by the array entries:

$$qmatval[qmatbeg[j]],...,qmatval[qmatbeg[j]+qmatcnt[j]-1]$$

The corresponding column/index entries are:

$$qmatind[qmatbeg[j]],...,qmatind[qmatbeg[j]+qmatcnt[j]-1]$$
The entries in \texttt{qmatind}[k] are not required to be in column order. Duplicate entries in \texttt{qmatind} within a single column are not allowed. Note that any column \( j \) that has only a linear objective term has \texttt{qmatcnt}[j] = 0 and no entries in \texttt{qmatind} and \texttt{qmatval}.

**Example**

```c
status = CPXcopyquad (env, lp, qmatbeg, qmatcnt, qmatind,
                     qmatval);
```

**See Also**

CPXreadcopyqp

**Parameters**

- \texttt{env}
  - A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- \texttt{lp}
  - A pointer to a CPLEX problem object as returned by CPXcreateprob.
- \texttt{qmatbeg}
  - An array that with \texttt{qmatcnt}, \texttt{qmatind}, and \texttt{qmatval} defines the quadratic coefficient matrix.
- \texttt{qmatcnt}
  - An array that with \texttt{qmatbeg}, \texttt{qmatind}, and \texttt{qmatval} defines the quadratic coefficient matrix.
- \texttt{qmatind}
  - An array that with \texttt{qmatbeg}, \texttt{qmatcnt}, and \texttt{qmatval} defines the quadratic coefficient matrix.
- \texttt{qmatval}
  - An array that with \texttt{qmatbeg}, \texttt{qmatcnt}, and \texttt{qmatind} defines the quadratic coefficient matrix. The arrays \texttt{qmatbeg} and \texttt{qmatcnt} should be of length at least \texttt{CPXgetnumcols(env,lp)}. The arrays \texttt{qmatind} and \texttt{qmatval} should be of length at least \texttt{qmatbeg}[numcols-1]+\texttt{qmatcnt}[numcols-1]. CPLEX requires only the nonzero coefficients grouped by column in the array \texttt{qmatval}. The nonzero elements of every column must be stored in sequential locations in this array with \texttt{qmatbeg}[j] containing the index of the beginning of column \( j \) and \texttt{qmatcnt}[j] containing the number of entries in column \( j \). Note that the components of \texttt{qmatbeg} must be in ascending order. For each \( k \), \texttt{qmatind}[k] indicates the column number of the corresponding coefficient, \texttt{qmatval}[k]. These arrays are accessed as explained above.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
**CPXcopysos**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXcopysos (CPXCENVptr env,  
CPXLPptr lp,  
int numsos,  
int numssosnz,  
const char * sostype,  
const int * sospri,  
const int * sosbeg,  
const int * sosind,  
const double * soswt);
```

**Description**  
The routine `CPXcopysos` is used to copy Special Ordered Set (SOS) information to a problem object of type CPXPROB_MILP, CPXPROB_MIQP, or CPXPROB_MIQCP.

**Table 1: Settings for sostype**

<table>
<thead>
<tr>
<th>CPX_TYPE_SOS1</th>
<th>'1'</th>
<th>Type 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_TYPE_SOS2</td>
<td>'2'</td>
<td>Type 2</td>
</tr>
</tbody>
</table>

**Example**

```c
status = CPXcopysos (env,  
lp,  
numsos,  
umssosnz,  
sostype,  
sospri,  
sosbeg,  
sosind,  
soswt);
```

**See Also**  
`CPXreadcopysos`

**Parameters**

- `env`  
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
- `lp`  
  A pointer to the CPLEX problem object.
A pointer to a CPLEX problem object as returned by CPXcreateprob.

numsos

The number of SOS sets. If numsos is equal to zero, CPXcopysos removes all the SOSs from the LP object.

numsosnz

The total number of members in all sets. CPXcopysos with numsosnz equal to zero removes all the SOSs from the LP object.

sostype

An array containing SOS type information for the sets. sostype[i] specifies the SOS type of set i, according to the settings in Table 1. The length of this array must be at least numsos.

sospri

An array containing priority values for each set. sospri[i] specifies the priority for set i, and may take any nonnegative value. May be NULL; otherwise, its length must be at least numsos.

soswt

Arrays describing the indices and weights for the sets. For every set, the indices and weights must be stored in sequential locations in sosind and soswt, respectively, with sosbeg[j] containing the index of the beginning of set j. The weights must be unique in their array. For j < numsos-1 the indices of set j must be stored in sosind[sosbeg[j]],..., sosind[sosbeg[j+1]-1] and the weights in soswt[sosbeg[j]],..., soswt[sosbeg[j+1]-1]. For the last set, j = numsos-1, the indices must be stored in sosind[sosbeg[numsos-1]],..., sosind[numsosnz-1] and the corresponding weights in soswt[sosbeg[numsos-1]]...soswt[numsosnz-1]. Hence, sosbeg must be of length at least numsos, while sosind and soswt must be of length at least numsosnz.

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXcopystart

Category  Global Function
Definition File  cplex.h
Include Files  cplex.h

Synopsis

```c
int CPXPUBLIC CPXcopystart (CPXCENVptr env,
    CPXLPptr lp,
    const int * cstat,
    const int * rstat,
    const double * cprim,
    const double * rprim,
    const double * cdual,
    const double * rdual);
```

Description

The routine CPXcopystart is used to provide starting information for use in a subsequent call to a simplex optimization routine (CPXlpopt with CPX_PARAM_LPMETHOD or CPX_PARAM_QPMETHOD set to CPX_ALG_PRIMAL or CPX_ALG_DUAL, CPXdualopt, CPXprimopt, or CPXhybnetopt). Starting information is not applicable to the barrier optimizer or the mixed integer optimizer.

When a basis (parameters cstat and rstat) is installed for a linear problem and CPXlpopt is used with CPX_PARAM_LPMETHOD set to CPX_ALG_AUTOMATIC, CPLEX will use the primal simplex algorithm if the basis is primal feasible and the dual simplex method otherwise.

Any of three different kinds of starting points can be provided: a starting basis (cstat, rstat), starting primal values (cprim, rprim), and starting dual values (cdual, rdual). Only a starting basis is applicable to a CPXhybnetopt call, but for Dual Simplex and Primal Simplex any combination of these three types of information can be used in providing a starting point. If no starting-point is provided, this routine returns an error; otherwise, any resident starting information in the CPLEX problem object is freed and the new information is copied into it.

If you provide a starting basis, then both cstat and rstat must be specified. It is permissible to provide cprim with or without rprim, or rdual with or without cdual; arrays not being provided must be passed as NULL pointers.

```
Note: The starting information is ignored by the optimizers if the parameter CPX_PARAM_ADVIND is off.
```
Example

```c
status = CPXcopystart (env,
    lp,
    cstat,
    rstat,
    cprim,
    rprim,
    cdual,
    rdual);
```

Parameters

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **lp**
  A pointer to a CPLEX problem object as returned by CPXcreateprob.
- **cstat**
  A pointer to a CPLEX problem object as returned by CPXcreateprob.
An array containing the basis status of the columns in the constraint matrix. The length of the array is equal to the number of columns in the CPLEX problem object. If this array is NULL, rstat must be NULL. Table 1 shows the possible values.

*rstat*

An array containing the basis status of the slack/surplus/artificial variable associated with each row in the constraint matrix. The array's length is equal to the number of rows in the LP problem. For rows other than ranged rows, the array element rstat[i] can be set according to Table 2. For ranged rows, the array element rstat[i] can be set according to Table 3. If this array is NULL, cstat must be NULL.

*cprim*

An array containing the initial primal values of the column variables. The length of the array must be no less than the number of columns in the CPLEX problem object. If this array is NULL, rprim must be NULL.

*rprim*

An array containing the initial primal values of the slack (row) variables. The length of the array must be no less than the number of rows in the CPLEX problem object. This array may be NULL.

*cdual*

An array containing the initial values of the reduced costs for the column variables. The length of the array must be no less than the number of columns in the CPLEX problem object. This array may be NULL.

*rdual*

An array containing the initial values of the dual variables for the rows. The length of the array must be no less than the number of rows in the CPLEX problem object. If this array is NULL, cdual must be NULL.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
**CPXcreateprob**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**

```c
CPXLPtr CPXPUBLIC CPXcreateprob(CPXCENVptr env,
    int * status_p,
    const char * probname_str)
```

**Description**
The routine CPXcreateprob creates a CPLEX problem object in the CPLEX environment. The arguments to CPXcreateprob define an LP problem name. The problem that is created is an LP minimization problem with zero constraints, zero variables, and an empty constraint matrix. The CPLEX problem object exists until the routine CPXfreeprob is called.

To define the constraints, variables, and nonzero entries of the constraint matrix, any of the CPLEX LP problem modification routines may be used. In addition, any of the routines beginning with the prefix CPXcopy may be used to copy data into the CPLEX problem object. New constraints or new variables can be created with the routines CPXnewrows or CPXnewcols, respectively.

**Example**

```c
lp = CPXcreateprob (env, &status, "myprob");
```

See also all the Callable Library examples (except those pertaining to networks) in the *ILOG CPLEX User’s Manual*.

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **status_p**
  A pointer to an integer used to return any error code produced by this routine.

- **probname_str**
  A character string that specifies the name of the problem being created.

**Returns**
If successful, CPXcreateprob returns a pointer that can be passed to other CPLEX routines to identify the problem object that is created. If not successful, a NULL pointer
is returned, and an error status is returned in the variable *status_p. If the routine is successful, *status_p is 0 (zero).
CPXcutcallbackadd

Category       Global Function
Definition File cplex.h
Include Files  cplex.h

Synopsis
int CPXPUBLIC CPXcutcallbackadd(CPXCENVptr env,
void * cbdata,
int wherefrom,
int nzcnt,
double rhs,
int sense,
const int * cutind,
const double * cutval)

Description
The routine CPXcutcallbackadd adds cuts during MIP branch & cut. This routine may be called only from within user-written cut callbacks; thus it may be called only when the value of its wherefrom argument is CPX_CALLBACK_MIP_CUT.

The cut may be for the original problem if the parameter CPX_PARAM_MIPCBREDLP was set to CPX_OFF before the call to CPXmipopt that calls the callback. Otherwise, the cut is used on the presolved problem.

Example
status = CPXcutcallbackadd (env,
cbdata,
wherefrom,
mnzcnt,
myrhs,
'L',
mycutind,
mycutval);

See also the example admipex5.c in the standard distribution.

Parameters

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbdata
The pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.

wherefrom
An integer value that indicates where the user-written callback was called from. This parameter must be the value of `wherefrom` passed to the user-written callback.

`nzcnt`

An integer value that indicates the number of coefficients in the cut, or equivalently, the length of the arrays `cutind` and `cutval`.

`rhs`

A double value that indicates the value of the right-hand side of the cut.

`sense`

An integer value that indicates the sense of the cut.

`cutind`

An array containing the column indices of cut coefficients.

`cutval`

An array containing the values of cut coefficients.

**Returns**

The routine returns zero on success and nonzero if an error occurs. One possible error is indicated by the symbolic constant `CPXERR_NO_SPACE`. That error occurs when the number of cuts added reaches the maximum allowed, as set by the parameter `CPX_PARAM_CUTSFACCTOR`. 
**CPXcutcallbackaddlocal**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXcutcallbackaddlocal(CPXENVptr env,
    void * cbdata,
    int wherefrom,
    int nzcnt,
    double rhs,
    int sense,
    const int * cutind,
    const double * cutval)
```

**Description**  
The routine CPXcutcallbackaddlocal adds local cuts during MIP branch & cut. A local cut is one that applies to the current nodes and the subtree rooted at this node. Global cuts, that is, cuts that apply throughout the branch & cut tree, are added with the routine CPXcutcallbackadd. This routine may be called only from within user-written cut callbacks; thus it may be called only when the value of its wherefrom argument is CPX_CALLBACK_MIP_CUT.

The cut may be for the original problem if the parameter CPX_PARAM_MIPCBREDLP was set to CPX_OFF before the call to CPXmipopt that calls the callback. Otherwise, the cut is used on the presolved problem.

**Example**  
```c
status = CPXcutcallbackaddlocal (env,
    cbdata,
    wherefrom,
    mynzcnt,
    myrhs,
    'L',
    mycutind,
    mycutval);
```

**See Also**  
CPXcutcallbackadd, CPXgetcutcallbackfunc, CPXsetcutcallbackfunc

**Parameters**

e
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbdata
The pointer passed to the user-written callback. This parameter must be the value of `cbdata` passed to the user-written callback.

`wherefrom`
An integer value that indicates where the user-written callback was called from. This parameter must be the value of `wherefrom` passed to the user-written callback.

`nzcnt`
An integer value that indicates the number of coefficients in the cut, or equivalently, the length of the arrays `cutind` and `cutval`.

`rhs`
A double value that indicates the value of the right-hand side of the cut.

`sense`
An integer value that indicates the sense of the cut.

`cutind`
An array containing the column indices of cut coefficients.

`cutval`
An array containing the values of cut coefficients.

**Returns**
The routine returns zero on success and nonzero if an error occurs.
CPXdelchannel

Category               Global Function
Definition File         cplex.h
Include Files          cplex.h
Synopsis                void CPXPUBLIC CPXdelchannel(CPXENVptr env, CPXCHANNELptr * channel_p)
Description             The routine CPXdelchannel flushes all message destinations for a channel, clears the message destination list, and frees the memory allocated to the channel. On completion, the pointer to the channel is set to NULL.
Example                 CPXdelchannel (env, &mychannel);

See also lpex5.c in the ILOG CPLEX User's Manual.

Parameters
env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
channel_p
A pointer to the pointer to the channel containing the message destinations to be flushed, cleared, and destroyed.

Returns               This routine does not have a return value.
**Category**
Global Function

**Definition File**
cplex.h

**Synopsis**
```c
int CPXPUBLIC CPXdelcols(CPXENVptr env,
                          CPXLPptr lp,
                          int begin,
                          int end)
```

**Description**
The routine `CPXdelcols` is used to delete all the columns in a specified range. The range is specified using a lower and an upper index that represent the first and last column to be deleted, respectively. The indices of the columns following those deleted are decreased by the number of columns deleted.

**Example**
```c
status = CPXdelcols (env, lp, 10, 20);
```

**Parameters**
- **env**
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
- **lp**
  A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
- **begin**
  An integer that indicates the numeric index of the first column to be deleted.
- **end**
  An integer that indicates the numeric index of the last column to be deleted.

**Returns**
The routine returns zero on success and nonzero if an error occurs.
CPXdelfpdest

Category  Global Function
Definition File  cplex.h
Include Files  cplex.h
Synopsis  
```c
int CPXPUBLIC CPXdelfpdest(CPXENVptr env,
  CPXCHANNELptr channel,
  CPXFILEptr fileptr)
```
Description  The routine CPXdelfpdest removes a file from the list of message destinations for a channel. Failure occurs when the channel does not exist or the file pointer is not in the message destination list.
Example  
```c
CPXdelfpdest (env, mychannel, fileptr);
```
Parameters  
- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **channel**
  The pointer to the channel for which destinations are to be deleted.
- **fileptr**
  A CPXFILEptr for the file to be removed from the destination list.
Returns  The routines return zero on success and nonzero if an error occurs.
CPXdelfuncdest

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXdelfuncdest(CPXENVptr env,
CPXCHANNELptr channel,
void * handle,
void(CPXPUBLIC *msgfunction)(void *, const char *) )
```

Description:

The routine CPXdelfuncdest removes the function msgfunction from the list of message destinations associated with a channel. Use CPXdelfuncdest to remove functions that were added to the list using CPXaddfuncdest.

To illustrate, consider an application in which a developer wishes to trap CPLEX error messages and display them in a dialog box that prompts the user for an action. Use CPXaddfuncdest to add the address of a function to the list of message destinations associated with the cpxerror channel. Then write the msgfunction routine. It must contain the code that controls the dialog box. When CPXmsg is called with cpxerror as its first argument, it calls the msgfunction routine, which then displays the error message.

**Note:** The handle parameter is a generic pointer that can be used to hold information needed by the msgfunction routine to avoid making such information global to all routines.

Example:

```c
void msgfunction (void *handle, char *msg_string)
{
    FILE *fp;
    fp = (FILE *)handle;
    fprintf (fp, "%s", msg_string);
}
status = CPXdelfuncdest (env, mychannel, fileptr, msgfunction);
```

Parameters:

- env

A pointer to the CPLEX environment as returned by CPXopenCPLEX.

channel
The pointer to the channel to which the function destination is to be added.

handle
A void pointer that can be used in the msgfunction routine to direct the message to a file, the screen, or a memory location.

msgfunction
The pointer to the function to be called when a message is sent to a channel. For details about this callback function, see CPXaddfuncdest.

See Also
CPXaddfuncdest

Returns
The routines return zero on success and nonzero if an error occurs. Failure occurs when msgfunction is not in the message destination list or the channel does not exist.
## CPXdelnames

**Category**  
Global Function

**Definition File**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXdelnames(CPXENVptr env, CPXLPptr lp)
```

**Description**  
The routine **CPXdelnames** removes all names that have been previously assigned to rows and columns. The memory that was used by those names is released.

Names can be assigned to rows and columns in a variety of ways, and this routine allows them to be removed. For example, if the problem is read from a file in LP or MPS format, names are also read from the file. Names can be assigned by the user by calling one of the routines **CPXchgrowname**, **CPXchgcolname**, or **CPXchgname**.

**Example**

```c
CPXdelnames (env, lp);
```

**Parameters**

- **env**  
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **lp**  
A pointer to a CPLEX problem object as returned by CPXcreateprob.

**Returns**  
The routine returns zero on success and nonzero if an error occurs.
CPXdelqconstrs

Category Global Function

Definition File cplex.h

Include Files cplex.h

Synopsis

int CPXPUBLIC CPXdelqconstrs(CPXENVptr env,
    CPXLPptr lp,
    int begin,
    int end)

Description

The routine CPXdelqconstrs deletes a range of quadratic constraints. The range is specified by a lower and upper index that represent the first and last quadratic constraints to be deleted, respectively. The indices of the constraints following those deleted are decreased by the number of deleted constraints.

Example

status = CPXdelqconstrs (env, lp, 10, 20);

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

begin
An integer that indicates the numeric index of the first quadratic constraint to be deleted.

end
An integer that indicates the numeric index of the last quadratic constraint to be deleted.

Returns

The routine returns zero on success and nonzero if an error occurs.
**CPXdelrows**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**
```c
int CPXPUBLIC CPXdelrows(CPXENVptr env,
            CPXLPptr lp,
            int begin,
            int end)
```

**Description**
The routine `CPXdelrows` deletes a range of rows. The range is specified using a lower and upper index that represent the first and last row to be deleted, respectively. The indices of the rows following those deleted are decreased by the number of deleted rows.

**Example**
```c
status = CPXdelrows (env, lp, 10, 20);
```

**Parameters**
- **env**
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
- **lp**
  A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
- **begin**
  An integer that indicates the numeric index of the first row to be deleted.
- **end**
  An integer that indicates the numeric index of the last row to be deleted.

**Returns**
The routine returns zero on success and nonzero if an error occurs.
**CPXdelsetcols**

**Category** Global Function

**Definition File** cplex.h

**Include Files** cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXdelsetcols(CPXENVptr env,
                    CPXLPptr lp,
                    int * delstat)
```

**Description**

The routine `CPXdelsetcols` is used to delete a set of columns from a CPLEX problem object. Unlike the routine `CPXdelcols`, `CPXdelsetcols` does not require the columns to be in a contiguous range. After the deletion occurs, the remaining columns are indexed consecutively starting at 0, and in the same order as before the deletion.

**Note:** The `delstat` array must have at least `CPXgetnumcols(env, lp)` elements.

**Example**

```c
status = CPXdelsetcols (env, lp, delstat);
```

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
- **lp**
  A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
- **delstat**
  An array indicating the columns to be deleted. The routine `CPXdelsetcols` deletes each column `j` for which `delstat[j] = 1`. The deletion of columns results in a renumbering of the remaining columns. After termination, `delstat[j]` is either -1 for columns that have been deleted or the new index number that has been assigned to the remaining columns.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
**CPXdelsetrows**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXdelsetrows (CPXCENVptr env,  
CPXLPptr lp,  
int * delstat)
```

**Description**  
The routine `CPXdelsetrows` deletes a set of rows. Unlike the routine `CPXdelrows`, `CPXdelsetrows` does not require the rows to be in a contiguous range. After the deletion occurs, the remaining rows are indexed consecutively starting at 0, and in the same order as before the deletion.

```
Note: The delstat array must have at least CPXgetnumrows(env, lp) elements.
```

**Example**  
```c
status = CPXdelsetrows (env, lp, delstat);
```

**Parameters**  
- **env**  
A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
- **lp**  
A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
- **delstat**  
An array indicating the rows to be deleted. The routine `CPXdelsetrows` deletes each row `i` for which `delstat[i] = 1`. The deletion of rows results in a renumbering of the remaining rows. After termination, `delstat[i]` is either -1 for rows that have been deleted or the new index number that has been assigned to the remaining rows.

**Returns**  
The routine returns zero on success and nonzero if an error occurs.
CPXdelsetsos

Category          Global Function
Definition File   cplex.h
Include Files     cplex.h
Synopsis          int CPXPUBLIC CPXdelsetsos(CPXENVptr env,
                           CPXLPptr lp,
                           int * delset)
Description       The routine CPXdelsetsos is used to delete a group of special ordered sets (SOSs) from a CPLEX problem object.

Note: The delstat array must have at least CPXgetnumsos(env,lp) elements.

Example

status = CPXdelsetsos (env, lp, delstat);

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.
delset
An array indicating the SOSs to be deleted. The routine CPXdelsetsos deletes each SOS j for which delstat[j] = 1. The deletion of SOSs results in a renumbering of the remaining SOSs. After termination, delstat[j] is either -1 for SOSs that have been deleted or the new index number that has been assigned to the remaining SOSs.

Note: The delstat array must have at least CPXgetnumsos(env,lp) elements.

Example
status = CPXdelsetsos (env, lp, delstat);

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXdisconnectchannel

Category       Global Function
Definition File cplex.h
Include Files  cplex.h
Synopsis        void CPXPUBLIC CPXdisconnectchannel(CPXENVptr env,
                                          CPXCHANNELptr channel)
Description    The routine CPXdisconnectchannel flushes all message destinations associated
                with a channel and clears the corresponding message destination list.
Example         CPXdisconnectchannel (env, mychannel);

Parameters

  env
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

  channel
  A pointer to the channel containing the message destinations to be flushed and cleared.

Returns         This routine does not have a return value.
CPXdisplayiis

Category          Global Function
Definition File   cplex.h
Include Files    cplex.h
Synopsis          int CPXPUBLIC CPXdisplayiis(CPXENVptr env,
                      CPXCLPptr lp,
                      CPXCHANNELptr channel,
                      int display)

Description       The routine CPXdisplayiis is used to send IIS output to a CPLEX message channel. The IIS must already have been computed using a call to CPXfindiis or CPXiiswrite. Two different levels of output are available, corresponding to the output written to the log file and an IIS file in the CPLEX Interactive Optimizer. Thus, CPXdisplayiis enables the user to use CPLEX IIS output formats in a Callable Library application. If neither of these formats is appropriate, the information provided by the routine CPXgetiis can be used to create customized IIS output.

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CPXIIS_TERSE</td>
<td>Displays the names of the rows and columns in the IIS</td>
</tr>
<tr>
<td>2</td>
<td>CPXIIS_VERBOSE</td>
<td>Displays an LP format file containing the IIS</td>
</tr>
</tbody>
</table>

Example

status = CPXdisplayiis (env, lp, mychannel, CPXIIS_TERSE);

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.
channel
The pointer to the message channel receiving the IIS output.
display
An integer indicating the type of output desired.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXdperwrite

Category                Global Function
Definition File         cplex.h
Include Files           cplex.h
Synopsis                int CPXPUBLIC CPXdperwrite (CPXCENVptr env,
                        CPXLPptr lp,
                        const char * filename_str,
                        double epsilon)

Description             When solving degenerate linear programs with the dual simplex method, CPLEX may
                        initiate a perturbation of the objective function of the problem in order to improve
                        performance. The routine CPXdperwrite writes a similarly perturbed problem to a
                        binary SAV format file.
Example                  status = CPXdperwrite (env, lp, "myprob.dpe", epsilon);

Parameters              env
                        A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
                        A pointer to a CPLEX problem object as returned by CPXcreateprob.
filename_str
                        A character string containing the name of the file to which the perturbed LP problem
                        should be written.
epsilon
                        The perturbation constant.

Returns                 The routine returns zero on success and nonzero if an error occurs.
CPXdualopt

Category Global Function
Definition File cplex.h
Include Files cplex.h
Synopsis int CPXPUBLIC CPXdualopt (CPXCENVptr env, CPXLPptr lp)
Description The routine CPXdualopt may be used at any time after a linear program has been created via a call to CPXcreateprob to find a solution to that problem using the dual simplex algorithm. When this function is called, the CPLEX dual simplex optimization routines attempt to optimize the specified problem. The results of the optimization are recorded in the CPLEX problem object.
Example
status = CPXdualopt (env, lp);
Parameters
env A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp A pointer to a CPLEX problem object as returned by CPXcreateprob.
Returns The routine returns zero unless an error occurred during the optimization. Examples of errors include exhausting available memory (CPXERR_NO_MEMORY) or encountering invalid data in the CPLEX problem object (CPXERR_NO_PROBLEM). Exceeding a user-specified CPLEX limit, or proving the model infeasible or unbounded, are not considered errors. Note that a zero return value does not necessarily mean that a solution exists. Use query routines CPXsolninfo, CPXgetstat, and CPXsolution to obtain further information about the status of the optimization.
CPXdualwrite

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXdualwrite(CPXENVptr env, 
   CPXCLPptr lp, 
   const char * filename_str, 
   double * objshift_p)
```

Description:
The routine CPXdualwrite is used to write a dual formulation of the current CPLEX problem object. MPS format is used. This function can only be applied to a linear program; it generates an error for other problem types.

Note:
Any fixed variables in the primal are removed before the dual problem is written to a file. Each fixed variable with a nonzero objective coefficient causes the objective value to shift. As a result, if fixed variables are present, the optimal objective obtained from solving the dual problem created using CPXdualwrite may not be the same as the optimal objective of the primal problem. The `objshift_p` parameter can be used to reconcile this difference.

Example:

```c
status = CPXdualwrite (env, lp, "myfile.dua", &objshift);
```

Parameters:

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **lp**
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

- **filename_str**
  A character string containing the name of the file to which the dual problem should be written.

- **objshift_p**
A pointer to a variable of type double to hold the change in the objective function resulting from the removal of fixed variables in the primal.

Returns

The routine returns zero on success and nonzero if an error occurs.
# CPXembwrite

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXembwrite (CPXCENVptr env,  
CPXLPptr lp,  
const char * filename_str)
```

**Description**  
The routine `CPXembwrite` is used to write out the network embedded in the selected problem object. MPS format is used. The specific network extracted depends on the current setting of the `CPX_PARAM_NETFIND` parameter.

**Example**  
```c
status = CPXembwrite (env, lp, "myfile.emb");
```

**Parameters**

- **env**  
A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- **lp**  
A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

- **filename_str**  
A character string containing the name of the file to which the embedded network should be written.

**Example**  
```c
status = CPXembwrite (env, lp, "myfile.emb");
```

**Returns**  
The routine returns zero on success and nonzero if an error occurs.
CPXfclose

**Category**       Global Function

**Definition File**  cplex.h

**Include Files**  cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXfclose(CPXFILEptr stream)
```

**Description**

The routine `CPXfclose` closes files that are used in conjunction with the routines `CPXaddfpdest`, `CPXdelfpdest`, and `CPXsetlogfile`. It is used in the same way as the standard C library function `fclose`. Files that are opened with the routine `CPXfopen` must be closed with the routine `CPXfclose`.

**Example**

```c
CPXfclose(fp);
```

See `lpex5.c` in the *Cplex User's Manual*.

**Parameters**

- **stream**
  
  A pointer to a file opened by the routine `CPXfopen`.

**Returns**

This routine returns zero on success and nonzero if a failure occurs. The syntax is identical to the standard C library routine `fclose`.
**CPXfeasopt**

**Category**  
Global Function

**Definition File**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXfeasopt (CPXCENVptr env,  
                        CPXLPptr lp,  
                        double * rhs,  
                        double * rng,  
                        double * lb,  
                        double * ub,  
                        int optind)
```

**Description**  
The routine `CPXfeasopt` computes a minimal relaxation of the bounds of variable bounds or the right-hand sides of constraints that make the active model feasible. For each bound, the user may specify a preferred value indicating how much relaxation of that bound is acceptable. Similarly, for every right-hand side value or (where applicable) range value, the user may specify a preferred value indicating how much relaxation of that value is acceptable. A negative or zero (0) value indicates that the corresponding bound must not be relaxed. Typically, values greater than or equal to one should be used.

If enough variables or constraints were allowed to be relaxed, the routine returns with suggested relaxed values that would make the active model feasible. These bounds are chosen in such a way that the relaxation is minimal in the sense that the sum of (relaxation amount)/(preference value) over all variables and constraints is minimized.

A solution vector for the model relaxed in this way will be available.

When the argument `optind` is `true`, this method also tries to find the optimal solution of the minimally relaxed model.

**Parameters**

- **env**  
  Pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- **lp**  
  Pointer to a CPLEX problem object as returned by `CPXcreateprob`.

- **rhs**  
  An array of doubles of length at least equal to the number of rows in the problem. NULL may be specified if no rhs values are allowed to be relaxed. When a non-NULL array is passed and the function successfully finds a relaxation, the elements of the
array will contain the new values for the right-hand side that constitute the feasible relaxation found by CPXfeasopt.

rng
An array of doubles of length at least equal to the number of rows in the problem. NULL may be specified if no range values are allowed to be relaxed or none are present in the active model. When a non-NULL array is passed and the function successfully finds a relaxation, the elements of the array will contain the new values for the ranges that constitute the feasible relaxation found by CPXfeasopt.

lb
An array of doubles of length at least equal to the number of columns in the problem. NULL may be passed if no lower bound of any variable is allowed to be relaxed. When a non-NULL array is passed and the function successfully finds a relaxation, the elements of the array will contain the new values for the lower bounds that constitute the feasible relaxation found by CPXfeasopt.

ub
An array of doubles of length at least equal to the number of columns in the problem. NULL may be passed if no upper bound of any variable is allowed to be relaxed. When a non-NULL array is passed and the function successfully finds a relaxation, the elements of the array will contain the new values for the upper bounds that constitute the feasible relaxation found by CPXfeasopt.

optind
Optimization indicator. If this parameter is true, then CPXfeasopt tries to find the optimal solution of the minimally relaxed model.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXfindiis

Category       Global Function
Definition File cplex.h
Include Files  cplex.h
Synopsis        int CPXPUBLIC CPXfindiis(CPXCENVptr env,  
                             CPXLPptr lp,  
                             int * iisnumrows_p,  
                             int * iisnumcols_p)
Description     The routine CPXfindiis is used to determine an irreducibly inconsistent set (IIS) for an infeasible LP problem. On successful completion, CPXfindiis returns the number of rows and columns in the IIS. To obtain information about the individual rows and columns, use the routine CPXgetiis, CPXiiswrite or CPXdisplayiis.
Example         status = CPXfindiis (env, lp, &iisnumrows, &iisnumcols);

Parameters      env
                A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
                A pointer to a CPLEX problem object as returned by CPXcreateprob.
                iisnumrows_p
                A pointer to an integer to contain the number of rows in the IIS.
                iisnumcols_p
                A pointer to an integer to contain the number of columns in the IIS.

Returns         The routine returns zero on success and nonzero if an error occurs.
CPXflushchannel

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis: void CPXPUBLIC CPXflushchannel(CPXENVptr env, CPXCHANNELptr channel)

Description: The routine CPXflushchannel flushes (outputs and clears the buffers of) all message destinations for a channel. Use this routine in cases when it is important to have output written to disk immediately after it is generated. For most applications this routine need not be used.

Example:

CPXflushchannel (env, mychannel);

Parameters:

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

channel
A pointer to the channel containing the message destinations to be flushed.

Returns: This routine does not return a value.
CPXflushstdchannels

**Category**          Global Function

**Definition File**   cplex.h

**Include Files**     cplex.h

**Synopsis**          int CPXPUBLIC CPXflushstdchannels(CPXENVptr env)

**Description**       The routine CPXflushstdchannels flushes the output buffers of the four standard channels cpxresults, cpxwarning, cpxerror, and cpxlog. Use this routine where it is important to see all of the output created by CPLEX either on the screen or in a disk file without calling CPXflushchannel for each of the four channels.

**Example**

```c
status = CPXflushstdchannels (env);
```

**Parameters**        env

A pointer to the CPLEX environment as returned by CPXopenCPLEX.

**Returns**           The routine returns zero on success and nonzero if an error occurs.
CPXfopen

Category          Global Function
Definition File  cplex.h
Include Files    cplex.h
Synopsis          CPXFILEptr CPXPUBLIC CPXfopen(const char * filename_str,
                                           const char * type_str)
Description       The routine CPXfopen is used to open files to be used in conjunction with the routines
                  CPXaddfpdest, CPXdelfpdest and CPXsetlogfile. It has the same
                  arguments as the standard C library function fopen.
Example           fp = CPXfopen ("mylog.log", "w");

See also lpex5.c in the ILOG CPLEX User's Manual.

Parameters

  filename_str  A pointer to a character string that contains the name of the file to be opened.

  type_str     A pointer to a character string, containing characters according to the syntax of the
               standard C function fopen.

Returns           The routine returns a pointer to an object representing an open file, or NULL if the file
                  could not be opened. A CPXFILEptr is analogous to FILE *type in C language.
CPXfputs

Category: Global Function
Definition File: cplex.h
Include Files: cplex.h

Synopsis

```c
int CPXPUBLIC CPXfputs(const char * s_str, CPXFILEptr stream)
```

Description

The routine CPXfputs can be used to write output to a file opened with CPXfopen. The purpose of this routine is to allow user-defined output in a file to be interspersed with the output created by using the routines CPXaddfpdest or CPXsetlogfile. The syntax of CPXfputs is the same as the standard C library function fputs.

Example

```c
CPXfputs("Solved first problem.
```

Parameters

s_str
A pointer to a string to be output to the file.

stream
A pointer to a file opened by the routine CPXfopen.

Returns

This routine returns a nonnegative value if successful. Otherwise, it returns the system constant EOF (end of file).
CPXfreelazyconstraints

Category Global Function

Definition File cplex.h

Include Files cplex.h

Synopsis int CPXPUBLIC CPXfreelazyconstraints(CPXENVptr env, CPXLPptr lp)

Description The routine CPXfreelazyconstraints is used to clear the list of lazy constraints that have been previously specified through calls to CPXaddlazyconstraints.

Example

status = CPXfreelazyconstraints (env, lp);

Parameters env
An pointer to the CPLEX environment, as returned by CPXopenCPLEX.

lp
A pointer to a CPLEX LP problem object, as returned by CPXcreateprob.

Returns The routine returns zero on success and nonzero if an error occurs.
CPXfreeprob

Category: Global Function
Definition File: cplex.h
Include Files: cplex.h
Synopsis:
```c
int CPXPUBLIC CPXfreeprob(CPXENVptr env,
                          CPXLPptr * lp_p)
```
Description:
The routine CPXfreeprob removes the specified CPLEX problem object from the
CPLEX environment and frees the associated memory used internally by CPLEX. The
routine is used when the user has no need for further access to the specified problem
data.

Example:
```c
status = CPXfreeprob (env, &lp);
```
See also the example lpex1.c in the ILOG CPLEX User's Manual and in the standard
distribution.

Parameters:
- env
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- lp_p
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

Example:
```c
status = CPXfreeprob (env, &lp);
```
See also the example lpex1.c in the ILOG CPLEX User's Manual and in the standard
distribution.

Returns:
The routine returns zero on success and nonzero if an error occurs.
CPXfreeusercuts

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:
```c
int CPXPUBLIC CPXfreeusercuts (CPXCENVptr env, CPXLPptr lp)
```

Description:
The routine `CPXfreeusercuts` is used to clear the list of user cuts that have been previously specified through calls to `CPXaddusercuts`.

Example:
```c
status = CPXfreeusercuts (env, lp);
```

Parameters:
- `env`: A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
- `lp`: A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.

Returns:
The routine returns zero on success and nonzero if an error occurs.
**CPXgetax**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXgetax(CPXENVptr env,  
CPXCLPptr lp,  
double * x,  
int begin,  
int end)
```

**Description**  
The routine `CPXgetax` is used to access row activity levels for a range of constraints of a linear or quadratic program. The beginning and end of the range must be specified. A row activity is the inner product of a row in the constraint matrix and the structural variables in the problem.

The array must be of length at least `(end-begin+1)`. If successful, `x[0]` through `x[end-begin]` contain the row activities.

**Parameters**

- **env**  
A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- **lp**  
A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

- **x**  
An array to receive the values of the row activity levels for each of the constraints in the specified range.

  The array must be of length at least `(end-begin+1)`. If successful, `x[0]` through `x[end-begin]` contain the row activities.

- **begin**  
An integer indicating the beginning of the range of row activities to be returned.

- **end**  
An integer indicating the end of the range of row activities to be returned.

**Returns**  
The routine returns zero on success and nonzero if an error occurs.
**CPXgetbaritcnt**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**
```
int CPXPUBLIC CPXgetbaritcnt (CPXCENVptr env, CPXCLPptr lp)
```

**Description**
The routine `CPXgetbaritcnt` is used to access the total number of Barrier iterations to solve an LP problem.

**Example**
```
itcnt = CPXgetbaritcnt (env, lp);
```

**Parameters**
- `env`
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
- `lp`
  A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

**Example**
```
itcnt = CPXgetbaritcnt (env, lp);
```

**Returns**
The routine returns the total iteration count if a solution exists. It returns zero if no solution exists or any other type of error occurs.
CPXgetbase

Category       Global Function
Definition File cplex.h
Include Files  cplex.h
Synopsis        

int CPXPUBLIC CPXgetbase(CPXCENVptr env,  
                        CPXCLPptr lp,  
                        int * cstat,  
                        int * rstat)

Description     The routine CPXgetbase is used to get the basis resident in a CPLEX problem object. Either of the arguments cstat or rstat may be NULL, if only one set of statuses is needed.

<table>
<thead>
<tr>
<th></th>
<th>Values of elements of cstat</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_AT_LOWER</td>
<td>0 variable at lower bound</td>
</tr>
<tr>
<td>CPX_BASIC</td>
<td>1 variable is basic</td>
</tr>
<tr>
<td>CPX_AT_UPPER</td>
<td>2 variable at upper bound</td>
</tr>
<tr>
<td>CPX_FREE_SUPER</td>
<td>3 variable free and nonbasic</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Values of elements of rstat in rows other than ranged rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_AT_LOWER</td>
<td>0 associated slack/surplus/artificial variable nonbasic at value 0.0</td>
</tr>
<tr>
<td>CPX_BASIC</td>
<td>1 associated slack/surplus/artificial variable basic</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Values of elements of rstat for ranged rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_AT_LOWER</td>
<td>0 associated slack/surplus/artificial variable at its lower bound</td>
</tr>
<tr>
<td>CPX_BASIC</td>
<td>1 associated slack/surplus/artificial variable basic</td>
</tr>
</tbody>
</table>
Example

```
status = CPXgetbase (env, lp, cstat, rstat);
```

See also the example `lpex6.c` in the *ILOG CPLEX User's Manual* and in the standard distribution.

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- **lp**
  A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

- **cstat**
  An array to receive the basis status of the columns in the CPLEX problem object. The length of the array must be no less than the number of columns in the matrix. The array element `cstat[i]` has the meaning indicated in Table 1.

- **rstat**
  An array to receive the basis status of the artificial/slack/surplus variable associated with each row in the constraint matrix. The array’s length must be no less than the number of rows in the CPLEX problem object. For rows other than ranged rows, the array element `rstat[i]` has the meaning indicated in Table 2. For ranged rows, the array element `rstat[i]` has the meaning indicated in Table 3.

**Returns**

The routine returns zero if a basis exists. It returns zero if no solution exists or any other type of error occurs.

<table>
<thead>
<tr>
<th>CPX_AT_UPPER</th>
<th>2</th>
<th>associated slack/surplus/artificial variable nonbasic at upper bound</th>
</tr>
</thead>
</table>
**CPXgetbestobjval**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**
```c
int CPXPUBLIC CPXgetbestobjval(CPXENVptr env,
                              CPXCLPptr lp,
                              double * objval_p)
```

**Description**
The routine CPXgetbestobjval accesses the currently best known bound on the optimal solution value of the problem at the time the invoking callback is called by an instance of IloCplex while solving a MIP. When a model has been solved to optimality, this value matches the optimal solution value. Otherwise, this value is computed for a minimization (maximization) problem as the minimum (maximum) objective function value of all remaining unexplored nodes.

**Example**
```c
status = CPXgetbestobjval (env, lp, &objval);
```

**Parameters**
- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **lp**
  A pointer to a CPLEX problem object as returned by CPXcreateprob.
- **objval**
  A pointer to the location where the best node objective value is returned. If the branch & cut tree has been exhausted, the best node value is set to a large number.

**Returns**
The routine returns zero on success and nonzero if an error occurs.
CPXgetbranchcallbackfunc

Category                     Global Function
Definition File             cplex.h
Include Files               cplex.h
Synopsis
void CPXPUBLIC CPXgetbranchcallbackfunc(CPXENVptr env,
     int CPXPUBLIC **branchcallback_p)(CALLBACK_BRANCH_ARGSS) ,
     void ** cbhandle_p)

Description

Note: This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetbranchcallbackfunc accesses the user-written callback routine to be called during MIP optimization after a branch has been selected but before the branch is carried out. ILOG CPLEX uses the callback routine to change its branch selection.

Example

CPXgetbranchcallbackfunc(env, &current_callback,
     &current_handle);

See also Advanced MIP Control Interface in the ILOG CPLEX User's Manual.

Parameters

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.
branchcallback_p
The address of the pointer to the current user-written branch callback. If no callback has been set, the returned pointer evaluates to NULL.
cbhandle_p
The address of a variable to hold the user's private pointer.
Callback description

```c
int callback (CPXCENVptr env,
    void       *cbdata,
    int        wherefrom,
    void       *cbhandle,
    int        type,
    int        sos,
    int        nodecnt,
    int        bdcnt,
    double     *nodeest,
    int        *nodebeg,
    int        *indices,
    char       *lu,
    int        *bd,
    int        *useraction_p);
```

The call to the branch callback occurs after a branch has been selected but before the branch is carried out. This function is written by the user. On entry to the callback, the ILOG CPLEX-selected branch is defined in the arguments. The arguments to the callback specify a list of changes to make to the bounds of variables when child nodes are created. One, two, or zero child nodes can be created, so one, two, or zero lists of changes are specified in the arguments. The first branch specified is considered first. The callback is called with zero lists of bound changes when the solution at the node is integer feasible.

Custom branching strategies can be implemented by calling the CPLEX function `CPXbranchcallbackbranchbds` and setting the `useraction` variable to `CPX_CALLBACK_SET`. Then CPLEX will carry out these branches instead of the CPLEX-selected branches.

Branch variables are in terms of the original problem if the parameter `CPX_PARAM_MIPCBREDLP` is set to `CPX_OFF` before the call to `CPXmipopt` that calls the callback. Otherwise, branch variables are in terms of the presolved problem.

Callback return value

The callback returns zero on success and nonzero if an error occurs.

Callback arguments

`env`

A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.

`cbdata`

A pointer passed from the optimization routine to the user-written callback that identifies the problem being optimized. The only purpose of this pointer is to pass it to the callback information routines.

**wherefrom**

An integer value indicating where in the optimization this function was called. It will have the value CPX_CALLBACK_MIP_BRANCH.

**cbhandle**

A pointer to user-private data.

**int type**

An integer that indicates the type of branch. This table summarizes possible values.

### Branch Types Returned from a User-Written Branch Callback

<table>
<thead>
<tr>
<th>Symbolic Constant</th>
<th>Value</th>
<th>Branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_TYPE_VAR</td>
<td>0</td>
<td>variable branch</td>
</tr>
<tr>
<td>CPX_TYPE_SOS1</td>
<td>1</td>
<td>SOS1 branch</td>
</tr>
<tr>
<td>CPX_TYPE_SOS2</td>
<td>2</td>
<td>SOS2 branch</td>
</tr>
<tr>
<td>CPX_TYPE_USER</td>
<td>X</td>
<td>user-defined</td>
</tr>
</tbody>
</table>

**sos**

An integer that indicates the special ordered set (SOS) used for this branch. A value of –1 indicates that this branch is not an SOS-type branch.

**nodecnt**

An integer that indicates the number of nodes CPLEX will create from this branch. Possible values are:

- 0 (zero), or
- 1, or
- 2.

If the argument is 0, the node will be fathomed unless user-specified branches are made; that is, no child nodes are created and the node itself is discarded.

**bdcnt**

An integer that indicates the number of bound changes defined in the arrays indices, lu, and bd that define the CPLEX-selected branch.
An array with `nodecnt` entries that contains estimates of the integer objective-function value that will be attained from the created node.

`nodebeg`  
An array with `nodecnt` entries. The i-th entry is the index into the arrays `indices`, `lu`, and `bd` of the first bound changed for the i-th node.

`indices`  
Together with `lu` and `bd`, this array defines the bound changes for each of the created nodes. The entry `indices[i]` is the index for the variable.

`lu`  
Together with `indices` and `bd`, this array defines the bound changes for each of the created nodes. The entry `lu[i]` is one of the three possible values indicating which bound to change:

- L for lower bound, or
- U for upper bound, or
- B for both bounds.

`bd`  
Together with `indices` and `lu`, this array defines the bound changes for each of the created nodes. The entry `bd[i]` indicates the new value of the bound.

`useraction_p`  
A pointer to an integer indicating the action for ILOG CPLEX to take at the completion of the user callback. The table summarizes the possible actions.

### Actions to be Taken After a User-Written Branch Callback

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CPX_CALLBACK_DEFAULT</td>
<td>Use CPLEX-selected branch</td>
</tr>
<tr>
<td>1</td>
<td>CPX_CALLBACK_FAIL</td>
<td>Exit optimization</td>
</tr>
</tbody>
</table>
| 2     | CPX_CALLBACK_SET                   | Use user-selected branch, as defined by calls to `CPXcallbackbranchbranchbd`
| 3     | CPX_CALLBACK_NO_SPACE              | Allocate more space and call callback again                            |

**See Also**  
CPXsetbranchcallbackfunc

**Returns**  
This routine does not return a result.
CPXgetcallbackctype

Category                Global Function
Definition File         cplex.h
Include Files          cplex.h
Synopsis
int CPXPUBLIC CPXgetcallbackctype(CPXENVptr env,
   void * cbdata,
   int wherefrom,
   char * xctype,
   int begin,
   int end)

Description

Note: This is an advanced routine. Advanced routines typically demand a profound
understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher
risk of incorrect behavior in your application, behavior that can be difficult to
detach. Therefore, ILOG encourages you to consider carefully whether you can
accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetcallbackctype is used to get the ctypes for the MIP problem
from within a user-written callback during MIP optimization. The values are from the
original problem if CPX_PARAM_MIPCBREDLP is set to CPX_OFF. Otherwise, they
are from the presolved problem.

This routine may be called only when the value of the wherefrom argument is one of the
following:

- CPX_CALLBACK_MIP,
- CPX_CALLBACK_MIP_BRANCH,
- CPX_CALLBACK_MIP_INCUMBENT,
- CPX_CALLBACK_MIP_NODE,
- CPX_CALLBACK_MIP_HEURISTIC,
- CPX_CALLBACK_MIP_SOLVE, or
- CPX_CALLBACK_MIP_CUT.

Example
status = CPXgetcallbackctype (env, cbdata, wherefrom, 
    prectype, 0, precols-1);

Parameters

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbdata
The pointer passed to the user-written callback. This parameter must be the value of 
cbdata passed to the user-written callback.

wherefrom
An integer value indicating from where the user-written callback was called. The 
parameter must be the value of wherefrom passed to the user-written callback.

xctype
An array where the ctype values for the MIP problem will be returned. The array must be 
of length at least (end - begin + 1). If successful, xctype[0] through 
xctype[end-begin] contain the variable types.

begin
An integer indicating the beginning of the range of ctype values to be returned.

end
An integer indicating the end of the range of ctype values to be returned.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXgetcallbackgloballb

Category       Global Function
Definition File cplex.h
Include Files  cplex.h
Synopsis

```c
int CPXPUBLIC CPXgetcallbackgloballb(CPXENVptr env,
                                       void * cbdata,
                                       int wherefrom,
                                       double * lb,
                                       int begin,
                                       int end)
```

Description

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetcallbackgloballb is used to get the best known global lower bound values during MIP optimization from within a user-written callback. The values are from the original problem if CPX_PARAM_MIPCBREDLP is set to CPX_OFF, otherwise they are from the presolved problem.

This routine may be called only when the value of the wherefrom argument is one of the following:
- CPX_CALLBACK_MIP,
- CPX_CALLBACK_MIP_BRANCH,
- CPX_CALLBACK_MIP_INCUMBENT,
- CPX_CALLBACK_MIP_NODE,
- CPX_CALLBACK_MIP_HEURISTIC,
- CPX_CALLBACK_MIP_SOLVE, or
- CPX_CALLBACK_MIP_CUT.

Example
```c
status = CPXgetcallbackglobalb (env, cbdata, wherefrom,
    lb, 0, cols-1);
```

**Parameters**

- `env` 
  A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
- `cbdata` 
  The pointer passed to the user-written callback. This parameter must be the value of `cbdata` passed to the user-written callback.
- `wherefrom` 
  An integer value indicating from where the user-written callback was called. The parameter must be the value of `wherefrom` passed to the user-written callback.
- `lb` 
  An array to receive the values of the global lower bound values. This array must be of length at least (`end - begin + 1`). If successful, `lb[0]` through `lb[end-begin]` contain the global lower bound values.
- `begin` 
  An integer indicating the beginning of the range of lower bound values to be returned.
- `end` 
  An integer indicating the end of the range of lower bound values to be returned.

**Returns** 

The routine returns zero on success and nonzero if an error occurs.
### CPXgetcallbackglobalub

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXgetcallbackglobalub(CPXENVptr env,
    void * cbdata,
    int wherefrom,
    double * ub,
    int begin,
    int end)
```

**Description**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetcallbackglobalub is used to get the best known global upper bound values during MIP optimization from within a user-written callback. The values are from the original problem if `CPX_PARAM_MIPCBREDLP` is set to `CPX_OFF`, otherwise they are from the presolved problem.

This routine may be called only when the value of the `wherefrom` argument is one of the following:

- CPX_CALLBACK_MIP,
- CPX_CALLBACK_MIP_BRANCH,
- CPX_CALLBACK_MIP_INCUMBENT,
- CPX_CALLBACK_MIP_NODE,
- CPX_CALLBACK_MIP_HEURISTIC,
- CPX_CALLBACK_MIP_SOLVE, or
- CPX_CALLBACK_MIP_CUT.

**Example**
status = CPXgetcallbackglobalub (env, cbdata, wherefrom, 
gub, 0, cols-1);

Parameters

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbdata
The pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.

wherefrom
An integer value indicating from where the user-written callback was called. The parameter must be the value of wherefrom passed to the user-written callback.

ub
An array to receive the values of the global upper bound values. This array must be of length at least (end - begin + 1). If successful, ub[0] through ub[end-begin] contain the global upper bound values.

begin
An integer indicating the beginning of the range of upper bound values to be returned.

dend
An integer indicating the end of the range of upper bound values to be returned.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXgetcallbackincumbent

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXgetcallbackincumbent (CPXCENVptr env, 
    void * cbdata, 
    int wherefrom, 
    double * x, 
    int begin, 
    int end)
```

**Description**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetcallbackincumbent is used to get the incumbent values during MIP optimization from within a user-written callback. The values are from the original problem if CPX_PARAM_MIPCBREDLP is set to CPX_OFF. Otherwise, they are from the presolved problem.

This routine may be called only when the value of the wherefrom argument is one of the following:

- CPX_CALLBACK_MIP,
- CPX_CALLBACK_MIP_BRANCH,
- CPX_CALLBACK_MIP_INCUMBENT,
- CPX_CALLBACK_MIP_NODE,
- CPX_CALLBACK_MIP_HEURISTIC,
- CPX_CALLBACK_MIP_SOLVE, or
- CPX_CALLBACK_MIP_CUT.

**Example**
status = CPXgetcallbackincumbent (env, cbdata, wherefrom,
                                   bestx, 0, cols-1);

Parameters

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbdata
The pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.

wherefrom
An integer value indicating from where the user-written callback was called. The parameter must be the value of wherefrom passed to the user-written callback.

x
An array to receive the values of the incumbent (best available) integer solution. This array must be of length at least (end - begin + 1). If successful, x[0] through x[end-begin] contain the incumbent values.

begin
An integer indicating the beginning of the range of incumbent values to be returned.

end
An integer indicating the end of the range of incumbent values to be returned.

Returns
The routine returns zero on success and nonzero if an error occurs.
**CPXgetcallbackinfo**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**
```c
int CPXPUBLIC CPXgetcallbackinfo(CPXENVptr env,
   void * cbdata,
   int wherefrom,
   int whichinfo,
   void * result_p)
```

**Description**
The routine CPXgetcallbackinfo is used to access information about the current optimization process from within a user-written callback function.

**Note:** This routine is the only routine that can access optimization status information from within a non-advanced user-written callback function. It is also the only Callable Library routine that may be called from within a non-advanced user-written callback function, and in fact, may only be called from the callback function.

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **cbdata**
  The cbdata pointer passed to the user-written callback function. The parameter cbdata MUST be the value of cbdata passed to the user-written callback function.

- **wherefrom**
  An integer value indicating the optimization algorithm from which the user-written callback function was called. The parameter wherefrom MUST be the value of wherefrom passed to the user-written callback function. See CPXgetlpcallbackfunc, CPXgetmipcallbackfunc, and CPXgetnetcallbackfunc for possible values of wherefrom and their meaning.

- **whichinfo**
An integer value indicating the specific information that should be returned by CPXgetcallbackinfo to the result argument. Values for whichinfo, the type of the information returned into *result_p, plus a description appear in the table.

result_p
A generic pointer to a variable of type double or int, dependent on the value of whichinfo, as documented in the following tables.

For LP algorithms:

<table>
<thead>
<tr>
<th>whichinfo</th>
<th>type of *result_p</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_CALLBACK_INFO_PRIMAL_OBJ</td>
<td>double</td>
<td>primal objective value</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_DUAL_OBJ</td>
<td>double</td>
<td>dual objective value</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_PRIMAL_INFMEAS</td>
<td>double</td>
<td>measure of primal infeasibility</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_DUAL_INFMEAS</td>
<td>double</td>
<td>measure of dual infeasibility</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_PRIMAL_FEAS</td>
<td>int</td>
<td>1 if primal feasible, 0 if not</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_DUAL_FEAS</td>
<td>int</td>
<td>1 if dual feasible, 0 if not</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_ITCOUNT</td>
<td>int</td>
<td>iteration count</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_CROSSOVER_PPUSH</td>
<td>int</td>
<td>primal push crossover itn. count</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_CROSSOVER_PEXCH</td>
<td>int</td>
<td>primal exchange crossover itn. count</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_CROSSOVER_DPUSH</td>
<td>int</td>
<td>dual push crossover itn. count</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_CROSSOVER_DEXCH</td>
<td>int</td>
<td>dual exchange crossover itn. count</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_USER_PROBLEM</td>
<td>CPXCLPptr</td>
<td>returns pointer to original user problem; available for primal, dual, barrier, mip</td>
</tr>
</tbody>
</table>

For Network algorithms:

<table>
<thead>
<tr>
<th>whichinfo</th>
<th>type of *result_p</th>
<th>description</th>
</tr>
</thead>
</table>
### For Presolve algorithms:

<table>
<thead>
<tr>
<th>whichinfo</th>
<th>type of *result_p</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_CALLBACK_INFO_PRESOLVE</td>
<td>int</td>
<td>number of rows eliminated</td>
</tr>
<tr>
<td>LVE_ROWSGONE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_PRESOLVE</td>
<td>int</td>
<td>number of columns eliminated</td>
</tr>
<tr>
<td>LVE_COLSGONE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_PRESOLVE</td>
<td>int</td>
<td>number of aggregator substitutions</td>
</tr>
<tr>
<td>LVE_AGGSUBST</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_PRESOLVE</td>
<td>int</td>
<td>number of modified coefficients</td>
</tr>
<tr>
<td>LVE_COEFFS</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### For MIP algorithms:

<table>
<thead>
<tr>
<th>whichinfo</th>
<th>type of *result_p</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_CALLBACK_INFO_BEST_INTEGER</td>
<td>double</td>
<td>obj. value of best integer solution</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_BEST_REMAINING</td>
<td>double</td>
<td>obj. value of best remaining node</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_NODE_COUNT</td>
<td>int</td>
<td>total number of nodes solved</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_NODES_LEFT</td>
<td>int</td>
<td>number of remaining nodes</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_NODES_LEFT</td>
<td>int</td>
<td>number of remaining nodes</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_MIP_ITERATIONS</td>
<td>int</td>
<td>total number of MIP iterations</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_MIP_FEAS</td>
<td>int</td>
<td>returns 1 if feasible solution exists; otherwise, 0</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_CUTOFF</td>
<td>double</td>
<td>updated cutoff value</td>
</tr>
</tbody>
</table>
### CPXgetcallbackinfo

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_CALLBACK_INFO_CLIQU_E_COUNT</td>
<td>int</td>
<td>number of clique cuts added</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_COVER_COUNT</td>
<td>int</td>
<td>number of cover cuts added</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_DISJUNCTIVE_COUNT</td>
<td>int</td>
<td>number of disjunctive cuts added</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_FLOWCOVER_COUNT</td>
<td>int</td>
<td>number of flow cover cuts added</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFOFlowLayout_COUNT</td>
<td>int</td>
<td>number of flow path cuts added</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_FRACTIONAL_COUNT</td>
<td>int</td>
<td>number of Gomory fractional cuts added</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_GUBCOVER_COUNT</td>
<td>int</td>
<td>number of GUB cover cuts added</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_IMPLBOUND_COUNT</td>
<td>int</td>
<td>number of implied bound cuts added</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_MIXEDROUND_COUNT</td>
<td>int</td>
<td>number of mixed integer rounding cuts added</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_USER_PROBLEM</td>
<td>CPXCLPptr</td>
<td>returns pointer to original user problem; available for primal, dual, barrier, MIP</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_PROBE_PHASE</td>
<td>int</td>
<td>current phase of probing (0-3)</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_PROBE_PROGRESS</td>
<td>double</td>
<td>fraction of probing phase completed (0.0-1.0)</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_FRACTIONAL_PROGRESS</td>
<td>double</td>
<td>fraction of Gomory cut generation for the pass completed (0.0 - 1.0)</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_DISJUNCTIVE_PROGRESS</td>
<td>double</td>
<td>fraction of disjunctive cut generation for the pass completed (0.0 - 1.0)</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_FLOWMIR_PROGRESS</td>
<td>double</td>
<td>fraction of flow cover and MIR cut generation for the pass completed (0.0 - 1.0)</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_MY_THREAD</td>
<td>int</td>
<td>identifier of the parallel thread making this call</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_USER_THREADS</td>
<td>int</td>
<td>total number of parallel threads currently running</td>
</tr>
</tbody>
</table>

### Example

See lpex4.c in the *CPLEX User's Manual*. 
Suppose you want to know the objective value on each iteration for a graphical user display. In addition, if primal simplex is not feasible after 1000 iterations, you want to stop the optimization. The function `mycallback` is a callback function to do this.

```c
int mycallback (CPXCENVptr env, void *cbdata, int wherefrom, void *cbhandle)
{
    int itcount;
    double objval;
    int ispfeas;
    int status = 0;
    if ( wherefrom == CPX_CALLBACK_PRIMAL ) {
        status = CPXgetcallbackinfo (env, cbdata, wherefrom,
                                    CPX_CALLBACK_INFO_PRIMAL_FEAS,
                                    &ispfeas);
        if ( status ) {
            fprintf (stderr, "error %d in CPXgetcallbackinfo
                       status = 1;
                       goto TERMINATE;
        }
        if ( ispfeas ) {
            status = CPXgetcallbackinfo (env, cbdata, wherefrom,
                                          CPX_CALLBACK_INFO_PRIMAL_OBJ,
                                          &objval);
            if ( status ) {
                fprintf (stderr, "error %d in CPXgetcallbackinfo
                           status = 1;
                           goto TERMINATE;
            }
            /* Do some graphics with the value of objval */
        }
        else {
            status = CPXgetcallbackinfo (env, cbdata, wherefrom,
                                          CPX_CALLBACK_INFO_ITCOUNT, &itcount);
            if ( status ) {
                fprintf (stderr, "error %d in CPXgetcallbackinfo
                           status = 1;
                           goto TERMINATE;
            }
            if ( itcount > 1000 ) status = 1;
        }
    }
    TERMINATE:
    return (status);
}
```
Returns

The routine returns zero on success and nonzero if an error occurs. If nonzero, the requested value may not be available for the specific optimization algorithm. For example, the dual objective is not available from primal simplex.
CPXgetcallbacklp

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXgetcallbacklp(CPXENVptr env,
    void * cbdata,
    int wherefrom,
    CPXCLPptr * lp_p)
```

Description:

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetcallbacklp is used to get the pointer to the MIP problem that is in use when the user-written callback function is called. It is the original MIP if CPX_PARAM_MIPCBREDLP is set to CPX_OFF; otherwise, it is the presolved MIP. In contrast, the function CPXgetcallbacknodelp returns a pointer to the node subproblem, which is an LP. Generally, this pointer may be used only in CPLEX Callable Library query routines, such as CPXsolution or CPXgetrows.

This routine may be called only when the value of the `wherefrom` argument is one of the following:

- CPX_CALLBACK_MIP,
- CPX_CALLBACK_MIP_BRANCH,
- CPX_CALLBACK_MIP_INCUMBENT,
- CPX_CALLBACK_MIP_NODE,
- CPX_CALLBACK_MIP_HEURISTIC,
- CPX_CALLBACK_MIP_SOLVE, or
- CPX_CALLBACK_MIP_CUT.

Example:
status = CPXgetcallbacklp (env, cbdata, wherefrom, &origlp);

See also admipex1.c, admipex2.c, and admipex3.c in the standard distribution.

**Parameters**

- **env**
  A pointer to the CPLEX environment, as returned by CPXopenCplex.

- **cbdata**
  The pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.

- **wherefrom**
  An integer value indicating from where the user-written callback was called. The parameter must be the value of wherefrom passed to the user-written callback.

- **lp_p**
  A pointer to a variable of type CPXLPptr to receive the pointer to the LP problem object, which is a MIP.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXgetcallbacknodeinfo

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXgetcallbacknodeinfo(CPXENVptr env,
    void * cbdata,
    int wherefrom,
    int nodenum,
    int whichinfo,
    void * result_p)
```

**Description**

The routine `CPXgetcallbacknodeinfo` is called from within user-written callbacks during a MIP optimization and accesses information about nodes.

The primary purpose of this routine is to examine nodes in order to select one from which to proceed. In this case the `wherefrom` argument is `CPX_CALLBACK_MIP_NODE`, and a node with any `nodeindex` value can be queried. A secondary purpose of this routine is to obtain information about the current node. When the `wherefrom` argument is any one of the following values, only the current node can be queried.

- `CPX_CALLBACK_MIP_CUT`
- `CPX_CALLBACK_MIP_INCUMBENT`
- `CPX_CALLBACK_MIP_HEURISTIC`
- `CPX_CALLBACK_MIP_SOLVE`
- `CPX_CALLBACK_MIP_BRANCH`

To query the current node, specify a `nodeindex` value of 0. Other values of the `wherefrom` argument are invalid for this routine. An invalid `nodeindex` value or `wherefrom` argument value will result in an error return value.
Example

```c
status = CPXgetnodecallbackinfo(env,
        curlp,
        wherefrom,
        0,
        CPX_CALLBACK_INFO_NODE_SIINF,
        &numiinf);
```

**Note:** The values returned for CPX_CALLBACK_INFO_NODE_SIINF and CPX_CALLBACK_INFO_NODE_NIINF for the current node are the values that applied to the node when it was stored and thus before the branch was solved. As a result, these values should not be used to assess the feasibility of the node.

This routine cannot retrieve information about nodes that have been moved to node files. For more information about node files, see the *ILOG CPLEX User's Manual*. If the argument `nodeindex` refers to a node in a node file, CPXgetnodecallbackinfo returns the value CPXERR_NODE_ON_DISK. Nodes still in memory have the lowest index numbers so a user can loop through the nodes until CPXgetcallbacknodeinfo returns an error, and then exit the loop.

### Table 1: Information Requested for a User-Written Node Callback

<table>
<thead>
<tr>
<th>Symbolic Constant</th>
<th>C Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_CALLBACK_INFO_NODE_SIINF</td>
<td>double</td>
<td>sum of integer infeasibilities</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_NODE_NIINF</td>
<td>int</td>
<td>number of integer infeasibilities</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_NODE_ESTIMATE</td>
<td>double</td>
<td>estimated integer objective</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_NODE_DEPTH</td>
<td>int</td>
<td>depth of node in branch &amp; cut tree</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_NODE_OBJJVAL</td>
<td>double</td>
<td>objective value of LP subproblem</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_NODE_TYPE</td>
<td>int</td>
<td>type of branch at this node; see Table 2</td>
</tr>
</tbody>
</table>
Table 1: Information Requested for a User-Written Node Callback

<table>
<thead>
<tr>
<th>Symbolic Constant</th>
<th>Value</th>
<th>Branch Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_CALLBACK_INFO_NODE_VAR</td>
<td>int</td>
<td>for nodes of type CPX_TYPE_VAR, the branching variable for this node; for other types, -1 is returned</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_NODE_SOS</td>
<td>int</td>
<td>for nodes of type CPX_TYPE_SOS1 or CPX_TYPE_SOS2 the number of the SOS used in branching; -1 otherwise</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_NODE_SEQNUM</td>
<td>int</td>
<td>sequence number of the node</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_NODE_NODENUM</td>
<td>int</td>
<td>node index of the node (only available for CPXgetcallbackseqinfo)</td>
</tr>
</tbody>
</table>

Table 2: Branch Types Returned when whichinfo = CPX_CALLBACK_INFO_NODE_TYPE

See also Advanced MIP Control Interface in the ILOG CPLEX User’s Manual.

See Also

CPXgetcallbackinfo, CPXgetcallbackseqinfo

Parameters

env

A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbd

The pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.

wherefrom

An integer value indicating where the user-written callback was called from. This parameter must be the value of wherefrom passed to the user-written callback.
nodenum

The index of the node for which information is requested. Nodes are indexed from 0 (zero) to (nodecount - 1) where nodecount is obtained from the callback information function CPXgetcallbackinfo, with a whichinfo value of CPX_CALLBACK_INFO_NODES_LEFT.

whichinfo

An integer indicating which information is requested. Table 1 summarizes the possible values. Table 2 summarizes possible values returned when the type of information requested is branch type (that is, whichinfo = CPX_CALLBACK_INFO_NODE_TYPE).

result_p

A generic pointer to a variable of type double or int, representing the value returned by whichinfo. (The column “C Type” in Table 1 indicates the type of various values returned by whichinfo.)

Returns

The routine returns zero on success and nonzero if an error occurs. The return value CPXERR_NODE_ON_DISK indicates an attempt to access a node currently located in a node file on disk.
CPXgetcallbacknodeintfeas

Category: Global Function
Definition File: cplex.h
Include Files: cplex.h
Synopsis:

```c
int CPXPUBLIC CPXgetcallbacknodeintfeas(CPXENVptr env,
                                        void * cbdata,
                                        int wherefrom,
                                        int * feas,
                                        int begin,
                                        int end)
```

Description:

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetcallbacknodeintfeas is used to get indicators for each variable of whether or not the variable is integer feasible in the node subproblem. It can be used in a user-written callback during MIP optimization. The indicators are from the original problem if CPX_PARAM_MIPCBREDLP is set to CPX_OFF. Otherwise, they are from the presolved problem.

**Example**

```c
status = CPXgetcallbacknodeintfeas(env, cbdata, wherefrom, feas, 0, cols-1);
```

See admipex1.c and admipex2.c in the standard distribution.

This routine may be called only when the value of the wherefrom argument is one of the following:

- CPX_CALLBACK_MIP,
- CPX_CALLBACK_MIP_BRANCH,
- CPX_CALLBACK_MIP_INCUMBENT,
- CPX_CALLBACK_MIP_NODE,
◆ CPX_CALLBACK_MIP_HEURISTIC, or  
◆ CPX_CALLBACK_MIP_CUT.

**Indicators of feasibility for a node of the subproblem**

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_INTEGER_FEASIBLE</td>
<td>0</td>
<td>variable j+begin is integer-valued</td>
</tr>
<tr>
<td>CPX_INTEGER_INFEASIBLE</td>
<td>1</td>
<td>variable j+begin is not integer-valued</td>
</tr>
<tr>
<td>CPX_IMPLIED_INTEGER_FEASIBLE</td>
<td>2</td>
<td>variable j+begin may have a fractional value in the current solution, but it will take on an integer value when all integer variables still in the problem have integer values. It should not be branched upon.</td>
</tr>
</tbody>
</table>

**Parameters**

- **env**
  A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

- **cbdata**
  The pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.

- **wherefrom**
  An integer value indicating from where the user-written callback was called. The parameter must be the value of wherefrom passed to the user-written callback.

- **feas**
  An array to receive an indicator of feasibility for the node subproblem. This array must be of length at least (end - begin + 1). If successful, feas[0] through feas[end-begin] will contain the indicators. The indicators of feasibility for a node of the subproblem appear in the table.

- **begin**
  An integer indicating the beginning of the range of feasibility indicators to be returned.

- **end**
  An integer indicating the end of the range of feasibility indicators to be returned.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
**CPXgetcallbacknodelb**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXgetcallbacknodelb(CPXENVptr env,
       void * cbdata,
       int wherefrom,
       double * lb,
       int begin,
       int end)
```

**Description**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetcallbacknodelb is used to get the lower bound values for the subproblem at the current node during MIP optimization from within a user-written callback. The values are from the original problem if CPX_PARAM_MIPCBREDLP is set to CPX_OFF; otherwise, they are from the presolved problem.

This routine may be called only when the value of the wherefrom argument is one of the following:

- CPX_CALLBACK_MIP,
- CPX_CALLBACK_MIP_BRANCH,
- CPX_CALLBACK_MIP_INCUMBENT,
- CPX_CALLBACK_MIP_NODE,
- CPX_CALLBACK_MIP_HEURISTIC,
- CPX_CALLBACK_MIP_SOLVE, or
- CPX_CALLBACK_MIP_CUT.

**Example**
status = CPXgetcallbacknodeLB (env, cbdata, wherefrom, lb, 0, cols-1);

Parameters

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbdata
The pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.

wherefrom
An integer value indicating from where the user-written callback was called. The parameter must be the value of wherefrom passed to the user-written callback.

lb
An array to receive the values of the lower bound values. This array must be of length at least (end - begin + 1). If successful, lb[0] through lb[end-begin] contain the lower bound values for the current subproblem.

begin
An integer indicating the beginning of the range of lower bounds to be returned.

dend
An integer indicating the end of the range of lower bounds to be returned.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXgetcallbacknodelp

Category                      Global Function
Definition File              cplex.h
Include Files                cplex.h
Synopsis                      int CPXPUBLIC CPXgetcallbacknodelp(CPXCENVptr env,
                               void * cbdata,
                               int wherefrom,
                               CPXLPptr * nodelp_p)

Description

Note: This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetcallbacknodelp accesses the lp pointer indicating the currently defined linear programming subproblem (LP) from within user-written callbacks. Generally, this pointer may be used only in ILOG CPLEX Callable Library query routines, such as CPXsolution or CPXgetrows.

Example

    status = CPXgetcallbacknodelp (env, cbdata, &nodelp);

See also the example admipex1.c and admipex6.c in the standard distribution.

CPXgetcallbacknodelp may be called only when its wherefrom argument has one of the following values:

- CPX_CALLBACK_MIP,
- CPX_CALLBACK_MIP_BRANCH,
- CPX_CALLBACK_MIP_CUT,
- CPX_CALLBACK_MIP_HEURISTIC,
- CPX_CALLBACK_MIP_INCUMBENT, or
- CPX_CALLBACK_MIP_SOLVE.
When the `wherefrom` argument has the value `CPX_CALLBACK_MIP_SOLVE`, the subproblem pointer may also be used in ILOG CPLEX optimization routines.

**Note:** *Any modification to the subproblem may result in corruption of the problem and of the ILOG CPLEX environment.*

### Parameters

- **env**
  
  A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.

- **cbdata**
  
  The `cbdata` pointer passed to the user-written callback. This parameter must be the value of `cbdata` passed to the user-written callback.

- **wherefrom**
  
  An integer value indicating where the user-written callback was called from. This parameter must be the value of the `wherefrom` passed to the user-written callback.

- **nodelp**
  
  The `lp` pointer indicating the current subproblem. If no subproblem is defined, the pointer is set to NULL.

### Returns

The routine returns zero on success and nonzero if an error occurs. A nonzero return value may mean that the requested value is not available.
**CPXgetcallbacknodeobjval**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXgetcallbacknodeobjval (CPXCENVptr env,
    void * cbdata,
    int wherefrom,
    double * objval_p)
```

**Description**

- **Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetcallbacknodeobjval` is used to get the objective value for the subproblem at the current node during MIP optimization from within a user-written callback.

This routine may be called only when the value of the `wherefrom` argument is one of the following:

- `CPX_CALLBACK_MIP`,
- `CPX_CALLBACK_MIP_BRANCH`,
- `CPX_CALLBACK_MIP_INCUMBENT`,
- `CPX_CALLBACK_MIP_NODE`,
- `CPX_CALLBACK_MIP_HEURISTIC`, or
- `CPX_CALLBACK_MIP_CUT`.

**Example**

```c
status = CPXgetcallbacknodeobjval (env, cbdata, wherefrom, &objval);
```

See also `admipex1.c` and `admipex3.c` in the standard distribution.
Parameters

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbdata
The pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.

wherefrom
An integer value indicating from where the user-written callback was called. The parameter must be the value of wherefrom passed to the user-written callback.

objval_p
A pointer to a variable of type double where the objective value of the node subproblem is to be stored.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXgetcallbacknodestat

Category Global Function

Definition File cplex.h

Include Files cplex.h

Synopsis

```c
int CPXPUBLIC CPXgetcallbacknodestat (CPXCENVptr env,
                                void * cbdata,
                                int wherefrom,
                                int * nodestat_p)
```

Description

The routine `CPXgetcallbacknodestat` is used to get the optimization status of the subproblem at the current node from within a user-written callback during MIP optimization.

The optimization status will be either optimal or unbounded. An unbounded status can occur when some of the constraints are being treated as lazy constraints. When the node status is unbounded, then the function `CPXgetcallbacknodex` returns a ray that can be used to decide which lazy constraints need to be added to the subproblem.

This routine may be called only when the value of the `wherefrom` argument is `CPX_CALLBACK_MIP_CUT`.

Example

```c
status = CPXgetcallbacknodestat (env, cbdata, wherefrom,
                              &nodestatus);
```

Parameters

- **env**
  
  A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.

- **cbdata**
  
  The pointer passed to the user-written callback. This parameter must be the value of `cbdata` passed to the user-written callback.

Note: This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.
wherefrom

An integer value indicating from where the user-written callback was called. The parameter must be the value of *wherefrom* passed to the user-written callback.

nodestat_p

A pointer to an integer where the node subproblem optimization status is to be returned. The values of *nodestat_p* may be CPX_STAT_OPTIMAL or CPX_STAT_UNBOUNDED.

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXgetcallbacknodeub

Category Global Function

Definition File cplex.h

Include Files cplex.h

Synopsis

```
int CPXPUBLIC CPXgetcallbacknodeub(CPXCENVptr env,
void * cbdata,
int wherefrom,
double * ub,
int begin,
int end)
```

Description

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetcallbacknodeub is used to get the upper bound values for the subproblem at the current node during MIP optimization from within a user-written callback. The values are from the original problem if CPX_PARAM_MIPCBREDLP is set to CPX_OFF; otherwise, they are from the presolved problem.

This routine may be called only when the value of the wherefrom argument is one of the following:

- CPX_CALLBACK_MIP,
- CPX_CALLBACK_MIP_BRANCH,
- CPX_CALLBACK_MIP_INCUMBENT,
- CPX_CALLBACK_MIP_NODE,
- CPX_CALLBACK_MIP_HEURISTIC,
- CPX_CALLBACK_MIP_SOLVE, or
- CPX_CALLBACK_MIP_CUT.

Example
status = CPXgetcallbacknodeub (env, cbdata, wherefrom, ub, 0, cols-1);

Parameters

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbdata
The pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.

wherefrom
An integer value indicating from where the user-written callback was called. The parameter must be the value of wherefrom passed to the user-written callback.

ub
An array to receive the values of the upper bound values. This array must be of length at least (end - begin + 1). If successful, ub[0] through ub[end-begin] contain the upper bound values for the current subproblem.

begin
An integer indicating the beginning of the range of upper bound values to be returned.

end
An integer indicating the end of the range of upper bound values to be returned.

Returns
The routine returns zero on success and nonzero if an error occurs.
**CPXgetcallbacknodex**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXgetcallbacknodex (CPXCENVptr env,  
   void * cbdata,  
   int wherefrom,  
   double * x,  
   int begin,  
   int end)
```

**Description**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetcallbacknodex is used to get the primal variable (x) values for the subproblem at the current node during MIP optimization from within a user-written callback. The values are from the original problem if CPX_PARAM_MIPCBREDLP is set to CPX_OFF; otherwise, they are from the presolved problem.

This routine may be called only when the value of the `wherefrom` argument is one of the following:

- CPX_CALLBACK_MIP,
- CPX_CALLBACK_MIP_BRANCH,
- CPX_CALLBACK_MIP_INCUMBENT,
- CPX_CALLBACK_MIP_NODE,
- CPX_CALLBACK_MIP_HEURISTIC, or
- CPX_CALLBACK_MIP_CUT.

**Example**

```c
status = CPXgetcallbacknodex (env, cbdata, wherefrom,  
   nodex, 0, cols-1);
```
See also admipex1.c, admipex3.c, and admipex5.c in the standard distribution.

**Parameters**

- **env**
  A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

- **cbdata**
  The pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.

- **wherefrom**
  An integer value indicating from where the user-written callback was called. The parameter must be the value of wherefrom passed to the user-written callback.

- **x**
  An array to receive the values of the primal variables for the node subproblem. This array must be of length at least \((end - begin + 1)\). If successful, \(x[0]\) through \(x[end-begin]\) contain the primal values.

- **begin**
  An integer indicating the beginning of the range of primal variable values for the node subproblem to be returned.

- **end**
  An integer indicating the end of the range of primal variable values for the node subproblem to be returned.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
**CPXgetcallbackorder**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**
int CPXPUBLIC CPXgetcallbackorder(CPXENVptr env,  
void * cbdata,  
int wherefrom,  
int * priority,  
int * direction,  
int begin,  
int end)

**Description**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetcallbackorder is used to get MIP priority order information during MIP optimization from within a user-written callback. The values are from the original problem if CPX_PARAM_MIPCBREDLP is set to CPX_OFF. Otherwise, they are from the presolved problem.

This routine may be called only when the value of the wherefrom argument is one of the following values:

◆ CPX_CALLBACK_MIP,
◆ CPX_CALLBACK_MIP_BRANCH,
◆ CPX_CALLBACK_MIP_INCUMBENT,
◆ CPX_CALLBACK_MIP_NODE,
◆ CPX_CALLBACK_MIP_HEURISTIC,
◆ CPX_CALLBACK_MIP_SOLVE, or
◆ CPX_CALLBACK_MIP_CUT.

**Example**
status = CPXgetcallbackorder (env, cbdata, wherefrom, 
    priority, NULL, 0, cols-1);

## Branching direction

<table>
<thead>
<tr>
<th>Branching Direction</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_BRANCH_GLOBAL</td>
<td>0</td>
<td>use global branching direction setting CPX_PARAM_BRDIR</td>
</tr>
<tr>
<td>CPX_BRANCH_DOWN</td>
<td>-1</td>
<td>branch down first on variable j+begin</td>
</tr>
<tr>
<td>CPX_BRANCH_UP</td>
<td>1</td>
<td>branch up first on variable j+begin</td>
</tr>
</tbody>
</table>

### Parameters

**env**
- A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

**cbdata**
- The pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.

**wherefrom**
- An integer value indicating from where the user-written callback was called. The parameter must be the value of wherefrom passed to the user-written callback.

**priority**
- An array where the priority values are to be returned. This array must be of length at least (end - begin + 1). If successful, priority[0] through priority[end-begin] contain the priority order values. May be NULL. The value of direction[j] will be a value from the table of branching directions.

**begin**
- An integer indicating the beginning of the range of priority order information to be returned.

**end**
- An integer indicating the end of the range of priority order information to be returned.

### Returns

The routine returns zero on success and nonzero if an error occurs.
CPXgetcallbackpseudocosts

Category          Global Function
Definition File   cplex.h
Include Files     cplex.h
Synopsis          int CPXPUBLIC CPXgetcallbackpseudocosts(CPXENVptr env,
                                          void * cbdata,
                                          int wherefrom,
                                          double * uppc,
                                          double * downpc,
                                          int begin,
                                          int end)

Description       Note: This is an advanced routine. Advanced routines typically demand a profound
                  understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

                  The routine CPXgetcallbackpseudocosts is used to get the pseudo-cost values during MIP optimization from within a user-written callback. The values are from the original problem if CPX_PARAM_MIPCBREDLP is set to CPX_OFF. Otherwise, they are from the presolved problem.

                  Note: When pseudo-costs are retrieved for the original problem variables, pseudo-costs are zero for variables that have been removed from the problem, since they are never used for branching.

                  This routine may be called only when the value of the wherefrom argument is one of the following:
                  ♦ CPX_CALLBACK_MIP,
                  ♦ CPX_CALLBACK_MIP_BRANCH,
                  ♦ CPX_CALLBACK_MIP_INCUMBENT,
                  ♦ CPX_CALLBACK_MIP_NODE,
◆ CPX_CALLBACK_MIP_HEURISTIC, 
◆ CPX_CALLBACK_MIP_SOLVE, or 
◆ CPX_CALLBACK_MIP_CUT.

Example

status = CPXgetcallbackpseudocosts (env, cbdata, wherefrom, 
                                        upcost, downcost, 
                                        j, k);

Parameters

env

A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbdata

The pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.

wherefrom

An integer value indicating from where the user-written callback was called. The parameter must be the value of wherefrom passed to the user-written callback.

uppc

An array to receive the values of up pseudo-costs. This array must be of length at least (end - begin + 1). If successful, uppc[0] through uppc[end-begin] will contain the up pseudo-costs. May be NULL.

downpc

An array to receive the values of the down pseudo-costs. This array must be of length at least (end - begin + 1). If successful, downpc[0] through downpc[end-begin] will contain the down pseudo-costs. May be NULL.

begin

An integer indicating the beginning of the range of pseudo-costs to be returned.

dend

An integer indicating the end of the range of pseudo-costs to be returned.

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXgetcallbackseqinfo

Category                                    Global Function
Definition File                             cplex.h
Include Files                              cplex.h

Synopsis

int CPXPUBLIC CPXgetcallbackseqinfo(CPXCENVptr env,
void * cbdata,
int wherefrom,
int seqid,
int whichinfo,
void * result_p)

Description

The routine CPXgetcallbackseqinfo accesses information about nodes during the MIP optimization from within user-written callbacks. This routine may be called only when the value of its wherefrom argument is CPX_CALLBACK_MIP_NODE. The information accessed from this routine can also be accessed with the routine CPXgetcallbacknodeinfo. Nodes are not stored by sequence number but by node number, so using the routine CPXgetcallbackseqinfo can be much more time-consuming than using the routine CPXgetcallbacknodeinfo. A typical use of this routine would be to obtain the node number of a node for which the sequence number is known and then use that node number to select the node with the node callback.

This argument must be the value of cbdata passed to the user-written callback.

Note: This routine cannot retrieve information about nodes that have been moved to node files. (For more information about node files, see the ILOG CPLEX User's Manual.) If the argument seqnum refers to a node in a node file, CPXgetcallbacknodeinfo returns the value CPXERR_NODE_ON_DISK.

This argument must be the value of wherefrom passed to the user-written callback.

For a summary of possible values, refer to the table titled Information Requested for a User-Written Node Callback in the description of CPXgetcallbacknodeinfo.

Parameters

env

A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbdata
The pointer passed to the user-written callback. This argument must be the value of cbdata passed to the user-written callback.

wherefrom

An integer value indicating where the user-written callback was called from. This argument must be the value of wherefrom passed to the user-written callback.

seqid

The sequence number of the node for which information is requested.

whichinfo

An integer indicating which information is requested. For a summary of possible values, refer to the table titled \textit{Information Requested for a User-Written Node Callback} in the description of CPXgetcallbacknodeinfo.

result_p

A generic pointer to a variable of type double or int. The variable represents the value returned by whichinfo. The column \textit{C Type} in the table titled \textit{Information Requested for a User-Written Node Callback} indicates the type of various values returned by whichinfo.

**Returns**

The routine returns zero on success and nonzero if an error occurs. The return value CPXERR_NODE_ON_DISK indicates an attempt to access a node currently located in a node file on disk.
CPXgetcallbacksosinfo

Category Global Function

Definition File cplex.h

Include Files cplex.h

Synopsis

int CPXPUBLIC CPXgetcallbacksosinfo(CPXCENVptr env,
void * cbdata,
int wherefrom,
int sosindex,
int member,
int whichinfo,
void * result_p)

Description

The routine CPXgetcallbacksosinfo accesses information about special ordered sets (SOSs) during MIP optimization from within user-written callbacks. This routine may be called only when the value of its wherefrom argument is one of these values:

- CPX_CALLBACK_MIP_HEURISTIC,
- CPX_CALLBACK_MIP_BRANCH,
- CPX_CALLBACK_MIP_INCUMBENT, or
- CPX_CALLBACK_MIP_CUT.

The information returned is for the original problem if the parameter CPX_PARAM_MIPCBREDLP is set to CPX_OFF before the call to CPXmipopt that calls the callback. Otherwise, it is for the presolved problem.

Example

status = CPXgetcallbacksosinfo(env, curlp, wherefrom, 6, 4,
CPX_CALLBACK_INFO_SOS_IS_FEASIBLE,
&isfeasible);

See also the example admipex3.c in the standard distribution.

Table 1: Information Requested for a User-Written SOS Callback

<table>
<thead>
<tr>
<th>Symbolic Constant</th>
<th>C Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_CALLBACK_INFO_SOS_NUM</td>
<td>int</td>
<td>number of SOSs</td>
</tr>
</tbody>
</table>
Table 1: Information Requested for a User-Written SOS Callback

<table>
<thead>
<tr>
<th>CPX_CALLBACK_INFO_SOS_TYPE</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_CALLBACK_INFO_SOS_SIZE</td>
<td>int</td>
<td>Size of SOS</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_SOS_IS_FEASIBLE</td>
<td>int</td>
<td>1 if SOS is feasible, 0 if SOS is not</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_SOS_PRIORITY</td>
<td>int</td>
<td>Priority value of SOS</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_SOS_MEMBER_INDEX</td>
<td>int</td>
<td>Variable index of member of SOS</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_SOS_MEMBER_REFVAL</td>
<td>double</td>
<td>Reference value (weight) of this member</td>
</tr>
</tbody>
</table>

Table 2: SOS Types Returned when whichinfo = CPX_CALLBACK_INFO_SOS_TYPE

<table>
<thead>
<tr>
<th>Symbolic Constant</th>
<th>SOS Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_SOS1</td>
<td>type 1</td>
</tr>
<tr>
<td>CPX_SOS2</td>
<td>type 2</td>
</tr>
</tbody>
</table>

Parameters

**env**
A pointer to the CPLEX environment, as returned by CPXopenCplex.

**cbdata**
The pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.

**wherefrom**
An integer value indicating where the user-written callback was called from. This parameter must be the value of wherefrom passed to the user-written callback.

**sosindex**
The index of the special ordered set (SOS) for which information is requested. SOSs are indexed from zero to (numsets - 1) where numsets is the result of calling this routine with a whichinfo value of CPX_CALLBACK_INFO_SOS_NUM.

**member**
The index of the member of the SOS for which information is requested.

**whichinfo**
An integer indicating which information is requested. Table 1 summarizes the possible values. Table 2 summarizes possible values returned when the type of information
requested is the SOS type (that is, \texttt{whichinfo = CPX_CALLBACK_INFO_SOS_TYPE}).
\begin{verbatim}
result_p
\end{verbatim}
A generic pointer to a variable of type \texttt{double}, \texttt{int}, or \texttt{char}. The variable represents
the value returned by \texttt{whichinfo}. (The column C Type in the table indicates the type
of various values returned by \texttt{whichinfo}.)

**Returns**

This routine returns zero on success and nonzero if an error occurs. If the return value is nonzero, the requested value may not be available.
CPXgetchannels

Category          Global Function
Definition File   cplex.h
Include Files    cplex.h
Synopsis          int CPXPUBLIC CPXgetchannels(CPXENVptr env,  
                  CPXCHANNELptr * cpxresults_p,  
                  CPXCHANNELptr * cpxwarning_p,  
                  CPXCHANNELptr * cpxerror_p,  
                  CPXCHANNELptr * cpxlog_p)

Description       The routine CPXgetchannels obtains pointers to the four  default channels created when CPXopenCPLEX is called. To manipulate the messages for any of these channels, this routine must be called.

Example           status = CPXgetchannels (env, &cpxresults, &cpxwarning,  
                  &cpxerror, &cpxlog);

See also lpex5.c in the ILOG CPLEX User's Manual.

Parameters        env
                  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

                  cpxresults_p
                  A pointer to a variable of type CPXCHANNELptr to hold the address of the channel  
                  corresponding to cpxresults. May be NULL.

                  cpxwarning_p
                  A pointer to a variable of type CPXCHANNELptr to hold the address of the channel  
                  corresponding to cpxwarning. May be NULL.

                  cpxerror_p
                  A pointer to a variable of type CPXCHANNELptr to hold the address of the channel  
                  corresponding to cpxerror. May be NULL.

                  cpxlog_p
                  A pointer to a variable of type CPXCHANNELptr to hold the address of the channel  
                  corresponding to cpxlog. May be NULL.

Returns          The routine returns zero on success and nonzero if an error occurs.
CPXgetclqcnt

Category Global Function

Definition File cplex.h

Include Files cplex.h

Synopsis int CPXPUBLIC CPXgetclqcnt (CPXCENVptr env, CPXCLPptr lp)

Description The routine CPXgetclqcnt is used to access the number of clique cuts added to the problem object during mixed integer optimization.

Example clique_cuts = CPXgetclqcnt (env, lp);

Parameters

env A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp A pointer to a CPLEX problem object as returned by CPXcreateprob.

Example clique_cuts = CPXgetclqcnt (env, lp);

Returns This routine returns zero if no solution, problem, or environment exists. Otherwise, the number of clique cuts added is returned.
CPXgetcoef

Category    Global Function
Definition File    cplex.h
Include Files    cplex.h

Synopsis

```
int CPXPUBLIC CPXgetcoef (CPXCENVptr env,
     CPXCLPptr lp,
     int i,
     int j,
     double * coef_p)
```

Description

The routine CPXgetcoef is used to access a single constraint matrix coefficient of a CPLEX problem object. The row and column indices must be specified.

Example

```
status = CPXgetcoef (env, lp, 10, 20, &coef);
```

Parameters

- env
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- lp
  A pointer to a CPLEX problem object as returned by CPXcreateprob.
- i
  An integer indicating the numeric index of the row.
- j
  An integer indicating the numeric index of the column.
- coef_p
  A pointer to a double to contain the specified matrix coefficient.

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXgetcolindex

Category          Global Function
Definition File   cplex.h
Include Files     cplex.h
Synopsis
int CPXPUBLIC CPXgetcolindex(CPXENVptr env, 
                              CPXCLPptr lp, 
                              const char * lname_str, 
                              int * index_p)
Description
The routine CPXgetcolindex searches for the index number of the specified column 
in a CPLEX problem object.
Example
    status = CPXgetcolindex (env, lp, "power43", &colindex);
Parameters
env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.
lname_str
A column name to search for.
index_p
A pointer to an integer to hold the index number of the column with name lname_str. 
If the routine is successful, *index_p contains the index number; otherwise, 
*index_p is undefined.
Returns
The routine returns zero on success and nonzero if an error occurs.
CPXgetcolName

Category       Global Function
Definition File cplex.h
Include Files  cplex.h

Synopsis

int CPXPUBLIC CPXgetcolname (CPXCENVptr env,
              CPXCLPptr lp,
              char ** name,
              char * namestore,
              int storespace,
              int * surplus_p,
              int begin,
              int end)

Description

The routine CPXgetcolname is used to access a range of column names or, equivalently, the variable names of a CPLEX problem object. The beginning and end of the range, along with the length of the array in which the column names are to be returned, must be specified.

Note: If the value of storespace is 0, the negative of the value of *surplus_p returned indicates the total number of characters needed for the array namestore.

Example

status = CPXgetcolname (env, lp, cur_colname, cur_colnamestore,
                         cur_storespace, &surplus, 0,
                         cur_numcols-1);

See also the example lpex7.c in the ILOG CPLEX User’s Manual and in the standard distribution.

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

name
An array of pointers to the column names stored in the array namestore. This array must be of length at least \((\text{end} - \text{begin} + 1)\). The pointer to the name of column \(j\) is returned in \(\text{name}[j - \text{begin}]\).

namestore

An array of characters where the specified column names are to be returned. May be NULL if storespace is 0.

storespace

An integer indicating the length of the array namestore. May be 0.

surplus_p

A pointer to an integer to contain the difference between storespace and the total amount of memory required to store the requested names. A nonnegative value of *surplus_p indicates that storespace was sufficient. A negative value indicates that it was insufficient and that the routine could not complete its task. In that case, CPXgetcolname returns the value CPXERR_NEGATIVE_SURPLUS, and the negative value of the variable *surplus_p indicates the amount of insufficient space in the array namestore.

begin

An integer indicating the beginning of the range of column names to be returned.

end

An integer indicating the end of the range of column names to be returned.

Returns

The routine returns zero on success and nonzero if an error occurs. The value CPXERR_NEGATIVE_SURPLUS indicates that insufficient space was available in the namestore array to hold the names.
**CPXgetcols**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXgetcols(CPXENVptr env,  
CPXCLPptr lp,  
int * nzcnt_p,  
int * cmatbeg,  
int * cmatind,  
double * cmatval,  
int cmatspace,  
int * surplus_p,  
int begin,  
int end)
```

**Description**  
The routine `CPXgetcols` is used to access a range of columns of the constraint matrix of a CPLEX problem object. The beginning and end of the range, along with the length of the arrays in which the nonzero entries of these columns are to be returned, must be specified.

**Note:** *If the value of cmatspace is zero, the negative of the value of *surplus_p* returned indicates the length needed for the arrays cmatind and cmatval.*

**Example**  
```c
status = CPXgetcols (env, lp, &nzcnt, cmatbeg, cmatind,  
cmatval, cmatspace, &surplus, 0,  
cur_numcols-1);
```

**Parameters**

- **env**  
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **lp**  
A pointer to a CPLEX problem object as returned by CPXcreateprob.

- **nzcnt_p**  
An integer pointing to the number of nonzeros in the constraint matrix.

- **cmatbeg**  
An integer array pointing to the beginning of each row in the constraint matrix.

- **cmatind**  
An integer array pointing to the indices of the nonzero entries in each row.

- **cmatval**  
A double array pointing to the values of the nonzero entries in each row.

- **cmatspace**  
The space available for the arrays.

- **surplus_p**  
The surplus of the problem.

- **begin**  
The beginning of the range.

- **end**  
The end of the range.
A pointer to an integer to contain the number of nonzeros returned; that is, the true length of the arrays cmatind and cmatval.

```plaintext
cmatbeg
```

An array to contain indices indicating where each of the requested columns begins in the arrays cmatval and cmatind. Specifically, column \( j \) consists of the entries in cmatval and cmatind in the range from cmatbeg[\( j - begin \)] to cmatbeg[\( (j + 1) - begin \)] - 1. (Column end consists of the entries from cmatbeg[end - begin] to nzcnt_p - 1.) This array must be of length at least (end - begin + 1).

```plaintext
cmatind
```

An array to contain the row indices associated with the elements of cmatval. May be NULL if cmatspace is zero.

```plaintext
cmatval
```

An array to contain the nonzero coefficients of the specified columns. May be NULL if cmatspace is zero.

```plaintext
cmatspace
```

An integer indicating the length of the arrays cmatind and cmatval. May be zero.

```plaintext
surplus_p
```

A pointer to an integer to contain the difference between cmatspace and the number of entries in each of the arrays cmatind and cmatval. A nonnegative value of *surplus_p indicates that the length of the arrays was sufficient. A negative value indicates that the length was insufficient and that the routine could not complete its task. In this case, CPXgetcols returns the value CPXERR_NEGATIVE_SURPLUS, and the negative value of *surplus_p indicates the amount of insufficient space in the arrays.

```plaintext
begin
```

An integer indicating the beginning of the range of columns to be returned.

```plaintext
end
```

An integer indicating the end of the range of columns to be returned.

### Returns

The routine returns zero on success and nonzero if an error occurs. The value CPXERR_NEGATIVE_SURPLUS indicates that insufficient space was available in the arrays cmatind and cmatval to hold the nonzero coefficients.
CPXgetcovcnt

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis: int CPXPUBLIC CPXgetcovcnt (CPXCENVptr env, CPXCLPptr lp)

Description: The routine CPXgetcovcnt is used to access the number of cover cuts added to the problem object during mixed integer optimization.

Example:

```
cover_cuts = CPXgetcovcnt (env, lp);
```

Parameters:

- env
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- lp
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

Example:

```
cover_cuts = CPXgetcovcnt (env, lp);
```

Returns: The routine returns zero if no solution, problem, or environment exists. Otherwise, the number of cover cuts added is returned.
**CPXgetcrossdxchcnt**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXgetcrossdxchcnt(CPXENVptr env, 
     CPXCLPptr lp)
```

**Description**  
The routine **CPXgetcrossdxchcnt** is used to access the number of dual exchange iterations in the crossover method. An exchange occurs when a nonbasic variable is forced to enter the basis as it is pushed toward a bound.

**Example**
```c
itcnt = CPXgetcrossdxchcnt (env, lp);
```

**Parameters**

- **env**  
A pointer to the CPLEX environment as returned by **CPXopenCPLEX**.

- **lp**  
A pointer to a CPLEX problem object as returned by **CPXcreateprob**.

**Example**
```c
itcnt = CPXgetcrossdxchcnt (env, lp);
```

**Returns**  
The routine returns the dual exchange iteration count if a solution exists. If no solution exists, it returns zero.
**CPXgetcrossdpushcnt**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```
int CPXPUBLIC CPXgetcrossdpushcnt (CPXCENVptr env,  
CPXCLPptr lp)
```

**Description**  
The routine `CPXgetcrossdpushcnt` is used to access the number of dual push iterations in the crossover method. A push occurs when a nonbasic variable switches bounds and does not enter the basis.

**Example**  
```
itcnt = CPXgetcrossdpushcnt (env, lp);
```

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- **lp**
  A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

**Example**  
```
itcnt = CPXgetcrossdpushcnt (env, lp);
```

**Returns**  
The routine returns the dual push iteration count if a solution exists. If no solution exists, it returns zero.
**CPXgetcrosspexchcnt**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXgetcrosspexchcnt (CPXCENVptr env, 
                                  CPXCLPptr lp)
```

**Description**
The routine CPXgetcrosspexchcnt is used to access the number of primal exchange iterations in the crossover method. An exchange occurs when a nonbasic variable is forced to enter the basis as it is pushed toward a bound.

**Example**

```c
itcnt = CPXgetcrosspexchcnt (env, lp);
```

**Parameters**

- `env`:
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- `lp`:
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

**Example**

```c
itcnt = CPXgetcrosspexchcnt (env, lp);
```

**Returns**
The routine returns the primal exchange iteration count if a solution exists. If no solution exists, it returns zero.
**CPXgetcrossppushcnt**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXgetcrossppushcnt(CPXENVptr env,
                                 CPXCLPptr lp)
```

**Description**  
The routine CPXgetcrossppushcnt is used to access the number of primal push iterations in the crossover method. A push occurs when a nonbasic variable switches bounds and does not enter the basis.

**Example**  
```c
itcnt = CPXgetcrossppushcnt (env, lp);
```

**Parameters**  
- **env**  
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **lp**  
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

**Example**  
```c
itcnt = CPXgetcrossppushcnt (env, lp);
```

**Returns**  
The routine returns the primal push iteration count if a solution exists. If no solution exists, it returns zero.
CPXgetctype

Category Global Function

Definition File cplex.h

Include Files cplex.h

Synopsis

```c
int CPXPUBLIC CPXgetctype(CPXENVptr env,
    CPXCLPptr lp,
    char * xctype,
    int begin,
    int end)
```

Description

The routine CPXgetctype is used to access the types for a range of variables in a problem object. The beginning and end of the range must be specified.

Example

```c
status = CPXgetctype (env, lp, ctype, 0, cur_numcols-1);
```

See Also

CPXcopyctypetype

Parameters

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **lp**
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

- **xctype**
  An array where the specified types are to be returned. This array must be of length (end-begin+1). The type of variable \( j \) is returned in \( \text{ctype}[j\text{-begin}] \). See the routine CPXcopyctypetype for a list of possible values for the variables in \( \text{ctype} \).

- **begin**
  An integer indicating the beginning of the range of types to be returned

- **end**
  An integer indicating the end of the range of types to be returned.

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXgetcutcallbackfunc

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
void CPXPUBLIC CPXgetcutcallbackfunc(CPXCENVptr env,  
int CPXPUBLIC **cutcallback_p)(CALLBACK_CUT_ARGS) ,  
void ** cbhandle_p)

**Description**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine **CPXgetcutcallbackfunc** accesses the user-written callback for adding cuts. The user-written callback is called by ILOG CPLEX during MIP branch & cut for every node that has an LP optimal solution with objective value below the cutoff and that is integer infeasible. The callback routine adds globally valid cuts to the LP subproblem.

**Example**

```
CPXgetcutcallbackfunc(env, &current_cutfunc,  
&current_data);
```

See also *Advanced MIP Control Interface* in the *ILOG CPLEX User's Manual*.

For documentation of callback arguments, see the routine **CPXsetcutcallbackfunc**.

**Parameters**

- **env**
  
  A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

- **cutcallback_p**
  
  The address of the pointer to the current user-written cut callback. If no callback has been set, the pointer evaluates to NULL.
cbhandle_p

The address of a variable to hold the user's private pointer.

See Also

CPXcutcallbackadd, CPXsetcutcallbackfunc

Returns

This routine does not return a result.
CPXgetcutoff

Category          Global Function
Definition File   cplex.h
Include Files     cplex.h
Synopsis

    int CPXPUBLIC CPXgetcutoff (CPXCENVptr env,
                               CPXCLPtr lp,
                               double * cutoff_p)

Description

The routine CPXgetcutoff is used to access the MIP cutoff value being used during mixed integer optimization. The cutoff is updated with the objective function value, each time an integer solution is found during branch & cut.

Example

    status = CPXgetcutoff (env, lp, &cutoff);

Parameters

    env
    A pointer to the CPLEX environment as returned by CPXopenCPLEX.

    lp
    A pointer to a CPLEX problem object as returned by CPXcreateprob.

    cutoff_p
    A pointer to a location where the value of the cutoff is returned.

Example

    status = CPXgetcutoff (env, lp, &cutoff);

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXgetdblparam

Category: Global Function
Definition File: cplex.h
Include Files: cplex.h

Synopsis:
```c
int CPXPUBLIC CPXgetdblparam(CPXENVptr env,
                                 int whichparam,
                                 double * value_p)
```

Description:
The routine CPXgetdblparam is used to obtain the current value of a CPLEX parameter of type double.

The reference manual ILOG CPLEX Parameters provides a list of parameters with their types, options, and default values.

Example:
```c
status = CPXgetdblparam (env, CPX_PARAM_TILIM, &curtilim);
```

Parameters:
- **env**
The pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **whichparam**
The symbolic constant (or reference number) of the parameter for which the value is to be obtained.
- **value_p**
A pointer to a variable of type double to hold the current value of the CPLEX parameter.

Returns:
The routine returns zero on success and nonzero if an error occurs.
CPXgetdblquality

Category Global Function

Definition File cplex.h

Include Files cplex.h

Synopsis

```
int CPXPUBLIC CPXgetdblquality(CPXCENVptr env,
    CPXCLPptr lp,
    double * quality_p,
    int what)
```

Description

The routine CPXgetdblquality is used to access double-valued information about the quality of the current solution of a problem. A solution, though not necessarily a feasible or optimal one, must be available in the CPLEX problem object. The quality values are returned in the double variables pointed to by the argument quality_p.

The maximum bound infeasibility identifies the largest bound violation, which helps determine the cause of an infeasible problem. If it exceeds the feasibility tolerance by only a small amount, it may be possible to obtain a feasible solution to the problem by increasing the feasibility tolerance. If a problem is optimal, it gives insight into the smallest setting for the feasibility tolerance that would not cause the problem to terminate infeasible.

If an error occurs, the value remains unchanged.

The possible quality values for a solution are listed in the group optim.cplex.callable.solutionquality in the ILOG CPLEX Reference Manual.

Example

```
status = CPXgetdblquality (env, lp, &max_x, CPX_MAX_X);
```

Parameters

env
A pointer to the CPLEX environment as returned by the CPXopenCPLEX routine.

lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

quality_p
A pointer to a double variable in which the requested quality value is to be stored.

If an error occurs, the value remains unchanged.

what
A symbolic constant indicating the quality value to be retrieved.

The possible quality values for a solution are listed in the group optim.cplex.callable.solutionquality in the *ILOG CPLEX Reference Manual.*

**Example**

```c
status = CPXgetdblquality (env, lp, &max_x, CPX_MAX_X);
```

**Returns**

The routine returns zero on success and nonzero if an error occurs.
**CPXgetdeletenodecallbackfunc**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
void CPXPUBLIC CPXgetdeletenodecallbackfunc(CPXCENVptr env,
   void(CPXPUBLIC **deletecallback_p)(CALLBACK_DELETENODE_ARGS) ,
   void ** cbhandle_p)
```

**Description**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine **CPXgetdeletenodecallbackfunc** accesses the user-written callback to be called during MIP optimization when a node is to be deleted. Nodes are deleted when a branch is carried out from that node, when the node relaxation is infeasible, or when the node relaxation objective value is worse than the cutoff. This callback can be used to delete user data associated with a node.

**Example**

```c
CPXgetdeletenodecallbackfunc(env,
   &current_callback,
   &current_cbdata);
```

See also *Advanced MIP Control Interface* in the ILOG CPLEX User's Manual.

For documentation of callback arguments, see the routine **CPXsetdeletenodecallbackfunc**.

**Parameters**

- `env`
  - A pointer to the CPLEX environment, as returned by **CPXopenCPLEX**.
  - `deletenodecallback_p`
The address of the pointer to the current user-written delete-node callback. If no callback has been set, the pointer evaluates to NULL.

`cbhandle_p`

The address of a variable to hold the user's private pointer.

**See Also**

CPXsetdeletenodecallbackfunc, CPXbranchcallbackbranchbds, CPXbranchcallbackbranchconstraints, CPXbranchcallbackbranchgeneral

**Returns**

This routine does not return a result.
**CPXgetdj**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```
int CPXPUBLIC CPXgetdj (CPXCENVptr env,  
CPXCLPptr lp,  
double * dj,  
int begin,  
int end)
```

**Description**  
The routine `CPXgetdj` is used to access the reduced costs for a range of the variables of a linear or quadratic program. The beginning and end of the range must be specified.

**Example**  
```
status = CPXgetdj (env, lp, dj, 0, CPXgetnumcols(env,lp)-1);
```

**Parameters**

- **env**  
A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- **lp**  
A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

- **dj**  
An array to receive the values of the reduced costs for each of the variables. This array must be of length at least `(end - begin + 1)`. If successful, `dj[0]` through `dj[end-begin]` contain the values of the reduced costs.

- **begin**  
An integer indicating the beginning of the range of reduced-cost values to be returned.

- **end**  
An integer indicating the end of the range of reduced-costs values to be returned.

**Example**  
```
status = CPXgetdj (env, lp, dj, 0, CPXgetnumcols(env,lp)-1);
```

**Returns**  
The routine returns zero on success and nonzero if an error occurs.
**CPXgetdsbcnt**

**Category**          Global Function

**Definition File**  cplex.h

**Synopsis**          
```c
int CPXPUBLIC CPXgetdsbcnt (CPXCENVptr env,
                            CPXCLPptr lp)
```

**Description** The routine CPXgetdsbcnt is used to access the number of dual super-basic variables in the current solution.

**Example**

```c
itcnt = CPXgetdsbcnt (env, lp);
```

**Parameters**

**env**
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

**lp**
A pointer to a CPLEX problem object as returned by CPXcreateprob.

**Example**

```c
itcnt = CPXgetdsbcnt (env, lp);
```

**Returns** If a solution exists, CPXgetdsbcnt returns the number of dual super-basic variables. If no solution exists, CPXgetdsbcnt returns the value 0 (zero).
CPXgeterrorstring

Category Global Function
Definition File cplex.h
Synopsis

    CPXCCHARptr CPXPUBLIC CPXgeterrorstring(CPXENVptr env,
        int errcode,
        char * buffer_str)

Description

The routine CPXgeterrorstring returns an error message string corresponding to an error code. Error codes are returned by CPLEX routines when an error occurs.

Note: This routine allows the CPLEX environment parameter to be NULL so that errors caused by the routine CPXopenCplex can be translated.

Example

    char *errstr;
    errstr = CPXgeterrorstring (env, errcode, buffer);
    if ( errstr != NULL ) {
        printf ("%s
"
    }
    else {
        printf ("CPLEX Error %5d:  Unknown error code.  
            errcode);
    }

Parameters

    env
    A pointer to the CPLEX environment as returned by CPXopenCplex.

    errcode
    The error code to be translated.

    buffer_str
    A character string buffer. This buffer must be at least 510 characters to hold the error string.

Returns

    This routine returns a pointer to the parameter buffer if the string does exist. In that case, buffer contains the error message string. It returns NULL if the error code does not have a corresponding string.
**CPXgetgenclqcnt**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```
int CPXPUBLIC CPXgetgenclqcnt (CPXCENVptr env, CPXCLPptr lp)
```

**Description**  
The routine `CPXgetgenclqcnt` is used to access the number of clique inequalities in the "clique table", generated by CPLEX at the start of a mixed integer optimization.

**Example**  
```
gen_clique_cuts = CPXgetgenclqcnt (env, lp);
```

**Parameters**  
- `env`  
A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.  
- `lp`  
A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

**Example**  
```
gen_clique_cuts = CPXgetgenclqcnt (env, lp);
```

**Returns**  
If no solution, problem, or environment exists, `CPXgetgenclqcnt` returns 0. Otherwise, it returns the number of clique inequalities generated.
CPXgetgrad

Category          Global Function
Definition File   cplex.h
Include Files    cplex.h

Synopsis
int CPXPUBLIC CPXgetgrad(CPXENVptr env,
                CPXCLPptr lp,
                int j,
                int * head,
                double * y)

Description
The routine CPXgetgrad can be used, after an LP has been solved and a basis is available, to access information useful for different types of post-solution analysis. CPXgetgrad provides two arrays that can be used to project the impact of making changes to optimal variable values or objective function coefficients.

For a unit change in the value of the $j$th variable, the value of the $i$th basic variable, sometimes referred to as the variable basic in the $i$th row, changes by the amount $y[i]$. Also, for a unit change of the objective function coefficient of the $i$th basic variable, the reduced-cost of the $j$th variable changes by the amount $y[i]$. The vector $y$ is equal to the product of the inverse of the basis matrix and the column $j$ of the constraint matrix. Thus, $y$ can be thought of as the representation of the $j$th column in terms of the basis.

Example

```c
status = CPXgetgrad (env, lp, 13, head, y);
```

Parameters

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **lp**
  A pointer to a CPLEX problem object as returned by CPXcreateprob.
- **j**
  An integer indicating the index of the column of interest. A negative value for $j$ indicates a column representing the slack or artificial variable for row $-j-1$.
- **head**
  An array to contain a listing of the indices of the basic variables in the order in which they appear in the basis. This listing is sometimes called the basis header. The $i$th entry in this list is also sometimes viewed as the variable in the $i$th row of the basis. If the $i$th
basic variable is a structural variable, head[i] simply contains the column index of that variable. If it is a slack variable, head[i] contains one less than the negative of the row index of that slack variable. This array should be of length at least CPXgetnumrows(env, lp). May be NULL.

\( \text{y} \)

An array to contain the coefficients of the \( j \)th column relative to the current basis. See the discussion above on how to interpret the entries in \( y \). This array should be of length at least CPXgetnumrows(env, lp). May be NULL.

**Example**

\[
\text{status} = \text{CPXgetgrad} (\text{env}, \text{lp}, 13, \text{head}, \text{y});
\]

**Returns**

The routine returns zero on success and nonzero if an error occurs. This routine fails if no basis exists.
**CPXgetheuristiccallbackfunc**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**
```c
void CPXPUBLIC CPXgetheuristiccallbackfunc(CPXENVptr env,
   int(CPXPUBLIC **heuristiccallback_p)(CALLBACK_HEURISTIC_ARGS) ,
   void ** cbhandle_p)
```

**Description**

Note: This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetheuristiccallbackfunc accesses the user-written callback to be called by ILOG CPLEX during MIP optimization after the subproblem has been solved to optimality. That callback is not called when the subproblem is infeasible or cut off. The callback supplies ILOG CPLEX with heuristically-derived integer solutions.

**Example**

```c
CPXgetheuristiccallbackfunc(env, &current_callback,
   &current_handle);
```

See also **Advanced MIP Control Interface** in the **ILOG CPLEX User's Manual**.

For documentation of callback arguments, see the routine **CPXsetheuristiccallbackfunc**.

**Parameters**

- **env**
  
  A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

- **heuristiccallback_p**
  
  The address of the pointer to the current user-written heuristic callback. If no callback has been set, the pointer evaluates to NULL.

- **cbhandle_p**
The address of a variable to hold the user’s private pointer.

See Also

CPXsetheuristiccallbackfunc

Returns

This routine does not return a result.
CPXgetiis

**Category**  Global Function

**Definition File**  cplex.h

**Include Files**  cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXgetiis(CPXCENVptr env,
    CPXCLPptr lp,
    int * iisstat_p,
    int * rowind,
    int * rowbdstat,
    int * iisnumrows_p,
    int * colind,
    int * colbdstat,
    int * iisnumcols_p)
```

**Description**

The routine CPXgetiis is used to examine the IIS for an infeasible LP problem. The IIS must already have been computed by a call to CPXfindiis or CPXiiswrite. On successful completion, the CPXgetiis arguments return information about the IIS that can be used to diagnose the cause of infeasibility. The number of rows and bound constraints in the IIS, as well as the indices of the IIS members, are returned.

**Table 1: Values of iisstat_p**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CPXIIS_COMPLETE</td>
</tr>
<tr>
<td>2</td>
<td>CPXIIS_PARTIAL</td>
</tr>
</tbody>
</table>

**Table 2: Values of elements of rowbdstat**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CPXIIS_AT_LOWER</td>
</tr>
<tr>
<td>1</td>
<td>CPXIIS_FIXED</td>
</tr>
<tr>
<td>2</td>
<td>CPXIIS_AT_UPPER</td>
</tr>
</tbody>
</table>

**Table 3: Values of elements of colbdstat (status of column bounds)**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CPXIIS_AT_LOWER</td>
</tr>
<tr>
<td>1</td>
<td>CPXIIS_FIXED</td>
</tr>
<tr>
<td>2</td>
<td>CPXIIS_AT_UPPER</td>
</tr>
</tbody>
</table>
Example

```
status = CPXgetiis (env, lp, &iisstat, rowind, rowbdstat,
                   &iisnumrows, colind, colbdstat,
                   &iisnumcols);
```

Parameters

- **env**
  A pointer to the CPLEX environment as returned by the CPXopenCPLEX routine.

- **lp**
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

- **iisstat_p**
  A pointer to an integer to contain the result of the IIS algorithm. The specific values that *iisstat_p* can take and their meanings appear in Table 1.

- **rowind**
  An array to contain the indices of the rows in the IIS. The length of the array must be at least as large as the number of rows in the IIS.

- **rowbdstat**
  An array that identifies the right-hand side value for each row in the IIS that is causing the infeasibility. Possible values appear in Table 2. The length of the array must be at least as large as the number of rows in the IIS. This information is needed only for ranged rows. For all other row senses, the value is uniquely determined by the sense.

- **iisnumrows_p**
  A pointer to an integer to contain the number of rows in the IIS.

- **colind**
  An array to contain the indices of the columns in the IIS. The length of the array must be at least as large as the number of columns in the IIS.

- **colbdstat**
  An array that identifies the bound for each column in the IIS that is causing the infeasibility. Possible values appear in Table 3. The length of the array must be at least as large as the number of columns in the IIS.

- **iisnumcols_p**
  A pointer to an integer to contain the number of columns in the IIS.

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXgetincumbentcallbackfunc

Category  Global Function
Definition File  cplex.h
Include Files  cplex.h
Synopsis  

```c
void CPXPUBLIC CPXgetincumbentcallbackfunc(CPXCENVptr env,
   int(CPXPUBLIC **incumbentcallback_p)(CALLBACK_INCUMBENT_ARGS) ,
   void ** cbhandle_p)
```

Description

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetincumbentcallbackfunc accesses the user-written callback to be called by CPLEX during MIP optimization after an integer solution has been found but before this solution replaces the incumbent. This callback can be used to discard solutions that do not meet criteria beyond that of the mixed integer programming formulation.

**Example**

```c
CPXgetincumbentcallbackfunc(env, &current_incumbentcallback,
   &current_handle);
```

See also *Advanced MIP Control Interface* in the *ILOG CPLEX User's Manual*.

For documentation of callback arguments, see the routine CPXsetincumbentcallbackfunc.

**Parameters**

- **env**
  A pointer to the CPLEX environment, as returned by CPXopenCPLEX.
- **incumbentcallback_p**
  The address of the pointer to the current user-written incumbent callback. If no callback has been set, the pointer evaluates to NULL.
cbhandle_p

The address of a variable to hold the user's private pointer.

See Also

CPXsetincumbentcallbackfunc

Returns

This routine does not return a result.
CPXgetintparam

Category                     Global Function
Definition File             cplex.h
Include Files               cplex.h
Synopsis
int CPXPUBLIC CPXgetintparam(CPXENVptr env,
                           int whichparam,
                           int * value_p)

Description
The routine CPXgetintparam is used to obtain the current value of a CPLEX parameter of type int.

The reference manual ILOG CPLEX Parameter provides a list of parameters with their types, options, and default values.

Example

status = CPXgetintparam (env, CPX_PARAM_PREIND, &curpreind);

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

whichparam
The symbolic constant (or reference number) of the parameter for which the value is to be obtained.

value_p
A pointer to an integer variable to hold the current value of the CPLEX parameter.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXgetintquality

Category               Global Function
Definition File        cplex.h
Include Files          cplex.h
Synopsis               int CPXPUBLIC CPXgetintquality(CPXENVptr env,
                               CPXCLPptr lp,
                               int * quality_p,
                               int what)

Description            The routine CPXgetintquality is used to access integer-valued information about the quality of the current solution of a problem. A solution, though not necessarily a feasible or optimal one, must be available in the CPLEX problem object. The quality values are returned in the integer variables pointed to by the argument quality_p.

The possible quality values for a solution are listed in the group optim.cplex.callable.solutionquality in the ILOG CPLEX Reference Manual.

Example

    status = CPXgetintquality (env, lp, &max_x_ind, CPX_MAX_X);

Parameters             env
                        A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp                      A pointer to a CPLEX problem object as returned by CPXcreateprob.
quality_p               A pointer to an integer variable in which the requested quality value is to be stored.
what                    A symbolic constant indicating the quality value to be retrieved.

The possible quality values for a solution are listed in the group optim.cplex.callable.solutionquality in the ILOG CPLEX Reference Manual.

Example

    status = CPXgetintquality (env, lp, &max_x_ind, CPX_MAX_X);
Returns

The routine returns zero on success and nonzero if an error occurs.
**CPXgetitcnt**

**Category**    Global Function

**Definition File**    cplex.h

**Include Files**    cplex.h

**Synopsis**

```
int CPXPUBLIC CPXgetitcnt (CPXCENVptr env, CPXCLPptr lp)
```

**Description**

The routine `CPXgetitcnt` is used to access the total number of simplex iterations to solve an LP problem, or the number of crossover iterations in the case that Barrier is used.

**Example**

```
int itcnt = CPXgetitcnt (env, lp);
```

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- **lp**
  A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

**Example**

```
int itcnt = CPXgetitcnt (env, lp);
```

**Returns**

If a solution exists, `CPXgetitcnt` returns the total iteration count. If no solution exists, `CPXgetitcnt` returns the value 0.

See `lpex6.c` in the *CPLEX User's Manual*. 
CPXgetlb

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXgetlb(CPXENVptr env,
CPXCLPptr lp,
double * lb,
int begin,
int end)
```

Description:
The routine CPXgetlb is used to access a range of lower bounds on the variables of a CPLEX problem object. The beginning and end of the range must be specified.

Example:

```c
status = CPXgetlb (env, lp, lb, 0, cur_numcols-1);
```

Parameters:

- `env`:
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- `lp`:
  A pointer to a CPLEX problem object as returned by CPXcreateprob.
- `lb`:
  An array where the specified lower bounds on the variables are to be returned. This array must be of length at least `(end - begin + 1)`. The lower bound of variable `j` is returned in `lb[j - begin]`.
- `begin`:
  An integer indicating the beginning of the range of lower bounds to be returned.
- `end`:
  An integer indicating the end of the range of lower bounds to be returned.

Returns:
The routine returns zero on success and nonzero if an error occurs.
CPXgetlogfile

Category             Global Function
Definition File      cplex.h
Include Files        cplex.h
Synopsis             int CPXPUBLIC CPXgetlogfile(CPXENVptr env,
                                  CPXFILEptr * logfile_p)
Description           The routine CPXgetlogfile accesses the log file to which messages from all four
                      CPLEX-defined channels are written.
Example               status = CPXgetlogfile (env, &logfile);
Parameters
                      env
                      A pointer to the CPLEX environment as returned by CPXopenCPLEX.
                      logfile_p
                      The address of a CPXFILEptr variable. This routine sets logfile_p to be the file
                      pointer for the current log file.
Returns               The routine returns zero on success and nonzero if an error occurs.
**CPXgetlpcallbackfunc**

**Category** Global Function

**Definition File** cplex.h

**Include Files** cplex.h

**Synopsis**

```c
int CPXIPUBLIC CPXgetlpcallbackfunc(CPXCENVptr env,
                                   int(CPXIPUBLIC **callback_p)(CPXCENVptr, void *, int, void *) ,
                                   void ** cbhandle_p)
```

**Description**

The routine CPXgetlpcallbackfunc is used to access the user-written callback routine to be called after each iteration during the optimization of a linear or quadratic program, and also periodically during the CPLEX presolve algorithm.

**Callback description**

```c
int callback (CPXENVptr env,
              void   *cbdata,
              int     wherefrom,
              void   *cbhandle);
```

This is the user-written callback routine.

**Callback return value**

A nonzero terminates the optimization.

**Callback arguments**

- **env**
  A pointer to the CPLEX environment that was passed into the associated optimization routine.

- **cbdata**
  A pointer passed from the optimization routine to the user-written callback function that identifies the LP problem being optimized. The only purpose for the cbdata pointer is to pass it to the routine CPXgetcallbackinfo.

- **wherefrom**
  An integer value indicating which optimization algorithm the user-written callback function was called from. Possible values and their meaning appear in the table.
cbhandle

Pointer to user private data, as passed to CPXsetlpcallbackfunc.

Parameters

env

A pointer to the CPLEX environment as returned by CPXopenCPLEX.

callback_p

The address of the pointer to the current user-written callback function. If no callback function has been set, the pointer evaluates to NULL.

cbhandle_p

The address of a variable to hold the user’s private pointer.

Example

status = CPXgetlpcallbackfunc (env, mycallback, NULL);

See Also

CPXgetcallbackinfo

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXgetmethod

Category          Global Function
Definition File   cplex.h
Include Files    cplex.h
Synopsis          int CPXPUBLIC CPXgetmethod(CPXENVptr env, CPXCLPptr lp)

Description       The routine CPXgetmethod returns an integer indicating the solution algorithm used to solve the resident LP or QP problem.
                    
Example            method = CPXgetmethod (env, lp);

Parameters        env
                    A pointer to the CPLEX environment as returned by CPXopenCPLEX.
                    lp
                    A pointer to a CPLEX problem object as returned by CPXcreateprob.

Returns           The possible return values are summarized in the table.

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CPX_ALG_NONE</td>
<td>None</td>
</tr>
<tr>
<td>1</td>
<td>CPX_ALG_PRIMAL</td>
<td>Primal simplex</td>
</tr>
<tr>
<td>2</td>
<td>CPX_ALG_DUAL</td>
<td>Dual simplex</td>
</tr>
<tr>
<td>4</td>
<td>CPX_ALG_BARRIER</td>
<td>Barrier optimizer (no crossover)</td>
</tr>
</tbody>
</table>
**CPXgetmipcallbackfunc**

**Category**  
Global Function

**Definition File**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXgetmipcallbackfunc(CPXENVptr env,
        int (CPXPUBLIC **callback_p)(CPXENVptr, void *, int, void *) ,
        void ** cbhandle_p)
```

**Description**  
The routine `CPXgetmipcallbackfunc` is used to access the user-written callback routine to be called prior to solving each subproblem in the branch & cut tree during the optimization of a mixed integer program.

This routine works in the same way as the routine `CPXgetlpcallbackfunc`. It enables the user to create a separate callback function to be called during the solution of mixed integer programming problems. The prototype for the callback function is identical to that of `CPXgetlpcallbackfunc`.

**Parameters**

- **env**  
A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- **callback_p**  
The address of the pointer to the current user-written callback function. If no callback function has been set, the pointer evaluates to NULL.

- **cbhandle_p**  
The address of a variable to hold the user's private pointer.

**Example**

```c
status = CPXgetmipcallbackfunc (env, mycallback, NULL);
```

**Callback description**

```c
int callback (CPXENVptr env,
        void *cbdata,
        int wherefrom,
        void *cbhandle);
```

This is the user-written callback routine.
Callback return value
A nonzero terminates the optimization.

Callback arguments

env
A pointer to the CPLEX environment that was passed into the associated optimization routine.

cbdata
A pointer passed from the optimization routine to the user-written callback function that identifies the LP problem being optimized. The only purpose for the cbdata pointer is to pass it to the routine CPXgetcallbackinfo.

wherefrom
An integer value indicating from which optimization algorithm the user-written callback function was called. Possible values and their meaning appear in this table.

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>CPX_CALLBACK_MIP</td>
<td>From mipopt</td>
</tr>
<tr>
<td>107</td>
<td>CPX_CALLBACK_MIP_PROBE</td>
<td>From probing or clique merging</td>
</tr>
<tr>
<td>108</td>
<td>CPX_CALLBACK_MIP_FRACCUT</td>
<td>From Gomory fractional cuts</td>
</tr>
<tr>
<td>109</td>
<td>CPX_CALLBACK_MIP_DISJCUT</td>
<td>From disjunctive cuts</td>
</tr>
<tr>
<td>110</td>
<td>CPX_CALLBACK_MIP_FLOWMI</td>
<td>From Mixed Integer Rounding cuts</td>
</tr>
</tbody>
</table>

cbhandle
Pointer to user private data, as passed to CPXsetmipcallbackfunc.

See Also
CPXgetcallbackinfo

Returns
The routine returns zero on success and nonzero if an error occurs.
**CPXgetmipitcnt**

**Category**  
Global Function

**Definition File**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXgetmipitcnt (CPXCENVptr env, CPXCLPptr lp)
```

**Description**  
The routine CPXgetmipitcnt is used to access the cumulative number of simplex iterations used to solve a mixed integer problem.

**Example**  
```c
itcnt = CPXgetmipitcnt (env, lp);
```

**Parameters**

- **env**  
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **lp**  
A pointer to a CPLEX problem object as returned by CPXcreateprob.

**Example**  
```c
itcnt = CPXgetmipitcnt (env, lp);
```

**Returns**  
If a solution exists, CPXgetmipitcnt returns the total iteration count. If no solution, problem, or environment exists, CPXgetmipitcnt returns the value 0.
CPXgetmipobjval

Category          Global Function
Definition File   cplex.h
Synopsis          int CPXPUBLIC CPXgetmipobjval (CPXCENVptr env,
                   CPXCLPptr lp,
                   double * objval_p)
Description       The routine CPXgetmipobjval is used to access the mixed integer solution objective value.
Example            status = CPXgetmipobjval (env, lp, &objval);

See also the example mipex1.c in the standard distribution.

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.
objval_p
A pointer to the location where the mixed integer objective value is stored.
Example

status = CPXgetmipobjval (env, lp, &objval);

See also the example mipex1.c in the standard distribution.

Returns

The routine returns zero on success and nonzero if an error occurs. If no integer solution has been found, the value CPXERR_NO_INT_SOLN is returned.
CPXgetmipqconstrslack

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXgetmipqconstrslack(CPXENVptr env,
   CPXCLPptr lp,
   double * qcslack,
   int begin,
   int end)
```

**Description**
The routine CPXgetmipqconstrslack is used to access the slack values for a range of the quadratic constraints of an MIQCP. The beginning and end of the range must be specified.

**Example**

```c
status = CPXgetmipqconstrslack (env, lp, qcslack, 0,
   CPXgetnumqconstrs(env,lp)-1);
```

**Parameters**

- `env`
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- `lp`
A pointer to a CPLEX problem object as returned by CPXcreateprob.

- `qcslack`
An array to contain the values of the quadratic constraint `qcslack` variables for the problem. This array must be of length at least (end-begin+1). If successful, `qcslack[0]` through `qcslack[end-begin]` contain the quadratic constraint slack variables `begin` through `end`.

- `begin`
An integer indicating the beginning of the range of quadratic constraint slack values to be returned.

- `end`
An integer indicating the end of the range of quadratic constraint slack values to be returned.


Returns

The routine returns zero on success and nonzero if an error occurs. If no integer solution has been found, the value CPXERR_NO_INT_SOLUTION is returned.
CPXgetmipslack

Category            Global Function
Definition File     cplex.h
Include Files       cplex.h
Synopsis
int CPXPUBLIC CPXgetmipslack (CPXCENVptr env, 
   CPXCLPptr lp, 
   double * slack, 
   int begin, 
   int end)

Description
The routine CPXgetmipslack is used to access a range of slack variables for the 
current mixed integer solution. The beginning and end of the range must be specified.
The routine returns zero on success and nonzero if an error occurs. If no integer solution 
has been found, the value CPXERR_NO_INT_SOLUTION is returned.

Example
status = CPXgetmipslack (env, lp, slack, 0, 
   CPXgetnumrows(env,lp)-1);

See also the example mipex1.c in the standard distribution.

Parameters
env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.
slack
An array to contain the values of the primal slack variables for the problem. This array 
must be of length at least (end-begin+1). If successful, slack[0] through 
slack[end-begin] contain the slack variables begin through end.
begin
An integer indicating the beginning of the range of slack values to be returned.
end
An integer indicating the end of the range of slack values to be returned.

Example
status = CPXgetmipslack (env, lp, slack, 0,
   CPXgetnumrows(env, lp)-1);

See also the example mipex1.c in the standard distribution.
CPXgetmipstart

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXgetmipstart (CPXCENVptr env,
                             CPXCLPptr lp,
                             int * cnt_p,
                             int * indices,
                             double * value,
                             int mipstarts_p,
                             int * surplus_p)
```

Description:
The routine CPXgetmipstart is used to access MIP start information stored in a CPLEX problem object. Values are returned for all integer, binary, semi-continuous, and nonzero SOS variables when the MIP start is the result of a call to CPXmipopt.

Note: If the value of `startspace` is 0 (zero), then the negative of the value of `*surplus_p` returned indicates the length needed for the arrays `indices` and `values`.

Example:

```c
status = CPXgetmipstart (env, lp, &listsize, indices, values,
                          numcols, &surplus);
```

Parameters:

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **lp**
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

- **cnt_p**
  A pointer to an integer to contain the number of MIP start entries returned; that is, the true length of the arrays `indices` and `values`.

- **indices**
An array to contain the indices of the variables in the MIP start. indices[k] is the index of the variable which is entry k in the MIP start information. Must be of length no less than startspace.

value

An array to contain the MIP start values. The start value corresponding to indices[k] is returned in values[k]. Must be of length at least startspace.

mipstartspace

An integer indicating the length of the non-NULL array indices and values; startspace may be 0 (zero).

surplus_p

A pointer to an integer to contain the difference between startspace and the number of entries in each of the arrays indices and values. A nonnegative value of *surplus_p indicates that the length of the arrays was sufficient. A negative value indicates that the length was insufficient and that the routine could not complete its task. In this case, the routine CPXgetmipstart returns the value CPXERR_NEGATIVE_SURPLUS, and the negative value of *surplus_p indicates the amount of insufficient space in the arrays.

Returns

The routine returns zero on success and nonzero if an error occurs. The value CPXERR_NEGATIVE_SURPLUS indicates that insufficient space was available in the indices and values arrays to hold the MIP start information.
CPXgetmipx

Category       Global Function
Definition File cplex.h
Include Files  cplex.h
Synopsis
int CPXPUBLIC CPXgetmipx (CPXCENVptr env, 
CPXCLPptr lp, 
double * x, 
int begin, 
int end)

Description   The routine CPXgetmipx is used to access a range of mixed integer solution values. The beginning and end of the range must be specified.

Example
status = CPXgetmipx (env, lp, x, 0, CPXgetnumcols(env,lp)-1);

See also the example mipex1.c in the standard distribution.

Parameters
env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

x
An array to contain the values of the primal variables for the problem. This array must be of length at least (end-begin+1). If successful, x[0] through x[end-begin] contain the solution values begin through end.

begin
An integer indicating the beginning of the range of variable values to be returned.

end
An integer indicating the end of the range of variable values to be returned.

Example
status = CPXgetmipx (env, lp, x, 0, CPXgetnumcols(env,lp)-1);
See also the example `mipex1.c` in the standard distribution.

**Returns**

The routine returns zero on success, and nonzero if an error occurs. If no integer solution has been found, the value `CPXERR_NO_INT_SOLL` is returned.
CPXgetnetcallbackfunc

Category             Global Function
Definition File      cplex.h

Synopsis

```c
int CPXPUBLIC CPXgetnetcallbackfunc(CPXENVptr env,
    int(CPXPUBLIC **callback_p)(CPXENVptr, void *, int, void *) ,
    void ** cbhandle_p)
```

Description

The CPXgetnetcallbackfunc is used to access the user-written callback routine to be called each time a log message is issued during the optimization of a network problem. If the display log is turned off, the callback routine is still called.

This routine works in the same way as the routine CPXgetlpcallbackfunc. It enables the user to create a separate callback function to be called during the solution of a network problem. The prototype for the callback function is identical to that of CPXgetlpcallbackfunc.

Callback description

```c
int callback (CPXENVptr env,
    void *cbdata,
    int wherefrom,
    void *cbhandle);
```

This is the user-written callback routine.

Callback return value

A nonzero terminates the optimization.

Callback arguments

**env**

A pointer to the CPLEX environment that was passed into the associated optimization routine.

**cbdata**

A pointer passed from the optimization routine to the user-written callback function that identifies the problem being optimized. The only purpose for the cbdata pointer is to pass it to the routine CPXgetcallbackinfo.

**wherefrom**
An integer value indicating which optimization algorithm the user-written callback function was called from. Possible values and their meaning appear in the table.

<table>
<thead>
<tr>
<th>Value?</th>
<th>Symbolic Constant?</th>
<th>Meaning?</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>CPX_CALLBACK_NETWORK</td>
<td>From network simplex?</td>
</tr>
</tbody>
</table>

cbhandle

Pointer to user private data, as passed to CPXsetlpcallbackfunc.

**Parameters**

`env`

A pointer to the CPLEX environment as returned by CPXopenCPLEX.

`callback`

The address of the pointer to the current user-written callback function. If no callback function has been set, the pointer evaluates to NULL.

`cbhandle_p`

The address of a variable to hold the private pointer of the user.

**Example**

```c
status = CPXgetnetcallbackfunc (env, mycallback, NULL);
```

**See Also**

CPXgetcallbackinfo

**Returns**

A nonzero terminates the optimization.
CPXgetnodecallbackfunc

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
void CPXPUBLIC CPXgetnodecallbackfunc(CPXENVptr env,
int (CPXPUBLIC **)nodecallback_p)(CALLBACK_NODE_ARGS),
void ** cbhandle_p)
```

Description:

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetnodecallbackfunc accesses the user-written callback to be called during MIP optimization after ILOG CPLEX has selected a node to explore, but before this exploration is carried out. The callback routine can change the node selected by ILOG CPLEX to a node selected by the user.

For documentation of callback arguments, see the routine CPXsetnodecallbackfunc.

Example:

```c
CPXgetnodecallbackfunc(env, &current_callback,
&current_handle);
```

See also the example admipex1.c in the standard distribution.

Parameters:

- env
  - A pointer to the CPLEX environment, as returned by CPXopenCPLEX.
- nodecallback_p
  - The address of the pointer to the current user-written node callback. If no callback has been set, the pointer will evaluate to NULL.
- cbhandle_p
The address of a variable to hold the user's private pointer.

**Returns**

This routine does not return a result.
CPXgetnodecnt

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis: int CPXBASIC CPXgetnodecnt (CPXENVptr env,
CPXCLPptr lp)

Description: The routine CPXgetnodecnt is used to access the number of nodes used to solve a mixed integer problem.

Example:
nodecount = CPXgetnodecnt (env, lp);

Parameters:
env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

Example:
nodecount = CPXgetnodecnt (env, lp);

Returns: If a solution exists, CPXgetnodecnt returns the node count. If no solution, problem, or environment exists, CPXgetnodecnt returns the value 0.
**CPXgetnodeint**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXgetnodeint (CPXCENVptr env,
                                CPXCLPptr lp)
```

**Description**  
The routine CPXgetnodeint is used to access the node number of the best known integer solution.

**Example**  
```c
nodeint = CPXgetnodeint (env, lp);
```

**Parameters**  
- **env**  
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **lp**  
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

**Example**  
```c
nodeint = CPXgetnodeint (env, lp);
```

**Returns**  
If no solution, problem, or environment exists, CPXgetnodeint returns a value of -1; otherwise, CPXgetnodeint returns the node number.
**CPXgetnodeleftcnt**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPX_PUBLIC CPXgetnodeleftcnt(CPXENVptr env, CPXCLPptr lp)
```

**Description**  
The routine CPXgetnodeleftcnt is used to access the number of unexplored nodes left in the branch & cut tree.

**Example**  
```c
nodes_left = CPXgetnodeleftcnt(env, lp);
```

**Parameters**  
- `env`  
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- `lp`  
A pointer to a CPLEX problem object as returned by CPXcreateprob.

**Returns**  
If no solution, problem, or environment exists, CPXgetnodeleftcnt returns 0 (zero); otherwise, CPXgetnodeleftcnt returns the number of unexplored nodes left in the branch & cut tree.
CPXgetnumbin

Category      Global Function
Definition File     cplex.h
Include Files     cplex.h

Synopsis         int CPXPUBLIC CPXgetnumbin(CPXCENVptr env, CPXCLPptr lp)

Description      The routine CPXgetnumbin is used to access the number of binary variables in a CPLEX problem object.

Example          nubin = CPXgetnumbin (env, lp);

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

Example          nubin = CPXgetnumbin (env, lp);

Returns         If the problem object or environment does not exist, CPXgetnumbin returns zero. Otherwise, it returns the number of binary variables in the problem object.
CPXgetnumcols

Category   Global Function
Definition File   cplex.h
Include Files   cplex.h
Synopsis   int CPXPUBLIC CPXgetnumcols(CPXCNVptr env, CPXCLPptr lp)
Description   The routine CPXgetnumcols is used to access the number of columns in the constraint matrix, or equivalently, the number of variables in the CPLEX problem object.

Example

        cur_numcols = CPXgetnumcols (env, lp);

See also the example lpxex1.c in the ILOG CPLEX User's Manual and in the standard distribution.

Parameters

 env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

 lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

Example

        cur_numcols = CPXgetnumcols (env, lp);

See also the example lpxex1.c in the ILOG CPLEX User's Manual and in the standard distribution.

Returns   If the problem object or environment does not exist, CPXgetnumcols returns the value 0 (zero); otherwise, it returns the number of columns (variables).
**CPXgetnumint**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXgetnumint (CPXCENVptr env, CPXCLPptr lp)
```

**Description**
The routine **CPXgetnumint** is used to access the number of general integer variables in a CPLEX problem object.

**Example**

```c
numint = CPXgetnumint (env, lp);
```

**Parameters**
- **env**
  A pointer to the CPLEX environment as returned by **CPXopenCPLEX**.
- **lp**
  A pointer to a CPLEX problem object as returned by **CPXcreateprob**.

**Example**

```c
numint = CPXgetnumint (env, lp);
```

**Returns**
If the problem object or environment does not exist, **CPXgetnumint** returns zero. Otherwise, it returns the number of general integer variables in the problem object.
CPXgetnumnz

Category          Global Function

Definition File   cplex.h

Include Files    cplex.h

Synopsis          int CPXPUBLIC CPXgetnumnz (CPXCENVptr env, 
                  CPXCLPptr lp)

Description       The routine CPXgetnumnz is used to access the number of nonzero elements in the 
                  constraint matrix of a CPLEX problem object, not including the objective function, 
                  quadratic constraints, or the bounds constraints on the variables.

Example            cur_numnz = CPXgetnumnz (env, lp);

Parameters

   env
   A pointer to the CPLEX environment as returned by CPXopenCPLEX.

   lp
   A pointer to a CPLEX problem object as returned by CPXcreateprob.

Returns            If the problem object or environment does not exist, CPXgetnumnz returns the value 0 
                  (zero); otherwise, it returns the number of nonzero elements.
**CPXgetnumqconstrs**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**
```c
int CPXPUBLIC CPXgetnumqconstrs(CPXENVptr env, CPXCLPptr lp)
```

**Description**
The routine CPXgetnumqconstrs is used to access the number of quadratic constraints in a CPLEX problem object.

**Example**
```c
cur_numqconstrs = CPXgetnumqconstrs (env, lp);
```

**Parameters**
- **env**
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **lp**
A pointer to a CPLEX problem object as returned by CPXcreateprob.

**Returns**
If the problem object or environment does not exist, CPXgetnumqconstrs returns the value 0 (zero); otherwise, it returns the number of quadratic constraints.
CPXgetnumqpnz

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXgetnumqpnz(CPXENVptr env,
 CPXCLPptr lp)
```

Description:

The routine `CPXgetnumqpnz` returns the number of nonzeros in the Q matrix of a problem object.

Example:

```c
numqpnz = CPXgetnumqpnz (env, lp);
```

Parameters:

- `env`
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- `lp`
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

Returns:

If successful, the routine returns the number of nonzeros in the Q matrix. If an error occurs, zero is returned.
**CPXgetnumquad**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXgetnumquad(CPXENVptr env,  
                           CPXCLPptr lp)
```

**Description**  
The routine CPXgetnumquad returns the number of variables that have quadratic objective coefficients in a CPLEX problem object.

**Example**
```c
numquad = CPXgetnumquad (env, lp);
```

**Parameters**

- `env`  
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- `lp`  
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

**Returns**  
If successful, the routine returns the number of variables having quadratic coefficients. If an error occurs, 0 is returned.
CPXgetnumrows

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:
```c
int CPXPUBLIC CPXgetnumrows(CPXENVptr env,
CPXCLIptr lp)
```

Description:
The routine CPXgetnumrows is used to access the number of rows in the constraint matrix, not including the objective function, quadratic constraints, or the bounds constraints on the variables.

Example:
```c
cur_numrows = CPXgetnumrows (env, lp);
```

See also the example lpex1.c in the ILOG CPLEX User's Manual and in the standard distribution.

Parameters:
- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **lp**
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

Returns:
If the CPLEX problem object or environment does not exist, CPXgetnumrows returns the value 0 (zero); otherwise, it returns the number of rows.
**CPXgetnumsemicont**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**
```c
int CPXPUBLIC CPXgetnumsemicont (CPXCENVptr env,
                                CPXCLPptr lp)
```

**Description**
The routine `CPXgetnumsemicont` is used to access the number of semi-continuous variables in a CPLEX problem object.

**Example**
```c
numsc = CPXgetnumsemicont (env, lp);
```

**Parameters**
- **env**
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- **lp**
  A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

**Returns**
If the problem object or environment does not exist, `CPXgetnumsemicont` returns the value 0 (zero); otherwise, it returns the number of semi-continuous variables in the problem object.
CPXgetnumsemiint

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis: int CPXPUBLIC CPXgetnumsemiint (CPXCENVptr env, CPXCLPptr lp)

Description: The routine CPXgetnumsemiint is used to access the number of semi-integer variables in a CPLEX problem object.

Example:

numsc = CPXgetnumsemiint (env, lp);

Parameters:

- env
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- lp
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

Returns: If the problem object or environment does not exist, CPXgetnumsemiint returns the value 0 (zero); otherwise, it returns the number of semi-integer variables in the problem object.
**CPXgetnumsos**

**Category**  Global Function

**Definition File**  cplex.h

**Include Files**  cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXgetnumsos (CPXCENVptr env, 
CPXCLPptr lp)
```

**Description**  The routine CPXgetnumsos is used to access the number of Special Ordered Sets in a CPLEX problem object.

**Example**

```c
numsos = CPXgetnumsos (env, lp);
```

**Parameters**

- **env**  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **lp**  A pointer to a CPLEX problem object as returned by CPXcreateprob.

**Example**

```c
numsos = CPXgetnumsos (env, lp);
```

**Returns**  If the problem object or environment does not exist, or the problem is not a mixed integer problem, the routine returns the value 0; otherwise, it returns the number of Special Ordered Sets in the problem object.
CPXgetobj

Category  Global Function
Definition File  cplex.h
Include Files  cplex.h
Synopsis

```c
int CPXPUBLIC CPXgetobj(CPXENVptr env,
                      CPXCLPptr lp,
                      double * obj,
                      int begin,
                      int end)
```

Description

The routine CPXgetobj is used to access a range of objective function coefficients of a CPLEX problem object. The beginning and end of the range must be specified.

Example

```c
status = CPXgetobj (env, lp, obj, 0, cur_numcols-1);
```

Parameters

- `env`
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- `lp`
  A pointer to a CPLEX problem object as returned by CPXcreateprob.
- `obj`
  An array where the specified objective coefficients are to be returned. This array must be of length at least \((\text{end} - \text{begin} + 1)\). The objective function coefficient of variable \(j\) is returned in \(\text{obj}[j - \text{begin}]\).
- `begin`
  An integer indicating the beginning of the range of objective function coefficients to be returned.
- `end`
  An integer indicating the end of the range of objective function coefficients to be returned.

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXgetobjname

Category       Global Function
Definition File cplex.h
Include Files  cplex.h
Synopsis
   int CPXPUBLIC CPXgetobjname(CPXENVptr env,
                               CPXCLPptr lp,
                               char * buf_str,
                               int bufspace,
                               int * surplus_p)
Description
   The routine CPXgetobjname is used to access the name of the objective row of a CPLEX problem object.

   Note: If the value of bufspace is 0, then the negative of the value of *surplus_p returned indicates the total number of characters needed for the array buf_str.

Example
   status = CPXgetobjname(env, lp, cur_objname, lenname,
                           &surplus);

Parameters

   env
   A pointer to the CPLEX environment as returned by CPXopenCPLEX.

   lp
   A pointer to a CPLEX problem object as returned by CPXcreateprob.

   buf_str
   A pointer to a buffer of size bufspace. May be NULL if bufspace is 0.

   bufspace
   An integer indicating the length of the array buf_str. May be 0.

   surplus_p
   A pointer to an integer to contain the difference between bufspace and the amount of memory required to store the objective row name. A nonnegative value of *surplus_p
indicates that the length of the array buf_str was sufficient. A negative value indicates that the length of the array was insufficient and that the routine could not complete its task. In this case, CPXgetobjname returns the value CPXERR_NEGATIVE_SURPLUS, and the negative value of the variable *surplus_p indicates the amount of insufficient space in the array buf_str.

**Returns**

The routine returns zero on success and nonzero if an error occurs. The value CPXERR_NEGATIVE_SURPLUS indicates that insufficient space was available in the buf_str array to hold the objective name.
**CPXgetobjsen**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXgetobjsen(CPXCENVptr env,
                           CPXCLPptr lp)
```

**Description**  
The routine CPXgetobjsen is used to access whether the objective function sense of a CPLEX problem object is maximization or minimization.

**Example**  
```c
cur_objsen = CPXgetobjsen (env, lp);
```

**Parameters**
- **env**  
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **lp**  
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

**Returns**  
A value of CPX_MIN=1 is returned for minimization and CPX_MAX=-1 is returned for maximization. If the problem object or environment does not exist, a 0 is returned.
CPXgetobjval

Category                     Global Function
Definition File             cplex.h
Include Files               cplex.h
Synopsis
int CPXPUBLIC CPXgetobjval(CPXENVptr env,
                           CPXCLPptr lp,
                           double * objval_p)
Description                  The routine CPXgetobjval is used to return the LP, QCP or QP solution objective value.
Example
status = CPXgetobjval (env, lp, &objval);

See also the example lpex2.c in the ILOG CPLEX User's Manual and in the standard distribution.

Parameters
env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.
objval_p
A pointer to a variable of type double where the objective value is stored.
Example
status = CPXgetobjval (env, lp, &objval);

See also the example lpex2.c in the ILOG CPLEX User's Manual and in the standard distribution.

Returns
The routine returns zero on success and nonzero if no solution exists.
CPXgetorder

Category  Global Function
Definition File  cplex.h
Include Files  cplex.h
Synopsis  
```c
int CPXPUBLIC CPXgetorder(CPXENVptr env,
    CPXCLPptr lp,
    int * cnt_p,
    int * indices,
    int * priority,
    int * direction,
    int ordspace,
    int * surplus_p)
```

Description  The routine CPXgetorder is used to access all the MIP priority order information stored in a CPLEX problem object. A priority order is generated if there is no order and parameter CPX_PARAM_MIPORDTYPE is nonzero.

**Note:** If the value of ordspace is 0, then the negative of the value of *surplus_p returned indicates the length needed for the arrays indices, priority, and direction.

Example  
```c
status = CPXgetorder (env, lp, &listsize, indices, priority, 
    direction, numcols, &surplus);
```

Possible settings for direction

| CPX_BRANCH_GLOBAL | (0) | use global branching direction setting CPX_PARAM_BRDIR |
| CPX_BRANCH_DOWN   | (1) | branch down first on variable indices[k] |
| CPX_BRANCH_UP     | (2) | branch up first on variable indices[k] |

Parameters  
- `env`
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

cnt_p
A pointer to an integer to contain the number of order entries returned; i.e., the true length of the arrays indices, priority, and direction.

indices
An array where the indices of the variables in the order are to be returned. indices[k] is the index of the variable which is entry k in the order information.

priority
An array where the priority values are to be returned. The priority corresponding to the indices[k] is returned in priority[k]. May be NULL. If priority is not NULL, it must be of length at least ordspace.

direction
An array where the preferred branching directions are to be returned. The direction corresponding to indices[k] is returned in direction[k]. May be NULL. If direction is not NULL, it must be of length at least ordspace. Possible settings for direction[k] appear in the table.

ordspace
An integer indicating the length of the non-NULL arrays indices, priority, and direction. May be 0.

surplus_p
A pointer to an integer to contain the difference between ordspace and the number of entries in each of the arrays indices, priority, and direction. A nonnegative value of *surplus_p indicates that the length of the arrays was sufficient. A negative value indicates that the length was insufficient and that the routine could not complete its task. In this case, the routine CPXgetorder returns the value CPXERR_NEGATIVE_SURPLUS, and the negative value of *surplus_p indicates the amount of insufficient space in the arrays.

**Returns**

The routine returns zero on success and nonzero if an error occurs. The value CPXERR_NEGATIVE_SURPLUS indicates that insufficient space was available in the indices, priority, and direction arrays to hold the priority order information.
**CPXgetparamname**

**Category** Global Function

**Definition File** cplex.h

**Include Files** cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXgetparamname(CPXENVptr env,
                               int whichparam,
                               char * name_str)
```

**Description**

The routine `CPXgetparamname` returns the name of a CPLEX parameter, given the symbolic constant (or reference number) for it.

The reference manual *ILOG CPLEX Parameters* provides a list of parameters with their types, options, and default values.

**Example**

```c
status = CPXgetparamname (env, CPX_PARAM_ADVIND, param_string);
```

**Parameters**

- **env**
  - A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.  

- **whichparam**
  - An integer indicating the symbolic constant (or reference number) of the desired parameter.

- **name_str**
  - A character array to receive the name of the selected parameter.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
**CPXgetparamnum**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXgetparamnum(CPXENVptr env,
                               const char * name_str,
                               int * whichparam_p)
```

**Description**  
The routine `CPXgetparamnum` returns the reference number of a CPLEX parameter, given a character string containing the name for it.

The reference manual *ILOG CPLEX Parameters* provides a list of parameters with their types, options, and default values.

**Example**
```c
status = CPXgetparamnum (env, "CPX_PARAM_ADVIND", param_number);
```

**Parameters**

- **env**  
A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- **name_str**  
A character array containing the name of the target parameter.

- **whichparam_p**  
A pointer to an integer to receive the reference number.

**Returns**  
The routine returns zero on success and nonzero if an error occurs.
CPXgetphase1cnt

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXgetphase1cnt(CPXCENVptr env, 
CPXCLPptr lp)
```

Description:

The routine CPXgetphase1cnt is used to access the number of Phase I iterations to solve a problem using the primal or dual simplex method.

Example:

```c
itcnt = CPXgetphase1cnt (env, lp);
```

Parameters:

- `env`:
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- `lp`:
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

Example:

```c
itcnt = CPXgetphase1cnt (env, lp);
```

Returns:

If a solution exists, CPXgetphase1cnt returns the Phase I iteration count. If no solution exists, CPXgetphase1cnt returns the value 0.
CPXgetpi

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:
```c
int CPXPUBLIC CPXgetpi(CPXENVptr env,
                        CPXCLPptr lp,
                        double * pi,
                        int begin,
                        int end)
```

Description:
The routine CPXgetpi is used to access the dual values for a range of the constraints of a linear or quadratic program. The beginning and end of the range must be specified.

Example:
```c
status = CPXgetpi (env, lp, pi, 0, CPXgetnumrows(env,lp)-1);
```

Parameters:
- **env**: A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **lp**: A pointer to a CPLEX problem object as returned by CPXcreateprob.
- **pi**: An array to receive the values of the dual variables for each of the constraints. This array must be of length at least (end - begin + 1). If successful, pi[0] through pi[end - begin] contain the dual values.
- **begin**: An integer indicating the beginning of the range of dual values to be returned.
- **end**: An integer indicating the end of the range of dual values to be returned.

Example:
```c
status = CPXgetpi (env, lp, pi, 0, CPXgetnumrows(env,lp)-1);
```

Returns:
The routine returns zero on success and nonzero if an error occurs.
CPXgetprobname

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:
```c
int CPXPUBLIC CPXgetprobname(CPXENVptr env, CPXCLPptr lp, char * buf_str, int bufspace, int * surplus_p)
```

Description:
The routine CPXgetprobname is used to access the name of the problem set via the call to CPXcreateprob.

Example:
```c
status = CPXgetprobname (env, lp, cur_probname, lenname, &surplus);
```

Parameters:
- `env`:
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- `lp`:
  A pointer to a CPLEX problem object as returned by CPXcreateprob.
- `buf_str`:
  A pointer to a buffer of size `bufspace`. May be NULL if `bufspace` is 0.
- `bufspace`:
  An integer indicating the length of the array `buf_str`. May be 0.
- `surplus_p`:
  A pointer to an integer to contain the difference between `bufspace` and the amount of memory required to store the problem name. A nonnegative value of `*surplus_p`
indicates that the length of the array \texttt{buf\_str} was sufficient. A negative value indicates that the length of the array was insufficient and that the routine could not complete its task. In this case, \texttt{CPXgetprobnanme} returns the value \texttt{CPXERR\_NEGATIVE\_SURPLUS}, and the negative value of the variable \texttt{*surplus\_p} indicates the amount of insufficient space in the array \texttt{buf\_str}.

**Returns**

The routine returns zero on success and nonzero if an error occurs. The value \texttt{CPXERR\_NEGATIVE\_SURPLUS} indicates that insufficient space was available in the \texttt{buf\_str} array to hold the problem name.
CPXgetprobtype

Category                  Global Function
Definition File           cplex.h
Include Files             cplex.h
Synopsis                  int CPXPUBLIC CPXgetprobtype(CPXENVptr env, CPXCLPptr lp)
Description               The routine CPXgetprobtype is used to access the problem type that is currently stored in a CPLEX problem object.
Example                   probtype = CPXgetprobtype (env, lp);

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>?</td>
<td>Error: no problem or environment.</td>
</tr>
<tr>
<td>0</td>
<td>CPXPROB_LP</td>
<td>Linear program; no quadratic data or ctype information stored.</td>
</tr>
<tr>
<td>1</td>
<td>CPXPROB_MILP</td>
<td>Problem with ctype information.</td>
</tr>
<tr>
<td>3</td>
<td>CPXPROB_FIXEDMILP</td>
<td>Problem with ctype information, integer variables fixed.</td>
</tr>
<tr>
<td>5</td>
<td>CPXPROB_QP</td>
<td>Problem with quadratic data stored.</td>
</tr>
<tr>
<td>7</td>
<td>CPXPROB_MIQP</td>
<td>Problem with quadratic data and ctype information.</td>
</tr>
<tr>
<td>8</td>
<td>CPXPROB_FIXEDMIQP</td>
<td>Problem with quadratic data and ctype information, integer variables fixed.</td>
</tr>
<tr>
<td>10</td>
<td>CPXPROB_QCP</td>
<td>Problem with quadratic constraints.</td>
</tr>
</tbody>
</table>
Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>CPXPROB_MIQCP</td>
<td>Problem with quadratic constraints and ctype information.</td>
</tr>
</tbody>
</table>

See Also

CPXchgprobtype

Parameters

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **ip**
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

Returns

The values returned by CPXgetprobtype appear in the table.
CPXgetpsbcnt

Category          Global Function
Definition File   cplex.h
Synopsis

    int CPXPUBLIC CPXgetpsbcnt (CPXCENVptr env,  
                                   CPXCLp.ptr lp)

Description

The routine CPXgetpsbcnt is used to access the number of primal super-basic  
variables in the current solution.

Example

    itcnt = CPXgetpsbcnt (env, lp);

Parameters

    env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

    lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

Example

    itcnt = CPXgetpsbcnt (env, lp);

Returns

If a solution exists, CPXgetpsbcnt returns the number of primal super-basic  
variables. If no solution exists, CPXgetpsbcnt returns the value 0 (zero).
CPXgetqconstr

Category          Global Function
Definition File   cplex.h
Include Files    cplex.h

Synopsis

```
int CPXPUBLIC CPXgetqconstr (CPXCENVptr env,
    CPXCLPptr lp,
    int * linnzcnt_p,
    int * quadnzcnt_p,
    double * rhs_p,
    char * sense_p,
    int * linind,
    double * linval,
    int linspace,
    int * linsurplus_p,
    int * quadrow,
    int * quadcol,
    double * quadval,
    int quadspace,
    int * quadsurplus_p,
    int which)
```

Description

The routine CPXgetqconstr is used to access a specified quadratic constraint on the variables of a CPLEX problem object. The length of the arrays in which the nonzero linear and quadratic coefficients of the constraint are to be returned must be specified.

Note: If the value of linspace is 0 then the negative of the value of *linsurplus_p returned indicates the length needed for the arrays linval and rmatind.

Note: If the value of quadspace is 0 then the negative of the value of *quadsurplus_p returned indicates the length needed for the arrays quadrow, quadcol and quadval.

Example

```
status = CPXgetqconstr (env, lp, &linnzcnt, &quadnzcnt, linind, linval,
    linspace, &linsurplus, quadrow, quadcol, quadval,
    quadspace, &quadsurplus, 0);
```
**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by the CPXopenCPLEX routine.

- **lp**
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

- **linznzcnt_p**
  A pointer to an integer to contain the number of linear coefficients returned; that is, the true length of the arrays linind and linval.

- **quadznzcnt_p**
  A pointer to an integer to contain the number of quadratic coefficients returned; that is, the true length of the arrays quadrow, quadcol and quadval.

- **linind**
  An array to contain the variable indices of the entries of linval. May be NULL if linspace is 0.

- **linval**
  An array to contain the linear coefficients of the specified constraint. May be NULL if linspace is 0.

- **linspace**
  An integer indicating the length of the arrays linind and linval. May be 0.

- **linsurplus_p**
  A pointer to an integer to contain the difference between linspace and the number of entries in each of the arrays linind and linval. A nonnegative value of *linsurplus_p* indicates that the length of the arrays was sufficient. A negative value indicates that the length was insufficient and that the routine could not complete its task. In this case, the routine CPXgetqconstr returns the value CPXERR_NEGATIVE_SURPLUS, and the negative value of *linsurplus_p* indicates the amount of insufficient space in the arrays. May be NULL if linspace is 0.

- **quadrow**
  An array to contain the variable indices of the entries of quadval. If the quadratic coefficients were stored in a matrix, quadrow would give the row indexes of the quadratic terms. May be NULL if quadspace is 0.

- **quadcol**
An array to contain the variable indices of the entries of quadval. If the quadratic coefficients were stored in a matrix, quadcol would give the column indexes of the quadratic terms. May be NULL if quadspace is 0.

quadval

An array to contain the quadratic coefficients of the specified constraint. May be NULL if quadspace is 0.

quadspace

An integer indicating the length of the arrays quadrow, quadcol and quadval. May be 0.

quadsurplus_p

A pointer to an integer to contain the difference between quadspace and the number of entries in each of the arrays quadrow, quadcol and quadval. A nonnegative value of *quadsurplus_p indicates that the length of the arrays was sufficient. A negative value indicates that the length was insufficient and that the routine could not complete its task. In this case, the routine CPXgetqconstr returns the value CPXERR_NEGATIVE_SURPLUS, and the negative value of *quadsurplus_p indicates the amount of insufficient space in the arrays. May be NULL if quadspace is 0.

which

An integer indicating which quadratic constraint to return.

Returns

The routine returns zero on success and nonzero if an error occurs. The value CPXERR_NEGATIVE_SURPLUS indicates that insufficient space was available in either the arrays linind and linval or quadrow, quadcol, and quadval to hold the nonzero coefficients.
CPXgetqconstrindex

Category       Global Function
Definition File cplex.h
Include Files  cplex.h
Synopsis        int CPXPUBLIC CPXgetqconstrindex(CPXENVptr env,
                                             CPXCLPptr lp,
                                             const char * lname_str,
                                             int * index_p)
Description     The routine CPXgetqconstrindex searches for the index number of the specified
                 quadratic constraint in a CPLEX problem object.
Example         status = CPXgetqconstrindex (env, lp, "resource89", &qconstrindex);

Parameters      env
                A pointer to the CPLEX environment as returned by CPXopenCPLEX.
                lp
                A pointer to a CPLEX problem object as returned by CPXcreateprob.
                lname_str
                A quadratic constraint name to search for.
                index_p
                A pointer to an integer to hold the index number of the quadratic constraint with name
                lname_str. If the routine is successful, *index_p contains the index number;
                otherwise, *index_p is undefined.

Returns         The routine returns zero on success and nonzero if an error occurs.
CPXgetqconstrname

Category  Global Function
Definition File  cplex.h
Include Files  cplex.h
Synopsis
int CPXPUBLIC CPXgetqconstrname(CPXENVptr env,
    CPXCLPptr lp,
    char * buf_str,
    int bufspace,
    int * surplus_p,
    int which)
Description
The routine CPXgetqconstrname is used to access the name of a specified quadratic constraint of a CPLEX problem object.

Note: If the value of bufspace is 0, then the negative of the value of *surplus_p returned indicates the total number of characters needed for the array buf_str.

Example
status = CPXgetqconstrname (env, lp, qname, lenqname,
    &surplus, 5);

Parameters
env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.
buf_str
A pointer to a buffer of size bufspace. May be NULL if bufspace is 0.
bufspace
An integer indicating the length of the array buf_str. May be 0.
surplus_p
A pointer to an integer to contain the difference between `bufspace` and the amount of memory required to store the quadratic constraint name. A nonnegative value of `*surplus_p` indicates that the length of the array `buf_str` was sufficient. A negative value indicates that the length of the array was insufficient and that the routine could not complete its task. In this case, `CPXgetqconstrname` returns the value `CPXERR_NEGATIVE_SURPLUS`, and the negative value of the variable `*surplus_p` indicates the amount of insufficient space in the array `buf_str`.

An integer indicating the index of the quadratic constraint for which the name is to be returned.

**Returns**

The routine returns zero on success and nonzero if an error occurs. The value `CPXERR_NEGATIVE_SURPLUS` indicates that insufficient space was available in the `buf_str` array to hold the quadratic constraint name.
CPXgetqconstrslack

**Category**  Global Function

**Definition File**  cplex.h

**Include Files**  cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXgetqconstrslack(CPXCENVptr env,
                                  CPXCLPptr lp,
                                  double * qcslack,
                                  int begin,
                                  int end)
```

**Description**  
The routine `CPXgetqconstrslack` is used to access the slack values for a range of the quadratic constraints of a quadratically constrained program. The beginning and end of the range must be specified. The slack values returned consist of the right-hand side minus the constraint activity level.

**Example**  
```c
status = CPXgetqconstrslack (env, lp, qcslack, 0,
                               CPXgetnumqconstrs(env,lp)-1);
```

**Parameters**  
- **env**  
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
- **lp**  
  A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
- **qcslack**  
  An array to receive the values of the slack or surplus variables for each of the constraints. This array must be of length at least \((end - begin + 1)\). If successful, `qcslack[0]` through `qcslack[end - begin]` contain the values of the slacks.
- **begin**  
  An integer indicating the beginning of the range of slack values to be returned.
- **end**  
  An integer indicating the end of the range of slack values to be returned.

**Returns**  
The routine returns zero on success and nonzero if an error occurs.
CPXgetqpcoef

Category               Global Function
Definition File        cplex.h
Include Files          cplex.h
Synopsis               int CPXPUBLIC CPXgetqpcoef (CPXCENVptr env,
                        CPXCLPptr lp,
                        int rownum,
                        int colnum,
                        double * coef_p)
Description             The routine CPXgetqpcoef accesses the quadratic coefficient in the
                        matrix Q of a CPLEX problem object for the variable pair
                        indexed by (rownum, colnum). The result is stored in *coef_p.
Example                 status = CPXgetqpcoef (env, lp, 10, 20, &coef);
Parameters              env
                        A pointer to the CPLEX environment as returned by CPXopenCPLEX.
                        lp
                        A pointer to a CPLEX problem object as returned by CPXcreateprob.
                        rownum
                        The first variable number (row number in Q).
                        colnum
                        The second variable number (column number in Q).
                        coef_p
                        A pointer to a double where the coefficient should be stored.
Returns                 The routine returns zero on success and nonzero if an error occurs.
CPXgetquad

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXgetquad(CPXENVptr env,
                        CPXCLPptr lp,
                        int * nzcnt_p,
                        int * qmatbeg,
                        int * qmatind,
                        double * qmatval,
                        int qmatspace,
                        int * surplus_p,
                        int begin,
                        int end)
```

Description:

The routine CPXgetquad is used to access a range of columns of the matrix Q of a model with a quadratic objective function. The beginning and end of the range, along with the length of the arrays in which the nonzero entries of these columns are to be returned, must be specified.

Specifically, column \( j \) consists of the entries in \( qmatval \) and \( qmatind \) in the range from \( qmatbeg[j - begin] \) to \( qmatbeg[(j + 1) - begin]-1 \). (Column end consists of the entries from \( qmatbeg[end - begin] \) to \( nzcnt_p-1 \).) This array must be of length at least \( (end - begin + 1) \).

A nonnegative value of \( surplus_p \) indicates that the length of the arrays was sufficient. A negative value indicates that the length was insufficient and that the routine could not complete its task. In this case, CPXgetquad returns the value CPXERR_NEGATIVE_SURPLUS, and the negative value of \( surplus_p \) indicates the amount of insufficient space in the arrays.

**Note:** If the value of \( qmatspace \) is zero, the negative of the value of \( surplus_p \) returned indicates the length needed for the arrays \( qmatind \) and \( qmatval \).

Example:

```c
status = CPXgetquad (env, lp, &nzcnt, qmatbeg, qmatind,
                     qmatval, qmatspace, &surplus, 0,
```


Parameters

- env
  A pointer to the CPLEX environment as returned by CPXopenCplex.

- lp
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

- nzcnt_p
  A pointer to an integer to contain the number of nonzeros returned; that is, the true length of the arrays qmatind and qmatval.

- qmatbeg
  An array to contain indices indicating where each of the requested columns of Q begins in the arrays qmatval and qmatind.

- qmatind
  An array to contain the row indices associated with the elements of qmatval. May be NULL if cmatspace is zero.

- qmatval
  An array to contain the nonzero coefficients of the specified columns. May be NULL if cmatspace is zero.

- cmatspace
  An integer indicating the length of the arrays qmatind and qmatval. May be zero.

- surplus_p
  A pointer to an integer to contain the difference between cmatspace and the number of entries in each of the arrays qmatind and qmatval.

- begin
  An integer indicating the beginning of the range of columns to be returned.

- end
  An integer indicating the end of the range of columns to be returned.

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXgetrhs

Category  
Global Function

Definition File  
cplex.h

Include Files  
cplex.h

Synopsis  
int CPXPUBLIC CPXgetrhs(CPXENVptr env,  
CPXCLPptr lp,  
double * rhs,  
int begin,  
int end)

Description  
The routine CPXgetrhs is used to access the right-hand side coefficients for a range of constraints in a CPLEX problem object. The beginning and end of the range must be specified.

Example  
status = CPXgetrhs (env, lp, rhs, 0, cur_numrows-1);

Parameters  

env  
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp  
A pointer to a CPLEX problem object as returned by CPXcreateprob.

rhs  
An array where the specified right-hand side coefficients are to be returned. This array must be of length at least (end - begin + 1). The right-hand side of constraint i is returned in rhs[i - begin].

begin  
An integer indicating the beginning of the range of right-hand side terms to be returned.

del  
An integer indicating the end of the range of right-hand side terms to be returned.

Returns  
The routine returns zero on success and nonzero if an error occurs.
CPXgetrngval

**Category**  Global Function

**Definition File**  cplex.h

**Include Files**  cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXgetrngval(CPXCENVptr env,
            CPXCLPptr lp,
            double * rngval,
            int begin,
            int end)
```

**Description**

The routine CPXgetrngval is used to access the RHS range coefficients for a set of constraints in a CPLEX problem object. The beginning and end of the set must be specified. CPXgetrngval checks if ranged constraints are present in the problem object.

**Example**

```c
    status = CPXgetrngval (env, lp, rngval, 0, cur_numrows-1);
```

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **lp**
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

- **rngval**
  An array where RHS range coefficients are returned. This array must be of length at least \((end - begin + 1)\). A value of 0 for any entry means that the corresponding row is not ranged.

- **begin**
  An integer indicating the beginning of the set of rows for which RHS range coefficients are returned.

- **end**
  An integer indicating the end of the set of rows for which RHS range coefficients are returned.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXgetrowindex

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXgetrowindex (CPXCENVptr env, CPXCLPptr lp, const char * lname_str, int * index_p);
```

Description:

The routine CPXgetrowindex searches for the index number of the specified row in a CPLEX problem object.

Example:

```c
status = CPXgetrowindex (env, lp, "resource89", &rowindex);
```

Parameters:

- `env`
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- `lp`
  A pointer to a CPLEX problem object as returned by CPXcreateprob.
- `lname_str`
  A row name to search for.
- `index_p`
  A pointer to an integer to hold the index number of the row with name `lname_str`. If the routine is successful, `*index_p` contains the index number; otherwise, `*index_p` is undefined.

Returns:

The routine returns zero on success and nonzero if an error occurs.
CPXgetrowname

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXgetrowname(CPXENVptr env,
    CPXCLPptr lp,
    char ** name,
    char * namestore,
    int storespace,
    int * surplus_p,
    int begin,
    int end)
```

Description:

The routine CPXgetrowname is used to access a range of row names or, equivalently, the constraint names of a CPLEX problem object. The beginning and end of the range, along with the length of the array in which the row names are to be returned, must be specified.

**Note:** If the value of `storespace` is 0, then the negative of the value of `*surplus_p` returned indicates the total number of characters needed for the array `namestore`.

Example:

```c
status = CPXgetrowname (env, lp, cur_rowname, cur_rownamestore,
    cur_storespace, &surplus, 0,
    cur_numrows-1);
```

Parameters:

- `env`:
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- `lp`:
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

- `name`:
  A pointer to a CPLEX problem object as returned by CPXcreateprob.
An array of pointers to the row names stored in the array `namestore`. This array must be of length at least \((end - begin + 1)\). The pointer to the name of row \(i\) is returned in `name[i-begin]`.

`namestore`

An array of characters where the specified row names are to be returned. May be NULL if `storespace` is 0.

`storespace`

An integer indicating the length of the array `namestore`. May be 0.

`surplus_p`

A pointer to an integer to contain the difference between `storespace` and the total amount of memory required to store the requested names. A nonnegative value of `*surplus_p` indicates that `storespace` was sufficient. A negative value indicates that it was insufficient and that the routine could not complete its task. In that case, `CPXgetrownames` returns the value `CPXERR_NEGATIVE_SURPLUS`, and the negative value of the variable `*surplus_p` indicates the amount of insufficient space in the array `namestore`.

`begin`

An integer indicating the beginning of the range of row names to be returned.

`end`

An integer indicating the end of the range of row names to be returned.

**Returns**

The routine returns zero on success and nonzero if an error occurs. The value `CPXERR_NEGATIVE_SURPLUS` indicates that insufficient space was available in the `namestore` array to hold the names.
CPXgetrows

Category                       Global Function
Definition File                cplex.h
Include Files                  cplex.h

Synopsis
int CPXPUBLIC CPXgetrows(CPXENVptr env, 
                         CPXCLPptr lp, 
                         int * nzcnt_p, 
                         int * rmatbeg, 
                         int * rmatind, 
                         double * rmatval, 
                         int rmatspace, 
                         int * surplus_p, 
                         int begin, 
                         int end)

Description
The routine CPXgetrows is used to access a range of rows of the constraint matrix, not including the objective function or the bounds constraints on the variables of a CPLEX problem object. The beginning and end of the range, along with the length of the arrays in which the nonzero entries of these rows are to be returned, must be specified.

Note: If the value of rmatspace is 0 then the negative of the value of *surplus_p returned indicates the length needed for the arrays rmatval and rmatind.

Example

    status = CPXgetrows (env, lp, &nzcnt, rmatbeg, rmatind, rmatval, 
                         rmatspace, &surplus, 0, cur_numrows-1);

Parameters

    env
A pointer to the CPLEX environment as returned by the CPXopenCPLEX routine.

    lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

    nzcnt_p
A pointer to an integer to contain the number of nonzeros returned; that is, the true length of the arrays rmatind and rmatval.

rmatbeg

An array to contain indices indicating where each of the requested rows begins in the arrays rmatval and rmatind. Specifically, row i consists of the entries in rmatval and rmatind in the range from rmatbeg[i - begin] to rmatbeg[(i + 1) - begin] - 1. (Row end consists of the entries from rmatbeg[end - begin] to *nzcnt_p - 1.) This array must be of length at least(end - begin + 1).

rmatind

An array to contain the column indices of the entries of rmatval. May be NULL if rmatspace is 0.

rmatval

An array to contain the nonzero entries of the specified rows. May be NULL if rmatspace is 0.

rmatspace

An integer indicating the length of the arrays rmatind and rmatval. May be 0.

surplus_p

A pointer to an integer to contain the difference between rmatspace and the number of entries in each of the arrays rmatind and rmatval. A nonnegative value of *surplus_p indicates that the length of the arrays was sufficient. A negative value indicates that the length was insufficient and that the routine could not complete its task. In this case, the routine CPXgetrows returns the value CPXERR_NEGATIVE_SURPLUS, and the negative value of *surplus_p indicates the amount of insufficient space in the arrays.

begin

An integer indicating the beginning of the range of rows to be returned.

end

An integer indicating the end of the range of rows to be returned.

Returns

The routine returns zero on success and nonzero if an error occurs. The value CPXERR_NEGATIVE_SURPLUS indicates that insufficient space was available in the arrays rmatind and rmatval to hold the nonzero coefficients.
**CPXgetsense**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXgetsense(CPXENVptr env,
    CPXCLPptr lp,
    char * sense,
    int begin,
    int end)
```

**Description**  
The routine `CPXgetsense` is used to access the sense for a range of constraints in a CPLEX problem object. The beginning and end of the range must be specified.

**Example**  
```c
status = CPXgetsense (env, lp, sense, 0, cur_numrows-1);
```

**Values of sense**

<table>
<thead>
<tr>
<th>sense[i]</th>
<th>= 'L'</th>
<th>≤ constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>sense[i]</td>
<td>= 'E'</td>
<td>= constraint</td>
</tr>
<tr>
<td>sense[i]</td>
<td>= 'G'</td>
<td>≥ constraint</td>
</tr>
<tr>
<td>sense[i]</td>
<td>= 'R'</td>
<td>ranged constraint</td>
</tr>
</tbody>
</table>

**Parameters**

- **env**  
A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- **lp**  
A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

- **sense**  
An array where the specified constraint senses are to be returned. This array must be of length at least (end - begin + 1). The sense of constraint i is returned in sense[i - begin]. Possible values appear in the table.

- **begin**  
An integer indicating the beginning of the range of constraint senses to be returned.

- **end**  

An integer indicating the end of the range of constraint senses to be returned.

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXgetSiftItCnt

Category  Global Function
Definition File  cplex.h
Include Files  cplex.h
Synopsis  
int CPXPUBLIC CPXgetSiftItCnt (CPXENVptr env,
               CPXCLPptr lp)
Description  The routine CPXgetSiftItCnt is used to access the total number of sifting iterations to solve an LP problem.
Example  
   itcnt = CPXgetSiftItCnt (env, lp);
Parameters  
   env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
   lp
A pointer to a CPLEX LP problem object as returned by CPXcreateprob.
Example  
   itcnt = CPXgetSiftItCnt (env, lp);
Returns  The routine returns the total iteration count if a solution exists. It returns zero if no solution exists or any other type of error occurs.
**CPXgetsiftphase1cnt**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXgetsiftphase1cnt(CPXENVptr env, 
CPXCLPptr lp)
```

**Description**  
The routine `CPXgetsiftphase1cnt` is used to access the number of Phase I sifting iterations to solve an LP problem.

**Example**  
```c
itcnt = CPXgetsiftphase1cnt (env, lp);
```

**Parameters**  

- **env**  
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
- **lp**  
  A pointer to a CPLEX LP problem object as returned by `CPXcreateprob`.

**Example**  
```c
itcnt = CPXgetsiftphase1cnt (env, lp);
```

**Returns**  
The routine returns the Phase I iteration count if a solution exists. It returns zero if no solution exists or any other type of error occurs.
CPXgetslack

Category            Global Function
Definition File     cplex.h
Include Files       cplex.h
Synopsis

        int CPXPUBLIC CPXgetslack (CPXCENVptr env, 
        CPXCLPptr lp, 
        double * slack, 
        int begin, 
        int end)

Description

The routine CPXgetslack is used to access the slack values for a range of the constraints of a linear or quadratic program. The beginning and end of the range must be specified. Except for ranged rows, the slack values returned consist of the right-hand side minus the row activity level. For ranged rows, the value returned is the row activity level minus the right-hand side, or, equivalently, the value of the internal structural variable that CPLEX creates to represent ranged rows.

Example

        status = CPXgetslack (env, lp, slack, 0, CPXgetnumrows(env, lp)-1);

Parameters

        env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

        lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

        slack
An array to receive the values of the slack or surplus variables for each of the constraints. This array must be of length at least (end-begin+1). If successful, slack[0] through slack[end-begin] contain the values of the slacks.

        begin
An integer indicating the beginning of the range of slack values to be returned.

        end
An integer indicating the end of the range of slack values to be returned.

Example
Returns

The routine returns zero on success and nonzero if an error occurs.
CPXgetsolvecallbackfunc

Category Global Function

Definition File cplex.h

Include Files cplex.h

Synopsis

void CPX_PUBLIC CPXgetsolvecallbackfunc(CPXENVptr env, int(CPX_PUBLIC **solvecallback_p)(CALLBACK_SOLVE_ARGS) , void ** cbhandle_p)

Description

Note: This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetsolvecallbackfunc accesses the user-written callback to be called during MIP optimization to optimize the subproblem.

Example

CPXgetsolvecallbackfunc(env, &current_callback, &current_cbdata);

See also Advanced MIP Control Interface in the ILOG CPLEX User’s Manual.

For documentation of callback arguments, see the routine CPXsetsolvecallbackfunc.

Parameters

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

solvecallback_p
The address of the pointer to the current user-written solve callback. If no callback has been set, the pointer evaluates to NULL.

cbhandle_p
The address of a variable to hold the user’s private pointer.
See Also

CPXgetcallbacknodelp, CPXsetsolvecallbackfunc

Returns

This routine does not return a result.
**CPXgetsos**

**Category**  Global Function

**Definition File**  cplex.h

**Include Files**  cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXgetsos(CPXENVptr env,
                           CPXCLPptr lp,
                           int * numsosnz_p,
                           char * sostype,
                           int * sospri,
                           int * sosbeg,
                           int * sosind,
                           double * soswt,
                           int sosspace,
                           int * surplus_p,
                           int begin,
                           int end)
```

**Description**

The routine CPXgetsos is used to access a range of Special Ordered Set (SOS) definitions stored in a CPLEX problem object. The beginning and end of the range, along with the length of the array in which the definitions are to be returned, must be provided.

**Example**

```c
status = CPXgetsos (env, lp, &numsosnz, sostype, sospri, sosbeg,
                    sosind, soswt, sosspace, 0, numsos-1);
```

**Parameters**

- `env`
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- `lp`
  A pointer to a CPLEX problem object as returned by CPXcreateprob.
- `numsosnz_p`

---

**Note:** If the value of sosspace is 0 (zero), then the negative of the value of *surplus_p returned indicates the length needed for the arrays sosind and soswt.
A pointer to an integer to contain the number of set members returned; i.e., the true length of the arrays sosind and soswt.

sostype
An array to contain the types of the requested SOSs. The type of set k is returned in sostype[k-begin]. This array must be of length at least (end - begin + 1). Contains either CPX_TYPE_SOS1 ('1') or CPX_TYPE_SOS2 ('2'), for a type 1 or type 2 SOS respectively.

sospri
An array to contain the priorities of the SOSs. The priority of set i is returned in sospri[i-begin]. This array must be of length at least (end - begin + 1). May be NULL.

sosbeg
An array to contain indices indicating where each of the requested SOSs begins in the arrays sosind and soswt. Specifically, set k consists of the entries in sosind and soswt in the range from sosbeg[k-begin] to sosbeg[(k + 1) - begin] - 1. (Set end consists of the entries from sosbeg[end - begin] to *num sosnz_p - 1.) This array must be of length at least (end - begin + 1).

sosind
An array to contain the variable indices of the SOS members. May be NULL if sosspace is 0.

soswt
An array to contain the reference values (weights) for SOS members. May be NULL if sosspace is 0. Weight soswt[k] corresponds to sosind[k].

sosspace
An integer indicating the length of the arrays sosind and soswt. May be 0.

surplus_p
A pointer to an integer to contain the difference between sosspace and the number of entries in each of the arrays sosind and soswt. A nonnegative value of *surplus_p indicates that the length of the arrays was sufficient. A negative value indicates that the length was insufficient and that the routine could not complete its task. In this case, the routine CPXgetsos returns the value CPXERR_NEGATIVE_SURPLUS, and the negative value of *surplus_p indicates the amount of insufficient space in the arrays.

begin
An integer indicating the beginning of the range of SOSs to be returned.

end
An integer indicating the end of the range of SOSs to be returned.

**Returns**

The routine returns zero on success and nonzero if an error occurs. The value CPXERR_NEGATIVE_SURPLUS indicates that insufficient space was available in the arrays sosind and soswt to hold the SOS definition.
CPXgetstat

Category          Global Function
Definition File   cplex.h
Include Files     cplex.h
Synopsis          int CPXPUBLIC CPXgetstat(CPXENVptr env,
                           CPXCLPptr lp)
Description       The routine CPXgetstat is used to access the solution status of the problem after an
                   LP, QP, QCP, or mixed integer optimization.
Example            lpstat = CPXgetstat (env, lp);

Parameters

  env
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

  lp
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

Returns            The routine returns the solution status of the most recent optimization performed on the
                   CPLEX problem object. Return values are as shown in group optim.cplex.solutionstatus. For
                   status code CPX_STAT_NUM_BEST, the algorithm could not converge to the requested tolerances due to numeric
difficulties. The best solution found can be retrieved by the routine CPXsolution. Similarly, when an abort status is returned,
                   the last solution computed before the algorithm aborted can be retrieved using CPXsolution. Use the query
                   routines CPXsolninfo and CPXsolution to obtain further information about the current solution of an LP or QP.
**CPXgetstatstring**

**Category**       Global Function  
**Definition File** cplex.h  
**Synopsis**       

```c
CPXCHARptr CPXPUBLIC CPXgetstatstring(CPXCENVptr env,  
    int statind,  
    char * buffer_str)
```

**Description**  The routine CPXgetstatstring is used to place in a buffer, a string corresponding to the value of statind as returned by the CPXgetstat function. The buffer to hold the string can be up to 510 characters maximum.

**Example**  

```c
statind = CPXgetstat (env, lp);
p = CPXgetstatstring (env, statind, buffer);
```

**Parameters**  

- **env**  
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **statind**  
  An integer indicating the status value to return.

- **buffer_str**  
  A pointer to a buffer to hold the string corresponding to the value of statind.

**Example**  

```c
statind = CPXgetstat (env, lp);
p = CPXgetstatstring (env, statind, buffer);
```

**Returns**  The routine returns a pointer to a buffer if the statind value corresponds to a valid string. Otherwise, it returns NULL.
**CPXgetstrparam**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**
```c
int CPXPUBLIC CPXgetstrparam(CPXENVptr env,
                              int whichparam,
                              char * value_str)
```

**Description**
The routine CPXgetstrparam is used to obtain the current value of a CPLEX string parameter.

**Example**
```c
status = CPXgetstrparam (env, CPX_PARAM_NODEFILEDIR, dirname);
```

**Parameters**
- `env`
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- `whichparam`
  The symbolic constant (or reference number) of the parameter for which the value is to be obtained.
- `value_str`
  A pointer to a buffer of length at least `CPX_STR_PARAM_MAX` to hold the current value of the CPLEX parameter.

**Returns**
The routine returns zero on success and nonzero if an error occurs.
CPXgetsubmethod

Category Global Function
Definition File cplex.h
Include Files cplex.h
Synopsis
int CPXPUBLIC CPXgetsubmethod(CPXENVptr env, CPXCLPptr lp)
Description
The routine CPXgetsubmethod is used to access the solution method of the last subproblem optimization, in the case of an error termination during mixed integer optimization.
Example
    submethod = CPXgetsubmethod (env, lp);
Parameters
    env
    A pointer to the CPLEX environment as returned by CPXopenCPLEX.
    lp
    A pointer to a CPLEX problem object as returned by CPXcreateprob.
    Example
    submethod = CPXgetsubmethod (env, lp);
Returns
The possible return values are summarized below.

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CPX_ALG_NONE</td>
<td>None</td>
</tr>
<tr>
<td>1</td>
<td>CPX_ALG_PRIMAL</td>
<td>Primal simplex</td>
</tr>
<tr>
<td>2</td>
<td>CPX_ALG_DUAL</td>
<td>Dual simplex</td>
</tr>
<tr>
<td>4</td>
<td>CPX_ALG_BARRIER</td>
<td>Barrier optimizer (no crossover)</td>
</tr>
</tbody>
</table>
CPXgetsubstat

Category  Global Function
Definition File  cplex.h
Include Files  cplex.h
Synopsis  int CPXPUBLIC CPXgetsubstat(CPXCENVptr env, CPXCLPptr lp)
Description  The routine CPXgetsubstat is used to access the solution status of the last subproblem optimization, in the case of an error termination during mixed integer optimization.
Example  substatus = CPXgetsubstat (env, lp);
See Also  CPXgetsubmethod
Parameters  env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.
Example  substatus = CPXgetsubstat (env, lp);
Returns  The routine returns zero if no solution exists. A nonzero return value indicates there was an error termination where a subproblem could not be solved to completion. The values returned are documented in the group optim.cplex.callable.solutionstatus in the reference manual of the API.
CPXgetub

Category              Global Function
Definition File       cplex.h
Include Files         cplex.h
Synopsis
int CPXPUBLIC CPXgetub(CPXENVptr env, CPXCLPptr lp, double * ub, int begin, int end)

Description
The routine CPXgetub is used to access a range of upper bounds on the variables of a CPLEX problem object. The beginning and end of the range must be specified.

Example
status = CPXgetub (env, lp, ub, 0, cur_numcols-1);

Parameters
env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.
ub
An array where the specified upper bounds on the variables are to be returned. This array must be of length at least (end - begin+1). The upper bound of variable j is returned in ub[j-begin].
begin
An integer indicating the beginning of the range of upper bounds to be returned.
end
An integer indicating the end of the range of upper bounds to be returned.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXgetx

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXgetx(CPXENVptr env,
                        CPXCLPIptr lp,
                        double * x,
                        int begin,
                        int end)
```

Description:
The routine CPXgetx is used to access the solution values for a range of problem variables of a linear, quadratically constrained, or quadratic program. The beginning and end of the range must be specified.

Example:

```c
status = CPXgetx (env, lp, x, 0, CPXgetnumcols(env, lp)-1);
```

See also the example lpex2.c in the ILOG CPLEX User's Manual and in the standard distribution.

Parameters:

- `env`:
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- `lp`:
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

- `x`:
  An array to receive the values of the primal variables for the problem. This array must be of length at least `(end - begin + 1)`. If successful, `x[0]` through `x[end-begin]` contains the solution values.

- `begin`:
  An integer indicating the beginning of the range of variable values to be returned.

- `end`:
  An integer indicating the end of the range of variable values to be returned.

Example:
status = CPXgetx (env, lp, x, 0, CPXgetnumcols(env, lp)-1);

See also the example lpex2.c in the *ILOG CPLEX User's Manual* and in the standard distribution.

**Returns**  
The routine returns zero on success and nonzero if an error occurs.
CPXgetxqxax

Category  Global Function

Definition File  cplex.h

Include Files  cplex.h

Synopsis  

```c
int CPXPUBLIC CPXgetxqxax(CPXENVptr env,
  CPXCLPptr lp,
  double * xqxax,
  int begin,
  int end)
```

Description  The routine CPXgetxqxax is used to access quadratic constraint activity levels for a range of quadratic constraints in a quadratically constrained program (QCP). The beginning and end of the range must be specified. Quadratic constraint activity is the sum of the linear and quadratic terms of the constraint evaluated with the values of the structural variables in the problem.

Parameters

- **env**  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **lp**  A pointer to a CPLEX problem object as returned by CPXcreateprob.
- **xqxax**  An array to receive the values of the quadratic constraint activity levels for each of the constraints in the specified range. The array must be of length at least (end-begin+1). If successful, x[0] through x[end-begin] contain the quadratic constraint activities.
- **begin**  An integer indicating the beginning of the range of quadratic constraint activities to be returned.
- **end**  An integer indicating the end of the range of quadratic constraint activities to be returned.

Returns  The routine returns zero on success and nonzero if an error occurs.
CPXhybbaropt

Category Global Function
Definition File cplex.h
Include Files cplex.h
Synopsis int CPXPUBLIC CPXhybbaropt(CPXCENVptr env, CPXLPptr lp, int method)
Description The routine CPXhybbaropt may be used, at any time after a linear program has been created via a call to CPXcreateprob, to find a solution to that problem. When this function is called, the specified problem is solved using CPLEX Barrier followed by an automatic crossover to a basic solution if barrier determines that the problem is both primal and dual feasible. Otherwise, crossover is not performed. In this case, a call to CPXprimopt or CPXdualopt can force a crossover to occur. The results of the optimization are recorded in the problem object.

Methods of CPXhybbaropt

<table>
<thead>
<tr>
<th>method</th>
<th>use</th>
</tr>
</thead>
<tbody>
<tr>
<td>= 0</td>
<td>CPX_PARAM_BARCROSSALG to choose a crossover method</td>
</tr>
<tr>
<td>= CPX_ALG_PRIMAL</td>
<td>primal crossover</td>
</tr>
<tr>
<td>= CPX_ALG_DUAL</td>
<td>dual crossover</td>
</tr>
<tr>
<td>= CPX_ALG_NONE</td>
<td>no crossover</td>
</tr>
</tbody>
</table>

Example

status = CPXhybbaropt (env, lp, CPX_ALG_PRIMAL);

See also the example lpex2.c in the standard distribution.

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.
method
Crossover method to be implemented, according to the table.
**Returns**

The routine returns zero unless an error occurred during the optimization. Examples of errors include exhausting available memory (CPXERR_NO_MEMORY) or encountering invalid data in the CPLEX problem object (CPXERR_NO_PROBLEM). Exceeding a user-specified CPLEX limit, or proving the model infeasible or unbounded, are not considered errors. Note that a zero return value does not necessarily mean that a solution exists. Use query routines CPXsolninfo, CPXgetstat, and CPXsolution to obtain further information about the status of the optimization.
CPXhybnetopt

Category  Global Function
Definition File  cplex.h
Include Files  cplex.h
Synopsis

```c
int CPXPUBLIC CPXhybnetopt (CPXCENVptr env,
   CPXLPptr lp,
   int method)
```

Description

The routine CPXhybnetopt, given a linear program that has been created via a call to CPXcreateprob, extracts an embedded network, uses the CPLEX Network Optimizer to attempt to obtain an optimal basis to the network, and optimizes the entire linear program using one of the CPLEX simplex methods. CPLEX takes the network basis as input for the optimization of the whole linear program.

<table>
<thead>
<tr>
<th>method</th>
<th>= CPX_ALG_PRIMAL</th>
<th>primal Simplex</th>
</tr>
</thead>
<tbody>
<tr>
<td>method</td>
<td>= CPX_ALG_DUAL</td>
<td>dual Simplex</td>
</tr>
</tbody>
</table>

Example

```c
status = CPXhybnetopt (env, lp, CPX_ALG_DUAL);
```

See also the example lpeX3.c in the ILOG CPLEX User's Manual and in the standard distribution.

Parameters

- env
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- lp
  A pointer to a CPLEX problem object as returned by CPXcreateprob.
- method
  The type of simplex method to follow the network optimization.
Example

Example

status = CPXhybnetopt (env, lp, CPX_ALG_DUAL);

See also the example lpex3.c in the *ILOG CPLEX User's Manual* and in the standard distribution.

Returns

The routine returns zero unless an error occurred during the optimization. Examples of errors include exhausting available memory (CPXERR_NO_MEMORY) or encountering invalid data in the CPLEX problem object (CPXERR_NO_PROBLEM). Exceeding a user-specified CPLEX limit, or proving the model infeasible or unbounded, are not considered errors. Note that a zero return value does not necessarily mean that a solution exists. Use query routines CPXsolninfo, CPXgetstat, and CPXsolution to obtain further information about the status of the optimization.
CPXiiswrite

Category   Global Function
Definition File cplex.h
Include Files cplex.h
Synopsis    
    int CPXPUBLIC CPXiiswrite(CPXENVptr env,
                           CPXLPptr lp,
                           const char * filename_str)

Description The routine CPXiiswrite is used to write an LP format file containing the rows and columns in the Irreducibly Inconsistent Set (IIS) of an infeasible LP. Note that the infeasibility must previously have been detected by a simplex optimizer. If an IIS has already been obtained using the same method as is currently specified by the IIS algorithm indicator CPXPARAM_IISIND, CPXiiswrite simply writes an LP format file containing the existing IIS. If no IIS has been computed, or the IIS algorithm differs from the one used to compute the available IIS, CPXiiswrite first computes an IIS, then writes its LP format representation out.

Example

    status = CPXiiswrite (env, lp, "myprob.iis");

Parameters

    env
    A pointer to the CPLEX environment as returned by CPXopenCPLEX.

    lp
    A pointer to a CPLEX problem object as returned by CPXcreateprob.

    filename_str
    A character string containing the name of the file to which the IIS should be written.

Returns The routine returns zero on success and nonzero if an error occurs.
CPXinfodblparam

Category: Global Function
Definition File: cplex.h
Include Files: cplex.h

Synopsis:
```c
int CPXPUBLIC CPXinfodblparam(CPXENVptr env,
    int whichparam,
    double * defvalue_p,
    double * minvalue_p,
    double * maxvalue_p)
```

Description:
The routine CPXinfodblparam is used to obtain the default, minimum, and maximum values of a CPLEX parameter of type double.

**Note:** Values of zero obtained for both the minimum and maximum values of a parameter of type double indicate that the parameter has no limit.

The reference manual *ILOG CPLEX Parameters*, provides a list of parameters with their types, options, and default values.

**Example**

```c
status = CPXinfodblparam (env, CPX_PARAM_TILIM, &default_tilim,
    &min_tilim, &max_tilim);
```

Parameters:
- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **whichparam**
  The symbolic constant (or reference number) of the parameter value to be obtained.
- **defvalue_p**
  A pointer to a variable of type double to hold the default value of the CPLEX parameter. May be NULL.
- **minvalue_p**
  A pointer to a variable of type double to hold the minimum value of the CPLEX parameter. May be NULL.
maxvalue_p
A pointer to a variable of type double to hold the maximum value of the CPLEX parameter. May be NULL.

Returns
The routine returns zero on success and nonzero if an error occurs.
**CPXinfointparam**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXinfointparam(CPXCENVptr env,
    int whichparam,
    int * defvalue_p,
    int * minvalue_p,
    int * maxvalue_p)
```

**Description**  
The routine `CPXinfointparam` is used to obtain the default, minimum, and maximum values of a CPLEX parameter of type `int`.

The reference manual *ILOG CPLEX Parameters* provides a list of parameters with their types, options, and default values.

**Example**  
```c
status = CPXinfointparam (env, CPX_PARAM_PREIND, &default_preind,
                        &min_preind, &max_preind);
```

**Parameters**  
- **env**  
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
- **whichparam**  
  The symbolic constant (or reference number) of the parameter for which the value is to be obtained.
- **defvalue_p**  
  A pointer to an integer variable to hold the default value of the CPLEX parameter. May be NULL.
- **minvalue_p**  
  A pointer to an integer variable to hold the minimum value of the CPLEX parameter. May be NULL.
- **maxvalue_p**  
  A pointer to an integer variable to hold the maximum value of the CPLEX parameter. May be NULL.

**Returns**  
The routine returns zero on success and nonzero if an error occurs.
**CPinfostrparam**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXinfostrparam(CPXENVptr env,
                               int whichparam,
                               char * defvalue_str)
```

**Description**
The routine `CPXinfostrparam` is used to obtain the default value of a CPLEX string parameter.

**Example**

```c
status = CPXinfostrparam (env, CPX_PARAM_NODEFILEDIR, defdirname);
```

**Parameters**

- `env`
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- `whichparam`
  The symbolic constant (or reference number) of the parameter for which the default value is to be obtained.

- `defvalue_str`
  A pointer to a buffer of length at least `CPX_STR_PARAM_MAX` to hold the default value of the CPLEX parameter.

**Returns**
The routine returns zero on success and nonzero if an error occurs.
CPXlpopt

Category     Global Function
Definition File   cplex.h
Include Files   cplex.h
Synopsis   int CPXPUBLIC CPXlpopt (CPXCENVptr env, CPXLPptr lp)

Description The routine CPXlpopt may be used, at any time after a linear program has been created via a call to CPXcreateprob, to find a solution to that problem using one of the ILOG CPLEX linear optimizers. The parameter CPX_PARAM_LPMETHOD controls the choice of optimizer (dual simplex, primal simplex, barrier, network simplex, sifting, or concurrent optimization). Currently, with the default parameter setting of Automatic, CPLEX invokes the dual simplex method when no advanced basis or starting vector is loaded or when the advanced indication is zero. The behavior of the Automatic setting may change in the future.

Example

status = CPXlpopt (env, lp);

See also the example lplex1.c in Getting Started and in the standard distribution.

See Also   CPXgetstat, CPXsolninfo, CPXsolution

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
A pointer to the CPLEX problem object as returned by CPXcreateprob.

Returns The routine returns zero unless an error occurred during the optimization. Examples of errors include exhausting available memory (CPXERR_NO_MEMORY) or encountering invalid data in the CPLEX problem object (CPXERR_NO_PROBLEM). Exceeding a user-specified CPLEX limit, or proving the model infeasible or unbounded, are not considered errors. Note that a zero return value does not necessarily mean that a solution exists. Use query routines CPXsolninfo, CPXgetstat, and CPXsolution to obtain further information about the status of the optimization.
CPXmbasewrite

Category          Global Function
Definition File   cplex.h
Include Files     cplex.h

Synopsis          int CPXPUBLIC CPXmbasewrite(CPXENVptr env,
                                CPXCLPptr lp,         
                                const char * filename_str)

Description       The routine CPXmbasewrite is used to write the most current basis associated with a CPLEX problem object to a file. The file is saved in BAS format which corresponds to the industry standard MPS insert format for bases.

When CPXmbasewrite is invoked, the current basis is written to a file. This routine does not remove the basis from the problem object.

Example            status = CPXmbasewrite (env, lp, "myprob.bas");

Parameters         env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
A pointer to the CPLEX problem object as returned by CPXcreateprob.
filename_str
A character string containing the name of the file to which the basis should be written.

Returns            The routine returns zero on success and nonzero if an error occurs.
CPXmipopt

Category   Global Function
Definition File  cplex.h
Synopsis   int CPXPUBLIC CPXmipopt(CPXENVptr env,
CPXLPptr lp)
Description    The routine CPXmipopt may be used, at any time after a mixed integer program has been created by a call to CPXcreateprob, to find a solution to that problem.

CPXmipopt returns zero unless it encounters an error. Nonzero values are error codes indicating which type of failure occurred. Use the routine CPXgetstat to determine the status of the mixed integer optimization. One possible error is the inability to solve a subproblem satisfactorily, as indicated by CPXERR_SUBPROB_SOLVE. The solution status of the subproblem optimization can be obtained with the routine CPXgetsubstat.

An LP solution does not exist at the end of CPXmipopt, so post-solution information is available only through the special mixed integer solution routines. To obtain post-solution information for the LP subproblem associated with the integer solution, use the routine CPXchgprobtype.

Example

status = CPXmipopt (env, lp);

See also the example mipex1.c in the standard distribution.

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

Returns
This routine returns zero unless it encounters an error.
CPXmsg

Category: Global Function
Definition File: cplex.h
Include Files: cplex.h
Synopsis:

```c
int CPXMsg(CPXCHANNELptr channel,
            const char * format,
            ... )
```

Description:

The routine CPXmsg writes a message to a specified channel. Like the C function `printf`, it takes a variable number of arguments comprising the message to be written. The list of variables specified after the format string should be at least as long as the number of format codes in the format. The format string and variables are processed by the C library function `vprintf` or a substitute on systems that do not have the `vprintf` function.

The formatted string is limited to 1024 characters, and is usually output with the C library function `fputs` to each output destination in the output destination list for a channel, except when a function has been specified by the routine CPXaddfuncdest as a destination.

The CPLEX Callable Library uses CPXmsg for all message output. The CPXmsg routine may also be used in applications to send messages to either CPLEX-defined or user-defined channels.

**Note:** CPXmsg is the only non-advanced CPLEX routine not requiring the CPLEX environment parameter.

Example:

```c
CPXmsg (mychannel, "The objective value was %f.
```


Parameters:

- **channel**
  - The pointer to the channel receiving the message.
- **format**
The format string controlling the message output. This string is used in a way identical to the format string in a printf statement.

**Returns**

At completion, CPXmsg returns the number of characters in the formatted result string.
CPXmsgstr

Category Global Function

Definition File cplex.h

Include Files cplex.h

Synopsis

int CPXPUBLIC CPXmsgstr(CPXCHANNELptr channel,
                         const char * msg_str)

Description

The routine CPXmsgstr sends a character string to a CPLEX message channel. It is provided as an alternative to CPXmsg, which due to its variable-length argument list, cannot be used in some environments, such as Visual Basic.

Example

CPXmsgstr (p, q);

Parameters

channel
The pointer to the channel receiving the message.

msg_str
A pointer to a string that should be sent to the message channel.

Returns

The routine returns the number of characters in the string msg.
**CPXmstwrite**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXmstwrite(CPXENVptr env,
                           CPXCLPptr lp,
                           const char * filename_str)
```

**Description**
The routine **CPXmstwrite** is used to write a MIP start to an .mst file.

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **lp**
  A pointer to the CPLEX problem object as returned by CPXcreateprob.

- **filename_str**
  A character string containing the name of the file to which the MIP start information should be written.
CPXnewcols

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXnewcols (CPXCENVptr env,
                          CPXLPptr lp,
                          int ccnt,
                          const double * obj,
                          const double * lb,
                          const double * ub,
                          const char * xctype,
                          char ** colname)
```

Description:
The routine CPXnewcols adds empty columns to a specified CPLEX problem object. This routine may be called any time after a call to CPXcreateprob.

For each column, the user can specify the objective coefficient, the lower and upper bounds, the variable type, and name of the variable. The added columns are indexed to put them at the end of the problem. Thus, if `ccnt` columns are added to a problem object already having `k` columns, the new columns have indices `k, k+1, ..., k+ccnt-1`. The constraint coefficients in the new columns are zero; the constraint coefficients can be changed with calls to CPXchgcoef, CPXchgcoeflist or CPXaddrows.

The routine CPXnewcols is very similar to the routine CPXnewrows. It can be used to add variables to a problem object without specifying the matrix coefficients.

Types of new variables: values of ctype[j]

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_CONTINUOUS</td>
<td>'C'</td>
<td>continuous variable j</td>
</tr>
<tr>
<td>CPX_BINARY</td>
<td>'B'</td>
<td>binary variable j</td>
</tr>
<tr>
<td>CPX_INTEGER</td>
<td>'I'</td>
<td>general integer variable j</td>
</tr>
<tr>
<td>CPX_SEMICONT</td>
<td>'S'</td>
<td>semi-continuous variable j</td>
</tr>
<tr>
<td>CPX_SEMIINT</td>
<td>'N'</td>
<td>semi-integer variable j</td>
</tr>
</tbody>
</table>

Example:

```c
status = CPXnewcols (env, lp, ccnt, obj, lb, ub, NULL, NULL);
```

See also the example lpex8.c in the ILOG CPLEX User's Manual and in the standard distribution.
Parameters

- `env`: A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
- `lp`: A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
- `ccnt`: An integer that indicates the number of new variables being added to the problem object.
- `obj`: An array of length `ccnt` containing the objective function coefficients of the new variables. This array may be NULL, in which case the new objective coefficients are all set to 0.
- `lb`: An array of length `ccnt` containing the lower bound on each of the new variables. Any lower bound that is set to a value less than or equal to that of the constant `CPX_INFBOUND` is treated as -?. `CPX_INFBOUND` is defined in the header file `cplex.h`. This array may be NULL, in which case the new lower bounds are all set to 0 (zero).
- `ub`: An array of length `ccnt` containing the upper bound on each of the new variables. Any upper bound that is set to a value greater than or equal to that of the constant `CPX_INFBOUND` is treated as ?. `CPX_INFBOUND` is defined in the header file `cplex.h`. This array may be NULL, in which case the new upper bounds are all set to `CPX_INFBOUND`.
- `xctype`: An array of length `ccnt` containing the type of each of the new variables. Possible values appear in the table. This array may be NULL, in which case the new variables are created as continuous type.
- `colname`: An array of length `ccnt` containing pointers to character strings that specify the names of the new variables added to the problem object. May be NULL, in which case the new columns are assigned default names if the columns already resident in the problem object have names; otherwise, no names are associated with the variables. If column names are passed to `CPXnewcols` but existing variables have no names assigned, default names are created for the existing variables.

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXnewrows

Category    Global Function
Definition File  cplex.h
Include Files   cplex.h
Synopsis

```c
int CPXPUBLIC CPXnewrows (CPXCENVptr env,
    CPXLPptr lp,
    int rcnt,
    const double * rhs,
    const char * sense,
    const double * rngval,
    char ** rowname)
```

Description

The routine CPXnewrows adds empty constraints to a specified CPLEX problem object. This routine may be called any time after a call to CPXcreateprob.

For each row, the user can specify the sense, right-hand side value, range value and name of the constraint. The added rows are indexed to put them at the end of the problem. Thus, if \( rcnt \) rows are added to a problem object already having \( k \) rows, the new rows have indices \( k, k+1, \ldots, k+rcnt-1 \). The constraint coefficients in the new rows are zero; the constraint coefficients can be changed with calls to CPXchgcoef, CPXchgcoeflist or CPXaddcols.

Example

```c
status = CPXnewrows (env, lp, rcnt, rhs, sense, NULL, newrowname);
```

See also the example lpex1.c in the *ILOG CPLEX User’s Manual* and in the standard distribution.

Parameters

env

A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp

Table 1: Settings for elements of the array sense

<table>
<thead>
<tr>
<th>sense[i]</th>
<th>= 'L'</th>
<th>≤ constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>sense[i]</td>
<td>= 'E'</td>
<td>= constraint</td>
</tr>
<tr>
<td>sense[i]</td>
<td>= 'G'</td>
<td>≥ constraint</td>
</tr>
<tr>
<td>sense[i]</td>
<td>= 'R'</td>
<td>ranged constraint</td>
</tr>
</tbody>
</table>


A pointer to a CPLEX problem object as returned by CPXcreateprob.

rcnt
An integer that indicates the number of new rows to be added to the problem object.

rhs
An array of length rcnt containing the right-hand side term for each constraint to be added to the problem object. May be NULL, in which case the right-hand side terms are set to 0.0 for the new constraints.

sense
An array of length rcnt containing the sense of each constraint to be added to the problem object. This array may be NULL, in which case the sense of each constraint is set to 'E'. The values of the elements of this array appear in Table 1.

rngval
An array of length rcnt containing the range values for the new constraints. If a new constraint has sense[i] = 'R', the value of constraint i can be between rhs[i] and rhs[i] + rngval[i]. May be NULL, in which case the range values are all set to zero.

rowname
An array of length rcnt containing pointers to character strings that represent the names of the new rows, or equivalently, the constraint names. May be NULL, in which case the new rows are assigned default names if the rows already resident in the problem object have names; otherwise, no names are associated with the constraints. If row names are passed to CPXnewrows but existing constraints have no names assigned, default names are created for the existing constraints.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXobjsa

Category                  Global Function
Definition File           cplex.h
Include Files            cplex.h

Synopsis                  int CPXPUBLIC CPXobjsa(CPXENVptr env,
                          CPXCLPptr lp,
                          int begin,
                          int end,
                          double * lower,
                          double * upper)

Description               The routine CPXobjsa is used to access upper and lower sensitivity ranges for objective function coefficients for a specified range of variable indices. The beginning and end of the range of variable indices must be specified.

Note: Information for variable $j$, where $begin \leq j \leq end$, is returned in position $(j-begin)$ of the arrays lower and upper.

Example                   status = CPXobjsa (env, lp, 0, CPXgetnumcols(env,lp)-1,
                          lower, upper);

Parameters                env
                          A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
                          A pointer to a CPLEX problem object as returned by CPXcreateprob.
begin
                          An integer indicating the beginning of the range of ranges to be returned.
end
                          An integer indicating the end of the range of ranges to be returned.
lower
An array where the objective function lower range values are to be returned. This array must be of length at least \((end - begin + 1)\).

\[ \text{lower} \]

An array where the objective function upper range values are to be returned. This array must be of length at least \((end - begin + 1)\).

\[ \text{upper} \]

**Returns**

The routine returns zero on success and nonzero if an error occurs. This routine fails if no optimal basis exists.
CPXopenCPLEX

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis: CPXENVptr CPXPUBLIC CPXopenCPLEX(int * status_p)

Description: The routine CPXopenCPLEX initializes a CPLEX environment when accessing a license for CPLEX and works only if the computer is licensed for Callable Library use. The routine CPXopenCPLEX must be the first CPLEX routine called. The routine returns a pointer to a CPLEX environment. This pointer is used as a parameter to every other non-advanced CPLEX routine (except CPXmsg).

Example:

env = CPXopenCPLEX (&status);


Parameters:

status_p

A pointer to an integer, where an error code is placed by this routine.

Returns:

A pointer to the CPLEX environment. If an error occurs (including licensing problems), the value NULL is returned. The reason for the error is returned in the variable *status_p. If the routine is successful, then *status_p is 0 (zero).
CPXordwrite

Category       Global Function
Definition File cplex.h
Include Files  cplex.h
Synopsis       int CPXPUBLIC CPXordwrite(CPXENVptr env,  
                CPXCLPptr lp,  
                const char * filename_str)
Description    The routine CPXordwrite is used to write a priority order to an ORD file. If a priority order has been associated with the CPLEX problem object, or the parameter CPX_PARAM_MIPORDTYPE is nonzero, or a MIP feasible solution exists, this routine writes the priority order into a file.
Example        status = CPXordwrite (env, lp, "myfile.ord");

See also the example mipex3.c in the standard distribution.

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.
filename_str
A character string containing the name of the file to which the ORD information should be written.

Returns       The routine returns zero on success and nonzero if an error occurs.
CPXpperwrite

Category: Global Function
Definition File: cplex.h
Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXpperwrite(CPXENVptr env,
                          CPXLPptr ip,
                          const char * filename_str,
                          double epsilon)
```

Description:
When solving degenerate linear programs with the primal simplex method, CPLEX may initiate a perturbation of the bounds of the problem in order to improve performance. The routine CPXpperwrite writes a similarly perturbed problem to a binary SAV format file.

Example:

```c
status = CPXpperwrite (env, lp, "myprob.ppe", epsilon);
```

Parameters:
- `env`:
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- `lp`:
A pointer to a CPLEX problem object as returned by CPXcreateprob.
- `filename_str`:
A character string containing the name of the file to which the perturbed problem should be written.
- `epsilon`:
The perturbation constant.

Returns:
The routine returns zero on success and nonzero if an error occurs.
CPXpreslvwrite

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXpreslvwrite (CPXCENVptr env,  
CPXLPptr lp,  
const char * filename_str,  
double * objoff_p)
```

**Description**  
The routine `CPXpreslvwrite` is used to write a presolved version of the problem to a file. The file is saved in binary format, and can be read using the routine `CPXreadcopyprob`.

**Note:** Reductions done by the CPLEX presolve algorithms can cause the objective value to shift. As a result, the optimal objective obtained from solving the presolved problem created using `CPXpreslvwrite` may not be the same as the optimal objective of the original problem. The `objoff_p` parameter can be used to reconcile this difference.

**Example**  
```c
status = CPXpreslvwrite (env, lp, "myfile.pre", &objoff);
```

**Parameters**

- **env**  
A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- **lp**  
A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

- **filename_str**  
A character string containing the name of the file to which the presolved problem should be written.

- **objoff_p**  
A pointer to a double precision variable that is used to hold the objective value difference between the original problem and the presolved problem. That is:  
```c
original objective value = (*objoff_p) + presolved objective value
```
Returns

The routine returns zero on success and nonzero if an error occurs.
CPXprimopt

Category       Global Function
Definition File cplex.h
Include Files  cplex.h
Synopsis       int CPXPUBLIC CPXprimopt (CPXCENVptr env, CPXLPtr lp)
Description    The routine CPXprimopt may be used after a linear program has been created via a call to CPXcreateprob, to find a solution to that problem using the primal simplex method. When this function is called, the CPLEX primal simplex algorithm attempts to optimize the specified problem. The results of the optimization are recorded in the CPLEX problem object.
Example        status = CPXprimopt (env, lp);

Parameters
env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

Returns       The routine returns zero unless an error occurred during the optimization. Examples of errors include exhausting available memory (CPXERR_NO_MEMORY) or encountering invalid data in the CPLEX problem object (CPXERR_NO_PROBLEM). Exceeding a user-specified CPLEX limit or proving the model infeasible or unbounded are not considered errors. Note that a zero return value does not necessarily mean that a solution exists. Use query routines CPXsolninfo, CPXgetstat, and CPXsolution to obtain further information about the status of the optimization.
CPXputenv

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis: int CPXPUBLIC CPXputenv(const char * envsetting_str)

Description: The routine CPXputenv sets an environment variable to be used by CPLEX. Use it instead of the standard C Library putenv function to make sure your application ports properly to Windows. Be sure to allocate the memory dynamically for the string passed to CPXputenv.

As with the C putenv routine, the address of the character string goes directly into the environment. Therefore, the memory identified by the pointer must remain active throughout the remaining parts of the application where CPLEX runs. Since global or static variables are not thread safe, ILOG recommends dynamic memory allocation of the envsetting string.

Example:

```c
char *envstr = NULL;
envstr = (char *) malloc (256);
if ( envstr != NULL ) {
    strcpy (envstr, 
            "ILOG_LICENSE_FILE=c:\myapp\license\access.ilm");
    CPXputenv (envstr);
}
```

Parameters: envsetting_str

A string containing an environment variable assignment. This argument typically sets the ILOG_LICENSE_FILE environment variable that customizes the location of the license key.

Returns: The routine returns 0 (zero) when it executes successfully and -1 when it fails.
CPXqpindefcertificate

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis: int CPXPUBLIC CPXqpindefcertificate(CPXENVptr env, CPXCLPptr lp, double * x)

Description: The routine CPXqpindefcertificate computes a vector $x$ that satisfies the inequality $x^TQx < 0$. Such a vector demonstrates that the $Q$ matrix violates the assumption of positive semi-definiteness, and can be an aid in debugging a user's program if indefiniteness is an unexpected outcome.

Example:

status = CPXqpindefcertificate (env, lp, x);

Parameters:

- env: A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- lp: A pointer to a CPLEX problem object as returned by CPXcreateprob.
- x: An array to receive the values of the vector that is to be returned. The length of this array must be the same as the number of quadratic variables in the problem, which can be obtained by calling CPXgetnumquad for example.

Returns: The routine returns zero on success and nonzero if an error occurs.
CPXqpopt

Category            Global Function
Definition File     cplex.h
Include Files       cplex.h
Synopsis             int CPXPUBLIC CPXqpopt(CPXENVptr env,
                           CPXLPptr lp)
Description          The routine CPXqpopt may be used, at any time after a continuous quadratic program
                     has been created, to find a solution to that problem using one of CPLEX’s quadratic
                     optimizers. The parameter CPX_PARAM_QPMETHOD controls the choice of optimizer
                     (Dual Simplex, Primal Simplex, or Barrier). With the default setting of this parameter
                     (that is, Automatic) CPLEX invokes the barrier method because it is fastest on a
                     wide range of problems.
Example               status = CPXqpopt (env, lp);
See Also              CPXgetmethod
Parameters            env
                     A pointer to the CPLEX environment as returned by CPXopenCPLEX.
                     lp
                     A pointer to the CPLEX problem object as returned by CPXcreateprob.
Returns               The routine returns zero unless an error occurred during the optimization. Examples of
                     errors include exhausting available memory (CPXERR_NO_MEMORY) or
                     encountering invalid data in the CPLEX problem object (CPXERR_NO_PROBLEM).
                     Exceeding a user-specified CPLEX limit, or proving the model infeasible or unbounded,
                     are not considered errors. Note that a zero return value does not necessarily mean that a
                     solution exists. Use query routines CPXsolninfo, CPXgetstat, and CPXsolution to obtain
                     further information about the status of the optimization.
CPXqpwrite

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:
```
int CPXPUBLIC CPXqpwrite(CPXENVptr env,
    CPXCLPptr lp,
    const char * filename_str)
```

Description:
The routine CPXqpwrite is used to write a QP format file corresponding to the quadratic coefficient matrix of the problem object. This format is described in detail in the reference manual ILOG CPLEX File Formats.

Example:
```
status = CPXqpwrite (env, lp, "myprob.qp");
```

Parameters:
- **env**: A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **lp**: A pointer to a CPLEX problem object as returned by CPXcreateprob.
- **filename_str**: A character string containing the name of the file to which the QP information should be written.

Returns:
The routine returns zero on success and nonzero if an error occurs.
**CPXreadcopybase**

**Category**  Global Function

**Definition File**  cplex.h

**Include Files**  cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXreadcopybase(CPXENVptr env,
                               CPXLPptr lp,
                               const char * filename_str)
```

**Description**

The routine **CPXreadcopybase** reads a basis from a BAS file, and copies that basis into a CPLEX problem object. The parameter **CPX_PARAM_ADVIND** must be set to 1 (one), its default value, in order for the basis to be used for starting a subsequent optimization.

**Example**

```c
status = CPXreadcopybase (env, lp, "myprob.bas");
```

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by **CPXopenCPLEX**.

- **lp**
  A pointer to a CPLEX problem object as returned by **CPXcreateprob**.

- **filename_str**
  The name of the file from which the basis should be read.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
**CPXreadcopymipstart**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXreadcopymipstart(CPXENVptr env, 
    CPXLPptr lp, 
    const char * filename_str)
```

**Description**  
The routine `CPXreadcopymipstart` reads a MST file and copies the MIP start information into a CPLEX problem object. The parameter `CPX_PARAM_MIPSTART` must be set to `CPX_ON`, in order for the MIP start information to be used for starting a subsequent optimization. The default value for the parameter is `CPX_OFF`.

**Example**
```c
status = CPXreadcopystart(env, lp, "myprob.mst");
```

**Parameters**
- **env**  
A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
- **lp**  
A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
- **filename_str**  
A string containing the name of the MST file.

**Returns**  
The routine returns zero on success and nonzero if an error occurs.
**CPXreadcopyorder**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXreadcopyorder(CPXENVptr env,  
CPXLPptr lp,  
const char * filename_str)
```

**Description**  
The routine `CPXreadcopyorder` reads an ORD file and copies the priority order information into a CPLEX problem object. The parameter `CPX_PARAM_MIPORDIND` must be set to `CPX_ON` (its default value), in order for the priority order to be used for starting a subsequent optimization.

**Example**
```
status = CPXreadcopyorder (env, lp, "myprob.ord");
```

**Parameters**

- `env`  
A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- `lp`  
A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

- `filename_str`  
The name of the file from which the priority order and start values should be read.

**Returns**  
The routine returns zero on success and nonzero if an error occurs.
CPXreadcopyparam

Category         Global Function
Definition File  cplex.h
Include Files    cplex.h
Synopsis         int CPXPUBLIC CPXreadcopyparam(CPXENVptr env,
                           const char * filename_str)

Description      The routine CPXreadcopyparam reads parameter names and settings from the file indicated by filename_str and copies them into CPLEX.

This routine reads and copies files in the format created by CPXwriteparam. The file extension is .prm. The PRM format is documented in the reference manual ILOG CPLEX File Formats.

Parameters

env

A pointer to the CPLEX environment as returned by CPXopenCPLEX.

filename_str

Pointer to the file to read and copy into CPLEX.
CPXreadcopyprob

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXreadcopyprob(CPXENVptr env,
                              CPXLPptr lp,
                              const char * filename_str,
                              const char * filetype_str)
```

Description:

The routine `CPXreadcopyprob` reads an MPS, LP or SAV file into an existing CPLEX problem object. Any existing data associated with the problem object is destroyed. The problem can then be optimized via any one of the optimization routines. To determine the contents of the data, CPLEX query routines can be used.

The type of the file may be specified with the `filetype` argument. When the `filetype` argument is `NULL`, the end of the file name is checked for one of the strings `.lp`, `.lp.gz`, `.lp.z`, `.mps`, `.mps.gz`, `.mps.z`, `.sav`, `.sav.gz`, or `.sav.z`.

If one of these strings is present, `filetype` is set accordingly. If `filetype` is `NULL` and none of these strings is at the end of the file name, the routine automatically detects the type of the file by examining the first few bytes of the file.

If the file name ends in `.gz` or `.z` the file is read as a compressed file. Thus, a file name ending in `.sav` is read as a SAV format file, while a file name ending in `.sav.gz` is read as a compressed SAV format file.

If the file name does not end with a recognized type, CPLEX attempts to auto-detect the file type.

### Values of filetype_str

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;SAV&quot;</td>
<td>Use SAV format</td>
</tr>
<tr>
<td>&quot;MPS&quot;</td>
<td>Use MPS format</td>
</tr>
<tr>
<td>&quot;LP&quot;</td>
<td>Use LP format</td>
</tr>
</tbody>
</table>

Example:

```c
status = CPXreadcopyprob (env, lp, "myprob.mps", NULL);
```

See also the example `lpex2.c` in the *ILOG CPLEX User's Manual* and in the standard distribution.
Parameters

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **lp**
  A pointer to a CPLEX problem object as returned by CPXcreateprob.
- **filename_str**
  The name of the file from which the problem should be read.
- **filetype_str**
  A case insensitive string containing the type of the file, one of the strings in the table. May be NULL, in which case the file type is inferred from the last characters of the file name.

Returns

The routine returns zero on success and nonzero if an error occurs.
**CPXreadcopyqp**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXreadcopyqp(CPXENVptr env,
  CPXLPptr lp,
  const char * filename_str)
```

**Description**  
The routine CPXreadcopyqp reads in the QP file specified by the argument `filename` and copies the quadratic matrix data into the CPLEX problem object. This routine changes the problem type to QP if it is not already a QP. It may not be used on a problem whose type is MIP.

**Example**  
```c
status = CPXreadcopyqp (env, lp, "myprob.qp");
```

**Parameters**

- `env`  
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- `lp`  
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

- `filename_str`  
The name of the file from which the QP information should be read.

**Returns**  
The routine returns zero on success and nonzero if an error occurs.
**CPXreadcopy sos**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXreadcopy sos(CPXENVptr env,
    CPXLPptr lp,
    const char * filename_str)
```

**Description**  
The routine `CPXreadcopy sos` reads in SOS data from an SOS file, and copies the data into a MIP problem object.

This routine changes the problem type to MIP, if it is not already a MIP.

**Example**  
```c
status = CPXreadcopy sos (env, lp, "myprob.sos");
```

**Parameters**  
- `env`  
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- `lp`  
  A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

- `filename_str`  
  The name of the file from which the SOS information should be read.

**Returns**  
The routine returns zero on success and nonzero if an error occurs.
CPXreadcopytree

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis: int CPXPUBLIC CPXreadcopytree(CPXENVptr env, CPXLPptr lp, const char * filename_str)

Description: The routine CPXreadcopytree is used to read branch & cut progress information from a prior run, contained in a TRE format file, into a CPLEX problem object. The parameter CPX_PARAM_ADVIND must be set to CPX_ON (its default value), in order for the tree to be used for starting a subsequent optimization.

Example:

status = CPXreadcopytree (env, lp, "myprob.tre");

Parameters:

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

filename_str
The name of the file from which the tree information should be read.

Returns: The routine returns zero on success and nonzero if an error occurs.
CPXreadcopyvec

Category               Global Function
Definition File        cplex.h
Include Files          cplex.h
Synopsis               int CPXPUBLIC CPXreadcopyvec(CPXENVptr env,
                            CPXLPptr lp,
                            const char * filename_str)
Description             The routine CPXreadcopyvec reads Barrier solution information from a VEC format file and copies this information into a CPLEX problem object. This routine is typically used to initiate a crossover from the Barrier solution. The parameter CPX_PARAM_ADVIND must be set to 1 (one), its default value, in order for the vector file to take effect for starting a crossover.
Example                 status = CPXreadcopyvec (env, lp, "myprob.vec");
Parameters              env
                            A pointer to the CPLEX environment as returned by CPXopenCPLEX.
                             lp
                            A pointer to a CPLEX problem object as returned by CPXcreateprob.
                             filename_str
                            The name of the VEC file to be read.
Returns                 The routine returns zero on success and nonzero if an error occurs.
CPXrhssa

Category       Global Function
Definition File cplex.h
Include Files  cplex.h

Synopsis       int CPXPUBLIC CPXrhssa(CPXENVptr env,
                      CPXCLPptr lp,
                      int begin,
                      int end,
                      double * lower,
                      double * upper)

Description    The routine CPXrhssa is used to access a range of upper and lower ranges for right-hand side values. The beginning and end of the range must be specified.

Note: Information for constraint \( j \), where \( \text{begin} \leq j \leq \text{end} \), is returned in position \( (j-\text{begin}) \) of the arrays lower and upper.

Example

status = CPXrhssa (env, lp, 0, CPXgetnumrows(env,lp)-1,
                    lower, upper);

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

begin
An integer indicating the beginning of the range of ranges to be returned.

end
An integer indicating the end of the range of ranges to be returned.

lower
An array where the right-hand side lower range values are to be returned. This array must be of length at least \( (\text{end} - \text{begin} + 1) \).
An array where the right-hand side upper range values are to be returned. This array must be of length at least \((\text{end} - \text{begin} + 1)\).

**Returns**

The routine returns zero on success and nonzero if an error occurs. This routine fails if no optimal basis exists.
**CPXsetbranchcallbackfunc**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXsetbranchcallbackfunc(CPXENVptr env,
                int (CPXPUBLIC *branchcallback)(CALLBACK_BRANCH_ARGS) ,
                void * cbhandle)
```

**Description**

Note: This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXsetbranchcallbackfunc sets and modifies the user-written callback routine to be called after a branch has been selected but before the branch is carried out during MIP optimization. In the callback routine, the CPLEX-selected branch can be changed to a user-selected branch.

**Example**

```c
status = CPXsetbranchcallbackfunc (env, mybranchfunc, mydata);
```

See also the example admipex1.c in the standard distribution.

**Parameters**

- **env**
  A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

- **branchcallback**
  A pointer to a user-written branch callback. If the callback is set to NULL, no callback can be called during optimization.

- **cbhandle**
  A pointer to user private data. This pointer is passed to the callback.
Callback description

```c
int callback (CPXCENVptr env,
        void       *cbdata,
        int        wherefrom,
        void       *cbhandle,
        int        type,
        int        sos,
        int        nodecnt,
        int        bdcnt,
        double     *nodeest,
        int        *nodebeg,
        int        *indices,
        char       *lu,
        int        *bd,
        int        *useraction_p);
```

The call to the branch callback occurs after a branch has been selected but before the branch is carried out. This function is written by the user. On entry to the callback, the ILOG CPLEX-selected branch is defined in the arguments. The arguments to the callback specify a list of changes to make to the bounds of variables when child nodes are created. One, two, or zero child nodes can be created, so one, two, or zero lists of changes are specified in the arguments. The first branch specified is considered first. The callback is called with zero lists of bound changes when the solution at the node is integer feasible. ILOG CPLEX occasionally elects to branch by changing a number of variables bounds or by adding constraints to the node subproblem; the branch type is then CPX_TYPE_ANY. The details of the constraints added for a CPX_TYPE_ANY branch are not available to the user.

Custom branching strategies can be implemented by calling the CPLEX routine CPXbranchcallbackbranchbds, CPXbranchcallbackbranchconstraints, or CPXbranchcallbackbranchgeneral and setting the useraction variable to CPX_CALLBACK_SET. Then CPLEX will carry out these branches instead of the CPLEX-selected branches.

Branch variables are in terms of the original problem if the parameter CPX_PARAM_MIPCBREDLP is set to CPX_OFF before the call to CPXmipopt that calls the callback. Otherwise, branch variables are in terms of the presolved problem.

Callback return value

The callback returns zero on success and nonzero if an error occurs.

Callback arguments

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbdata
A pointer passed from the optimization routine to the user-written callback that identifies the problem being optimized. The only purpose of this pointer is to pass it to the callback information routines.

wherefrom
An integer value indicating where in the optimization this function was called. It will have the value CPX_CALLBACK_MIP_BRANCH.

cbhandle
A pointer to user-private data.

int type
An integer that indicates the type of branch. This table summarizes possible values.

<table>
<thead>
<tr>
<th>Symbolic Constant</th>
<th>Value</th>
<th>Branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_TYPE_VAR</td>
<td>'0'</td>
<td>variable branch</td>
</tr>
<tr>
<td>CPX_TYPE_SOS1</td>
<td>'1'</td>
<td>SOS1 branch</td>
</tr>
<tr>
<td>CPX_TYPE_SOS2</td>
<td>'2'</td>
<td>SOS2 branch</td>
</tr>
<tr>
<td>CPX_TYPE_ANY</td>
<td>'A'</td>
<td>multiple bound changes and/or constraints will be used for branching</td>
</tr>
</tbody>
</table>

sos
An integer that indicates the special ordered set (SOS) used for this branch. A value of -1 indicates that this branch is not an SOS-type branch.

nodecnt
An integer that indicates the number of nodes CPLEX will create from this branch. Possible values are:
◆ 0 (zero), or
◆ 1, or
◆ 2.

If the argument is 0, the node will be fathomed unless user-specified branches are made; that is, no child nodes are created and the node itself is discarded.

bdcnt
An integer that indicates the number of bound changes defined in the arrays indices, lu, and bd that define the CPLEX-selected branch.

nodeest
An array with nodecnt entries that contains estimates of the integer objective-function value that will be attained from the created node.

cnodebeg
An array with nodecnt entries. The i-th entry is the index into the arrays indices, lu, and bd of the first bound changed for the i-th node.

indices
Together with lu and bd, this array defines the bound changes for each of the created nodes. The entry indices[i] is the index for the variable.

lu
Together with indices and bd, this array defines the bound changes for each of the created nodes. The entry lu[i] is one of the three possible values indicating which bound to change:

◆ 'L' for lower bound, or
◆ 'U' for upper bound, or
◆ 'B' for both bounds.

bd
Together with indices and lu, this array defines the bound changes for each of the created nodes. The entry bd[i] indicates the new value of the bound.

useraction_p
A pointer to an integer indicating the action for ILOG CPLEX to take at the completion of the user callback. The table summarizes the possible actions.

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CPX_CALLBACK_DEFAULT</td>
<td>Use CPLEX-selected branch</td>
</tr>
<tr>
<td>1</td>
<td>CPX_CALLBACK_FAIL</td>
<td>Exit optimization</td>
</tr>
<tr>
<td>2</td>
<td>CPX_CALLBACK_SET</td>
<td>Use user-selected branch, as defined by calls to CPXbranchcallbackbranchbds</td>
</tr>
</tbody>
</table>

Returns
The routine returns zero on success and nonzero if an error occurs.
**CPXsetbranchnosolncallbackfunc**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXsetbranchnosolncallbackfunc(CPXENVptr env,  
   int (*branchnosolncallback)(CALLBACK_BRANCH_ARGS) ,  
   void * cbhandle)
```

**Description**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXsetbranchnosolncallbackfunc` sets the callback function that will be called instead of the branch callback when there is a failure due to such situations as an iteration limit being reached, unboundedness being detected, numeric difficulties being encountered, while the node LP is being solved. In consequence of the failure, whether the node is feasible or infeasible cannot be known and thus CPLEX routines such as `CPXsolution` may fail. In this situation, CPLEX will attempt to fix some variables and continue.

These conditions are rare (except when the user has set a very low iteration limit), so it is acceptable to let CPLEX follow its default action in these cases.
CPXsetcutcallbackfunc

Category Global Function

Definition File cplex.h

Include Files cplex.h

Synopsis

```c
int CPXPUBLIC CPXsetcutcallbackfunc(CPXENVptr env,
    int(CPXPUBLIC *cutcallback)(CALLBACK_CUT_ARGS) ,
    void * cbhandle)
```

Description

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXsetcutcallbackfunc sets and modifies the user-written callback for adding cuts. The user-written callback is called by ILOG CPLEX during MIP branch & cut for every node that has an LP optimal solution with objective value below the cutoff and is integer infeasible. The callback routine adds globally valid cuts to the LP subproblem. The cut may be for the original problem if the parameter CPX_PARAM_MIPCBREDLP was set to CPX_OFF before the call to CPXmipopt that calls the callback. Otherwise, the cut is for the presolved problem.

Within the user-written cut callback, the routine CPXgetcallbacknodelp and other query routines from the Callable Library access information about the subproblem. The routines CPXgetcallbacknodeintfeas and CPXgetcallbacksosinfo examines the status of integer entities.

The routine CPXcutcallbackadd adds cuts to the problem. Cuts added to the problem are first put into a cut pool, so they are not present in the subproblem LP until after the user-written cut callback is finished.

Any cuts that are duplicates of cuts already in the subproblem are not added to the subproblem. Cuts that are added remain part of all subsequent subproblems; there is no cut deletion.

If cuts have been added, the subproblem is re-solved and evaluated, and if the LP solution is still integer infeasible and not cut off, the cut callback is called again.
If the problem has names, user-added cuts have names of the form Xnumber where number is a sequence number among all cuts generated.

The parameter CPX_PARAM_REDUCE must be set to CPX_PREREDUCE_PRIMALONLY (1) or CPX_PREREDUCE_NOPRIMALORDUAL (0) if the constraints to be added in the callback are lazy constraints, that is, not implied by the constraints in the constraint matrix. The parameter CPX_PARAM_PRELINEAR must be set to 0 if the constraints to be added are in terms of the original problem and the constraints are valid cutting planes.

Example

    status = CPXsetcutcallbackfunc(env, mycutfunc, mydata);

See also the example admipex5.c in the standard distribution.

Parameters

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cutcallback
The pointer to the current user-written cut callback. If no callback has been set, the pointer evaluates to NULL.

cbhandle
A pointer to user private data. This pointer is passed to the user-written cut callback.

Callback description

    int callback (CPXENVptr env,
                   void      *cbdata,
                   int       wherefrom,
                   void      *cbhandle,
                   int       *useraction_p);

ILOG CPLEX calls the cut callback when the LP subproblem for a node has an optimal solution with objective value below the cutoff and is integer infeasible.

Callback return value
The callback returns zero on success and nonzero if an error occurs.

Callback arguments

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbdata
A pointer passed from the optimization routine to the user-written callback that identifies the problem being optimized. The only purpose of this pointer is to pass it to the callback information routines.

wherefrom
An integer value indicating where in the optimization this function was called. It has the value CPX_CALLBACK_MIP_CUT.

cbhandle
A pointer to user private data.

useraction_p
A pointer to an integer indicating the action for ILOG CPLEX to take at the completion of the user callback. The table summarizes possible actions.

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CPX_CALLBACK_DEFAULT</td>
<td>Use cuts as added</td>
</tr>
<tr>
<td>1</td>
<td>CPX_CALLBACK_FAIL</td>
<td>Exit optimization</td>
</tr>
<tr>
<td>2</td>
<td>CPX_CALLBACK_SET</td>
<td>Use cuts as added</td>
</tr>
</tbody>
</table>

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXsetdblparam

Category       Global Function
Definition File cplex.h
Include Files  cplex.h
Synopsis

int CPXPUBLIC CPXsetdblparam(CPXENVptr env,
                           int whichparam,
                           double newvalue)

Description    The routine CPXsetdblparam sets the value of a CPLEX parameter of type double.
                The reference manual ILOG CPLEX Parameters provides a list of parameters with their types, options, and default values.
Example

    status = CPXsetdblparam (env, CPX_PARAM_TILIM, 1000.0);

Parameters

    env
    A pointer to the CPLEX environment as returned by CPXopenCPLEX.
whichparam
    The symbolic constant (or reference number) of the parameter to change.
newvalue
    The new value of the parameter.

Returns        The routine returns zero on success and nonzero if an error occurs.
CPXsetdefaults

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis: int CPXPUBLIC CPXsetdefaults(CPXENVptr env)

Description: The routine CPXsetdefaults resets all CPLEX parameters and settings to default values (with the exception of the log file).

Note: This routine also resets the CPLEX callback functions to NULL.

The reference manual ILOG CPLEX Parameters provides a list of parameters with their types, options, and default values.

Example:

status = CPXsetdefaults(env);

Parameters:

env

A pointer to the CPLEX environment as returned by CPXopenCPLEX.

Returns: The routine returns zero on success and nonzero if an error occurs.
CPXsetdeletenodecallbackfunc

**Category**  
Global Function

**Definition File**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXsetdeletenodecallbackfunc(CPXENVptr env,  
    void(CPXPUBLIC *deletecallback)(CALLBACK_DELETENODE_ARGS) ,  
    void * cbhandle)
```

**Description**

The routine `CPXsetdeletenodecallbackfunc` sets and modifies the user-written callback to be called during MIP optimization when a node is to be deleted. Nodes are deleted in these circumstances:

- when a branch is carried out from that node, or
- when the node relaxation is infeasible, or
- when the node relaxation objective value is worse than the cutoff.

**Example**

```c
status = CPXsetdeletenodecallbackfunc (env,  
    mybranchfunc,  
    mydata);
```

See also the example `admipex1.c` in the standard distribution.

**Parameters**

env
A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.

deletecallback
A pointer to a user-written branch callback. If the callback is set to NULL, no callback is called during optimization.

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.
cbhandle
A pointer to user private data. This pointer is passed to the callback.

Callback description

```c
int callback (CPXENVptr env,
    void  *cbdata,
    int    wherefrom,
    void  *cbhandle,
    int    seqnum,
    void  *handle);
```

The call to the delete node callback routine occurs during MIP optimization when a node is to be deleted.

The main purpose of the callback is to provide an opportunity to free any user data associated with the node, thus preventing memory leaks.

Callback return value
The callback returns zero on success and nonzero if an error occurs.

Callback arguments

env
A pointer to the CPLEX environment, as returned by one of the CPXopenCPLEX routines.

cbdata
A pointer passed from the optimization routine to the user-written callback that identifies the problem being optimized. The only purpose of this pointer is to pass it to the callback information routines.

wherefrom
An integer value indicating where in the optimization this function was called. It will have the value CPX_CALLBACK_MIP_DELETENODE.

cbhandle
A pointer to user private data.

seqnum
The sequence number of the node that is being deleted.

handle
A pointer to the user private data that was assigned to the node when it was created with one of the callback branching routines:

- CPXbranchcallbackbranchbds, or
- CPXbranchcallbackbranchconstraints, or
- CPXbranchcallbackbranchgeneral.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXsetheuristiccallbackfunc

Category                      Global Function
Definition File               cplex.h
Include Files                 cplex.h

Synopsis

```c
int CPXPUBLIC CPXsetheuristiccallbackfunc(CPXENVptr env,
                                          int (CPXPUBLIC *heuristiccallback)(CALLBACK_HEURISTIC_ARGS),
                                          void * cbhandle)
```

Description

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXsetheuristiccallbackfunc` sets or modifies the user-written callback to be called by ILOG CPLEX during MIP optimization after the subproblem has been solved to optimality. That callback is not called when the subproblem is infeasible or cut off. The callback supplies ILOG CPLEX with heuristically-derived integer solutions.

If a linear program must be solved as part of a heuristic callback, make a copy of the node LP and solve the copy, not the CPLEX node LP.

**Example**

```c
status = CPXsetheuristiccallbackfunc(env, myheuristicfunc, mydata);
```

See also the example `admipex2.c` in the standard distribution.

**Parameters**

**env**

A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.

**heuristiccallback**

A pointer to a user-written heuristic callback. If this callback is set to NULL, no callback is called during optimization.

**cbhandle**
A pointer to the user's private data. This pointer is passed to the callback.

**Callback description**

```c
int callback (CPXENVptr env,
              void      *cbdata,
              int       wherefrom,
              void      *cbhandle,
              double    *objval_p,
              double    *x,
              int       *checkfeas_p,
              int       *useraction_p);
```

The call to the heuristic callback occurs after an optimal solution to the subproblem has been obtained. The user can provide that solution to start a heuristic for finding an integer solution. The integer solution provided to ILOG CPLEX replaces the incumbent if it has a better objective value. The basis that is saved as part of the incumbent is the optimal basis from the subproblem; it may not be a good basis for starting optimization of the fixed problem.

The integer solution returned to CPLEX is for the original problem if the parameter CPX_PARAM_MIPCBREDLP was set to CPX_OFF before the call to CPXmipopt that calls the callback. Otherwise, it is for the presolved problem.

**Callback return value**

The callback returns zero on success and nonzero if an error occurs.

**Callback arguments**

* env
  A pointer to the CPLEX environment, as returned by CPXopenCPLEX.
* cbdata
  A pointer passed from the optimization routine to the user-written callback to identify the problem being optimized. The only purpose of the cbdata pointer is to pass it to the callback information routines.
* wherefrom
  An integer value indicating at which point in the optimization this function was called. It has the value CPX_CALLBACK_MIP_HEURISTIC for the heuristic callback.
* cbhandle
  A pointer to user private data.
* objval_p
A pointer to a variable that on entry contains the optimal objective value of the subproblem and on return contains the objective value of the integer solution found, if any.

\texttt{x}

An array that on entry contains primal solution values for the subproblem and on return contains solution values for the integer solution found, if any.

\texttt{checkfeas\_p}

A pointer to an integer that indicates whether or not ILOG CPLEX should check the returned integer solution for integer feasibility. The solution is checked if \texttt{checkfeas\_p} is nonzero. When the solution is checked and found to be integer infeasible, it is discarded, and optimization continues.

\texttt{useraction\_p}

A pointer to an integer to contain the indicator for the action to be taken on completion of the user callback. The table summarizes the possible values.

### Actions to be Taken after a User-Written Heuristic Callback

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CPX_CALLBACK_DEFAULT</td>
<td>No solution found</td>
</tr>
<tr>
<td>1</td>
<td>CPX_CALLBACK_FAIL</td>
<td>Exit optimization</td>
</tr>
<tr>
<td>2</td>
<td>CPX_CALLBACK_SET</td>
<td>Use user solution as indicated in return values</td>
</tr>
</tbody>
</table>

### Returns

The routine returns zero on success and nonzero if an error occurs.
CPXsetincumbentcallbackfunc

Category Global Function
Definition File cplex.h
Include Files cplex.h
Synopsis

```c
int CPXPUBLIC CPXsetincumbentcallbackfunc(CPXENVptr env,
   int(CPXPUBLIC *incumbentallback)(CALLBACK_INCUMBENT_ARGS) ,
   void * cbhandle)
```

Description

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXsetincumbentcallbackfunc sets and modifies the user-written callback routine to be called when an integer solution has been found but before this solution replaces the incumbent. This callback can be used to discard solutions that do not meet criteria beyond that of the mixed integer programming formulation.

Variables are in terms of the original problem if the parameter CPX_PARAM_MIPCBREDLP is set to CPX_OFF before the call to CPXmipopt that calls the callback. Otherwise, variables are in terms of the presolved problem.

**Example**

```c
status = CPXsetincumbentcallbackfunc(env, myincumbentcheck, mydata);
```

See also *Advanced MIP Control Interface* in the *ILOG CPLEX User’s Manual.*

**Parameters**

- env
  A pointer to the CPLEX environment, as returned by CPXopenCPLEX.
- incumbentallback
  A pointer to a user-written incumbent callback. If the callback is set to NULL, no callback can be called during optimization.
cbhandle
A pointer to user private data. This pointer is passed to the callback.

Callback description

int callback (CPXENVptr env,
    void      *cbdata,
    int       wherefrom,
    void      *cbhandle,
    double    objval,
    double    *x,
    int       *isfeas_p,
    int       *useraction_p);

The incumbent callback is called when CPLEX has found an integer solution, but before this solution replaces the incumbent integer solution.

Variables are in terms of the original problem if the parameter CPX_PARAM_MIPCBREDLP is set to CPX_OFF before the call to CPXmipopt that calls the callback. Otherwise, variables are in terms of the presolved problem.

Callback return value
The callback returns zero on success and nonzero if an error occurs.

Callback arguments
env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbdata
A pointer passed from the optimization routine to the user-written callback that identifies the problem being optimized. The only purpose of this pointer is to pass it to the callback information routines.

wherefrom
An integer value indicating where in the optimization this function was called. It will have the value CPX_CALLBACK_MIP_BRANCH.

cbhandle
A pointer to user private data.

objval
A variable that contains the objective value of the integer solution.

x
An array that contains primal solution values for the integer solution.

`isfeas_p`

A pointer to an integer variable that indicates whether or not CPLEX should use the integer solution specified in `x` to replace the current incumbent. A nonzero value indicates that the incumbent should be replaced by `x`; a zero value indicates that it should not.

`useraction_p`

A pointer to an integer to contain the indicator for the action to be taken on completion of the user callback. The table summarizes the possible values.

### Actions to be Taken after a User-Written Incumbent Callback

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CPX_CALLBACK_DEFAULT</td>
<td>Proceed with optimization</td>
</tr>
<tr>
<td>1</td>
<td>CPX_CALLBACK_FAIL</td>
<td>Exit optimization</td>
</tr>
<tr>
<td>2</td>
<td>CPX_CALLBACK_SET</td>
<td>Proceed with optimization</td>
</tr>
</tbody>
</table>

See Also: CPXgetincumbentcallbackfunc

Returns: The routine returns zero on success and nonzero if an error occurs.
CPXsetintparam

Category       Global Function
Definition File cplex.h
Include Files  cplex.h
Synopsis
   int CPXPUBLIC CPXsetintparam(CPXENVptr env,
                                 int whichparam,
                                 int newvalue)
Description     The routine CPXsetintparam sets the value of a CPLEX parameter of type int.
The reference manual ILOG CPLEX Parameters provides a list of parameters with their
types, options, and default values.
Example
   status = CPXsetintparam (env, CPX_PARAM_SCRIND, CPX_ON);

See also lpex1.c in the CPLEX User's Manual.

Parameters
   env
   A pointer to the CPLEX environment as returned by CPXopenCPLEX.
   whichparam
   The symbolic constant (or reference number) of the parameter to change.
   newvalue
   The new value of the parameter.

Returns        The routine returns zero on success and nonzero if an error occurs.
CPXsetlogfile

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis: int CPXPUBLIC CPXsetlogfile(CPXENVptr env, CPXFILEptr lfile)

Description: The routine CPXsetlogfile modifies the log file to which messages from all four CPLEX-defined channels are written.

Note: A call to CPXset logfile is equivalent to directing output from the cpxresults, cpxwarning, cpxerror and cpxlog message channels to a single file.

Example:

status = CPXsetlogfile (env, logfile);

Parameters:

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lfile
A CPXFILEptr to the log file. This routine sets lfile to be the file pointer for the current log file. A NULL pointer may be passed if no log file is desired. NULL is the default value. Before calling this routine, obtain this pointer with a call to CPXfopen.

Returns: The routine returns zero on success and nonzero if an error occurs.
CPXsetlpcallbackfunc

Category   Global Function

Definition File  cplex.h

Synopsis   int CPXPUBLIC CPXsetlpcallbackfunc(CPXENVptr env,
                     int(CPXPUBLIC *callback)(CPXENVptr, void *, int, void *),
                     void * cbhandle)

Description   The routine CPXsetlpcallbackfunc is used to modify the user-written callback routine to be called after each iteration during the optimization of a linear program, and also periodically during the CPLEX presolve algorithm.

Callback description

int callback(CPXENVptr env,
             void *cbdata,
             int wherefrom,
             void *cbhandle);

This is the user-written callback routine.

Callback return value

A nonzero terminates the optimization.

Callback arguments

env

A pointer to the CPLEX environment that was passed into the associated optimization routine.

cbdata

A pointer passed from the optimization routine to the user-written callback function that identifies the problem being optimized. The only purpose for the cbdata pointer is to pass it to the routine CPXgetcallbackinfo.

wherefrom

An integer value indicating from which optimization algorithm the user-written callback function was called. Possible values and their meaning appear in the table below.
**Parameters**

**cbhandle**

Pointer to user private data, as passed to `CPXsetlpcallbackfunc`.

**env**

A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

**myfunc**

A pointer to a user-written callback function. Setting `callback` to NULL prevents any callback function from being called during optimization. The call to `callback` occurs after every iteration during optimization and periodically during the CPLEX presolve algorithms. This function is written by the user, and is prototyped as documented here.

**cbhandle**

A pointer to user private data. This pointer is passed to the callback function.

**Example**

```c
status = CPXsetlpcallbackfunc (env, myfunc, NULL);
```

**See Also**

`CPXgetcallbackinfo`, `CPXsetmipcallbackfunc`, `CPXsetnetcallbackfunc`

**Returns**

The routine returns zero on success and nonzero if an error occurs.
**CPXsetmipcallbackfunc**

**Category** Global Function

**Definition File** cplex.h

**Include Files** cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXsetmipcallbackfunc(CPXENVptr env,
      int(CPXPUBLIC *callback)(CPXENVptr, void *, int, void *) ,
      void * cbhandle)
```

**Description**

The routine `CPXsetmipcallbackfunc` is used to set the user-written callback routine to be called prior to solving each subproblem in the branch & cut tree, including the root node, during the optimization of a mixed integer program and during some cut generation routines. This routine works in the same way as the routine `CPXsetlpcallbackfunc`. It enables the user to create a separate callback function to be called during the solution of mixed integer programming problems. The prototype for the callback function is identical to that of `CPXsetlpcallbackfunc`.

**Example**

```c
status = CPXsetmipcallbackfunc (env, mycallback, NULL);
```

**Parameters**

- **env**
  A pointer to the CPLEX environment, as returned by one of the `CPXopenCPLEX` routines.

- **callback**
  A pointer to a user-written callback function. Setting `callback` to `NULL` will prevent any callback function from being called during optimization. The call to `callback` will occur after every node during optimization and periodically during the presolve algorithms. This function must be written by the user. Its prototype is explained in the Callback description.

- **cbhandle**
  A pointer to user private data. This pointer will be passed to the callback function.

**Callback description**

```c
int callback (CPXENVptr env,
      void *cbdata,
      int wherefrom,
```
This is the user-written callback routine.

**Callback return value**

A nonzero terminates the optimization.

**Callback arguments**

- `env`  
  A pointer to the CPLEX environment that was passed into the associated optimization routine.

- `cbdata`  
  A pointer passed from the optimization routine to the user-written callback function that identifies the problem being optimized. The only purpose for the `cbdata` pointer is to pass it to the routine `CPXgetcallbackinfo`.

- `wherefrom`  
  An integer value indicating from which optimization algorithm the user-written callback function was called. Possible values and their meaning appear in the table.

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>CPX_CALLBACK_MIP</td>
<td>From mipopt</td>
</tr>
<tr>
<td>107</td>
<td>CPX_CALLBACK_MIP_PROBE</td>
<td>From probing or clique merging</td>
</tr>
<tr>
<td>108</td>
<td>CPX_CALLBACK_MIP_FRACCU</td>
<td>From Gomory fractional cuts</td>
</tr>
<tr>
<td>109</td>
<td>CPX_CALLBACK_MIP_DISJCU</td>
<td>From disjunctive cuts</td>
</tr>
<tr>
<td>110</td>
<td>CPX_CALLBACK_MIP_FLOWMI</td>
<td>From Mixed Integer Rounding cuts</td>
</tr>
</tbody>
</table>

- `cbhandle`  
  A pointer to user private data as passed to `CPXsetmipcallbackfunc`.

**See Also**

- `CPXgetcallbackinfo`  
- `CPXsetlpcallbackfunc`
- `CPXsetnetcallbackfunc`

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXsetnetcallbackfunc

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:
int CPXPUBLIC CPXsetnetcallbackfunc(CPXENVptr env,
    int(CPXPUBLIC *callback)(CPXCENVptr, void *, int, void *) ,
    void * cbhandle)

Description:
The routine CPXsetnetcallbackfunc is used to set the user-written callback routine to be called each time a log message is issued during the optimization of a network program. If the display log is turned off, the callback routine will still be called. This routine works in the same way as the routine CPXsetlpcallbackfunc. It enables the user to create a separate callback function to be called during the solution of a network problem. The prototype for the callback function is identical to that of CPXsetlpcallbackfunc.

Callback description:

    int callback (CPXENVptr env,
        void *cbdata,
        int wherefrom,
        void *cbhandle);

This is the user-written callback routine.

Callback return value:
A nonzero terminates the optimization.

Callback arguments:

env
A pointer to the CPLEX environment that was passed into the associated optimization routine.

cbdata
A pointer passed from the optimization routine to the user-written callback function that identifies the problem being optimized. The only purpose for the cbdata pointer is to pass it to the routine CPXgetcallbackinfo.

wherefrom
An integer value indicating from which optimization algorithm the user-written callback function was called. Possible values and their meaning appear in the table.

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>CPX_CALLBACK_NETWORK</td>
<td>From network simplex</td>
</tr>
</tbody>
</table>

`cbhandle`

Pointer to user private data, as passed to CPXsetnetcallbackfunc.

**Parameters**

- `env`
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- `callback`
  A pointer to a user-written callback function. Setting `callback` to NULL prevents any callback function from being called during optimization. The call to `callback` occurs after every log message is issued during optimization and periodically during the CPLEX presolve algorithms. This function is written by the user.

- `cbhandle`
  A pointer to user private data. This pointer is passed to the callback function.

**Example**

```c
status = CPXsetnetcallbackfunc (env, myfunc, NULL);
```

**See Also**

CPXgetcallbackinfo, CPXsetlpcallbackfunc, CPXsetmipcallbackfunc

**Returns**

If the operation is successful, the routine returns zero; if not, it returns nonzero to indicate an error.
CPXsetnodecallbackfunc

Category                   Global Function
Definition File            cplex.h
Include Files              cplex.h
Synopsis                   int CPXPUBLIC CPXsetnodecallbackfunc(CPXENVptr env,
                                int(CPXPUBLIC *nodecallback)(CALLBACK_NODE_ARGS) ,
                                void * cbhandle)

Description

Note: This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXsetnodecallbackfunc sets and modifies the user-written callback to be called during MIP optimization after ILOG CPLEX has selected a node to explore, but before this exploration is carried out. The callback routine can change the node selected by ILOG CPLEX to a node selected by the user.

Example

status = CPXgetnodecallbackfunc(env, mynodefunc, mydata);

See also the example admipex1.c in the standard distribution.

Parameters

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

dnodecallback
A pointer to the current user-written node callback. If no callback has been set, the pointer evaluates to NULL.

cbhandle
A pointer to user private data. This pointer is passed to the user-written node callback.

Callback description
int callback (CPXENVptr env,
            void      *cbdata,
            int       wherefrom,
            void      *cbhandle,
            int       *nodeindex_p,
            int       *useraction_p);

ILOG CPLEX calls the node callback after selecting the next node to explore. The user can choose another node by setting the argument values of the callback.

**Callback return value**

The callback returns zero on success and nonzero if an error occurs.

**Callback arguments**

`env`

A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.

`cbdata`

A pointer passed from the optimization routine to the user-written callback that identifies the problem being optimized. The only purpose of this pointer is to pass it to the callback information routines.

`wherefrom`

An integer value indicating where in the optimization this function was called. It has the value `CPX_CALLBACK_MIP_NODE`.

`cbhandle`

A pointer to user private data.

`nodeindex_p`

A pointer to an integer that indicates the node number of the user-selected node. The node selected by ILOG CPLEX is node number 0 (zero). Other nodes are numbered relative to their position in the tree, and this number changes with each tree operation. The unchanging identifier for a node is its sequence number. To access the sequence number of a node, use the ILOG CPLEX Callable Library routine `CPXgetcallbacknodeinfo`. An error results if a user attempts to select a node that has been moved to a node file. (See the *ILOG CPLEX User’s Manual* for more information about node files.)

`useraction_p`

A pointer to an integer indicating the action to be taken on completion of the user callback. The table summarizes the possible actions.
Actions to be Taken after a User-Written Node Callback

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CPX_CALLBACK_DEFAULT</td>
<td>Use ILOG CPLEX-selected node</td>
</tr>
<tr>
<td>1</td>
<td>CPX_CALLBACK_FAIL</td>
<td>Exit optimization</td>
</tr>
<tr>
<td>2</td>
<td>CPX_CALLBACK_SET</td>
<td>Use user-selected node as defined in returned values</td>
</tr>
</tbody>
</table>

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXsetsolvecallbackfunc

Category  | Global Function
Definition File | cplex.h
Include Files | cplex.h
Synopsis

```
int CPXPUBLIC CPXsetsolvecallbackfunc(CPXENVptr env,
   int (CPXPUBLIC *solvecallback)(CALLBACK_SOLVE_ARGS) ,
   void * cbhandle)
```

Description

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXsetsolvecallbackfunc sets and modifies the user-written callback to be called during MIP optimization to optimize the subproblem.

**Example**

```
status = CPXsetsolvecallbackfunc(env, mysolvefunc, mydata);
```

See also the example admipex1.c in the standard distribution.

**Parameters**

- **env**
  A pointer to the CPLEX environment, as returned by CPXopenCPLEX.
- **solvecallback**
  A pointer to a user-written solve callback. If the callback is set to NULL, no callback is called during optimization.
- **cbhandle**
  A pointer to user private data. This pointer is passed to the callback.

**Callback description**

```
int callback (CPXENVptr env,
   void *cbdata,
```
ILOG CPLEX calls the solve callback before ILOG CPLEX solves the subproblem defined by the current node. The user can choose to solve the subproblem in the solve callback instead by setting the user action parameter of the callback. The optimization that the user provides to solve the subproblem must provide a CPLEX solution. That is, the Callable Library routine CPXgetstat must return a nonzero value. The user may access the lp pointer of the subproblem with the Callable Library routine CPXgetcallbacknodelp.

**Callback return value**

The callback returns zero on success and nonzero if an error occurs.

**Callback arguments**

- **env**
  A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

- **cbdata**
  A pointer passed from the optimization routine to the user-written callback that identifies the problem being optimized. The only purpose of this pointer is to pass it to the callback information routines.

- **wherefrom**
  An integer value indicating where in the optimization this function was called. It will have the value CPX_CALLBACK_MIP_SOLVE.

- **cbhandle**
  A pointer to user private data.

- **useraction_p**
  A pointer to an integer indicating the to be taken on completion of the user callback. Table 11 summarizes the possible actions.

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CPX_CALLBACK_DEFAULT</td>
<td>Use ILOG CPLEX subproblem optimizer</td>
</tr>
<tr>
<td>1</td>
<td>CPX_CALLBACK_FAIL</td>
<td>Exit optimization</td>
</tr>
</tbody>
</table>

Table 11 summarizes the possible actions.
Actions to be Taken after a User-Written Solve Callback

<table>
<thead>
<tr>
<th></th>
<th>CPX_CALLBACK_SET</th>
<th>The subproblem has been solved in the callback</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Returns**  
The routine returns zero on success and nonzero if an error occurs.
CPXsetstrparam

Category Global Function

Definition File cplex.h

Include Files cplex.h

Synopsis

int CPXPUBLIC CPXsetstrparam(CPXENVptr env,
   int whichparam,
   const char * newvalue_str)

Description

The routine CPXsetstrparam sets the value of a CPLEX string parameter.

Example

status = CPXsetstrparam (env, CPX_PARAM_WORKDIR, "mydir");

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

whichparam
The symbolic constant (or reference number) of the parameter to change.

newvalue_str
The new value of the parameter. The maximum length of newvalue_str, including the NULL terminator (the character '0' or char(0)), is CPX_STR_PARAM_MAX, defined in cplex.h. Setting newvalue_str to a string longer than this results in an error.

Returns
The routine returns zero on success and nonzero if an error occurs.
**CPXsolninfo**

**Category**      Global Function

**Definition File**  cplex.h

**Include Files**  cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXsolninfo(CPXENVptr env,
                          CPXCLPptr lp,
                          int * solnmethod_p,
                          int * solntype_p,
                          int * pfeasind_p,
                          int * dfeasind_p)
```

**Description**

The routine CPXsolninfo is used to access solution information produced by the routines CPXlpopt, CPXprimopt, CPXdualopt, CPXbaropt, CPXhybbaropt, CPXhybnetopt, and CPXqpopt. This information is maintained until the CPLEX problem object is freed via a call to CPXfreeprob or the solution is rendered invalid because of a call to one of the problem modification routines.

The arguments to CPXsolninfo are pointers to locations where data are to be written. Such data can include the optimization method used to produce the current solution, the type of solution available, and what is known about the primal and dual feasibility of the current solution. If any piece of information represented by an argument to CPXsolninfo is not required, a NULL pointer can be passed for that argument.

**Example**

```c
status = CPXsolninfo (env, lp, &solnmethod, &solntype,
                     &pfeasind, &dfeasind);
```

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **lp**
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

- **solnmethod_p**
  A pointer to an integer indicating the method used to produce the current solution. The specific values which *solnmethod_p* can take and their meanings are the same as the return values documented for CPXgetmethod.

- **solntype_p**
A pointer to an integer variable indicating the type of solution currently available. Possible return values are CPX_BASIC_SOLN, CPX_NONBASIC_SOLN, CPX_PRIMAL_SOLN, and CPX_NO_SOLN, meaning the model either has a simplex basis, has a primal and dual solution but no basis, has a primal solution but no corresponding dual solution, or has no solution, respectively.

pfeasind_p

A pointer to integer variables indicating whether the current solution is known to be primal feasible. Note that a false return value does not necessarily mean that the solution is not feasible. It simply means that the relevant algorithm was not able to conclude it was feasible when it terminated.

dfeasind_p

A pointer to integer variables indicating whether the current solution is known to be dual feasible. Note that a false return value does not necessarily mean that the solution is not feasible. It simply means that the relevant algorithm was not able to conclude it was feasible when it terminated.

**Returns**

The routine returns zero on success and it returns nonzero if an error occurs.
**CPXsolution**

**Category**  Global Function

**Definition File**  cplex.h

**Include Files**  cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXsolution(CPXENVptr env,
   CPXCLPptr lp,
   int * lpstat_p,
   double * objval_p,
   double * x,
   double * pi,
   double * slack,
   double * dj)
```

**Description**

The routine `CPXsolution` accesses the solution values produced by the routines `CPXlpopt`, `CPXprimopt`, `CPXdualopt`, `CPXbaropt`, `CPXhybbaropt`, `CPXhybnetopt`, or `CPXqpopt`. The solution is maintained until the CPLEX problem object is freed via a call to `CPXfreeprob` or the solution is rendered invalid because of a call to one of the problem modification routines.

The arguments to `CPXsolution` are pointers to locations where data are to be written. Such data can include the status of the optimization, the value of the objective function, the values of the primal variables, the dual variables, the slacks and the reduced costs. Although all of the above data exists after a successful call to one of the LP or QP optimizers, it is possible that the user needs only a subset of the data available. If any part of the solution represented by an argument to `CPXsolution` is not required, that argument can be passed with the value NULL in a call to `CPXsolution`. If only one part is required, it may be more convenient to use the CPLEX routine that accesses that part of the solution individually: `CPXgetstat`, `CPXgetobjval`, `CPXgetx`, `CPXgetpi`, `CPXgetslack`, `CPXgetdj`.

For barrier, the solution values for `x`, `pi`, `slack`, and `dj` correspond to the last iterate of the primal-dual algorithm, independent of solution status.

If optimization stopped with an infeasible solution, take care to interpret the meaning of the values in the returned arrays as described in the Arguments section.

**Example**

```c
status = CPXsolution (env, lp, &lpstat, &objval, x, pi,
   slack, dj);
```
See also the example lpex1.c in the *ILOG CPLEX User's Manual* and in the standard distribution.

### See Also

- [CPXsolninfo](#)

### Parameters

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **lp**
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

- **lpstat_p**
  A pointer to an integer indicating the result of the optimization. The specific values which *lpstat_p* can take and their meanings are the same as the return values documented for CPXgetstat and are found in the group optim.cplex.statuscodes of this reference manual.

- **objval_p**
  A pointer to a double precision variable where the objective function value is to be stored.

- **x**
  An array to receive the values of the primal variables for the problem. The length of the array must be at least as great as the number of columns in the problem object. If the solution was computed using the dual simplex optimizer, and the solution is not feasible, *x* values are calculated relative to the phase I RHS used by CPXdualopt.

- **pi**
  An array to receive the values of the dual variables for each of the constraints. The length of the array must be at least as great as the number of rows in the problem object. If the solution was computed using the primal simplex optimizer, and the solution is not feasible, *pi* values are calculated relative to the phase I objective (the infeasibility function).

- **slack**
  An array to receive the values of the slack or surplus variables for each of the constraints. The length of the array must be at least as great as the number of rows in the problem object. If the solution was computed by the dual simplex optimizer, and the solution is not feasible, *slack* values are calculated relative to the phase I RHS used by CPXdualopt.

- **dj**
  An array to receive the values of the reduced costs for each of the variables. The length of the array must be at least as great as the number of columns in the problem object. If
the solution was computed by the primal simplex optimizer, and the solution is not feasible, $d_j$ values are calculated relative to the phase I objective (the infeasibility function).

**Returns**

This routine returns zero if a solution exists. If no solution exists, or some other failure occurs, CPXsolution returns nonzero.
CPXsoswrite

Category Global Function
Definition File cplex.h
Include Files cplex.h

Synopsis
int CPXPUBLIC CPXsoswrite (CPXENVptr env,
                           CPXCLPptr lp,
                           const char * filename_str)

Description
The routine CPXsoswrite is used to write SOS information for a problem into an
SOS format file.

Example
status = CPXsoswrite (env, lp, "myprob.sos");

See also the example mipex3.c in the standard distribution.

Parameters
env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.
filename_str
A character string containing the name of the file to which the SOS information should
be written.

Returns
The routine returns zero on success and nonzero if an error occurs.
**CPXstrcpy**

**Category**  
Global Function

**Definition File**  
cplex.h

**Synopsis**  
CPXCHARptr CPXPUBLIC CPXstrcpy(char * s1_str,  
const char * s2_str)

**Description**  
The routine CPXstrcpy is used to copy strings. It is exactly the same as the standard C library routine strcpy. This routine is provided so that strings passed to the message function routines (see CPXaddfuncdest) can be copied by languages that do not allow dereferencing of pointers (for example, older versions of Visual Basic).

**Example**

CPXstrcpy (p, q);

**Parameters**

- **s1_str**  
A pointer to the string to hold the copy of the string pointed to by src.

- **s2_str**  
A pointer to a string to be copied to dest.

**Returns**  
The routine returns a pointer to the string being copied to.
CPXstrlen

Category       Global Function

Definition File cplex.h

Include Files  cplex.h

Synopsis       int CPXPUBLIC CPXstrlen(const char * s_str)

Description    The routine CPXstrlen determines the length of a string. It is exactly the same as the standard C library routine strlen. This routine is provided so that strings passed to the message function routines (see CPXaddfuncdest) can be analyzed by languages that do not allow dereferencing of pointers (for example, older versions of Visual Basic).

Example

len = CPXstrlen (p);

Parameters

s_str
A pointer to a character string.

Returns       The routine returns the length of the string.
CPXtreewrite

Category                     Global Function
Definition File             cplex.h
Include Files               cplex.h
Synopsis
int CPXPUBLIC CPXtreewrite(CPXENVptr env,
                CPXCLPptr lp,
                const char * filename_str)

Description
The routine CPXtreewrite is used to save a TRE file containing information about the branch & cut path or “tree” for the current problem.

Example
    status = CPXtreewrite (env, lp, "myprob.tre");

Parameters

    env
    A pointer to the CPLEX environment as returned by CPXopenCPLEX.

    lp
    A pointer to a CPLEX problem object as returned by CPXcreateprob.

    filename_str
    A character string containing the name of the file to which the tree information is to be written.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXvecwrite

Category           Global Function
Definition File    cplex.h
Include Files      cplex.h

Synopsis           int CPXPUBLIC CPXvecwrite(CPXENVptr env,
                                      CPXCLPptr lp,
                                      const char * filename_str)

Description        The routine CPXvecwrite is used to write solution information from a Barrier optimization (without crossover) into a VEC format file. The VEC file can later be read back into CPLEX for a crossover.

Example             status = CPXvecwrite (env, lp, "myprob.vec");

Parameters          env
                      A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
                      A pointer to a CPLEX problem object as returned by CPXcreateprob.
filename_str
                      A character string containing the name of the file to which the solution information is to be written.

Returns             The routine returns zero on success and nonzero if an error occurs.
CPXversion

Category        Global Function
Definition File cplex.h
Synopsis        CPXCHARptr CPXPUBLIC CPXversion(CPXENVptr env)
Description     The routine CPXversion returns a pointer to a string indicating the version of the CPLEX library linked with the application. The caller should not change the string returned by this function.

Example

printf ("CPLEX version is %s

Parameters  env

A pointer to the CPLEX environment as returned by CPXopenCPLEX.

Returns     The routine returns NULL if the environment does not exist and the pointer to a string otherwise.
CPXwriteparam

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```
int CPXPUBLIC CPXwriteparam(CPXCENVptr env, const char * filename_str)
```

Description:

The routine CPXwriteparam writes the name and current setting of CPLEX parameters that are not at their default setting in the environment indicated by `env`.

This routine writes a file in a format suitable for reading by CPXreadcopyparam, so you can save current, nondefault parameter settings for re-use in a later session. The file extension is .prm. The PRM format is documented in the reference manual ILOG CPLEX File Formats.

Parameters:

- `env`: A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- `filename_str`: Pointer to the file to write.
CPXwriteprob

Category     Global Function
Definition File cplex.h
Include Files cplex.h
Synopsis     int CPXPUBLIC CPXwriteprob (CPXCENVptr env, 
const char * filename_str, 
const char * filetype_str)
Description  The routine CPXwriteprob is used to write the CPLEX problem object to a file in one 

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAV</td>
<td>Binary matrix and basis file</td>
</tr>
<tr>
<td>MPS</td>
<td>MPS format</td>
</tr>
<tr>
<td>LP</td>
<td>CPLEX LP format</td>
</tr>
<tr>
<td>REW</td>
<td>MPS format, with all names changed to generic names</td>
</tr>
<tr>
<td>RMP</td>
<td>MPS format, with all names changed to generic names</td>
</tr>
<tr>
<td>RLP</td>
<td>LP format, with all names changed to generic names</td>
</tr>
</tbody>
</table>

When this routine is invoked, the current problem is written to a file. If the file name ends with .gz, a compressed file is written.

Example

```c
status = CPXwriteprob (env, lp, "myprob.sav", NULL);
```

See also the example lpex1.c in the ILOG CPLEX User's Manual and in the standard distribution.

Parameters

env

A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

filename_str
A character string containing the name of the file to which the problem is to be written, unless otherwise specified with the filetype argument. If the file name ends with .gz, a compressed file is written in accordance with the selected file type.

filetype_str
A character string containing the type of the file, which can be one of the values in the table. May be NULL, in which case the type is inferred from the file name. The string is not case sensitive.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXwritesol

Category         Global Function
Definition File  cplex.h
Include Files   cplex.h
Synopsis

    int CPXPUBLIC CPXwritesol(CPXENVptr env,
                   CPXCLPptr lp,
                   const char * filename_str,
                   const char * filetype_str)

Description

The routine CPXwritesol is used to write a solution file for the selected CPLEX
problem object. The routine can write files in:

- TXT format, which is the ASCII format
- BIN format, which is the binary format.

The ASCII and binary formats are documented in the reference manual ILOG CPLEX

Values of filetype_str

<table>
<thead>
<tr>
<th>'TXT'</th>
<th>ASCII solution file</th>
</tr>
</thead>
<tbody>
<tr>
<td>'BIN'</td>
<td>Binary solution file</td>
</tr>
</tbody>
</table>

Example

    status = CPXwritesol (env, lp, "myfile.txt", NULL);

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.
filename_str
A character string containing the name of the file to which the solution should be written.
filetype_str
A character string containing the type of the file. The type of the file can be one of the values in the table. May be NULL, in which case the type is inferred from the characters following the last period in the filename. The string is not case sensitive.

Returns The routine returns zero on success and nonzero if an error occurs.

**Group optim.cplex.callable.createdeletecopy**

The routines in the ILOG CPLEX Callable Library to create and delete problems and to copy data.

### Global Functions Summary

<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXcloneprob</td>
</tr>
<tr>
<td>CPXcompletelp</td>
</tr>
<tr>
<td>CPXcopybase</td>
</tr>
<tr>
<td>CPXcopyctype</td>
</tr>
<tr>
<td>CPXcopylp</td>
</tr>
<tr>
<td>CPXcopylpwnames</td>
</tr>
<tr>
<td>CPXcopymipstart</td>
</tr>
<tr>
<td>CPXcopynettolp</td>
</tr>
<tr>
<td>CPXcopyobjcname</td>
</tr>
<tr>
<td>CPXcopyorder</td>
</tr>
<tr>
<td>CPXcopyqpsep</td>
</tr>
<tr>
<td>CPXcopyquad</td>
</tr>
<tr>
<td>CPXcopystart</td>
</tr>
<tr>
<td>CPXcreateprob</td>
</tr>
<tr>
<td>CPXfreeprob</td>
</tr>
<tr>
<td>CPXNETaddarcs</td>
</tr>
<tr>
<td>CPXNETaddnodes</td>
</tr>
<tr>
<td>CPXNETcheckcopynet</td>
</tr>
<tr>
<td>CPXNETcopybase</td>
</tr>
<tr>
<td>CPXNETcopynet</td>
</tr>
<tr>
<td>CPXNETcreateprob</td>
</tr>
<tr>
<td>CPXNETextract</td>
</tr>
<tr>
<td>CPXNETfreeprob</td>
</tr>
<tr>
<td>CPXreadcopy sos</td>
</tr>
</tbody>
</table>

### Description

These routines create, populate, or delete a problem object. You can also populate a problem object by means of the routines for modifying a problem or for reading data from a file.
CPXNETaddarcs

Category
Global Function

Definition File
cplex.h

Include Files
cplex.h

Synopsis
int CPXPUBLIC CPXNETaddarcs (CPXCENVptr env, CPXNETptr net, int narcs, const int * fromnode, const int * tonode, const double * low, const double * up, const double * obj, char ** anames)

Description
The routine CPXNETaddarcs adds new arcs to the network stored in a network problem object.

Example
status = CPXNETaddarcs (env, net, narcs, fromnode, tonode, NULL, NULL, obj, NULL);

See Also
CPXNETgetnumnodes

Parameters
env
A pointer to the CPLEX environment as returned by CPXopenCplex.

net
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

narcs
Number of arcs to be added.

fromnode
Array of indices of the “from node” for the arcs to be added. All the indices must be greater than or equal to 0. If a node index is greater than or equal to the number of nodes currently in the network (see CPXNETgetnumnodes) new nodes are created implicitly with default supply values 0. The size of the fromnode array must be at least narcs.

tonode
Array of indices of the “to node” for the arcs to be added. All the indices must be greater than or equal to 0. If a node index is greater than or equal to the number of nodes currently in the network (see CPXNETgetnumnodes) new nodes are created implicitly with default supply values 0. The size of the tonode array must be at least narcs.

low

Pointer to an array of lower bounds on the flow through added arcs. If NULL is passed, all lower bounds default to 0 (zero). Otherwise, the size of the array must be at least narcs. Values less than or equal to -CPX_INFBOUND are considered as -?.

up

Pointer to an array of upper bounds on the flow of added arcs. If NULL is passed, all upper bounds default to CPX_INFBOUND. Otherwise, the size of the array must be at least narcs. Values greater than or equal to CPX_INFBOUND are considered as ?.

obj

Pointer to an array of objective values for the added arcs. If NULL is passed, all objective values default to 0. Otherwise, the size of the array must be at least narcs.

anames

Pointer to an array of names for added arcs. If NULL is passed and the existing arcs have names, default names are assigned to the added arcs. If NULL is passed and the existing arcs have no names, the new arcs are assigned no names. Otherwise, the size of the array must be at least narcs and every name in the array must be a string terminating in 0. If the existing arcs have no names and anames is not NULL, default names are assigned to the existing arcs.

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXNETaddnodes

Category: Global Function
Definition File: cplex.h
Include Files: cplex.h

Synopsis:
```
int CPXPUBLIC CPXNETaddnodes (CPXCENVptr env,
   CPXNETptr net,
   int nnodes,
   const double * supply,
   char ** name)
```

Description:
The routine CPXNETaddnodes adds new nodes to the network stored in a network problem object.

Example:
```
status = CPXNETaddnodes (env, net, nnodes, supply, NULL);
```

Parameters:

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCplex.

- **net**
  A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

- **nnodes**
  Number of nodes to add.

- **supply**
  Supply values for the added nodes. If NULL is passed, all supplies defaults to 0 (zero). Otherwise, the size of the array must be at least nnodes.

- **name**
  Pointer to an array of names for added nodes. If NULL is passed and the existing nodes have names, default names are assigned to the added nodes. If NULL is passed but the existing nodes have no names, the new nodes are assigned no names. Otherwise, the size of the array must be at least nnodes and every name in the array must be a string terminating in 0. If the existing nodes have no names and nnames is not NULL, default names are assigned to the existing nodes.

Returns:
The routine returns zero on success and nonzero if an error occurs.
CPXNETcheckcopynet

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```
int CPXPUBLIC CPXNETcheckcopynet(CPXENVptr env, 
    CPXNETptr net, 
    int objsen, 
    int nnodes, 
    const double * supply, 
    char ** nnames, 
    int narcs, 
    const int * fromnode, 
    const int * tonode, 
    const double * low, 
    const double * up, 
    const double * obj, 
    char ** aname)
```

Description:

The routine CPXNETcheckcopynet performs a consistency check on the argument passed to the routine CPXNETcopynet.

The CPXNETcheckcopynet routine has the same argument list as the CPXNETcopynet routine.

Example:

```
status = CPXNETcheckcopynet (env, net, CPX_MAX, nnodes, supply, 
    nnames, narcs, fromnode, tonode, 
    lb, ub, obj, anames);
```

Returns:

The routine returns zero on success and nonzero if an error occurs.
CPXNETcopybase

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXNETcopybase(CPXENVptr env,  
                            CPXNETptr net,  
                            const int * astat,  
                            const int * nstat)
```

**Description**  
The routine CPXNETcopybase can be used to set the network basis for a network problem object. It is not necessary to load a basis prior to optimizing a problem, but a very good starting basis may increase the speed of optimization significantly. A copied basis does not need to be feasible to be used by the network optimizer. Any solution information stored in the problem object is lost.

**Example**  
```c
status = CPXNETcopybase (env, net, arc_stat, node_stat);
```

**Table 1: Status of arcs in astat**

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_BASIC</td>
<td>If the arc is to be basic</td>
</tr>
<tr>
<td>CPX_AT_LOWER</td>
<td>If the arc is to nonbasic and its flow is on the lower bound</td>
</tr>
<tr>
<td>CPX_AT_UPPER</td>
<td>If the arc is to nonbasic and its flow is on the upper bound</td>
</tr>
<tr>
<td>CPX_FREE_SUPER</td>
<td>If the arc is to nonbasic but is free. In this case its flow is set to 0</td>
</tr>
</tbody>
</table>

**Table 2: Status of artificial arcs in nstat**

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_BASIC</td>
<td>If the arc is to be basic</td>
</tr>
<tr>
<td>CPX_AT_LOWER</td>
<td>If the arc is to nonbasic and its flow is set to 0</td>
</tr>
</tbody>
</table>

**Parameters**  
- `env`
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

net

A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

astat

Array of status values for network arcs. Each arc needs to be assigned one of the values in Table 1.

nstat

Array of status values for artificial arcs from each node to the root node. Each artificial arc needs to be assigned one of the values in Table 2. At least one of the artificial arcs must be assigned the status CPX_BASIC for a network basis.

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXNETcopynet

Category: Global Function
Definition File: cplex.h
Include Files: cplex.h

Synopsis:
```c
int CPXPUBLIC CPXNETcopynet(CPXENVptr env,
   CPXNETptr net,
   int objsen,
   int nnodes,
   const double * supply,
   char ** nnames,
   int narcs,
   const int * fromnode,
   const int * tonode,
   const double * low,
   const double * up,
   const double * obj,
   char ** anames)
```

Description:
The routine CPXNETcopynet copies a network to a network object, overriding any other network saved in the object. The network to be copied is specified by providing:

* the objective sense
* number of nodes
* supply values for each node
* names for each node
* number of arcs
* indices of the “from nodes” (or, equivalently, the tail nodes) for each arc
* indices of the “to nodes” (or, equivalently, the head nodes) for each arc
* lower and upper bounds on flow through each arc
* cost for flow through each arc
* names of each arc as parameters to the routine CPXNETcopynet.

The arcs are numbered according to the order given in the fromnode and tonode arrays. Some of the parameters are optional and replaced by default values if NULL is passed for them.

Example:
status = CPXNETcopynet (env, net, CPX_MAX, nnodes, supply, NULL, 
narcs, fromnode, tonode, NULL, NULL, obj, 
NULL);

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

net
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

objsen
Optimization sense of the network to be copied. It may take values CPX_MAX for a maximization problem or CPX_MIN for a minimization problem.

nnodes
Number of nodes to be copied to the network object.

supply
Supply values for the nodes. If NULL is passed all supply values default to 0 (zero). Otherwise, the size of the array must be at least nnodes.

nnames
Pointer to an array of names for the nodes. If NULL is passed, no names are assigned to the nodes. Otherwise, the size of the array must be at least nnodes and every name in the array must be a string terminating in 0.

narcs
Number of arcs to be copied to the network object.

fromnode
The array of indices in each arc’s “from node.” The indices must be in the range [0, nnodes-1]. The size of the array must be at least narcs.

tonode
The array of indices in each arc’s “to node.” The indices must be in the range [0, nnodes-1]. The size of the array must be at least narcs.

low
Pointer to an array of lower bounds on the flow through arcs. If NULL is passed, all lower bounds default to 0 (zero). Otherwise, the size of the array must be at least narcs. Values less than or equal to -CPX_INFBOUND are considered -infinity.
up
Pointer to an array of upper bounds on the flow through arcs. If NULL is passed, all lower bounds default to \texttt{CPX-INFBOUND}. Otherwise, the size of the array must be at least \texttt{narcs}. Values greater than or equal to \texttt{CPX-INFBOUND} are considered infinity.

\textbf{obj}
Pointer to an array of objective values for flow through arcs. If NULL is passed, all objective values default to 0 (zero). Otherwise, the size of the array must be at least \texttt{narcs}.

\textbf{anames}
Pointer to an array of names for the arcs. If NULL is passed, no names are assigned to the nodes. Otherwise, the size of the array must be at least \texttt{narcs}, and every name in the array must be a string terminating in 0 (zero).

\textbf{Returns}
The routine returns zero on success and nonzero if an error occurs.
CPXNETcreateprob

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
CPXNETptr CPXPUBLIC CPXNETcreateprob(CPXENVptr env,
    int * status_p,
    const char * name_str)
```

Description:
The routine CPXNETcreateprob constructs a new network problem object. The new object contains a minimization problem for a network with 0 nodes and 0 arcs. Other network problem data can be copied to a network with one of the routines CPXNETaddnodes, CPXNETaddarcs, CPXNETcopynet, CPXNETextract, or CPXNETreadcopyprob.

Example:

```c
CPXNETptr net = CPXNETcreateprob (env, &status, "mynet");
```

See Also:
CPXNETaddnodes, CPXNETaddarcs, CPXNETcopynet, CPXNETextract, CPXNETreadcopyprob

Parameters:

- `env`
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- `status_p`
  A pointer to an integer used to return any error code produced by this routine.

- `name_str`
  Name of the network to be created.

Returns:
If the operation is successful, CPXNETcreateprob returns the newly constructed network problem object; if not, it returns either NULL or a nonzero value to indicate an error. In case of an error, the value pointed to by `status_p` contains an integer indicating the cause of the error.
CPXNETextract

Category          Global Function

Definition File   cplex.h

Include Files    cplex.h

Synopsis

```c
int CPXPUBLIC CPXNETextract (CPXENVptr env,
                              CPXNETptr net,
                              CPXCLPptr lp,
                              int * colmap,
                              int * rowmap)
```

Description

The routine **CPXNETextract** finds an embedded network in the LP stored in a CPLEX problem object and copies it as a network to the network problem object, `net`. The extraction algorithm is controlled by the environment parameter `CPX_PARAM_NETFIND`.

If the CPLEX problem object has a basis, an attempt is made to copy the basis to the network object. However, this may fail if the statuses corresponding to the rows and columns of the subnetworks do not form a basis. Even if the entire LP is a network, it may not be possible to load the basis to the network object if none of the slack or artificial variables are basic.

The size of `colmap` must be at least `CPXnumcols(env, lp) + CPXnumrows(env, lp)`.

The size of `rowmap` must be at least `CPXnumrows(env, lp) + 1`.

Example

```c
status = CPXNETextract (env, net, lp, colmap, rowmap);
```

Parameters

- **env**
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- **net**
  A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.

- **lp**
  A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

- **colmap**
If not NULL, after completion \( \text{colmap}[i] \) contains the index of the LP column that has been mapped to arc \( i \). If \( \text{colmap}[i] < 0 \), arc \( i \) corresponds to the slack variable for row \(-\text{colmap}[i]-1\).

The size of \( \text{colmap} \) must be at least \( \text{CPXnumcols}(\text{env}, \text{lp}) + \text{CPXnumrows}(\text{env}, \text{lp}) \).

If not NULL, after completion \( \text{rowmap}[i] \) contains the index of the LP row that has been mapped to node \( i \). If \( \text{colmap}[i] < 0 \), node \( i \) is a dummy node that has no corresponding row in the LP.

The size of \( \text{rowmap} \) must be least \( \text{CPXnumrows}(\text{env}, \text{lp}) + 1 \).

**Example**

```c
status = CPXNETextract (env, net, lp, colmap, rowmap);
```

**Returns**

The routine returns zero on success and nonzero if an error occurs.
**CPXNETfreeprob**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXNETfreeprob (CPXENVptr env, 
    CPXNETptr * net_p)
```

**Description**  
The routine `CPXNETfreeprob` deletes the network problem object pointed to by `net_p`. This also deletes all network problem data and solution data stored in the network problem object.

**Example**  
```c
CPXNETfreeprob (env, &net);
```

**Parameters**  
- **env**  
A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- **net_p**  
CPLEX network problem object to be deleted.

**Returns**  
The routine returns zero on success and nonzero if an error occurs.
**CPXcloneprob**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
CPXLPtr CPXPUBLIC CPXcloneprob (CPXCENVptr env,  
CPXCLPptr lp,  
int * status_p)

**Description**  
The routine **CPXcloneprob** can be used to create a new CPLEX problem object and copy all the problem data from an existing problem object to it. Solution and starting information is not copied.

**Example**  

```c
    copy = CPXcloneprob (env, lp, &status);
```

**Parameters**

- **env**  
  A pointer to the CPLEX environment as returned by **CPXopenCPLEX**.
- **lp**  
  A pointer to a CPLEX problem object of which a copy is to be created.
- **status_p**  
  A pointer to an integer used to return any error code produced by this routine.

**Example**  

```c
    copy = CPXcloneprob (env, lp, &status);
```

**Returns**  
If successful, **CPXcloneprob** returns a pointer that can be passed to other CPLEX routines to identify the problem object that is created. If not successful, a NULL pointer (NULL) is returned, and an error status is returned in the variable *status_p. If the routine is successful, *status_p is zero.
**CPXcompletp**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXcompletp(CPXENVptr env, CPXLPtr lp)
```

**Description**  
The routine CPXcompletp is provided to allow users to handle those rare cases where modification steps need to be closely managed; for example, when careful timings are desired for the individual steps in a user's solution process, or more control of memory allocations for problem modifications is needed.

**Example**  
```c
status = CPXcompletp (env, lp);
```

**Parameters**

- **env**  
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **lp**  
A pointer to a CPLEX problem object as returned by CPXcreateprob.

**Returns**  
The routine returns zero on success and nonzero if an error occurs.
The routine CPXcopybase is used to copy a basis into a CPLEX problem object. It is not necessary to copy a basis prior to optimizing an LP problem, but a good initial basis can increase the speed of optimization significantly. A basis does not need to be primal or dual feasible to be used by the optimizer.

**Note:** The basis is ignored by the optimizer if CPX_PARAM_ADVIND is set to CPX_OFF.

### Table 1: Values of basis status for columns in cstat[j]

<table>
<thead>
<tr>
<th>Basis Status</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_AT_LOWER</td>
<td>0</td>
<td>variable at lower bound</td>
</tr>
<tr>
<td>CPX_BASIC</td>
<td>1</td>
<td>variable is basic</td>
</tr>
<tr>
<td>CPX_AT_UPPER</td>
<td>2</td>
<td>variable at upper bound</td>
</tr>
<tr>
<td>CPX_FREE_SUPER</td>
<td>3</td>
<td>variable free and nonbasic</td>
</tr>
</tbody>
</table>

### Table 2: Values of basis status for rows other than ranged rows in rstat[j]

<table>
<thead>
<tr>
<th>Basis Status</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_AT_LOWER</td>
<td>0</td>
<td>associated slack/surplus/artificial variable nonbasic at value 0.0</td>
</tr>
<tr>
<td>CPX_BASIC</td>
<td>1</td>
<td>associated slack/surplus/artificial variable basic</td>
</tr>
</tbody>
</table>
Example

\[
\text{status} = \text{CPXcopybase} (\text{env}, \text{lp}, \text{cstat}, \text{rstat});
\]

See Also

CPXreadcopybase

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

cstat
An array containing the basis status of the columns in the constraint matrix. The length of the array is equal to the number of columns in the problem object. Possible values of the basis status of columns appear in Table 1.

rstat
An array containing the basis status of the slack/surplus/artificial variable associated with each row in the constraint matrix. The array’s length is equal to the number of rows in the CPLEX problem object. For rows other than ranged rows, the array element \(rstat[i]\) has the meaning in Table 2. For ranged rows, the array element \(rstat[i]\) has the meaning in Table 3.

Returns
The routine returns zero on success and nonzero if an error occurs.
**CPXcopyctype**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```
int CPXPUBLIC CPXcopyctype(CPXENVptr env,  
            CPXLPptr lp,  
            const char * xtype)
```

**Description**  
The routine `CPXcopyctype` can be used to copy variable type information to a given problem. Variable types indicate whether a variable is continuous, integer, binary, semi-continuous, or semi-integer. If the type of the problem object is `CPXPROB_LP`, it is changed to `CPXPROB_MILP`. If the type of the problem object is `CPXPROB_QP`, it is changed to `CPXPROB_MIQP`. If the type of the problem object is `CPXPROB_QCP`, it is changed to `CPXPROB_MIQCP`.

This routine allows the types of all the variables to be set in one function call. When `CPXcopyctype` is called, any current solution information is freed.

**Example**  
```
status = CPXcopyctype (env, lp, xtype);
```
See also the example mipex1.c in the standard distribution.

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **lp**
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

- **xctype**
  An array of length CPXgetnumcols(env,lp) containing the type of each column in the constraint matrix. Possible values appear in Table 1.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
**CPXcopylp**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXcopylp(CPXENVptr env,
    CPXLPptr lp,
    int numcols,
    int numrows,
    int objsen,
    const double * obj,
    const double * rhs,
    const char * sense,
    const int * matbeg,
    const int * matcnt,
    const int * matind,
    const double * matval,
    const double * lb,
    const double * ub,
    const double * rngval)
```

**Description**

The routine CPXcopylp copies data that defines an LP problem to a CPLEX problem object. The arguments to CPXcopylp define an objective function, the constraint matrix, the right-hand side, and the bounds on the variables. The routine CPXcopylp does not copy names. Calling CPXcopylp destroys any existing data associated with the problem object.

The more comprehensive routine CPXcopylpwnames can be used in place of CPXcopylp to copy linear programs with associated names.

The arguments passed to CPXcopylp define a linear program. Since these arguments are copied into local arrays maintained by CPLEX, the LP problem data passed via CPXcopylp may be modified or freed after the call to CPXcopylp without affecting the state of the CPLEX problem object.

<table>
<thead>
<tr>
<th>objsen</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>(CPX_MIN) minimize</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>(CPX_MAX) maximize</td>
</tr>
</tbody>
</table>
The arrays \( \text{matbeg} \), \( \text{matcnt} \), \( \text{matind} \), and \( \text{matval} \) are accessed as follows. Suppose that CPLEX wants to access the entries in some column \( j \). These are assumed to be given by the array entries:

\[
\text{matval}[\text{matbeg}[j]], \ldots, \text{matval}[\text{matbeg}[j]+\text{matcnt}[j]-1]
\]

The corresponding row indices are:

\[
\text{matind}[\text{matbeg}[j]], \ldots, \text{matind}[\text{matbeg}[j]+\text{matcnt}[j]-1]
\]

Entries in \( \text{matind} \) are not required to be in row order. Duplicate entries in \( \text{matind} \) within a single column are not allowed. The length of the arrays \( \text{matbeg} \) and \( \text{matind} \) should be of at least \( \text{numcols} \). The length of arrays \( \text{matind} \) and \( \text{matval} \) should be of at least \( \text{matbeg}[\text{numcols}-1]+\text{matcnt}[\text{numcols}-1] \).

### Example

```c
status = CPXcopylp (env, lp, numcols, numrows, objsen, obj, rhs, 
                   sense, matbeg, matcnt, matind, matval, lb, 
                   ub, rngval);
```

See also the example `lpex1.c` in the *ILOG CPLEX User’s Manual* and in the standard distribution.

#### Parameters

- **env**
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
- **lp**
  A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
- **numcols**
  An integer that indicates the number of columns in the constraint matrix, or equivalently, the number of variables in the problem object.

---

### Table 2: Values of sense

<table>
<thead>
<tr>
<th>( \text{sense}[i] )</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>’L’</td>
<td>≤</td>
<td>constraint</td>
</tr>
<tr>
<td>’E’</td>
<td>=</td>
<td>constraint</td>
</tr>
<tr>
<td>’G’</td>
<td>≥</td>
<td>constraint</td>
</tr>
<tr>
<td>’R’</td>
<td>ranged constraint</td>
<td></td>
</tr>
</tbody>
</table>
numrows
An integer that indicates the number of rows in the constraint matrix, not including the
objective function or bounds on the variables.
objsen
An integer that indicates whether the problem is a minimization or maximization
problem.
objcen
An integer that indicates whether the problem is a minimization or maximization
problem.
rhs
An array of length at least numrows containing the right-hand side value for each
constraint in the constraint matrix.
sense
An array of at least length at least numrows containing the sense of each constraint in
the constraint matrix.
matbeg
An array that with matval, matcnt, and matind defines the constraint matrix.
matcnt
An array that with matbeg, matval, and matind defines the constraint matrix.
matind
An array that with matbeg, matcnt, and matval defines the constraint matrix.
matval
An array that with matbeg, matcnt, and matind defines the constraint matrix.
CPLEX needs to know only the nonzero coefficients. These are grouped by column in
the array matval. The nonzero elements of every column must be stored in sequential
locations in this array with matbeg[j] containing the index of the beginning of
column j and matcnt[j] containing the number of entries in column j. The
components of matbeg must be in ascending order. For each k, matind[k] indicates
the row number of the corresponding coefficient, matval[k].
lb
An array of at least length numcols containing the lower bound on each of the
variables. Any lower bound that is set to a value less than or equal to that of the constant
-CPX_INFBOUND is treated as negative ?. CPX_INFBOUND is defined in the header
file cplex.h.
ub
An array of at least length `numcols` containing the upper bound on each of the variables. Any upper bound that is set to a value greater than or equal to that of the constant `CPX_INFBOUND` is treated as ?. `CPX_INFBOUND` is defined in the header file `cplex.h`.

rngval
An array of at least length `numrows` containing the range value of each ranged constraint. Ranged rows are those designated by 'R' in the sense array. If the row is not ranged, the `rngval` array entry is ignored. If `rngval[i] > 0`, then row `i` activity is in `[rhs[i], rhs[i] + rngval[i]]`, and if `rngval[i] <= 0`, then row `i` activity is in `[rhs[i] + rngval[i], rhs[i]]`. This parameter may be NULL.

Returns
The routine returns zero on success and nonzero if an error occurs.
**CPXcopylpwnames**

**Category**  Global Function

**Definition File**  cplex.h

**Include Files**  cplex.h

**Synopsis**  

```c
int CPXPUBLIC CPXcopylpwnames(CPXENVptr env,
CPXLPptr lp,
int numcols,
int numrows,
int objsen,
const double * obj,
const double * rhs,
const char * sense,
const int * matbeg,
const int * matcnt,
const int * matind,
const double * matval,
const double * lb,
const double * ub,
const double * rngval,
char ** colname,
char ** rowname)
```

**Description**  
The routine `CPXcopylpwnames` copies LP data into a CPLEX problem object in the same way as the routine `CPXcopylp`, but using some additional arguments to specify the names of constraints and variables in the CPLEX problem object. The arguments to `CPXcopylpwnames` define an objective function, constraint matrix, variable bounds, right-hand side constraint senses, and range values. Unlike the routine `CPXcopylp`, `CPXcopylpwnames` also copies names. This routine is used in the same way as `CPXcopylp`.

**Table 1: Settings for objsen**

<table>
<thead>
<tr>
<th>objsen</th>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(CPX_MIN)</td>
<td>minimize</td>
</tr>
<tr>
<td>-1</td>
<td>(CPX_MAX)</td>
<td>maximize</td>
</tr>
</tbody>
</table>

**Table 2: Settings for sense**

<table>
<thead>
<tr>
<th>sense[i]</th>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'L'</td>
<td>$ \leq $ constraint</td>
<td></td>
</tr>
<tr>
<td>'E'</td>
<td>$ = $ constraint</td>
<td></td>
</tr>
<tr>
<td>'G'</td>
<td>$ \geq $ constraint</td>
<td></td>
</tr>
</tbody>
</table>
With respect to the parameters \texttt{matbeg} (beginning of the matrix), \texttt{matcnt} (count of the matrix), \texttt{matind} (indices of the matrix), and \texttt{matval} (values of the matrix), CPLEX needs to know only the nonzero coefficients. These are grouped by column in the array \texttt{matval}. The nonzero elements of every column must be stored in sequential locations in this array with \texttt{matbeg[j]} containing the index of the beginning of column \(j\) and \texttt{matcnt[j]} containing the number of entries in column \(j\). The components of \texttt{matbeg} must be in ascending order. For each \(k\), \texttt{matind[k]} indicates the row number of the corresponding coefficient, \texttt{matval[k]}.

These arrays are accessed as follows. Suppose that CPLEX wants to access the entries in some column \(j\). These are assumed to be given by the array entries:

\[
\texttt{matval[matbeg[j]]}, \ldots, \texttt{matval[matbeg[j]+matcnt[j]-1]}
\]

The corresponding row indices are:

\[
\texttt{matind[matbeg[j]]}, \ldots, \texttt{matind[matbeg[j]+matcnt[j]-1]}
\]

Entries in \texttt{matind} are not required to be in row order. Duplicate entries in \texttt{matind} and \texttt{matval} within a single column are not allowed. The length of the arrays \texttt{matbeg} and \texttt{matind} should be of at least \texttt{numcols}. The length of arrays \texttt{matind} and \texttt{matval} should be of at least \texttt{matbeg[numcols-1]+matcnt[numcols-1]}.

Example

```c
status = CPXcopylpwnames (env,
                         lp,
                         numcols,
                         numrows,
                         objsen,
                         obj,
                         rhs,
                         sense,
                         matbeg,
                         matcnt,
                         matind,
                         matval,
                         lb,
                         ub,
                         rngval,
                         colname,
                         rowname);
```

---

**Table 2: Settings for \texttt{sense}**

<table>
<thead>
<tr>
<th>\texttt{sense[i]}</th>
<th>= 'R'</th>
<th>ranged constraint</th>
</tr>
</thead>
</table>

---

**sense\[i\] = 'R' ranged constraint**
Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

numcols
An integer that indicates the number of columns in the constraint matrix, or equivalently, the number of variables in the problem object.

numrows
An integer that indicates the number of rows in the constraint matrix, not including the objective function or bounds on the variables.

objsen
An integer that indicates whether the problem is a minimization or maximization problem. Table 1 shows its possible settings.

objen
An integer that indicates whether the problem is a minimization or maximization problem. Table 1 shows its possible settings.

rhs
An array of length at least numrows containing the right-hand side value for each constraint in the constraint matrix.

sense
An array of at least length at least numrows containing the sense of each constraint in the constraint matrix. Table 2 shows the possible settings.

matbeg
An array that defines the constraint matrix.

matcnt
An array that defines the constraint matrix.

matind
An array that defines the constraint matrix.

matval
An array that defines the constraint matrix.
lb
An array of at least length `numcols` containing the lower bound on each of the variables. Any lower bound that is set to a value less than or equal to that of the constant `-CPX_INFBOUND` is treated as negative infinity. `CPX_INFBOUND` is defined in the header file `cplex.h`.

ub
An array of at least length `numcols` containing the upper bound on each of the variables. Any upper bound that is set to a value greater than or equal to that of the constant `CPX_INFBOUND` is treated as infinity. `CPX_INFBOUND` is defined in the header file `cplex.h`.

rngval
An array of at least length `numrows` containing the range value of each ranged constraint. Ranged rows are those designated by `R` in the `sense` array. If the row is not ranged, the `rngval` array entry is ignored. If `rngval[i] > 0`, then row `i` activity is in `[rhs[i], rhs[i] + rngval[i]]`, and if `rngval[i] ≤ 0`, then row `i` activity is in `[rhs[i] + rngval[i], rhs[i]]`. This parameter may be NULL.

colname
An array of length at least `numcols` containing pointers to character strings. Each string is terminated with the NULL character. These strings represent the names of the matrix columns or, equivalently, the variable names. May be NULL if no names are associated with the variables. If `colname` is not NULL, every variable must be given a name. The addresses in `colname` do not have to be in ascending order.

rowname
An array of length at least `numrows` containing pointers to character strings. Each string is terminated with the NULL character. These strings represent the names of the matrix rows or, equivalently, the constraint names. May be NULL if no names are associated with the constraints. If `rowname` is not NULL, every constraint must be given a name. The addresses in `rowname` do not have to be in ascending order.

**Returns**
zero on success and nonzero if an error occurs.
CPXcopymipstart

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXcopymipstart(CPXENVptr env,
     CPXLPptr lp,
     int cnt,
     const int * indices,
     const double * value)
```

Description:

The routine CPXcopymipstart is used to copy MIP starting values to a CPLEX problem object of type CPXPROB_MILP, CPXPROB_MIQP, or CPXPROB_MIQCP. Values must be specified for all integer and SOS member variables. Optionally, values may be specified for continuous variables. If the provided starting values are compatible with an integer feasible solution, that solution becomes the incumbent at the start of the optimizations. Otherwise, the starting values are ignored. The values for member variables of Special Ordered Sets are used to indicate which members of the set take on nonzero values.

The parameter CPX_PARAM_MIPSTART must be set to CPX_ON (1) for the values to be used. If it is set to zero, which is the default, the values are not used.

Example:

```c
status = CPXcopymipstart (env, lp, cnt, indices, value);
```

See Also: CPXreadcopyorder, CPXreadcopymipstart

Parameters:

- **env**: A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **lp**: A pointer to a CPLEX problem object as returned by CPXcreateprob.
- **cnt**: An integer giving the number of entries in the list.
- **indices**: An array of length cnt containing the numeric indices of the columns corresponding to the variables which are assigned starting values.
value

An array of length cnt containing the values to be used for the starting integer solution. The entry value[j] is the value assigned to variable indices[j]. An entry value[j] greater than or equal to CPX_INFBOUND indicates no value is set for variable indices[j].

Example

status = CPXcopymipstart (env, lp, cnt, indices, value);

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXcopynettolp

Category  Global Function

Definition File  cplex.h

Include Files  cplex.h

Synopsis  

int CPXPUBLIC CPXcopynettolp (CPXCENVptr env,  
                  CPXLPptr lp,  
                  CPXCNETptr net)

Description  The routine CPXcopynettolp is used to copy a network problem stored in a network problem object to a CPLEX problem object (as an LP). Any problem data previously stored in the CPLEX problem object is overridden.

Example  

status = CPXcopynettolp (env, lp, net);

Parameters  

env  
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp  
A pointer to a CPLEX problem object as returned by CPXcreateprob.

net  
A pointer to a CPLEX network problem object containing the network problem to be copied.

Example  

status = CPXcopynettolp (env, lp, net);

Returns  The routine returns zero on success and nonzero if an error occurs.
CPXcopyobjname

Category       Global Function
Definition File cplex.h
Include Files  cplex.h

Synopsis       int CPXPUBLIC CPXcopyobjname(CPXENVptr env,
               CPXLPptr lp,
               const char * objname_str)

Description    The routine CPXcopyobjname copies a name for the objective function into a CPLEX problem object. An argument to CPXcopyobjname defines the objective name.

Example

    status = CPXcopyobjname (env, lp, "Cost");

Parameters

    env
    A pointer to the CPLEX environment as returned by CPXopenCPLEX.
    ip
    A pointer to a CPLEX problem object as returned by CPXcreateprob.
    objname_str
    A pointer to a string representing the objective name.

Returns        The routine returns zero on success and nonzero if an error occurs.
**CPXcopyorder**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPX_PUBLIC CPXcopyorder (CPXCENVptr env,
    CPXLPptr lp,
    int cnt,
    const int * indices,
    const int * priority,
    const int * direction)
```

**Description**  
The routine `CPXcopyorder` is used to copy a priority order to a CPLEX problem object of type `CPXPROB_MILP`, `CPXPROB_MIQP`, or `CPXPROB_MIQCP`. During branching, integer variables with higher priorities are given preference over integer variables with lower priorities. Priorities must be positive integers. A preferred branching direction may also be specified for each variable.

The CPLEX parameter `CPX_PARAM_MIPORDIND` must be set to `CPX_ON`, its default value, for the priority order to be used in a subsequent optimization.

**Table 1: Settings for direction**

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_BRANCH_GLOBAL</td>
<td>use global branching direction when setting the parameter <code>CPX_PARAM_BRDIR</code></td>
</tr>
<tr>
<td>CPX/Branch_DOWN</td>
<td>branch down first on variable <code>indices[i]</code></td>
</tr>
<tr>
<td>CPX/Branch_UP</td>
<td>branch up first on variable <code>indices[i]</code></td>
</tr>
</tbody>
</table>

**Example**

```c
status = CPXcopyorder (env, lp, cnt, indices, priority, direction);
```

**See Also**  
`CPXreadcopyorder`

**Parameters**

- **env**  
  A pointer to the CPLEX environment as returned by `CPXopenCplex`.

- **lp**  
  A pointer to a CPLEX problem object as returned by `CPXcreateprob`. 
cnt
An integer giving the number of entries in the list.

indices
An array of length cnt containing the numeric indices of the columns corresponding to
the integer variables that are assigned priorities.

priority
An array of length cnt containing the priorities assigned to the integer variables. The
entry priority[j] is the priority assigned to variable indices[j]. May be NULL.

direction
An array of type integer containing the branching direction assigned to the integer
variables. The entry direction[j] is the direction assigned to variable indices
[j]. May be NULL. Possible settings for direction[j] appear in Table 1.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXcopyqpsep

Category      Global Function
Definition File cplex.h
Include Files cplex.h
Synopsis       int CPXPUBLIC CPXcopyqpsep(CPXCENVptr env, 
               CPXLPptr lp, 
               const double * qsepvec)
Description    The routine CPXcopyqpsep is used to copy the quadratic objective matrix Q for a separable QP problem. A separable QP problem is one where the coefficients of Q have no nonzero off-diagonal elements.

\[\textbf{Note:} \ CPLEX \text{ evaluates the corresponding objective with a factor of } 0.5 \text{ in front of the quadratic objective term.}\]

Example        status = CPXcopyqpsep (env, lp, qsepvec);

Parameters     env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.
qsepvec
An array of length CPXgetnumcols(env, lp).qsepvec[0], qsepvec[1],..., qsepvec[numcols-1] should contain the quadratic coefficients of the separable quadratic objective.

Returns        The routine returns zero on success and nonzero if an error occurs.
CPXcopyquad

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXcopyquad(CPXCENVptr env,
CPXLPptr lp,
const int * qmatbeg,
const int * qmatcnt,
const int * qmatind,
const double * qmatval)
```

**Description**  
The routine CPXcopyquad is used to copy quadratic objective matrix Q when Q is not diagonal. The arguments qmatbeg, qmatcnt, qmatind, and qmatval are used to specify the nonzero coefficients of the matrix Q. The meaning of these vectors is identical to the meaning of the corresponding vectors matbeg, matcnt, matind and matval, which are used to specify the structure of A in a call to CPXcopylp.

Q must be symmetric when copied by this function. Therefore, if the quadratic coefficient in algebraic form is $2x_1x_2$, then $x_2$ should be in the list for $x_1$, and $x_1$ should be in the list for $x_2$, and the coefficient would be 1.0 in each of those entries. See the corresponding example C program to review how the symmetry requirement is implemented.

**Note:** CPLEX evaluates the corresponding objective with a factor of 0.5 in front of the quadratic objective term.

**How the arrays are accessed**

Suppose that CPLEX wants to access the entries in a column $j$. These are assumed to be given by the array entries:

```
qmatval[qmatbeg[j]],...,qmatval[qmatbeg[j]+qmatcnt[j]-1]
```

The corresponding column/index entries are:

```
qmatind[qmatbeg[j]],...,qmatind[qmatbeg[j]+qmatcnt[j]-1]
```
The entries in \texttt{qmatind[k]} are not required to be in column order. Duplicate entries in \texttt{qmatind} within a single column are not allowed. Note that any column \textit{j} that has only a linear objective term has \texttt{qmatcnt[j]} = 0 and no entries in \texttt{qmatind} and \texttt{qmatval}.

**Example**

```cpp
status = CPXcopyquad (env, lp, qmatbeg, qmatcnt, qmatind, qmatval);
```

**See Also**

CPXreadcopyqp

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **lp**
  A pointer to a CPLEX problem object as returned by CPXcreateprob.
- **qmatbeg**
  An array that with \texttt{qmatcnt}, \texttt{qmatind}, and \texttt{qmatval} defines the quadratic coefficient matrix.
- **qmatcnt**
  An array that with \texttt{qmatbeg}, \texttt{qmatind}, and \texttt{qmatval} defines the quadratic coefficient matrix.
- **qmatind**
  An array that with \texttt{qmatbeg}, \texttt{qmatcnt}, and \texttt{qmatval} defines the quadratic coefficient matrix.
- **qmatval**
  An array that with \texttt{qmatbeg}, \texttt{qmatcnt}, and \texttt{qmatind} defines the quadratic coefficient matrix. The arrays \texttt{qmatbeg} and \texttt{qmatval} should be of length at least \texttt{CPXgetnumcols(env,lp)}. The arrays \texttt{qmatind} and \texttt{qmatval} should be of length at least \texttt{qmatbeg[numcols-1]+qmatcnt[numcols-1]}. CPLEX requires only the nonzero coefficients grouped by column in the array \texttt{qmatval}. The nonzero elements of every column must be stored in sequential locations in this array with \texttt{qmatbeg[j]} containing the index of the beginning of column \textit{j} and \texttt{qmatcnt[j]} containing the number of entries in column \textit{j}. Note that the components of \texttt{qmatbeg} must be in ascending order. For each \textit{k}, \texttt{qmatind[k]} indicates the column number of the corresponding coefficient, \texttt{qmatval[k]}. These arrays are accessed as explained above.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXcopystart

Category                Global Function
Definition File         cplex.h
Include Files           cplex.h
Synopsis

int CPXPUBLIC CPXcopystart(CPXENVptr env,
                           CPXLPptr lp,
                           const int * cstat,
                           const int * rstat,
                           const double * cprim,
                           const double * rprim,
                           const double * cdual,
                           const double * rdual)

Description

The routine CPXcopystart is used to provide starting information for use in a subsequent call to a simplex optimization routine (CPXlpopt with CPX_PARAM_LPMETHOD or CPX_PARAM_QPMETHOD set to CPX_ALG_PRIMAL or CPX_ALG_DUAL, CPXdualopt, CPXprimopt, or CPXhybnetopt). Starting information is not applicable to the barrier optimizer or the mixed integer optimizer.

When a basis (parameters cstat and rstat) is installed for a linear problem and CPXlpopt is used with CPX_PARAM_LPMETHOD set to CPX_ALG_AUTOMATIC, CPLEX will use the primal simplex algorithm if the basis is primal feasible and the dual simplex method otherwise.

Any of three different kinds of starting points can be provided: a starting basis (cstat, rstat), starting primal values (cprim, rprim), and starting dual values (cdual, rdual). Only a starting basis is applicable to a CPXhybnetopt call, but for Dual Simplex and Primal Simplex any combination of these three types of information can be used in providing a starting point. If no starting-point is provided, this routine returns an error; otherwise, any resident starting information in the CPLEX problem object is freed and the new information is copied into it.

If you provide a starting basis, then both cstat and rstat must be specified. It is permissible to provide cprim with or without rprim, or rdual with or without cdual; arrays not being provided must be passed as NULL pointers.

Note: The starting information is ignored by the optimizers if the parameter CPX_PARAM_ADVIND is off.
Example

```c
status = CPXcopystart (env,
                   lp,
                   cstat,
                   rstat,
                   cprim,
                   rprim,
                   cdual,
                   rdual);
```

Parameters

- `env`
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- `lp`
  A pointer to a CPLEX problem object as returned by CPXcreateprob.
- `cstat`
An array containing the basis status of the columns in the constraint matrix. The length of the array is equal to the number of columns in the CPLEX problem object. If this array is NULL, rstat must be NULL. Table 1 shows the possible values.

rstat

An array containing the basis status of the slack/surplus/artificial variable associated with each row in the constraint matrix. The array's length is equal to the number of rows in the LP problem. For rows other than ranged rows, the array element rstat[i] can be set according to Table 2. For ranged rows, the array element rstat[i] can be set according to Table 3. If this array is NULL, cstat must be NULL.

cprim

An array containing the initial primal values of the column variables. The length of the array must be no less than the number of columns in the CPLEX problem object. If this array is NULL, rprim must be NULL.

rprim

An array containing the initial primal values of the slack (row) variables. The length of the array must be no less than the number of rows in the CPLEX problem object. This array may be NULL.

cdual

An array containing the initial values of the reduced costs for the column variables. The length of the array must be no less than the number of columns in the CPLEX problem object. This array may be NULL.

rdual

An array containing the initial values of the dual variables for the rows. The length of the array must be no less than the number of rows in the CPLEX problem object. If this array is NULL, cdual must be NULL.

Returns

The routine returns zero on success and nonzero if an error occurs.
**CPXcreateprob**

**Category** Global Function

**Definition File** cplex.h

**Include Files** cplex.h

**Synopsis**

```c
CPXLPptr CPXPUBLIC CPXcreateprob(CPXENVptr env,
     int * status_p,
     const char * probname_str)
```

**Description**

The routine CPXcreateprob creates a CPLEX problem object in the CPLEX environment. The arguments to CPXcreateprob define an LP problem name. The problem that is created is an LP minimization problem with zero constraints, zero variables, and an empty constraint matrix. The CPLEX problem object exists until the routine CPXfreeprob is called.

To define the constraints, variables, and nonzero entries of the constraint matrix, any of the CPLEX LP problem modification routines may be used. In addition, any of the routines beginning with the prefix CPXcopy may be used to copy data into the CPLEX problem object. New constraints or new variables can be created with the routines CPXnewrows or CPXnewcols, respectively.

**Example**

```c
lp = CPXcreateprob (env, &status, "myprob");
```

See also all the Callable Library examples (except those pertaining to networks) in the *ILOG CPLEX User’s Manual*.

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **status_p**
  A pointer to an integer used to return any error code produced by this routine.
- **probname_str**
  A character string that specifies the name of the problem being created.

**Returns**

If successful, CPXcreateprob returns a pointer that can be passed to other CPLEX routines to identify the problem object that is created. If not successful, a NULL pointer
is returned, and an error status is returned in the variable *status_p. If the routine is successful, *status_p is 0 (zero).
CPXfreeprob

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis: int CPXPUBLIC CPXfreeprob(CPXENVptr env, CPXLPptr * lp_p)

Description: The routine CPXfreeprob removes the specified CPLEX problem object from the CPLEX environment and frees the associated memory used internally by CPLEX. The routine is used when the user has no need for further access to the specified problem data.

Example:

status = CPXfreeprob (env, &lp);

See also the example lpex1.c in the ILOG CPLEX User's Manual and in the standard distribution.

Parameters:

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp_p
A pointer to a CPLEX problem object as returned by CPXcreateprob.

Example:

status = CPXfreeprob (env, &lp);

See also the example lpex1.c in the ILOG CPLEX User's Manual and in the standard distribution.

Returns: The routine returns zero on success and nonzero if an error occurs.
CPXreadcopy sos

Category     Global Function
Definition File cplex.h
Include Files cplex.h

Synopsis     int CPXPUBLIC CPXreadcopy sos (CPXENVptr env,
               CPXLPtr lp,
               const char * filename_str)

Description The routine CPXreadcopy sos reads in SOS data from an SOS file, and copies the data into a MIP problem object.
This routine changes the problem type to MIP, if it is not already a MIP.

Example

    status = CPXreadcopy sos (env, lp, "myprob.sos");

Parameters

    env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

    lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

    filename_str
The name of the file from which the SOS information should be read.

Returns

The routine returns zero on success and nonzero if an error occurs.

Group optim.cplex.callable.optimizers

The routines in the ILOG CPLEX Callable Library to launch an optimizer.

<table>
<thead>
<tr>
<th>Global Functions Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXbaropt</td>
</tr>
<tr>
<td>CPXdualopt</td>
</tr>
<tr>
<td>CPXhybbaropt</td>
</tr>
<tr>
<td>CPXhybnetopt</td>
</tr>
<tr>
<td>Description</td>
</tr>
<tr>
<td>Continuous LP and QP problem objects can be optimized with simplex or barrier optimizers. Continuous QCP problem objects can be optimized with the barrier optimizer only.</td>
</tr>
<tr>
<td>For MIP problem objects, any appropriate continuous optimizer may be specified to solve the subproblems. You can also specify a different optimizer for solving the root LP subproblem and for the LP subproblems that occur at the nodes of the branch and cut tree.</td>
</tr>
<tr>
<td>For more information about parameters and their settings, see the reference manual ILOG CPLEX Parameters.</td>
</tr>
</tbody>
</table>
CPXNETprimopt

Category Global Function

Definition File cplex.h

Include Files cplex.h

Synopsis int CPXPUBLIC CPXNETprimopt(CPXENVptr env, CPXNETptr net)

Description The routine CPXNETprimopt can be called after a network problem has been copied to a network problem object, to find a solution to that problem using the primal network simplex method. When this function is called, the CPLEX primal network algorithm attempts to optimize the problem. The results of the optimization are recorded in the problem object and can be retrieved by calling the appropriate solution functions for that object.

Example

```
status = CPXNETprimopt (env, net);
```

See also the examples netex1.c and netex2.c in the standard distribution of the product.

Parameters

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCplex.

- **net**
  A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

Returns

The routine returns zero unless an error occurred during the optimization. Examples of errors include exhausting available memory (CPXERR_NO_MEMORY) or encountering invalid data in the CPLEX problem object (CPXERR_NO_PROBLEM). Exceeding a user-specified CPLEX limit, or proving the model infeasible or unbounded, are not considered errors. Note that a zero return value does not necessarily mean that a solution exists. Use query routines CPXNETsolninfo, CPXNETgetstat, and CPXNETsolution to obtain further information about the status of the optimization.
CPXbaropt

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXbaropt (CPXCENVptr env, CPXLPtr lp)
```

Description:

The routine CPXbaropt may be used to find a solution to a linear program (LP), quadratic program (QP), or quadratically constrained program (QCP) using the barrier algorithm, at any time after the problem is created via a call to CPXcreateprob. The optimization results are recorded in the CPLEX problem object.

Example:

```
status = CPXbaropt (env, lp);
```

Parameters:

- `env`: A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- `lp`: A pointer to a CPLEX problem object as returned by CPXcreateprob.

Returns:

The routine returns zero unless an error occurred during the optimization. Examples of errors include exhausting available memory (CPXERR_NO_MEMORY) or encountering invalid data in the CPLEX problem object (CPXERR_NO_PROBLEM). Exceeding a user-specified CPLEX limit, or proving the model infeasible or unbounded, are not considered errors. Note that a zero return value does not necessarily mean that a solution exists. Use query routines CPXsolninfo, CPXgetstat, and CPXsolution to obtain further information about the status of the optimization.
CPXdualopt

Category                Global Function
Definition File         cplex.h
Include Files           cplex.h
Synopsis                int CPXPUBLIC CPXdualopt (CPXCENVptr env, CPXLPptr lp)
Description             The routine CPXdualopt may be used at any time after a linear program has been created via a call to CPXcreateprob to find a solution to that problem using the dual simplex algorithm. When this function is called, the CPLEX dual simplex optimization routines attempt to optimize the specified problem. The results of the optimization are recorded in the CPLEX problem object.
Example                 status = CPXdualopt (env, lp);

Parameters             env
                      A pointer to the CPLEX environment as returned by CPXopenCPLEX.
                      lp
                      A pointer to a CPLEX problem object as returned by CPXcreateprob.
Returns                The routine returns zero unless an error occurred during the optimization. Examples of errors include exhausting available memory (CPXERR_NO_MEMORY) or encountering invalid data in the CPLEX problem object (CPXERR_NO_PROBLEM). Exceeding a user-specified CPLEX limit, or proving the model infeasible or unbounded, are not considered errors. Note that a zero return value does not necessarily mean that a solution exists. Use query routines CPXsolninfo, CPXgetstat, and CPXsolution to obtain further information about the status of the optimization.
CPXhybbaropt

Category Global Function

Definition File cplex.h

Include Files cplex.h

Synopsis int CPXPUBLIC CPXhybbaropt (CPXCENVptr env, CPXLPptr lp, int method)

Description The routine CPXhybbaropt may be used, at any time after a linear program has been created via a call to CPXcreateprob, to find a solution to that problem. When this function is called, the specified problem is solved using CPLEX Barrier followed by an automatic crossover to a basic solution if barrier determines that the problem is both primal and dual feasible. Otherwise, crossover is not performed. In this case, a call to CPXprimopt or CPXdualopt can force a crossover to occur. The results of the optimization are recorded in the problem object.

Methods of CPXhybbaropt

<table>
<thead>
<tr>
<th>method</th>
<th>use</th>
<th>Parameters</th>
</tr>
</thead>
</table>
| method = 0        | CPX_PARAM_BARCROSSALG to choose a crossover method | env
|                   |                                  | env                                             |
| method = CPX_ALG_PRIMAL | primal crossover                 | lp                                              |
| method = CPX_ALG_DUAL   | dual crossover                   | method                                          |
| method = CPX_ALG_NONE  | no crossover                     | Crossover method to be implemented, according to the table. |

Example

status = CPXhybbaropt (env, lp, CPX_ALG_PRIMAL);

See also the example lplex2.c in the standard distribution.

Parameters

env

A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp

A pointer to a CPLEX problem object as returned by CPXcreateprob.

method

Crossover method to be implemented, according to the table.
Returns

The routine returns zero unless an error occurred during the optimization. Examples of errors include exhausting available memory (CPXERR_NO_MEMORY) or encountering invalid data in the CPLEX problem object (CPXERR_NO_PROBLEM). Exceeding a user-specified CPLEX limit, or proving the model infeasible or unbounded, are not considered errors. Note that a zero return value does not necessarily mean that a solution exists. Use query routines CPXsolninfo, CPXgetstat, and CPXsolution to obtain further information about the status of the optimization.
CPXhybnetopt

Category                  Global Function
Definition File           cplex.h
Include Files            cplex.h
Synopsis                  int CPXPUBLIC CPXhybnetopt (CPXCENVptr env, 
                          CPXLPptr lp, 
                          int method)

Description               The routine CPXhybnetopt, given a linear program that has been created via a call to 
                          CPXcreateprob, extracts an embedded network, uses the CPLEX Network 
                          Optimizer to attempt to obtain an optimal basis to the network, and optimizes the entire 
                          linear program using one of the CPLEX simplex methods. CPLEX takes the network 
                          basis as input for the optimization of the whole linear program.

<table>
<thead>
<tr>
<th>method</th>
<th>= CPX_ALG_PRIMAL</th>
<th>primal Simplex</th>
</tr>
</thead>
<tbody>
<tr>
<td>method</td>
<td>= CPX_ALG_DUAL</td>
<td>dual Simplex</td>
</tr>
</tbody>
</table>

Example

    status = CPXhybnetopt (env, lp, CPX_ALG_DUAL);

See also the example lpex3.c in the ILOG CPLEX User's Manual and in the standard distribution.

Parameters

    env
    A pointer to the CPLEX environment as returned by CPXopenCPLEX.

    lp
    A pointer to a CPLEX problem object as returned by CPXcreateprob.

    method
    The type of simplex method to follow the network optimization.
Example

\[
\text{status} = \text{CPXhybnetopt (env, lp, CPX\_ALG\_DUAL)}; 
\]

See also the example \texttt{lpex3.c} in the \textit{ILOG CPLEX User's Manual} and in the standard distribution.

Returns

The routine returns zero unless an error occurred during the optimization. Examples of errors include exhausting available memory (CPXERR\_NO\_MEMORY) or encountering invalid data in the CPLEX problem object (CPXERR\_NO\_PROBLEM). Exceeding a user-specified CPLEX limit, or proving the model infeasible or unbounded, are not considered errors. Note that a zero return value does not necessarily mean that a solution exists. Use query routines \texttt{CPXsolninfo}, \texttt{CPXgetstat}, and \texttt{CPXsolution} to obtain further information about the status of the optimization.
**CPXlpopt**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXlpopt (CPXCENVptr env, 
CPXLPptr lp)
```

**Description**  
The routine CPXlpopt may be used, at any time after a linear program has been created via a call to CPXcreateprob, to find a solution to that problem using one of the ILOG CPLEX linear optimizers. The parameter CPX_PARAM_LPMETHOD controls the choice of optimizer (dual simplex, primal simplex, barrier, network simplex, sifting, or concurrent optimization). Currently, with the default parameter setting of Automatic, CPLEX invokes the dual simplex method when no advanced basis or starting vector is loaded or when the advanced indicator is zero. The behavior of the Automatic setting may change in the future.

**Example**
```c
status = CPXlpopt (env, lp);
```

See also the example lpex1.c in *Getting Started* and in the standard distribution.

**See Also**  
CPXgetstat, CPXsolninfo, CPXsolution

**Parameters**

- **env**  
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **lp**  
A pointer to the CPLEX problem object as returned by CPXcreateprob.

**Returns**

The routine returns zero unless an error occurred during the optimization. Examples of errors include exhausting available memory (CPXERR_NO_MEMORY) or encountering invalid data in the CPLEX problem object (CPXERR_NO_PROBLEM). Exceeding a user-specified CPLEX limit, or proving the model infeasible or unbounded, are not considered errors. Note that a zero return value does not necessarily mean that a solution exists. Use query routines CPXsolninfo, CPXgetstat, and CPXsolution to obtain further information about the status of the optimization.
CPXmipopt

Category          Global Function
Definition File   cplex.h
Synopsis          int CPXPUBLIC CPXmipopt(CPXENVptr env, CPXLPptr lp)
Description       The routine CPXmipopt may be used, at any time after a mixed integer program has been created by a call to CPXcreateprob, to find a solution to that problem.
CPXmipopt returns zero unless it encounters an error. Nonzero values are error codes indicating which type of failure occurred. Use the routine CPXgetstat to determine the status of the mixed integer optimization. One possible error is the inability to solve a subproblem satisfactorily, as indicated by CPXERR_SUBPROB_SOLVE. The solution status of the subproblem optimization can be obtained with the routine CPXgetsubstat.
An LP solution does not exist at the end of CPXmipopt, so post-solution information is available only through the special mixed integer solution routines. To obtain post-solution information for the LP subproblem associated with the integer solution, use the routine CPXchgsprobytype.

Example

    status = CPXmipopt (env, lp);

See also the example mipex1.c in the standard distribution.

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

ip
A pointer to a CPLEX problem object as returned by CPXcreateprob.

Returns

This routine returns zero unless it encounters an error.
CPXprimopt

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis: int CPXPUBLIC CPXprimopt (CPXCENVptr env, CPXLPptr lp)

Description: The routine CPXprimopt may be used after a linear program has been created via a call to CPXcreateprob, to find a solution to that problem using the primal simplex method. When this function is called, the CPLEX primal simplex algorithm attempts to optimize the specified problem. The results of the optimization are recorded in the CPLEX problem object.

Example:

status = CPXprimopt (env, lp);

Parameters:

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

Returns:

The routine returns zero unless an error occurred during the optimization. Examples of errors include exhausting available memory (CPXERR_NO_MEMORY) or encountering invalid data in the CPLEX problem object (CPXERR_NO_PROBLEM). Exceeding a user-specified CPLEX limit or proving the model infeasible or unbounded are not considered errors. Note that a zero return value does not necessarily mean that a solution exists. Use query routines CPXsolninfo, CPXgetstat, and CPXsolution to obtain further information about the status of the optimization.
CPXqpopt

Category          Global Function
Definition File   cplex.h
Include Files     cplex.h
Synopsis          int CPXPUBLIC CPXqpopt(CPXENVptr env,
                   CPXLPptr lp)
Description       The routine CPXqpopt may be used, at any time after a continuous quadratic program has been created, to find a solution to that problem using one of CPLEX's quadratic optimizers. The parameter CPX_PARAM_QPMETHOD controls the choice of optimizer (Dual Simplex, Primal Simplex, or Barrier). With the default setting of this parameter (that is, Automatic) CPLEX invokes the barrier method because it is fastest on a wide range of problems.
Example           status = CPXqpopt (env, lp);
See Also          CPXgetmethod
Parameters
                   env
                   A pointer to the CPLEX environment as returned by CPXopenCPLEX.
                   lp
                   A pointer to the CPLEX problem object as returned by CPXcreateprob.
Returns           The routine returns zero unless an error occurred during the optimization. Examples of errors include exhausting available memory (CPXERR_NO_MEMORY) or encountering invalid data in the CPLEX problem object (CPXERR_NO_PROBLEM). Exceeding a user-specified CPLEX limit, or proving the model infeasible or unbounded, are not considered errors. Note that a zero return value does not necessarily mean that a solution exists. Use query routines CPXsolninfo, CPXgetstat, and CPXsolution to obtain further information about the status of the optimization.

Group optim.cplex.callable.accesslpormqpresults

The routines in the ILOG CPLEX Callable Library to access LP or QP results.
Description

Solution query routines are used to access information about the results of applying an optimization method to a problem object. For LP and QP problem objects, you can access the values of variables, constraint slacks, reduced costs, and dual variables. Additionally, for an LP or QP problem object solved with a simplex method, you can query the simplex basis. Methods and routines are also available to retrieve other information about the optimization process (such as the iteration count).

<table>
<thead>
<tr>
<th>Global Functions Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXgetax</td>
</tr>
<tr>
<td>CPXgetbaritcnt</td>
</tr>
<tr>
<td>CPXgetbase</td>
</tr>
<tr>
<td>CPXgetcrossdexchcnt</td>
</tr>
<tr>
<td>CPXgetcrossdpushcnt</td>
</tr>
<tr>
<td>CPXgetcrossppushcnt</td>
</tr>
<tr>
<td>CPXgetdj</td>
</tr>
<tr>
<td>CPXgetdsbcnt</td>
</tr>
<tr>
<td>CPXgetitcnt</td>
</tr>
<tr>
<td>CPXgetmethod</td>
</tr>
<tr>
<td>CPXgetobjval</td>
</tr>
<tr>
<td>CPXgetphaselcnt</td>
</tr>
<tr>
<td>CPXgetpi</td>
</tr>
<tr>
<td>CPXgetpsbcnt</td>
</tr>
<tr>
<td>CPXgetsiftitcnt</td>
</tr>
<tr>
<td>CPXgetsiftphaselcnt</td>
</tr>
<tr>
<td>CPXgetslack</td>
</tr>
<tr>
<td>CPXgetstat</td>
</tr>
<tr>
<td>CPXgetstatstring</td>
</tr>
<tr>
<td>CPXgetx</td>
</tr>
<tr>
<td>CPXsolinfo</td>
</tr>
<tr>
<td>CPXsolution</td>
</tr>
</tbody>
</table>
CPXgetax

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXgetax(CPXENVptr env,
                        CPXCLPptr lp,
                        double * x,
                        int begin,
                        int end)
```

**Description**
The routine CPXgetax is used to access row activity levels for a range of constraints of a linear or quadratic program. The beginning and end of the range must be specified. A row activity is the inner product of a row in the constraint matrix and the structural variables in the problem.

The array must be of length at least (end-begin+1). If successful, x[0] through x[end-begin] contain the row activities.

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **lp**
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

- **x**
  An array to receive the values of the row activity levels for each of the constraints in the specified range.

  The array must be of length at least (end-begin+1). If successful, x[0] through x[end-begin] contain the row activities.

- **begin**
  An integer indicating the beginning of the range of row activities to be returned.

- **end**
  An integer indicating the end of the range of row activities to be returned.

**Returns**
The routine returns zero on success and nonzero if an error occurs.
CPXgetbaritcnt

Category                   Global Function
Definition File           cplex.h
Include Files             cplex.h
Synopsis                   int CPXPUBLIC CPXgetbaritcnt(CPXENVptr env, CPXCLPptr lp)
Description                The routine CPXgetbaritcnt is used to access the total number of Barrier iterations to solve an LP problem.
                           
                           Example
                           itcnt = CPXgetbaritcnt (env, lp);

Parameters

   env
   A pointer to the CPLEX environment as returned by CPXopenCPLEX.

   lp
   A pointer to a CPLEX problem object as returned by CPXcreateprob.

                           Example
                           itcnt = CPXgetbaritcnt (env, lp);

Returns                    The routine returns the total iteration count if a solution exists. It returns zero if no solution exists or any other type of error occurs.
**CPXgetbase**

**Category**  Global Function  
**Definition File**  cplex.h  
**Include Files**  cplex.h  
**Synopsis**  
```c  
int CPXPUBLIC CPXgetbase(CPXCENVptr env,  
                    CPXCLPptr lp,  
                    int * cstat,  
                    int * rstat)  
```

**Description**  The routine **CPXgetbase** is used to get the basis resident in a CPLEX problem object. Either of the arguments **cstat** or **rstat** may be NULL, if only one set of statuses is needed.

<table>
<thead>
<tr>
<th>Table 1: Values of elements of cstat</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_AT_LOWER</td>
</tr>
<tr>
<td>CPX_BASIC</td>
</tr>
<tr>
<td>CPX_AT_UPPER</td>
</tr>
<tr>
<td>CPX_FREE_SUPER</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2: Values of elements of rstat in rows other than ranged rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_AT_LOWER</td>
</tr>
<tr>
<td>CPX_BASIC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 3: Values of elements of rstat for ranged rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_AT_LOWER</td>
</tr>
<tr>
<td>CPX_BASIC</td>
</tr>
</tbody>
</table>
Table 3: Values of elements of rstat for ranged rows

| CPX_AT_UPPER | 2 | associated slack/surplus/artificial variable nonbasic at upper bound |

Example

```c
status = CPXgetbase (env, lp, cstat, rstat);
```

See also the example `lpex6.c` in the *ILOG CPLEX User’s Manual* and in the standard distribution.

Parameters

- **env**
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- **lp**
  A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

- **cstat**
  An array to receive the basis status of the columns in the CPLEX problem object. The length of the array must be no less than the number of columns in the matrix. The array element `cstat[i]` has the meaning indicated in Table 1.

- **rstat**
  An array to receive the basis status of the artificial/slack/surplus variable associated with each row in the constraint matrix. The array’s length must be no less than the number of rows in the CPLEX problem object. For rows other than ranged rows, the array element `rstat[i]` has the meaning indicated in Table 2. For ranged rows, the array element `rstat[i]` has the meaning indicated in Table 3.

Returns

The routine returns zero if a basis exists. It returns zero if no solution exists or any other type of error occurs.
**CPXgetcrossdexchcnt**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXgetcrossdexchcnt (CPXCENVptr env, CPXCLPptr lp)
```

**Description**  
The routine `CPXgetcrossdexchcnt` is used to access the number of dual exchange iterations in the crossover method. An exchange occurs when a nonbasic variable is forced to enter the basis as it is pushed toward a bound.

**Example**  
```c
itcnt = CPXgetcrossdexchcnt (env, lp);
```

**Parameters**  
- **env**  
A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
- **lp**  
A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

**Example**  
```c
itcnt = CPXgetcrossdexchcnt (env, lp);
```

**Returns**  
The routine returns the dual exchange iteration count if a solution exists. If no solution exists, it returns zero.
CPXgetcrossdpushcnt

Category            Global Function
Definition File     cplex.h
Include Files       cplex.h
Synopsis            int CPXPUBLIC CPXgetcrossdpushcnt(CPXENVptr env, CPXCLPtr lp)
Description          The routine CPXgetcrossdpushcnt is used to access the number of dual push iterations in the crossover method. A push occurs when a nonbasic variable switches bounds and does not enter the basis.
Example              itcnt = CPXgetcrossdpushcnt (env, lp);

Parameters
env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.
Example              itcnt = CPXgetcrossdpushcnt (env, lp);

Returns             The routine returns the dual push iteration count if a solution exists. If no solution exists, it returns zero.
CPXgetcrossppushcnt

Category: Global Function
Definition File: cplex.h
Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXgetcrossppushcnt(CPXENVptr env, CPXCLPptr lp)
```

Description:
The routine `CPXgetcrossppushcnt` is used to access the number of primal push iterations in the crossover method. A push occurs when a nonbasic variable switches bounds and does not enter the basis.

Example:

```c
itcnt = CPXgetcrossppushcnt (env, lp);
```

Parameters:
- **env**: A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
- **lp**: A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Example:

```c
itcnt = CPXgetcrossppushcnt (env, lp);
```

Returns:
The routine returns the primal push iteration count if a solution exists. If no solution exists, it returns zero.
CPXgetdj

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:
```
int CPXPUBLIC CPXgetdj(CPXENVptr env,
                   CPXCLPptr lp,
                   double * dj,
                   int begin,
                   int end)
```

Description:
The routine CPXgetdj is used to access the reduced costs for a range of the variables of a linear or quadratic program. The beginning and end of the range must be specified.

Example:
```
status = CPXgetdj (env, lp, dj, 0, CPXgetnumcols(env,lp)-1);
```

Parameters:
- env: A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- lp: A pointer to a CPLEX problem object as returned by CPXcreateprob.
- dj: An array to receive the values of the reduced costs for each of the variables. This array must be of length at least (end - begin + 1). If successful, dj[0] through dj[end-begin] contain the values of the reduced costs.
- begin: An integer indicating the beginning of the range of reduced-cost values to be returned.
- end: An integer indicating the end of the range of reduced-costs values to be returned.

Example:
```
status = CPXgetdj (env, lp, dj, 0, CPXgetnumcols(env,lp)-1);
```

Returns:
The routine returns zero on success and nonzero if an error occurs.
CPXgetdsbcnt

Category: Global Function
Definition File: cplex.h
Synopsis:
```
int CPXPUBLIC CPXgetdsbcnt (CPXCENVptr env, CPXCLPptr lp)
```
Description:
The routine CPXgetdsbcnt is used to access the number of dual super-basic variables in the current solution.

Example:
```
itcnt = CPXgetdsbcnt (env, lp);
```
Parameters:
- `env`:
  A pointer to the CPLEX environment as returned by CPXopenCplex.
- `lp`:
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

Example:
```
itcnt = CPXgetdsbcnt (env, lp);
```
Returns:
If a solution exists, CPXgetdsbcnt returns the number of dual super-basic variables. If no solution exists, CPXgetdsbcnt returns the value 0 (zero).


CPXgetitcnt

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis: int CPXPUBLIC CPXgetitcnt(CPXENVptr env, CPXCLPptr lp)

Description: The routine CPXgetitcnt is used to access the total number of simplex iterations to solve an LP problem, or the number of crossover iterations in the case that Barrier is used.

Example:

itcnt = CPXgetitcnt (env, lp);

Parameters:

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

Example:

itcnt = CPXgetitcnt (env, lp);

Returns: If a solution exists, CPXgetitcnt returns the total iteration count. If no solution exists, CPXgetitcnt returns the value 0.

CPXgetmethod

Category       Global Function

Definition File cplex.h

Include Files  cplex.h

Synopsis       int CPXPUBLIC CPXgetmethod(CPXENVptr env, 
               CPXCLPptr lp)

Description    The routine CPXgetmethod returns an integer indicating the solution algorithm used 
                to solve the resident LP or QP problem.

Example        method = CPXgetmethod (env, lp);

Parameters     env

                   A pointer to the CPLEX environment as returned by CPXopenCPLEX.

                   lp

                   A pointer to a CPLEX problem object as returned by CPXcreateprob.

Returns        The possible return values are summarized in the table.

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CPX_ALG_NONE</td>
<td>None</td>
</tr>
<tr>
<td>1</td>
<td>CPX_ALG_PRIMAL</td>
<td>Primal simplex</td>
</tr>
<tr>
<td>2</td>
<td>CPX_ALG_DUAL</td>
<td>Dual simplex</td>
</tr>
<tr>
<td>4</td>
<td>CPX_ALG_BARRIER</td>
<td>Barrier optimizer (no crossover)</td>
</tr>
</tbody>
</table>
**CPXgetobjval**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**
```c
int CPXPUBLIC CPXgetobjval(CPXENVptr env,
    CPXCLPptr lp,
    double * objval_p)
```

**Description**
The routine `CPXgetobjval` is used to return the LP, QCP or QP solution objective value.

**Example**
```c
    status = CPXgetobjval (env, lp, &objval);
```

See also the example `lpex2.c` in the *ILOG CPLEX User's Manual* and in the standard distribution.

**Parameters**
- **env**
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
- **lp**
  A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
- **objval_p**
  A pointer to a variable of type double where the objective value is stored.

**Example**
```c
    status = CPXgetobjval (env, lp, &objval);
```

See also the example `lpex2.c` in the *ILOG CPLEX User's Manual* and in the standard distribution.

**Returns**
The routine returns zero on success and nonzero if no solution exists.
CPXgetphase1cnt

Category Global Function

Definition File cplex.h

Include Files cplex.h

Synopsis

```c
int CPXPUBLIC CPXgetphase1cnt(CPXENVptr env, CPXCLPptr lp)
```

Description The routine CPXgetphase1cnt is used to access the number of Phase I iterations to solve a problem using the primal or dual simplex method.

Example

```c
itcnt = CPXgetphase1cnt (env, lp);
```

Parameters

e
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

Example

```c
itcnt = CPXgetphase1cnt (env, lp);
```

Returns If a solution exists, CPXgetphase1cnt returns the Phase I iteration count. If no solution exists, CPXgetphase1cnt returns the value 0.
CPXgetpi

Category  Global Function
Definition File  cplex.h
Include Files  cplex.h
Synopsis  
  int CPXPUBLIC CPXgetpi(CPXENVptr env,
                         CPXCLPptr lp,
                         double * pi,
                         int begin,
                         int end)
Description  The routine CPXgetpi is used to access the dual values for a range of the constraints of a linear or quadratic program. The beginning and end of the range must be specified.
Example  
  status = CPXgetpi (env, lp, pi, 0, CPXgetnumrows(env,lp)-1);
Parameters  
  env  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
  lp  A pointer to a CPLEX problem object as returned by CPXcreateprob.
  pi  An array to receive the values of the dual variables for each of the constraints. This array must be of length at least (end - begin + 1). If successful, pi[0] through pi[end - begin] contain the dual values.
  begin  An integer indicating the beginning of the range of dual values to be returned.
  end  An integer indicating the end of the range of dual values to be returned.
Example  
  status = CPXgetpi (env, lp, pi, 0, CPXgetnumrows(env,lp)-1);
Returns  The routine returns zero on success and nonzero if an error occurs.
CPXgetpsbcnt

Category  Global Function
Definition File  cplex.h
Synopsis  int CPXPUBLIC CPXgetpsbcnt(CPXENVptr env, CPXCLPptr lp)
Description  The routine CPXgetpsbcnt is used to access the number of primal super-basic variables in the current solution.
Example  
\[ \textit{itcnt} = \text{CPXgetpsbcnt}(\text{env}, \text{lp}); \]
Parameters  
- \textit{env}
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- \textit{lp}
  A pointer to a CPLEX problem object as returned by CPXcreateprob.
Example  
\[ \textit{itcnt} = \text{CPXgetpsbcnt}(\text{env}, \text{lp}); \]
Returns  If a solution exists, CPXgetpsbcnt returns the number of primal super-basic variables. If no solution exists, CPXgetpsbcnt returns the value 0 (zero).
CPXgetsiftitcnt

Category       Global Function
Definition File cplex.h
Include Files  cplex.h

Synopsis       int CPXPUBLIC CPXgetsiftitcnt (CPXENVptr env, CPXCLPptr lp)

Description    The routine CPXgetsiftitcnt is used to access the total number of sifting iterations to solve an LP problem.

Example
   itcnt = CPXgetsiftitcnt (env, lp);

Parameters      env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
A pointer to a CPLEX LP problem object as returned by CPXcreateprob.

Example
   itcnt = CPXgetsiftitcnt (env, lp);

Returns         The routine returns the total iteration count if a solution exists. It returns zero if no solution exists or any other type of error occurs.
CPXgetsiftphase1cnt

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:
```c
int CPXPUBLIC CPXgetsiftphase1cnt(CPXENVptr env, CPXCLPptr lp)
```

Description:
The routine CPXgetsiftphase1cnt is used to access the number of Phase I sifting iterations to solve an LP problem.

Example:
```c
itcnt = CPXgetsiftphase1cnt (env, lp);
```

Parameters:

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **lp**
  A pointer to a CPLEX LP problem object as returned by CPXcreateprob.

Example:
```c
itcnt = CPXgetsiftphase1cnt (env, lp);
```

Returns:
The routine returns the Phase I iteration count if a solution exists. It returns zero if no solution exists or any other type of error occurs.
CPXgetslack

Category                         Global Function
Definition File                  cplex.h
Include Files                    cplex.h
Synopsis                         int CPXPUBLIC CPXgetslack(CPXENVptr env,
                              CPXCLPptr lp,
                              double * slack,
                              int begin,
                              int end)

Description
The routine CPXgetslack is used to access the slack values for a range of the constraints of a linear or quadratic program. The beginning and end of the range must be specified. Except for ranged rows, the slack values returned consist of the right-hand side minus the row activity level. For ranged rows, the value returned is the row activity level minus the right-hand side, or, equivalently, the value of the internal structural variable that CPLEX creates to represent ranged rows.

Example

    status = CPXgetslack (env, lp, slack, 0, CPXgetnumrows(env,lp)-1);

Parameters

    env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

    lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

    slack
An array to receive the values of the slack or surplus variables for each of the constraints. This array must be of length at least (end-begin+1). If successful, slack[0] through slack[end-begin] contain the values of the slacks.

    begin
An integer indicating the beginning of the range of slack values to be returned.

    end
An integer indicating the end of the range of slack values to be returned.

Example
status = CPXgetslack (env, lp, slack, 0, CPXgetnumrows(env,lp)-1);

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXgetstat

Category    Global Function
Definition File    cplex.h
Include Files    cplex.h
Synopsis    int CPXPUBLIC CPXgetstat(CPXENVptr env, CPXCLPptr lp)
Description    The routine CPXgetstat is used to access the solution status of the problem after an LP, QP, QCP, or mixed integer optimization.
Example    

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

Returns    The routine returns the solution status of the most recent optimization performed on the CPLEX problem object. Return values are as shown in group optim.cplex.solutionstatus. For status code CPX_STAT_NUM_BEST, the algorithm could not converge to the requested tolerances due to numeric difficulties. The best solution found can be retrieved by the routine CPXsolution. Similarly, when an abort status is returned, the last solution computed before the algorithm aborted can be retrieved using CPXsolution. Use the query routines CPXsolninfo and CPXsolution to obtain further information about the current solution of an LP or QP.
CPXgetstatstring

Category: Global Function
Definition File: cplex.h

Synopsis:
CPXCHARptr CPXPUBLIC CPXgetstatstring(CPXENVptr env,
int statind,
char * buffer_str)

Description:
The routine CPXgetstatstring is used to place in a buffer, a string corresponding
to the value of statind as returned by the CPXgetstat function. The buffer to hold the
string can be up to 510 characters maximum.

Example:
statind = CPXgetstat (env, lp);
p = CPXgetstatstring (env, statind, buffer);

Parameters:
- env
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- statind
  An integer indicating the status value to return.
- buffer_str
  A pointer to a buffer to hold the string corresponding to the value of statind.

Example:
statind = CPXgetstat (env, lp);
p = CPXgetstatstring (env, statind, buffer);

Returns:
The routine returns a pointer to a buffer if the statind value corresponds to a valid
string. Otherwise, it returns NULL.
**CPXgetx**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  

```c
int CPXPUBLIC CPXgetx(CPXENVptr env,  
    CPXCLPptr lp,  
    double * x,  
    int begin,  
    int end)
```

**Description**  
The routine CPXgetx is used to access the solution values for a range of problem variables of a linear, quadratically constrained, or quadratic program. The beginning and end of the range must be specified.

**Example**

```c
status = CPXgetx (env, lp, x, 0, CPXgetnumcols(env, lp)-1);
```

See also the example lpex2.c in the *ILOG CPLEX User’s Manual* and in the standard distribution.

**Parameters**  
- **env**  
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **lp**  
A pointer to a CPLEX problem object as returned by CPXcreateprob.
- **x**  
An array to receive the values of the primal variables for the problem. This array must be of length at least (end - begin + 1). If successful, x[0] through x[end-begin] contains the solution values.
- **begin**  
An integer indicating the beginning of the range of variable values to be returned.
- **end**  
An integer indicating the end of the range of variable values to be returned.

**Example**
status = CPXgetx (env, lp, x, 0, CPXgetnumcols(env, lp)-1);

See also the example lpect2.c in the *ILOG CPLEX User's Manual* and in the standard distribution.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
**CPXsolninfo**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**

```c
#include <cplex.h>

int CPXPUBLIC CPXsolninfo (CPXCENVptr env,
    CPXCLPptr lp,
    int * solnmethod_p,
    int * solntype_p,
    int * pfeasind_p,
    int * dfeasind_p);
```

**Description**
The routine **CPXsolninfo** is used to access solution information produced by the routines **CPXlpopt**, **CPXprimopt**, **CPXdualopt**, **CPXbaropt**, **CPXhybbaropt**, **CPXhybnetopt**, and **CPXqopt**. This information is maintained until the CPLEX problem object is freed via a call to **CPXfreeprob** or the solution is rendered invalid because of a call to one of the problem modification routines.

The arguments to **CPXsolninfo** are pointers to locations where data are to be written. Such data can include the optimization method used to produce the current solution, the type of solution available, and what is known about the primal and dual feasibility of the current solution. If any piece of information represented by an argument to **CPXsolninfo** is not required, a NULL pointer can be passed for that argument.

**Example**

```c
status = CPXsolninfo (env, lp, &solnmethod, &solntype, &pfeasind, &dfeasind);
```

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by **CPXopenCPLEX**.

- **lp**
  A pointer to a CPLEX problem object as returned by **CPXcreateprob**.

- **solnmethod_p**
  A pointer to an integer indicating the method used to produce the current solution. The specific values which **solnmethod_p** can take and their meanings are the same as the return values documented for **CPXgetmethod**.

- **solntype_p**
A pointer to an integer variable indicating the type of solution currently available. Possible return values are CPX_BASIC_SOLN, CPX_NONBASIC_SOLN, CPX_PRIMAL_SOLN, and CPX_NO_SOLN, meaning the model either has a simplex basis, has a primal and dual solution but no basis, has a primal solution but no corresponding dual solution, or has no solution, respectively.

pfeasind_p

A pointer to integer variables indicating whether the current solution is known to be primal feasible. Note that a false return value does not necessarily mean that the solution is not feasible. It simply means that the relevant algorithm was not able to conclude it was feasible when it terminated.

dfeasind_p

A pointer to integer variables indicating whether the current solution is known to be dual feasible. Note that a false return value does not necessarily mean that the solution is not feasible. It simply means that the relevant algorithm was not able to conclude it was feasible when it terminated.

Returns

The routine returns zero on success and it returns nonzero if an error occurs.
CPXsolution

Category  Global Function
Definition File  cplex.h
Include Files  cplex.h
Synopsis  

int CPXPUBLIC CPXsolution (CPXCENVptr env,
   CPXCLPptr lp,
   int * lpstat_p,
   double * objval_p,
   double * x,
   double * pi,
   double * slack,
   double * dj)

Description  
The routine CPXsolution accesses the solution values produced by the routines CPXlpopt, CPXprimopt, CPXdualopt, CPXbaropt, CPXhybbaropt, CPXhybnetopt, or CPXqpopt. The solution is maintained until the CPLEX problem object is freed via a call to CPXfreeprob or the solution is rendered invalid because of a call to one of the problem modification routines.

The arguments to CPXsolution are pointers to locations where data are to be written. Such data can include the status of the optimization, the value of the objective function, the values of the primal variables, the dual variables, the slacks and the reduced costs. Although all of the above data exists after a successful call to one of the LP or QP optimizers, it is possible that the user needs only a subset of the data available. If any part of the solution represented by an argument to CPXsolution is not required, that argument can be passed with the value NULL in a call to CPXsolution. If only one part is required, it may be more convenient to use the CPLEX routine that accesses that part of the solution individually: CPXgetstat, CPXgetobjval, CPXgetx, CPXgetpi, CPXgetslack, CPXgetdj.

For barrier, the solution values for x, pi, slack, and dj correspond to the last iterate of the primal-dual algorithm, independent of solution status.

If optimization stopped with an infeasible solution, take care to interpret the meaning of the values in the returned arrays as described in the Arguments section.

Example  

    status = CPXsolution (env, lp, &lpstat, &objval, x, pi,
                           slack, dj);
See also the example `lpex1.c` in the *ILOG CPLEX User's Manual* and in the standard distribution.

### See Also

- `CPXsolninfo`

### Parameters

- **env**
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- **lp**
  A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

- **lpstat_p**
  A pointer to an integer indicating the result of the optimization. The specific values which `*lpstat_p` can take and their meanings are the same as the return values documented for `CPXgetstat` and are found in the group `optim.cplex.statuscodes` of this reference manual.

- **objval_p**
  A pointer to a double precision variable where the objective function value is to be stored.

- **x**
  An array to receive the values of the primal variables for the problem. The length of the array must be at least as great as the number of columns in the problem object. If the solution was computed using the dual simplex optimizer, and the solution is not feasible, `x` values are calculated relative to the phase I RHS used by `CPXdualopt`.

- **pi**
  An array to receive the values of the dual variables for each of the constraints. The length of the array must be at least as great as the number of rows in the problem object. If the solution was computed using the primal simplex optimizer, and the solution is not feasible, `pi` values are calculated relative to the phase I objective (the infeasibility function).

- **slack**
  An array to receive the values of the slack or surplus variables for each of the constraints. The length of the array must be at least as great as the number of rows in the problem object. If the solution was computed by the dual simplex optimizer, and the solution is not feasible, `slack` values are calculated relative to the phase I RHS used by `CPXdualopt`.

- **dj**
  An array to receive the values of the reduced costs for each of the variables. The length of the array must be at least as great as the number of columns in the problem object. If
the solution was computed by the primal simplex optimizer, and the solution is not feasible, \( d_j \) values are calculated relative to the phase I objective (the infeasibility function).

**Returns**

This routine returns zero if a solution exists. If no solution exists, or some other failure occurs, CPXsolution returns nonzero.

---

**Group optim.cplex.callable.accessmipresults**

The routines in the ILOG CPLEX Callable Library to access MIP results.

### Global Functions Summary

<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXgetbestobjval</td>
</tr>
<tr>
<td>CPXgetclqcnt</td>
</tr>
<tr>
<td>CPXgetcovcnt</td>
</tr>
<tr>
<td>CPXgetcutoff</td>
</tr>
<tr>
<td>CPXgetgenclqcnt</td>
</tr>
<tr>
<td>CPXgetmipitcnt</td>
</tr>
<tr>
<td>CPXgetmipobjval</td>
</tr>
<tr>
<td>CPXgetmipqconstrslack</td>
</tr>
<tr>
<td>CPXgetmipslack</td>
</tr>
<tr>
<td>CPXgetmipstart</td>
</tr>
<tr>
<td>CPXgetmipx</td>
</tr>
<tr>
<td>CPXgetnodecnt</td>
</tr>
<tr>
<td>CPXgetnodeint</td>
</tr>
<tr>
<td>CPXgetnodeleftcnt</td>
</tr>
<tr>
<td>CPXgetsubmethod</td>
</tr>
<tr>
<td>CPXgetsubstat</td>
</tr>
</tbody>
</table>

### Description

Solution query routines are used to access information about the results of applying an optimization method to a problem object. For MIP problem objects, you can access the values of variables and constraint slacks. Methods and routines are also available to retrieve other information about the optimization process (such as the number of nodes used).
CPXgetbestobjval

Category          Global Function
Definition File   cplex.h
Include Files     cplex.h
Synopsis          int CPXPUBLIC CPXgetbestobjval(CPXENVptr env, 
                  CPXCLPptr lp, 
                  double * objval_p)

Description       The routine CPXgetbestobjval accesses the currently best known bound on the 
                  optimal solution value of the problem at the time the invoking callback is called by an 
                  instance of IloCplex while solving a MIP. When a model has been solved to 
                  optimality, this value matches the optimal solution value. Otherwise, this value is 
                  computed for a minimization (maximization) problem as the minimum (maximum) 
                  objective function value of all remaining unexplored nodes.

Example

status = CPXgetbestobjval (env, lp, &objval);

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

objval_p
A pointer to the location where the best node objective value is returned. If the branch & 
cut tree has been exhausted, the best node value is set to a large number.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXgetclqcnt

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis: int CPXPUBLIC CPXgetclqcnt (CPXCENVptr env, CPXCLPptr lp)

Description: The routine CPXgetclqcnt is used to access the number of clique cuts added to the problem object during mixed integer optimization.

Example:

clique_cuts = CPXgetclqcnt (env, lp);

Parameters:

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

Example:

clique_cuts = CPXgetclqcnt (env, lp);

Returns: This routine returns zero if no solution, problem, or environment exists. Otherwise, the number of clique cuts added is returned.
**CPXgetcovcnt**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXgetcovcnt (CPXCENVptr env, CPXCLPptr lp)
```

**Description**  
The routine CPXgetcovcnt is used to access the number of cover cuts added to the problem object during mixed integer optimization.

**Example**  
```c
cover_cuts = CPXgetcovcnt (env, lp);
```

**Parameters**

- **env**  
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **lp**  
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

**Example**  
```c
cover_cuts = CPXgetcovcnt (env, lp);
```

**Returns**  
The routine returns zero if no solution, problem, or environment exists. Otherwise, the number of cover cuts added is returned.
CPXgetcutoff

Category         Global Function
Definition File  cplex.h
Include Files   cplex.h
Synopsis         int CPXPUBLIC CPXgetcutoff (CPXCENVptr env,
                    CPXCLPptr lp,
                    double * cutoff_p)
Description      The routine CPXgetcutoff is used to access the MIP cutoff value being used during mixed integer optimization. The cutoff is updated with the objective function value, each time an integer solution is found during branch & cut.
Example          status = CPXgetcutoff (env, lp, &cutoff);
Parameters
    env
    A pointer to the CPLEX environment as returned by CPXopenCPLEX.
    lp
    A pointer to a CPLEX problem object as returned by CPXcreateprob.
    cutoff_p
    A pointer to a location where the value of the cutoff is returned.
Example          status = CPXgetcutoff (env, lp, &cutoff);
Returns          The routine returns zero on success and nonzero if an error occurs.
**CPXgetgenclqcnt**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXgetgenclqcnt (CPXCENVptr env,
      CPXCLPptr lp)
```

**Description**  
The routine `CPXgetgenclqcnt` is used to access the number of clique inequalities in the “clique table”, generated by CPLEX at the start of a mixed integer optimization.

**Example**  
```c
gen_clique_cuts = CPXgetgenclqcnt (env, lp);
```

**Parameters**

- **env**  
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- **lp**  
  A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

**Example**  
```c
gen_clique_cuts = CPXgetgenclqcnt (env, lp);
```

**Returns**  
If no solution, problem, or environment exists, `CPXgetgenclqcnt` returns 0. Otherwise, it returns the number of clique inequalities generated.
CPXgetmipitcnt

Category: Global Function
Definition File: cplex.h

Synopsis:
```c
int CPXPUBLIC CPXgetmipitcnt(CPXENVptr env, CPXCLPptr lp)
```

Description:
The routine CPXgetmipitcnt is used to access the cumulative number of simplex iterations used to solve a mixed integer problem.

Example:
```c
itcnt = CPXgetmipitcnt (env, lp);
```

Parameters:
- `env`:
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- `lp`:
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

Example:
```c
itcnt = CPXgetmipitcnt (env, lp);
```

Returns:
- If a solution exists, CPXgetmipitcnt returns the total iteration count. If no solution, problem, or environment exists, CPXgetmipitcnt returns the value 0.
CPXgetmipobjval

Category          Global Function
Definition File   cplex.h
Synopsis
int CPXPUBLIC CPXgetmipobjval(CPXENVptr env, 
                           CPXCLPptr lp, 
                           double * objval_p)
Description      The routine CPXgetmipobjval is used to access the mixed integer solution objective

Example
status = CPXgetmipobjval (env, lp, &objval);

See also the example mipex1.c in the standard distribution.

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

objval_p
A pointer to the location where the mixed integer objective value is stored.

Example
status = CPXgetmipobjval (env, lp, &objval);

See also the example mipex1.c in the standard distribution.

Returns
The routine returns zero on success and nonzero if an error occurs. If no integer
solution has been found, the value CPXERR_NO_INT_SOLN is returned.
**CPXgetmipqconstrslack**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXgetmipqconstrslack (CPXCENVptr env,  
CPXCLPptr lp,  
double * qcsslack,  
int begin,  
int end)
```

**Description**  
The routine **CPXgetmipqconstrslack** is used to access the slack values for a range of the quadratic constraints of an MIQCP. The beginning and end of the range must be specified.

**Example**  
```c
status = CPXgetmipqconstrslack (env, lp, qcsslack, 0,  
CPXgetnumqconstrs(env,lp)-1);
```

**Parameters**

- **env**  
A pointer to the CPLEX environment as returned by **CPXopenCPLEX**.

- **lp**  
A pointer to a CPLEX problem object as returned by **CPXcreateprob**.

- **qcsslack**  
An array to contain the values of the quadratic constraint qcsslack variables for the problem. This array must be of length at least \((\text{end-end-begin})\). If successful, \(\text{qcsslack}[0]\) through \(\text{qcsslack}[\text{end-end-begin}]\) contain the quadratic constraint slack variables \(\text{begin}\) through \(\text{end}\).

- **begin**  
An integer indicating the beginning of the range of quadratic constraint slack values to be returned.

- **end**  
An integer indicating the end of the range of quadratic constraint slack values to be returned.
Returns

The routine returns zero on success and nonzero if an error occurs. If no integer solution has been found, the value CPXERR_NO_INT_SOLUTION is returned.
**CPXgetmipslack**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXgetmipslack(CPXENVptr env,
                               CPXCPLptr lp,
                               double * slack,
                               int begin,
                               int end)
```

**Description**  
The routine CPXgetmipslack is used to access a range of slack variables for the current mixed integer solution. The beginning and end of the range must be specified. The routine returns zero on success and nonzero if an error occurs. If no integer solution has been found, the value CPXERR_NO_INT_SOLUTION is returned.

**Example**  
```c
status = CPXgetmipslack (env, lp, slack, 0,
                          CPXgetnumrows(env,lp)-1);
```

See also the example mipex1.c in the standard distribution.

**Parameters**

- **env**  
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **lp**  
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

- **slack**  
  An array to contain the values of the primal slack variables for the problem. This array must be of length at least (end-begin+1). If successful, slack[0] through slack[end-begin] contain the slack variables begin through end.

- **begin**  
  An integer indicating the beginning of the range of slack values to be returned.

- **end**  
  An integer indicating the end of the range of slack values to be returned.

**Example**
status = CPXgetmipslack (env, lp, slack, 0,
                 CPXgetnumrows(env,lp)-1);

See also the example mipex1.c in the standard distribution.
# CPXgetmipstart

## Category
Global Function

## Definition File
cplex.h

## Include Files
cplex.h

## Synopsis
```c
int CPXPUBLIC CPXgetmipstart (CPXCENVptr env,
                             CPXCLPptr lp,
                             int * cnt_p,
                             int * indices,
                             double * value,
                             int mipstarts,  // non-CPLEX
                             int * surplus_p)
```

## Description
The routine `CPXgetmipstart` is used to access MIP start information stored in a CPLEX problem object. Values are returned for all integer, binary, semi-continuous, and nonzero SOS variables when the MIP start is the result of a call to `CPXmipopt`.

### Note
If the value of `startspace` is 0 (zero), then the negative of the value of `*surplus_p` returned indicates the length needed for the arrays `indices` and `values`.

## Example
```c
status = CPXgetmipstart (env, lp, &listsize, indices, values,
                          numcols, &surplus);
```

## Parameters
- **env**
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
- **lp**
  A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
- **cnt_p**
  A pointer to an integer to contain the number of MIP start entries returned; that is, the true length of the arrays `indices` and `values`.
- **indices**
- **value**
- **mipstarts**
  A non-CPLEX parameter.
- **surplus_p**
  A pointer to an integer to contain the number of surplus entries returned.
An array to contain the indices of the variables in the MIP start. \( \text{indices}[k] \) is the index of the variable which is entry \( k \) in the MIP start information. Must be of length no less than \( \text{startspace} \).

**values**

An array to contain the MIP start values. The start value corresponding to \( \text{indices}[k] \) is returned in \( \text{values}[k] \). Must be of length at least \( \text{startspace} \).

**mipstartspace**

An integer indicating the length of the non-NULL array \( \text{indices} \) and \( \text{values} \); \( \text{startspace} \) may be 0 (zero).

**surplus_p**

A pointer to an integer to contain the difference between \( \text{startspace} \) and the number of entries in each of the arrays \( \text{indices} \) and \( \text{values} \). A nonnegative value of \( *\text{surplus}_p \) indicates that the length of the arrays was sufficient. A negative value indicates that the length was insufficient and that the routine could not complete its task. In this case, the routine \( \text{CPXgetmipstart} \) returns the value \( \text{CPXERR_NEGATIVE_SURPLUS} \), and the negative value of \( *\text{surplus}_p \) indicates the amount of insufficient space in the arrays.

**Returns**

The routine returns zero on success and nonzero if an error occurs. The value \( \text{CPXERR_NEGATIVE_SURPLUS} \) indicates that insufficient space was available in the \( \text{indices} \) and \( \text{values} \) arrays to hold the MIP start information.
CPXgetmipx

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:
```c
int CPXPUBLIC CPXgetmipx(CPXENVptr env,
                         CPXCLPptr lp,
                         double * x,
                         int begin,
                         int end)
```

Description:
The routine CPXgetmipx is used to access a range of mixed integer solution values. The beginning and end of the range must be specified.

Example:
```c
status = CPXgetmipx (env, lp, x, 0, CPXgetnumcols(env,lp)-1);
```

See also the example mipex1.c in the standard distribution.

Parameters:

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **lp**
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

- **x**
  An array to contain the values of the primal variables for the problem. This array must be of length at least \((end-begin+1)\). If successful, \(x[0]\) through \(x[end-begin]\) contain the solution values \(begin\) through \(end\).

- **begin**
  An integer indicating the beginning of the range of variable values to be returned.

- **end**
  An integer indicating the end of the range of variable values to be returned.

Example:
```c
status = CPXgetmipx (env, lp, x, 0, CPXgetnumcols(env,lp)-1);
```
See also the example `mipex1.c` in the standard distribution.

**Returns**

The routine returns zero on success, and nonzero if an error occurs. If no integer solution has been found, the value `CPXERR_NO_INT_SOLN` is returned.
CPXgetnodecnt

Category          Global Function
Definition File   cplex.h
Include Files    cplex.h
Synopsis          int CPXPUBLIC CPXgetnodecnt (CPXCENVptr env, CPXCLPptr lp)
Description       The routine CPXgetnodecnt is used to access the number of nodes used to solve a mixed integer problem.
Example

nodecount = CPXgetnodecnt (env, lp);

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.
Example

nodecount = CPXgetnodecnt (env, lp);

Returns

If a solution exists, CPXgetnodecnt returns the node count. If no solution, problem, or environment exists, CPXgetnodecnt returns the value 0.
CPXgetnodeint

Category                  Global Function
Definition File          cplex.h
Include Files            cplex.h
Synopsis                  int CPXPUBLIC CPXgetnodeint(CPXCENVptr env,
                           CPXCLPptr lp)
Description               The routine CPXgetnodeint is used to access the node number of the best known integer solution.
Example                   nodeint = CPXgetnodeint (env, lp);
Parameters                env
                           A pointer to the CPLEX environment as returned by CPXopenCPLEX.
                           lp
                           A pointer to a CPLEX problem object as returned by CPXcreateprob.
Example                   nodeint = CPXgetnodeint (env, lp);
Returns                   If no solution, problem, or environment exists, CPXgetnodeint returns a value of -1; otherwise, CPXgetnodeint returns the node number.
CPXgetnodeleftcnt

Category          Global Function
Definition File   cplex.h
Include Files     cplex.h
Synopsis          int CPXPUBLIC CPXgetnodeleftcnt (CPXCENVptr env, CPXCLPtr lp)

Description       The routine CPXgetnodeleftcnt is used to access the number of unexplored nodes left in the branch & cut tree.

Example
nodes_left = CPXgetnodeleftcnt (env, lp);

Parameters
env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

Returns
If no solution, problem, or environment exists, CPXgetnodeleftcnt returns 0 (zero); otherwise, CPXgetnodeleftcnt returns the number of unexplored nodes left in the branch & cut tree.
CPXgetsubmethod

Category          Global Function
Definition File   cplex.h
Include Files     cplex.h
Synopsis
int CPXPUBLIC CPXgetsubmethod (CPXCENVptr env, CPXCLPptr lp)

Description
The routine CPXgetsubmethod is used to access the solution method of the last
subproblem optimization, in the case of an error termination during mixed integer
optimization.

Example
submethod = CPXgetsubmethod (env, lp);

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

Example
submethod = CPXgetsubmethod (env, lp);

Returns
The possible return values are summarized below.

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CPX_ALG_NONE</td>
<td>None</td>
</tr>
<tr>
<td>1</td>
<td>CPX_ALG_PRIMAL</td>
<td>Primal simplex</td>
</tr>
<tr>
<td>2</td>
<td>CPX_ALG_DUAL</td>
<td>Dual simplex</td>
</tr>
<tr>
<td>4</td>
<td>CPX_ALG_BARRIER</td>
<td>Barrier optimizer (no crossover)</td>
</tr>
</tbody>
</table>
CPXgetsubstat

Category       Global Function
Definition File cplex.h
Include Files  cplex.h
Synopsis       int CPXPUBLIC CPXgetsubstat(CPXENVptr env,
                              CPXCPtr lp)
Description    The routine CPXgetsubstat is used to access the solution status of the last subproblem optimization, in the case of an error termination during mixed integer optimization.

Example        substatus = CPXgetsubstat (env, lp);

See Also       CPXgetsubmethod

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

Example        substatus = CPXgetsubstat (env, lp);

Returns        The routine returns zero if no solution exists. A nonzero return value indicates there was an error termination where a subproblem could not be solved to completion. The values returned are documented in the group optim.cplex.callable.solutionstatus in the reference manual of the API.

Group optim.cplex.callable.accessnetworkresults

The routines in the ILOG CPLEX Callable Library to access network results.
### Description
Use these routines to access results after you have used the network optimizer on a problem object created as a network flow structure.

<table>
<thead>
<tr>
<th>Global Functions Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXNETgetbase</td>
</tr>
<tr>
<td>CPXNETgetdj</td>
</tr>
<tr>
<td>CPXNETgetitcnt</td>
</tr>
<tr>
<td>CPXNETgetobjval</td>
</tr>
<tr>
<td>CPXNETgetphasedcnt</td>
</tr>
<tr>
<td>CPXNETgetpi</td>
</tr>
<tr>
<td>CPXNETgetslack</td>
</tr>
<tr>
<td>CPXNETgetstat</td>
</tr>
<tr>
<td>CPXNETgetx</td>
</tr>
<tr>
<td>CPXNETsolveinfo</td>
</tr>
<tr>
<td>CPXNETsolutions</td>
</tr>
</tbody>
</table>
CPXNETgetbase

Category            Global Function
Definition File     cplex.h
Include Files       cplex.h
Synopsis
int CPXPUBLIC CPXNETgetbase (CPXCENVptr env, CPXCNETptr net, int * astat, int * nstat)

Description
The routine CPXNETgetbase is used to access the network basis for a network problem object. Either of the arguments astat or nstat may be NULL.
For this function to succeed, a solution must exist for the problem object.

Table 1: Status codes of network arcs

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_BASIC</td>
<td>If the arc is basic.</td>
</tr>
<tr>
<td>CPX_AT_LOWER</td>
<td>If the arc is nonbasic and its flow is on the lower bound.</td>
</tr>
<tr>
<td>CPX_AT_UPPER</td>
<td>If the arc is nonbasic and its flow is on the upper bound.</td>
</tr>
<tr>
<td>CPX_FREE_SUPER</td>
<td>If the arc is nonbasic but is free. In this case its flow is 0.</td>
</tr>
</tbody>
</table>

Table 2: Status of artificial arcs

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_BASIC</td>
<td>If the arc is basic.</td>
</tr>
<tr>
<td>CPX_AT_LOWER</td>
<td>If the arc is nonbasic and its flow is on the lower bound.</td>
</tr>
</tbody>
</table>

Example

status = CPXNETgetbase (env, net, astat, nstat);

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCLEX.

net
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
astat

An array in which the statuses for network arcs are to be written. After termination, `astat[i]` contains the status assigned to arc `i` of the network stored in `net`. The status may be one of the values in Table 1. If NULL is passed, no arc statuses are copied. Otherwise, `astat` must be an array of a size that is at least `CPXNETgetnumarcs`.

nstat

An array in which the statuses for artificial arcs from each node to the root node are to be written. After termination, `nstat[i]` contains the status assigned to the artificial arc from node `i` to the root node of the network stored in `net`. The status may be one of values in Table 2. If NULL is passed, no node statuses are copied. Otherwise, `nstat` must be an array of a size that is at least `CPXNETgetnumnodes`.

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXNETgetdj

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  

```c
int CPXPUBLIC CPXNETgetdj(CPXCENVptr env,  
CPXCNETptr net,  
double * dj,  
int begin,  
int end)
```

**Description**  
The routine CPXNETgetdj is used to access reduced costs for a range of arcs of the network stored in a network problem object.

For this function to succeed, a solution must exist for the problem object. If the solution is not feasible (CPXNETsolninfo returns 0 in argument pfeasind_p), the reduced costs are computed with respect to an objective function that penalizes infeasibilities.

**Example**

```c
status = CPXNETgetdj (env, net, dj, 10, 20);
```

**Parameters**

- **env**  
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **net**  
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

- **dj**  
Array in which to write requested reduced costs. If NULL is passed, no reduced cost values are returned. Otherwise, dj must point to an array of size at least (end-begin+1).

- **begin**  
Index of the first arc for which a reduced cost value is to be obtained.

- **end**  
Index of the last arc for which a reduced cost value is to be obtained.

**Example**

```c
status = CPXNETgetdj (env, net, dj, 10, 20);
```
Returns

The routine returns zero on success and nonzero if an error occurs.
CPXNETgetitcnt

Category                  Global Function
Definition File           cplex.h
Include Files            cplex.h
Synopsis                  int CPXPUBLIC CPXNETgetitcnt (CPXCENVptr env, 
                          CPXCNTPtr net)
Description               The routine CPXNETgetitcnt accesses the total number of network simplex 
                          iterations for a previous call to CPXNETprimopt, for a network problem object.
Example                   itcnt = CPXNETgetitcnt (env, net);
Parameters
                          env
                          A pointer to the CPLEX environment as returned by CFXopenCPLEX.
                          net
                          A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
Returns                   Returns the total number of network simplex iterations for the last call to 
                          CPXNETprimopt, for a network problem object. If CPXNETprimopt has not been 
                          called, zero is returned. If an error occurs, -1 is returned and an error message is issued.
CPXNETgetobjval

Category Global Function

Definition File cplex.h

Include Files cplex.h

Synopsis

```
int CPXPUBLIC CPXNETgetobjval(CPXENVptr env,
                           CPXNETptr net,
                           double * objval_p)
```

Description

The routine CPXNETgetobjval returns the objective value of the solution stored in a network problem object.

If the current solution is not feasible, the value returned depends on the setting of the parameter CPX_PARAM_NETDISPLAY. If this parameter is set to CPXNET_PENALIZED_OBJECTIVE (2), an objective function value is reported that includes penalty contributions for arcs on which the flow at termination violated the flow bounds on that arc.

Example

```
status = CPXNETgetobjval (env, net, &objval);
```

Parameters

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **net**
  A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

- **objval_p**
  Pointer to where the objective value is written. If NULL is passed, no objective value is returned.

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXNETgetphase1cnt

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXNETgetphase1cnt(CPXENVptr env, CPXNETptr net)
```

Description:

The routine CPXNETgetphase1cnt returns the number of phase 1 network simplex iterations for a previous call to CPXNETprimopt.

Example:

```c
phase1cnt = CPXNETgetphase1cnt (env, net);
```

Parameters:

- **env**: A pointer to the CPLEX environment as returned by CFXopenCPLEX.
- **net**: A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

Returns:

Returns the total number of phase 1 network simplex iterations for the last call to CPXNETprimopt. for a CPXNETptr object. If CPXNETprimopt has not been called, zero is returned. In an error occurs, -1 is returned and an error message is issued.
CPXNETgetpi

Category               Global Function
Definition File        cplex.h
Include Files          cplex.h
Synopsis

    int CPXPUBLIC CPXNETgetpi(CPXENVptr env, 
                              CPXNETptr net, 
                              double * pi, 
                              int begin, 
                              int end)

Description  The routine CPXNETgetpi is used to access dual values for a range of nodes in the network stored in a network problem object.
              For this function to succeed, a solution must exist for the problem object.
Example

    status = CPXNETgetpi (env, net, pi, 10, 20);

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

net
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

pi
Array in which to write solution dual values for requested nodes. If NULL is passed, no data is returned. Otherwise, pi must point to an array of size at least (end-begin+1).

begin
Index of the first node for which the dual value is to be obtained.

end
Index of the last node for which the dual value is to be obtained.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXNETgetslack

Category             Global Function
Definition File      cplex.h
Include Files        cplex.h
Synopsis              int CPXPUBLIC CPXNETgetslack(CPXENVptr env, 
                           CPXNETptr net, 
                           double * slack, 
                           int begin, 
                           int end)

Description          The routine CPXNETgetslack is used to access slack values or, equivalently, 
                      violations of supplies/demands for a range of nodes in the network stored in a network 
                      problem object.
                      For this function to succeed, a solution must exist for the problem object.

Example

                      status = CPXNETgetslack (env, net, slack, 10, 20);

Parameters           env
                      A pointer to the CPLEX environment as returned by CPXopenCPLEX.

                           net
                      A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

                           slack
                      Array in which to write solution slack variables for requested nodes. If NULL is passed, 
                      no data is returned. Otherwise, slack must point to an array of size at least (end− 
                      begin+1).

                           begin
                      Index of the first node for which a slack value is to be obtained.

                           end
                      Index of the last node for which a slack value is to be obtained.

Returns              The routine returns zero on success and nonzero if an error occurs.
CPXNETgetstat

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis: int CPXPUBLIC CPXNETgetstat (CPXCENVptr env, CPXCNETptr net)

Description: The routine CPXNETgetstat returns the solution status for a network problem object.

Example:
netstatus = CPXNETgetstat (env, net);

Parameters:

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

net
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

Returns: If no solution is available for the network problem object, CPXNETgetstat returns 0 (zero). When a solution exists, the possible return values are:

<table>
<thead>
<tr>
<th>CPX_STAT_OPTIMAL</th>
<th>Optimal solution found.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_STAT_UNBOUNDED</td>
<td>Problem has an unbounded ray.</td>
</tr>
<tr>
<td>CPX_STAT_INFEASIBLE</td>
<td>Problem is infeasible.</td>
</tr>
<tr>
<td>CPX_STAT_INFOrUNB</td>
<td>Problem is infeasible or unbounded.</td>
</tr>
<tr>
<td>CPX_STAT_ABORT_IT_LIM</td>
<td>Aborted due to iteration limit.</td>
</tr>
<tr>
<td>CPX_STAT_ABORT_TIME_LIM</td>
<td>Aborted due to time limit.</td>
</tr>
<tr>
<td>CPX_STAT_ABORT_USER</td>
<td>Aborted on user request.</td>
</tr>
</tbody>
</table>
**CPXNETgetx**

**Category** Global Function

**Definition File** cplex.h

**Include Files** cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXNETgetx(CPXENVptr env,
                         CPXNETptr net,
                         double * x,
                         int begin,
                         int end)
```

**Description**
The routine CPXNETgetx is used to access solution values or, equivalently, flow values for a range of arcs stored in a network problem object. For this routine to succeed, a solution must exist for the network problem object.

**Example**

```c
status = CPXNETgetx (env, net, x, 10, 20);
```

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **net**
  A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

- **x**
  Array in which to write solution (or flow) values for requested arcs. If NULL is passed, no solution vector is returned. Otherwise, x must point to an array of size at least (end-begin+1).

- **begin**
  Index of the first arc for which a solution (or flow) value is to be obtained.

- **end**
  Index of the last arc for which a solution (or flow) value is to be obtained.

**Returns**
The routine returns zero on success and nonzero if an error occurs.
CPXNETsolninfo

Category             Global Function
Definition File      cplex.h
Include Files        cplex.h
Synopsis              int CPXPUBLIC CPXNETsolninfo(CPXENVptr env,
                                   CPXNETptr net,
                                   int * pfeasind_p,
                                   int * dfeasind_p)
Description            The routine CPXNETsolninfo is used to access solution information computed by a previous call to CPXNETprimopt. The solution values are maintained in the object as long as no changes are applied to it with one of the CPXNETchg..., CPXNETcopy... or CPXNETadd... functions.

The arguments to CPXNETsolninfo are pointers to locations where data are to be written. The returned values indicate what is known about the primal and dual feasibility of the current solution. If either piece of information represented by an argument to CPXNETsolninfo is not required, a NULL pointer can be passed for that argument.

Example

    status = CPXNETsolninfo (env, lp, &pfeasind, &dfeasind);

Parameters

   env
       A pointer to the CPLEX environment as returned by CPXopenCPLEX.
   net
       A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
   pfeasind_p
       A pointer to an integer variables indicating whether the current solution is known to be primal feasible. Note that a false return value does not necessarily mean that the solution is not feasible. It simply means that the relevant algorithm was not able to conclude that it was feasible when it terminated.
   dfeasind_p
       A pointer to an integer variables indicating whether the current solution is known to be dual feasible. Note that a false return value does not necessarily mean that the solution is not feasible. It simply means that the relevant algorithm was not able to conclude that it was feasible when it terminated.
Returns

The routine returns zero on success and nonzero if an error occurs.
CPXNETsolution

Category Global Function

Definition File cplex.h

Include Files cplex.h

Synopsis

```c
int CPXPUBLIC CPXNETsolution (CPXCENVptr env,
    CPXNETptr net,
    int * netstat_p,
    double * objval_p,
    double * x,
    double * pi,
    double * slack,
    double * dj)
```

Description

The routine CPXNETsolution accesses solution values for a network problem object computed by a previous call to CPXNETprimopt for that object. The solution values are maintained in the object as long as no changes are applied to it with one of the CPXNETchg..., CPXNETcopy..., or CPXNETadd... functions. Whether or not a solution exists can be determined by CPXNETsolninfo.

The arguments to CPXNETsolution are pointers to locations where data is to be written. Such data includes the solution status, the value of the objective function, primal, dual and slack values and the reduced costs.

Although all the above data exists after a successful call to CPXNETprimopt, it is possible that the user only needs a subset of the available data. Thus, if any part of the solution represented by an argument to CPXNETsolution is not required, a NULL pointer can be passed for that argument.

Example

```c
status = CPXNETsolution (env, net, &netstatus, &objval, x, pi,
    slack, dj);
```

Parameters

- env
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- net
  A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
- netstat_p
Pointer to which the solution status is to be written. The specific values that *netstat_p can take and their meanings are the same as the return values documented for CPXNETgetstat.

`objval_p`

Pointer to which the objective value is to be written. If NULL is passed, no objective value is returned. If the solution status is one of the CPX_STAT_ABORT codes, the value returned depends on the setting of parameter CPX_PARAM_NETDISPLAY. If this parameter is set to 2, objective function values that are penalized for infeasible flows are used to compute the objective value of the solution. Otherwise, the true objective function values are used.

`x`

Array to which the solution (flow) vector is to be written. If NULL is passed, no solution vector is returned. Otherwise, `x` must point to an array of size at least that returned by CPXNETgetnumarcs.

`pi`

Array to which the dual values are to be written. If NULL is passed, no dual values are returned. Otherwise, `pi` must point to an array of size at least that returned by CPXNETgetnumnodes.

`slack`

Array to which the slack values (violations of supplies/demands) are to be written. If NULL is passed, no slack values are returned. Otherwise, `slack` must point to an array of size at least that returned by CPXNETgetnumnodes.

`dj`

Array to which the reduced cost values are to be written. If NULL is passed, no reduced cost values are returned. Otherwise, `dj` must point to an array of size at least that returned by CPXNETgetnumarcs.

**Returns**

If a solution exists, it returns zero; if not, it returns nonzero to indicate an error.

### Group optim.cplex.callable.analyzesolution

The routines in the ILOG CPLEX Callable Library to analyze solutions.

<table>
<thead>
<tr>
<th>Global Functions Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXboundsa</td>
</tr>
<tr>
<td>CPXdisplayiis</td>
</tr>
</tbody>
</table>
Description

Solution analysis routines give further information about a solution. As the solutions are computed with finite-precision arithmetic, there may be some numerical residuals; the quality routines give information about what these numerical residuals are. The sensitivity analysis routines give information about how the solution would change if some aspect of the problem is changed; these routines require a simplex basis, so they may be used only after a simplex optimization of an LP. The IIS routines isolate the infeasibilities in a problem to a smaller subproblem; these routines require a primal simplex basis.

<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXfeasopt</td>
</tr>
<tr>
<td>CPXfindiis</td>
</tr>
<tr>
<td>CPXgetdblquality</td>
</tr>
<tr>
<td>CPXgetgrad</td>
</tr>
<tr>
<td>CPXgetiis</td>
</tr>
<tr>
<td>CPXgetintquality</td>
</tr>
<tr>
<td>CPXobjsa</td>
</tr>
<tr>
<td>CPXqpindefcertificate</td>
</tr>
<tr>
<td>CPXrhssa</td>
</tr>
<tr>
<td>CPXsolwrite</td>
</tr>
</tbody>
</table>
CPXboundsa

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:
```c
int CPXPUBLIC CPXboundsa(CPXENVptr env,
                         CPXCLP ptr lp,
                         int begin,
                         int end,
                         double * lblower,
                         double * lbupper,
                         double * ublower,
                         double * ubupper)
```

Description:
The routine CPXboundsa is used to access ranges for lower and/or upper bound values. The beginning and end of the range must be specified. Information for variable \( j \), where \( \text{begin} \leq j \leq \text{end} \), is returned in position \((j-\text{begin})\) of the arrays \( \text{lblower} \), \( \text{lbupper} \), \( \text{ublower} \), and \( \text{ubupper} \).

**Note:** If only lower bound ranges are desired, then both \( \text{lblower} \) and \( \text{lbupper} \) should be non-NULL, and \( \text{ublower} \) and \( \text{ubupper} \) can be NULL.

Example:
```c
status = CPXboundsa (env, lp, 0, CPXgetnumcols(env,lp)-1,
                     lblower, lbupper, ublower, ubupper);
```

Parameters:
- **env**: A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **lp**: A pointer to a CPLEX problem object as returned by CPXcreateprob.
- **begin**: An integer indicating the beginning of the range of ranges to be returned.
- **end**: An integer indicating the end of the range of ranges to be returned.
**lblower**

An array where the lower bound lower range values are to be returned. The length of this array must be at least \((\text{end} - \text{begin} + 1)\). May be NULL.

**lbupper**

An array where the lower bound upper range values are to be returned. The length of this array must be at least \((\text{end} - \text{begin} + 1)\). May be NULL.

**ublower**

An array where the upper bound lower range values are to be returned. The length of this array must be at least \((\text{end} - \text{begin} + 1)\). May be NULL.

**ubupper**

An array where the upper bound upper range values are to be returned. The length of this array must be at least \((\text{end} - \text{begin} + 1)\). May be NULL.

**Example**

```c
status = CPXboundsa(env, lp, 0, CPXgetnumcols(env,lp)-1,
                     lblower, lbupper, ublower, ubupper);
```

**Returns**

The routine returns zero on success and nonzero if an error occurs. This routine fails if no basis exists.
CPXdisplayiis

Category                           Global Function
Definition File                    cplex.h
Include Files                      cplex.h
Synopsis                           int CPXPUBLIC CPXdisplayiis(CPXENVptr env,
                                   CPXCLPptr lp,
                                   CPXCHANNELptr channel,
                                   int display)

Description
The routine CPXdisplayiis is used to send IIS output to a CPLEX message channel. The IIS must already have been computed using a call to CPXfindiis or CPXiiswrite. Two different levels of output are available, corresponding to the output written to the log file and an IIS file in the CPLEX Interactive Optimizer. Thus, CPXdisplayiis enables the user to use CPLEX IIS output formats in a Callable Library application. If neither of these formats is appropriate, the information provided by the routine CPXgetiis can be used to create customized IIS output.

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CPXIIS_TERSE</td>
<td>Displays the names of the rows and columns in the IIS</td>
</tr>
<tr>
<td>2</td>
<td>CPXIIS_VERBOSE</td>
<td>Displays an LP format file containing the IIS</td>
</tr>
</tbody>
</table>

Example

```c
status = CPXdisplayiis (env, lp, mychannel, CPXIIS_TERSE);
```

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

channel
The pointer to the message channel receiving the IIS output.
display
An integer indicating the type of output desired.

**Returns**
The routine returns zero on success and nonzero if an error occurs.
CPXfeasopt

Category   Global Function
Definition File  cplex.h

Synopsis   
```
int CPXPUBLIC CPXfeasopt(CPXENVptr env,
    CPXLPptr lp,
    double * rhs,
    double * rng,
    double * lb,
    double * ub,
    int optind)
```

Description   The routine CPXfeasopt computes a minimal relaxation of the bounds of variable bounds or the right-hand sides of constraints that make the active model feasible. For each bound, the user may specify a preferred value indicating how much relaxation of that bound is acceptable. Similarly, for every right-hand side value or (where applicable) range value, the user may specify a preferred value indicating how much relaxation of that value is acceptable. A negative or zero (0) value indicates that the corresponding bound must not be relaxed. Typically, values greater than or equal to one should be used.

If enough variables or constraints were allowed to be relaxed, the routine returns with suggested relaxed values that would make the active model feasible. These bounds are chosen in such a way that the relaxation is minimal in the sense that the sum of (relaxation amount)/(preference value) over all variables and constraints is minimized.

A solution vector for the model relaxed in this way will be available.

When the argument optind is true, this method also tries to find the optimal solution of the minimally relaxed model.

Parameters
env
Pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
Pointer to a CPLEX problem object as returned by CPXcreateprob.
rhs
An array of doubles of length at least equal to the number of rows in the problem. NULL may be specified if no rhs values are allowed to be relaxed. When a non-NULL array is passed and the function successfully finds a relaxation, the elements of the
array will contain the new values for the right-hand side that constitute the feasible relaxation found by CPXfeasopt.

rng
An array of doubles of length at least equal to the number of rows in the problem. NULL may be specified if no range values are allowed to be relaxed or none are present in the active model. When a non-NULL array is passed and the function successfully finds a relaxation, the elements of the array will contain the new values for the ranges that constitute the feasible relaxation found by CPXfeasopt.

lb
An array of doubles of length at least equal to the number of columns in the problem. NULL may be passed if no lower bound of any variable is allowed to be relaxed. When a non-NULL array is passed and the function successfully finds a relaxation, the elements of the array will contain the new values for the lower bounds that constitute the feasible relaxation found by CPXfeasopt.

ub
An array of doubles of length at least equal to the number of columns in the problem. NULL may be passed if no upper bound of any variable is allowed to be relaxed. When a non-NULL array is passed and the function successfully finds a relaxation, the elements of the array will contain the new values for the upper bounds that constitute the feasible relaxation found by CPXfeasopt.

optind
Optimization indicator. If this parameter is true, then CPXfeasopt tries to find the optimal solution of the minimally relaxed model.

Returns
The routine returns zero on success and nonzero if an error occurs.
**CPXfindiis**

**Category** Global Function

**Definition File** cplex.h

**Include Files** cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXfindiis(CPXENVptr env,
                          CPXLPptr lp,
                          int * iisnumrows_p,
                          int * iisnumcols_p)
```

**Description**

The routine `CPXfindiis` is used to determine an irreducibly inconsistent set (IIS) for an infeasible LP problem. On successful completion, `CPXfindiis` returns the number of rows and columns in the IIS. To obtain information about the individual rows and columns, use the routine `CPXgetiis`, `CPXiswrite` or `CPXdisplayiis`.

**Example**

```c
status = CPXfindiis (env, lp, &iisnumrows, &iisnumcols);
```

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- **lp**
  A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

- **iisnumrows_p**
  A pointer to an integer to contain the number of rows in the IIS.

- **iisnumcols_p**
  A pointer to an integer to contain the number of columns in the IIS.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXgetdblquality

Category    Global Function
Definition File    cplex.h
Include Files    cplex.h
Synopsis

int CPXPUBLIC CPXgetdblquality(CPXENVptr env,
                      CPXCLPptr lp,
                      double * quality_p,
                      int what)

Description

The routine CPXgetdblquality is used to access double-valued information about
the quality of the current solution of a problem. A solution, though not necessarily a
feasible or optimal one, must be available in the CPLEX problem object. The quality
values are returned in the double variables pointed to by the argument quality_p.

The maximum bound infeasibility identifies the largest bound violation, which helps
determine the cause of an infeasible problem. If it exceeds the feasibility tolerance by
only a small amount, it may be possible to obtain a feasible solution to the problem by
increasing the feasibility tolerance. If a problem is optimal, it gives insight into the
smallest setting for the feasibility tolerance that would not cause the problem to
terminate infeasible.

If an error occurs, the value remains unchanged.

The possible quality values for a solution are listed in the group

Example

status = CPXgetdblquality (env, lp, &max_x, CPX_MAX_X);

Parameters

env
A pointer to the CPLEX environment as returned by the CPXopenCPLEX routine.
lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.
quality_p
A pointer to a double variable in which the requested quality value is to be stored.
If an error occurs, the value remains unchanged.
what
A symbolic constant indicating the quality value to be retrieved.

The possible quality values for a solution are listed in the group `optim.cplex.callable.solutionquality` in the *ILOG CPLEX Reference Manual*.

**Example**

```c
status = CPXgetdblquality (env, lp, &max_x, CPX_MAX_X);
```

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXgetgrad

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis

#include <cplex.h>

int CPXgetgrad (CPXCENVptr env, CPXCLPptr lp, int j, int * head, double * y);

Description

The routine CPXgetgrad can be used, after an LP has been solved and a basis is available, to access information useful for different types of post-solution analysis. CPXgetgrad provides two arrays that can be used to project the impact of making changes to optimal variable values or objective function coefficients.

For a unit change in the value of the jth variable, the value of the ith basic variable, sometimes referred to as the variable basic in the ith row, changes by the amount y[i]. Also, for a unit change of the objective function coefficient of the ith basic variable, the reduced-cost of the jth variable changes by the amount y[i]. The vector y is equal to the product of the inverse of the basis matrix and the column j of the constraint matrix. Thus, y can be thought of as the representation of the jth column in terms of the basis.

Example

status = CPXgetgrad (env, lp, 13, head, y);

Parameters

env

A pointer to the CPLEX environment as returned by CPXopenCplex.

lp

A pointer to a CPLEX problem object as returned by CPXcreateprob.

j

An integer indicating the index of the column of interest. A negative value for j indicates a column representing the slack or artificial variable for row -j-1.

head

An array to contain a listing of the indices of the basic variables in the order in which they appear in the basis. This listing is sometimes called the basis header. The ith entry in this list is also sometimes viewed as the variable in the ith row of the basis.
basic variable is a structural variable, `head[i]` simply contains the column index of that variable. If it is a slack variable, `head[i]` contains one less than the negative of the row index of that slack variable. This array should be of length at least `CPXgetnumrows(env, lp)`. May be NULL.

`y` 
An array to contain the coefficients of the jth column relative to the current basis. See the discussion above on how to interpret the entries in `y`. This array should be of length at least `CPXgetnumrows(env, lp)`. May be NULL.

**Example**

```
status = CPXgetgrad (env, lp, 13, head, y);
```

**Returns**

The routine returns zero on success and nonzero if an error occurs. This routine fails if no basis exists.
CPXgetiis

Category | Global Function
---|---
Definition File | cplex.h
Include Files | cplex.h

Synopsis

```c
int CPXPUBLIC CPXgetiis(CPXCENVptr env,
                          CPXCLPptr lp,
                          int * iisstat_p,
                          int * rowind,
                          int * rowbdstat,
                          int * iisnumrows_p,
                          int * colind,
                          int * colbdstat,
                          int * iisnumcols_p)
```

Description

The routine CPXgetiis is used to examine the IIS for an infeasible LP problem. The IIS must already have been computed by a call to CPXfindiis or CPXiiswrite. On successful completion, the CPXgetiis arguments return information about the IIS that can be used to diagnose the cause of infeasibility. The number of rows and bound constraints in the IIS, as well as the indices of the IIS members, are returned.

Table 1: Values of iisstat_p

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CPXIIS_COMPLETE</td>
</tr>
<tr>
<td>2</td>
<td>CPXIIS_PARTIAL</td>
</tr>
</tbody>
</table>

Table 2: Values of elements of rowbdstat

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CPXIIS_AT_LOWER</td>
</tr>
<tr>
<td>1</td>
<td>CPXIIS_FIXED</td>
</tr>
<tr>
<td>2</td>
<td>CPXIIS_AT_UPPER</td>
</tr>
</tbody>
</table>

Table 3: Values of elements of colbdstat (status of column bounds)

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CPXIIS_AT_LOWER</td>
</tr>
<tr>
<td>1</td>
<td>CPXIIS_FIXED</td>
</tr>
<tr>
<td>2</td>
<td>CPXIIS_AT_UPPER</td>
</tr>
</tbody>
</table>
Example

```c
status = CPXgetiis (env, lp, &iisstat, rowind, rowbdstat, 
    &iisnumrows, colind, colbdstat, 
    &iisnumcols);
```

Parameters

- **env**
  - A pointer to the CPLEX environment as returned by the CPXopenCLEX routine.
- **lp**
  - A pointer to a CPLEX problem object as returned by CPXcreateprob.
- **iisstat_p**
  - A pointer to an integer to contain the result of the IIS algorithm. The specific values that *iisstat_p* can take and their meanings appear in Table 1.
- **rowind**
  - An array to contain the indices of the rows in the IIS. The length of the array must be at least as large as the number of rows in the IIS.
- **rowbdstat**
  - An array that identifies the right-hand side value for each row in the IIS that is causing the infeasibility. Possible values appear in Table 2. The length of the array must be at least as large as the number of rows in the IIS. This information is needed only for ranged rows. For all other row senses, the value is uniquely determined by the sense.
- **iisnumrows_p**
  - A pointer to an integer to contain the number of rows in the IIS.
- **colind**
  - An array to contain the indices of the columns in the IIS. The length of the array must be at least as large as the number of columns in the IIS.
- **colbdstat**
  - An array that identifies the bound for each column in the IIS that is causing the infeasibility. Possible values appear in Table 3. The length of the array must be at least as large as the number of columns in the IIS.
- **iisnumcols_p**
  - A pointer to an integer to contain the number of columns in the IIS.

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXgetintquality

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXgetintquality (CPXCENVptr env,
                                CPXCLPptr lp,
                                int * quality_p,
                                int what)
```

Description:
The routine `CPXgetintquality` is used to access integer-valued information about the quality of the current solution of a problem. A solution, though not necessarily a feasible or optimal one, must be available in the CPLEX problem object. The quality values are returned in the integer variables pointed to by the argument `quality_p`.

The possible quality values for a solution are listed in the group optim.cplex.callable.solutionquality in the *ILOG CPLEX Reference Manual*.

Example:

```c
status = CPXgetintquality (env, lp, &max_x_ind, CPX_MAX_X);
```

Parameters:

- **env**
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- **lp**
  A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

- **quality_p**
  A pointer to an integer variable in which the requested quality value is to be stored.

- **what**
  A symbolic constant indicating the quality value to be retrieved.

The possible quality values for a solution are listed in the group optim.cplex.callable.solutionquality in the *ILOG CPLEX Reference Manual*.

Example:

```c
status = CPXgetintquality (env, lp, &max_x_ind, CPX_MAX_X);
```
The routine returns zero on success and nonzero if an error occurs.
CPXobjsa

Category        Global Function
Definition File cplex.h
Include Files   cplex.h

Synopsis        int CPXPUBLIC CPXobjsa(CPXENVptr env,
                          CPXCLPptr lp,
                          int begin,
                          int end,
                          double * lower,
                          double * upper)

Description     The routine CPXobjsa is used to access upper and lower sensitivity ranges for objective function coefficients for a specified range of variable indices. The beginning and end of the range of variable indices must be specified.

Note: Information for variable \( j \), where \( \text{begin} \leq j \leq \text{end} \), is returned in position \( (j-\text{begin}) \) of the arrays lower and upper.

Example

\[
\text{status} = \text{CPXobjsa} \{ \text{env}, \text{lp}, 0, \text{CPXgetnumcols}(\text{env}, \text{lp})-1, \\
\text{lower, upper}; \\
\]

Parameters       env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.
begin
An integer indicating the beginning of the range of ranges to be returned.
end
An integer indicating the end of the range of ranges to be returned.
lower
An array where the objective function lower range values are to be returned. This array must be of length at least \((\text{end} - \text{begin} + 1)\).

upper

An array where the objective function upper range values are to be returned. This array must be of length at least \((\text{end} - \text{begin} + 1)\).

**Returns**

The routine returns zero on success and nonzero if an error occurs. This routine fails if no optimal basis exists.
**CPXqpindefcertificate**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXqpindefcertificate(CPXENVptr env,
          CPXCLPptr lp,
          double * x)
```

**Description**  
The routine **CPXqpindefcertificate** computes a vector \( x \) that satisfies the inequality \( x'Qx < 0 \). Such a vector demonstrates that the \( Q \) matrix violates the assumption of positive semi-definiteness, and can be an aid in debugging a user's program if indefiniteness is an unexpected outcome.

**Example**  
```c
status = CPXqpindefcertificate (env, lp, x);
```

**Parameters**

- **env**  
  A pointer to the CPLEX environment as returned by **CPXopenCPLEX**.

- **lp**  
  A pointer to a CPLEX problem object as returned by **CPXcreateprob**.

- **x**  
  An array to receive the values of the vector that is to be returned. The length of this array must be the same as the number of quadratic variables in the problem, which can be obtained by calling **CPXgetnumquad** for example.

**Returns**  
The routine returns zero on success and nonzero if an error occurs.
CPXrhssa

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXrhssa(CPXENVptr env,
    CPXCLPptr lp,
    int begin,
    int end,
    double * lower,
    double * upper)
```

**Description**
The routine `CPXrhssa` is used to access a range of upper and lower ranges for right-hand side values. The beginning and end of the range must be specified.

**Note:** Information for constraint \( j \), where \( begin \leq j \leq end \), is returned in position \((j-begin)\) of the arrays `lower` and `upper`.

**Example**

```c
status = CPXrhssa (env, lp, 0, CPXgetnumrows(env,lp)-1,
    lower, upper);
```

**Parameters**

- `env`
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- `lp`
  A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

- `begin`
  An integer indicating the beginning of the range of ranges to be returned.

- `end`
  An integer indicating the end of the range of ranges to be returned.

- `lower`
  An array where the right-hand side lower range values are to be returned. This array must be of length at least \((end - begin + 1)\).
upper

An array where the right-hand side upper range values are to be returned. This array must be of length at least \((\text{end} - \text{begin} + 1)\).

Returns

The routine returns zero on success and nonzero if an error occurs. This routine fails if no optimal basis exists.
The routine CPXsolwrite is a generic routine for writing solutions. It performs all the calculations needed to produce a solution file, but it writes only through functions that the user provides to it, so that the user may choose the data representation, data selection, and file format.

The user must open the file before calling CPXsolwrite and close the file after calling CPXsolwrite.

The arguments to CPXsolwrite are functions it calls to write the file. CPXsolwrite does not “know” anything about the file or the type of output being written. The argument info, the last parameter, communicates information to the routines hsection, rsectionbeg, csectionbeg, write_entry, and sectionend.

The function indicated by hsection writes a header section in a formatted file. Its only arguments are the ILOG CPLEX environment pointer env, the problem pointer lp, and the info parameter. CPXsolwrite calls this function first. It uses the lp problem pointer to retrieve any information needed by the header-section function.

The function indicated by rsectionbeg writes information at the beginning of the row section of a formatted file. It is called after hsection and sectionend. Its only argument is the info parameter.
The function indicated by `csection` writes information at the beginning of the column section of a formatted file. It is called after all row entries have been completed. Its only argument is the `info` parameter.

The function indicated by `write_entry` is called once for each row and column in the problem. The following table summarizes its arguments.

The function indicated by `sectionend` is used at the end of each header, row, and column section. The only argument to this function is the `info` parameter.

<table>
<thead>
<tr>
<th>Type?</th>
<th>Name?</th>
<th>Meaning?</th>
</tr>
</thead>
<tbody>
<tr>
<td>void?</td>
<td>*info?</td>
<td>the info parameter of CPXsolwrite?</td>
</tr>
<tr>
<td>int?</td>
<td>aflag?</td>
<td>a for alternate optimum?</td>
</tr>
<tr>
<td>int?</td>
<td>num?</td>
<td>sequence number; cumulative over rows and columns?</td>
</tr>
<tr>
<td>char?</td>
<td>*name?</td>
<td>name of row or column?</td>
</tr>
<tr>
<td>char?</td>
<td>*state?</td>
<td>state of row or column; one of: UL, LL, BS, EQ, FR, **?</td>
</tr>
<tr>
<td>double?</td>
<td>val1?</td>
<td>for rows, row activity; for columns, column solution value?</td>
</tr>
<tr>
<td>double?</td>
<td>val2?</td>
<td>for rows, slack activity; for columns, objective coefficient?</td>
</tr>
<tr>
<td>double?</td>
<td>ll?</td>
<td>for rows, lower limit; for columns, lower bound?</td>
</tr>
<tr>
<td>double?</td>
<td>ul?</td>
<td>for rows, upper limit; for columns, upper bound?</td>
</tr>
<tr>
<td>double?</td>
<td>val3?</td>
<td>for rows, dual value; for columns, reduced cost?</td>
</tr>
</tbody>
</table>

**Parameters**

- `env`
  The pointer to the ILOG CPLEX environment, as returned by one of the CPXopenCPLEX routines.

- `lp`
  A pointer to a CPLEX LP problem object, as returned by CPXcreateprob.

- `info`
  A generic pointer that passes information to each of the functions called by CPXsolwrite.
Returns

The routine returns zero on success and nonzero if an error occurs.

Group optim.cplex.callable.querygeneralproblem

The routines in the ILOG CPLEX Callable Library to query general problem data.

<table>
<thead>
<tr>
<th>Global Functions Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXgetcoeff</td>
</tr>
<tr>
<td>CPXgetcolindex</td>
</tr>
<tr>
<td>CPXgetcolname</td>
</tr>
<tr>
<td>CPXgetcols</td>
</tr>
<tr>
<td>CPXgetlb</td>
</tr>
<tr>
<td>CPXgetnumcols</td>
</tr>
<tr>
<td>CPXgetnumNZ</td>
</tr>
<tr>
<td>CPXgetnumrows</td>
</tr>
<tr>
<td>CPXgetobj</td>
</tr>
<tr>
<td>CPXgetobjname</td>
</tr>
<tr>
<td>CPXgetobjsen</td>
</tr>
<tr>
<td>CPXgetprobsname</td>
</tr>
<tr>
<td>CPXgetprobsense</td>
</tr>
<tr>
<td>CPXgetrhs</td>
</tr>
<tr>
<td>CPXgetrngval</td>
</tr>
<tr>
<td>CPXgetrowindex</td>
</tr>
<tr>
<td>CPXgetrowname</td>
</tr>
<tr>
<td>CPXgetrows</td>
</tr>
<tr>
<td>CPXgetsense</td>
</tr>
<tr>
<td>CPXgetub</td>
</tr>
</tbody>
</table>

Description

These routines to access information about a problem object after it has been created can be used at any time, even after problem modifications.
**CPXgetcoef**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**

```
int CPXPUBLIC CPXgetcoef (CPXCENVptr env, 
CPXCLPptr lp, 
int i, 
int j, 
double * coef_p)
```

**Description**
The routine CPXgetcoef is used to access a single constraint matrix coefficient of a CPLEX problem object. The row and column indices must be specified.

**Example**

```
status = CPXgetcoef (env, lp, 10, 20, &coef);
```

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **lp**
  A pointer to a CPLEX problem object as returned by CPXcreateprob.
- **i**
  An integer indicating the numeric index of the row.
- **j**
  An integer indicating the numeric index of the column.
- **coef_p**
  A pointer to a double to contain the specified matrix coefficient.

**Returns**
The routine returns zero on success and nonzero if an error occurs.
CPXgetcolindex

Category          Global Function
Definition File   cplex.h
Include Files     cplex.h
Synopsis          int CPXPUBLIC CPXgetcolindex(CPXENVptr env,
                              CPXCLPptr lp,
                              const char * lname_str,
                              int * index_p)
Description       The routine CPXgetcolindex searches for the index number of the specified column
in a CPLEX problem object.
Example
    status = CPXgetcolindex (env, lp, "power43", &colindex);
Parameters
    env
    A pointer to the CPLEX environment as returned by CPXopenCPLEX.
    lp
    A pointer to a CPLEX problem object as returned by CPXcreateprob.
    lname_str
    A column name to search for.
    index_p
    A pointer to an integer to hold the index number of the column with name lname_str.
    If the routine is successful, *index_p contains the index number; otherwise,
    *index_p is undefined.
Returns
    The routine returns zero on success and nonzero if an error occurs.
CPXgetcolName

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXgetcolname(CPXCENVptr env,
CPXCLPptr lp,
char ** name,
char * namestore,
int storespace,
int * surplus_p,
int begin,
int end)
```

Description:

The routine CPXgetcolname is used to access a range of column names or, equivalently, the variable names of a CPLEX problem object. The beginning and end of the range, along with the length of the array in which the column names are to be returned, must be specified.

**Note:** If the value of `storespace` is 0, the negative of the value of `*surplus_p` returned indicates the total number of characters needed for the array `namestore`.

Example:

```c
status = CPXgetcolname (env, lp, cur_colname, cur_colnamestore,
cur_storespace, &surplus, 0,
cur_numcols-1);
```

See also the example `lpex7.c` in the *ILOG CPLEX User's Manual* and in the standard distribution.

Parameters:

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **lp**
  A pointer to a CPLEX problem object as returned by CPXcreateprob.
- **name**
An array of pointers to the column names stored in the array namestore. This array must be of length at least (end - begin + 1). The pointer to the name of column \( j \) is returned in name[begin + j].

namestore

An array of characters where the specified column names are to be returned. May be NULL if storespace is 0.

storespace

An integer indicating the length of the array namestore. May be 0.

surplus_p

A pointer to an integer to contain the difference between storespace and the total amount of memory required to store the requested names. A nonnegative value of *surplus_p indicates that storespace was sufficient. A negative value indicates that it was insufficient and that the routine could not complete its task. In that case, CPXgetcolname returns the value CPXERR_NEGATIVE_SURPLUS, and the negative value of the variable *surplus_p indicates the amount of insufficient space in the array namestore.

begin

An integer indicating the beginning of the range of column names to be returned.

end

An integer indicating the end of the range of column names to be returned.

Returns

The routine returns zero on success and nonzero if an error occurs. The value CPXERR_NEGATIVE_SURPLUS indicates that insufficient space was available in the namestore array to hold the names.
**CPXgetcols**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXgetcols (CPXCENVptr env,
    CPXCLPptr lp,
    int * nzcnt_p,
    int * cmatbeg,
    int * cmatind,
    double * cmatval,
    int cmatspace,
    int * surplus_p,
    int begin,
    int end)
```

**Description**  
The routine CPXgetcols is used to access a range of columns of the constraint matrix of a CPLEX problem object. The beginning and end of the range, along with the length of the arrays in which the nonzero entries of these columns are to be returned, must be specified.

**Note:** If the value of cmatspace is zero, the negative of the value of *surplus_p returned indicates the length needed for the arrays cmatind and cmatval.

**Example**  
```c
status = CPXgetcols (env, lp, &nzcnt, cmatbeg, cmatind,
    cmatval, cmatspace, &surplus, 0,
    cur_numcols-1);
```

**Parameters**  

**env**  
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

**lp**  
A pointer to a CPLEX problem object as returned by CPXcreateprob.

**nzcnt**
A pointer to an integer to contain the number of nonzeros returned; that is, the true length of the arrays cmatind and cmatval.

cmatbeg

An array to contain indices indicating where each of the requested columns begins in the arrays cmatval and cmatind. Specifically, column j consists of the entries in cmatval and cmatind in the range from cmatbeg[j - begin] to cmatbeg[(j + 1) - begin]-1. (Column end consists of the entries from cmatbeg[end - begin] to nzcnt_p-1.) This array must be of length at least (end - begin + 1).

cmatind

An array to contain the row indices associated with the elements of cmatval. May be NULL if cmatspace is zero.

cmatval

An array to contain the nonzero coefficients of the specified columns. May be NULL if cmatspace is zero.

cmatspace

An integer indicating the length of the arrays cmatind and cmatval. May be zero.

surplus_p

A pointer to an integer to contain the difference between cmatspace and the number of entries in each of the arrays cmatind and cmatval. A nonnegative value of &surplus_p indicates that the length of the arrays was sufficient. A negative value indicates that the length was insufficient and that the routine could not complete its task. In this case, CPXgetcols returns the value CPXERR_NEGATIVE_SURPLUS, and the negative value of &surplus_p indicates the amount of insufficient space in the arrays.

begin

An integer indicating the beginning of the range of columns to be returned.

depth

An integer indicating the end of the range of columns to be returned.

**Returns**

The routine returns zero on success and nonzero if an error occurs. The value CPXERR_NEGATIVE_SURPLUS indicates that insufficient space was available in the arrays cmatind and cmatval to hold the nonzero coefficients.
CPXgetlb

Category          Global Function
Definition File   cplex.h
Include Files    cplex.h
Synopsis          

```c
int CPXPUBLIC CPXgetlb(CPXENVptr env,
                        CPXCLPptr lp,
                        double * lb,
                        int begin,
                        int end)
```

Description       The routine CPXgetlb is used to access a range of lower bounds on the variables of a CPLEX problem object. The beginning and end of the range must be specified.

Example            

```c
status = CPXgetlb (env, lp, lb, 0, cur_numcols-1);
```

Parameters

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **lp**
  A pointer to a CPLEX problem object as returned by CPXcreateprob.
- **lb**
  An array where the specified lower bounds on the variables are to be returned. This array must be of length at least \((\text{end} - \text{begin} + 1)\). The lower bound of variable \( j \) is returned in \( lb[j - \text{begin}] \).
- **begin**
  An integer indicating the beginning of the range of lower bounds to be returned.
- **end**
  An integer indicating the end of the range of lower bounds to be returned.

Returns              The routine returns zero on success and nonzero if an error occurs.
**CPXgetnumcols**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXgetnumcols (CPXCENVptr env, CPXCLPptr lp)
```

**Description**  
The routine `CPXgetnumcols` is used to access the number of columns in the constraint matrix, or equivalently, the number of variables in the CPLEX problem object.

**Example**  
```c
cur_numcols = CPXgetnumcols (env, lp);
```

See also the example `lpex1.c` in the *ILOG CPLEX User’s Manual* and in the standard distribution.

**Parameters**  
- **env**  
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- **lp**  
  A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

**Example**  
```c
cur_numcols = CPXgetnumcols (env, lp);
```

See also the example `lpex1.c` in the *ILOG CPLEX User’s Manual* and in the standard distribution.

**Returns**  
If the problem object or environment does not exist, `CPXgetnumcols` returns the value 0 (zero); otherwise, it returns the number of columns (variables).
CPXgetnumnz

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXgetnumnz(CPXENVptr env, CPXCLPptr lp)
```

Description:

The routine CPXgetnumnz is used to access the number of nonzero elements in the constraint matrix of a CPLEX problem object, not including the objective function, quadratic constraints, or the bounds constraints on the variables.

Example:

```c
cur_numnz = CPXgetnumnz (env, lp);
```

Parameters:

- `env`: A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- `lp`: A pointer to a CPLEX problem object as returned by CPXcreateprob.

Returns:

If the problem object or environment does not exist, CPXgetnumnz returns the value 0 (zero); otherwise, it returns the number of nonzero elements.
CPXgetnumrows

Category: Global Function

Definition File: cplex.h
Include Files: cplex.h

Synopsis: int CPXPUBLIC CPXgetnumrows (CPXCENVptr env, CPXCLPptr lp)

Description: The routine CPXgetnumrows is used to access the number of rows in the constraint matrix, not including the objective function, quadratic constraints, or the bounds constraints on the variables.

Example:

cur_numrows = CPXgetnumrows (env, lp);

See also the example lpex1.c in the ILOG CPLEX User's Manual and in the standard distribution.

Parameters:

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

Returns: If the CPLEX problem object or environment does not exist, CPXgetnumrows returns the value 0 (zero); otherwise, it returns the number of rows.
CPXgetobj

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXgetobj(CPXENVptr env,
                         CPXCLPptr lp,
                         double * obj,
                         int begin,
                         int end)
```

Description:
The routine CPXgetobj is used to access a range of objective function coefficients of a CPLEX problem object. The beginning and end of the range must be specified.

Example:

```c
status = CPXgetobj (env, lp, obj, 0, cur_numcols-1);
```

Parameters:

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **lp**
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

- **obj**
  An array where the specified objective coefficients are to be returned. This array must be of length at least \((end - begin + 1)\). The objective function coefficient of variable \(j\) is returned in \(obj[j - begin]\).

- **begin**
  An integer indicating the beginning of the range of objective function coefficients to be returned.

- **end**
  An integer indicating the end of the range of objective function coefficients to be returned.

Returns:
The routine returns zero on success and nonzero if an error occurs.
CPXgetobjname

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXgetobjname(CPXENVptr env, CPXCLPptr lp, char * buf_str, int bufspace, int * surplus_p)
```

Description:
The routine CPXgetobjname is used to access the name of the objective row of a CPLEX problem object.

**Note:** If the value of `bufspace` is 0, then the negative of the value of `*surplus_p` returned indicates the total number of characters needed for the array `buf_str`.

Example:

```c
status = CPXgetobjname (env, lp, cur_objname, lenname, &surplus);
```

Parameters:

- `env`:
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- `lp`:
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

- `buf_str`:
  A pointer to a buffer of size `bufspace`. May be NULL if `bufspace` is 0.

- `bufspace`:
  An integer indicating the length of the array `buf_str`. May be 0.

- `surplus_p`:
  A pointer to an integer to contain the difference between `bufspace` and the amount of memory required to store the objective row name. A nonnegative value of `*surplus_p`
indicates that the length of the array `buf_str` was sufficient. A negative value indicates that the length of the array was insufficient and that the routine could not complete its task. In this case, `CPXgetobjname` returns the value `CPXERR_NEGATIVE_SURPLUS`, and the negative value of the variable `*surplus_p` indicates the amount of insufficient space in the array `buf_str`.

**Returns**

The routine returns zero on success and nonzero if an error occurs. The value `CPXERR_NEGATIVE_SURPLUS` indicates that insufficient space was available in the `buf_str` array to hold the objective name.
CPXgetobjsen

Category    Global Function
Definition File    cplex.h
Include Files    cplex.h
Synopsis    int CPXPUBLIC CPXgetobjsen(CPXCENVptr env, CPXCLPptr lp)

Description    The routine CPXgetobjsen is used to access whether the objective function sense of a CPLEX problem object is maximization or minimization.

Example    cur_objsen = CPXgetobjsen (env, lp);

Parameters

env    A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp    A pointer to a CPLEX problem object as returned by CPXcreateprob.

Returns    A value of CPX_MIN=1 is returned for minimization and CPX_MAX=-1 is returned for maximization. If the problem object or environment does not exist, a 0 is returned.
CPXgetprobname

Category                  Global Function
Definition File           cplex.h
Include Files            cplex.h
Synopsis                  int CPXPUBLIC CPXgetprobname (CPXCENVptr env,
                           CPXCLPptr lp,
                           char * buf_str,
                           int bufspace,
                           int * surplus_p)

Description               The routine CPXgetprobname is used to access the name of the problem set via the call to CPXcreateprob.

Note: If the value of bufspace is 0, then the negative of the value of *surplus_p returned indicates the total number of characters needed for the array buf_str.

Example

    status = CPXgetprobname (env, lp, cur_probname, lenname,
                            &surplus);

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.
buf_str
A pointer to a buffer of size bufspace. May be NULL if bufspace is 0.
bufspace
An integer indicating the length of the array buf_str. May be 0.
surplus_p
A pointer to an integer to contain the difference between bufspace and the amount of memory required to store the problem name. A nonnegative value of *surplus_p
indicates that the length of the array buf_str was sufficient. A negative value indicates that the length of the array was insufficient and that the routine could not complete its task. In this case, CPXgetprobnam returns the value CPXERR_NEGATIVE_SURPLUS, and the negative value of the variable *surplus_p indicates the amount of insufficient space in the array buf_str.

**Returns**

The routine returns zero on success and nonzero if an error occurs. The value CPXERR_NEGATIVE_SURPLUS indicates that insufficient space was available in the buf_str array to hold the problem name.
CPXgetprobtype

**Category**  Global Function

**Definition File**  cplex.h

**Include Files**  cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXgetprobtype(CPXENVptr env, CPXCLPptr lp)
```

**Description**

The routine `CPXgetprobtype` is used to access the problem type that is currently stored in a CPLEX problem object.

**Example**

```c
probtype = CPXgetprobtype (env, lp);
```

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>?</td>
<td>Error: no problem or environment.</td>
</tr>
<tr>
<td>0</td>
<td>CPXPROB_LP</td>
<td>Linear program; no quadratic data or ctype information stored.</td>
</tr>
<tr>
<td>1</td>
<td>CPXPROB_MILP</td>
<td>Problem with ctype information.</td>
</tr>
<tr>
<td>3</td>
<td>CPXPROB_FIXEDMILP</td>
<td>Problem with ctype information, integer variables fixed.</td>
</tr>
<tr>
<td>5</td>
<td>CPXPROB_QP</td>
<td>Problem with quadratic data stored.</td>
</tr>
<tr>
<td>7</td>
<td>CPXPROB_MIQP</td>
<td>Problem with quadratic data and ctype information.</td>
</tr>
<tr>
<td>8</td>
<td>CPXPROB_FIXEDMIQP</td>
<td>Problem with quadratic data and ctype information, integer variables fixed.</td>
</tr>
<tr>
<td>10</td>
<td>CPXPROB_QCP</td>
<td>Problem with quadratic constraints.</td>
</tr>
</tbody>
</table>
Return values

| 11 | CPXPROB_MIQCP | Problem with quadratic constraints and ctype information. |

See Also

CPXchgprobtype

Parameters

env

A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp

A pointer to a CPLEX problem object as returned by CPXcreateprob.

Returns

The values returned by CPXgetprobtype appear in the table.
CPXgetrhs

Category          Global Function
Definition File   cplex.h
Include Files     cplex.h
Synopsis
int CPXPUBLIC CPXgetrhs(CPXENVptr env,
            CPXCLPptr lp,
            double * rhs,
            int begin,
            int end)

Description       The routine CPXgetrhs is used to access the right-hand side coefficients for a range of constraints in a CPLEX problem object. The beginning and end of the range must be specified.
Example
    status = CPXgetrhs (env, lp, rhs, 0, cur_numrows-1);

Parameters
    env
    A pointer to the CPLEX environment as returned by CPXopenCPLEX.
    lp
    A pointer to a CPLEX problem object as returned by CPXcreateprob.
    rhs
    An array where the specified right-hand side coefficients are to be returned. This array must be of length at least (end - begin + 1). The right-hand side of constraint i is returned in rhs[i - begin].
    begin
    An integer indicating the beginning of the range of right-hand side terms to be returned.
    end
    An integer indicating the end of the range of right-hand side terms to be returned.

Returns       The routine returns zero on success and nonzero if an error occurs.
CPXgetrngval

Category         Global Function
Definition File  cplex.h
Include Files   cplex.h

Synopsis         int CPXPUBLIC CPXgetrngval(CPXENVptr env,
                            CPXCLPptr lp,
                            double * rngval,
                            int begin,
                            int end)

Description      The routine CPXgetrngval is used to access the RHS range coefficients for a set of constraints in a CPLEX problem object. The beginning and end of the set must be specified. CPXgetrngval checks if ranged constraints are present in the problem object.

Example          status = CPXgetrngval (env, lp, rngval, 0, cur_numrows-1);

Parameters       env
                  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
                  A pointer to a CPLEX problem object as returned by CPXcreateprob.
rngval
                  An array where RHS range coefficients are returned. This array must be of length at least (end - begin + 1). A value of 0 for any entry means that the corresponding row is not ranged.
begin
                  An integer indicating the beginning of the set of rows for which RHS range coefficients are returned.
end
                  An integer indicating the end of the set of rows for which RHS range coefficients are returned.

Returns          The routine returns zero on success and nonzero if an error occurs.
**CPXgetrowindex**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXgetrowindex(CPXENVptr env,  
                            CPXCLPptr lp,  
                            const char * lname_str,  
                            int * index_p)
```

**Description**  
The routine CPXgetrowindex searches for the index number of the specified row in a CPLEX problem object.

**Example**  
```c
status = CPXgetrowindex (env, lp, "resource89", &rowindex);
```

**Parameters**

- **env**  
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **lp**  
A pointer to a CPLEX problem object as returned by CPXcreateprob.

- **lname_str**  
A row name to search for.

- **index_p**  
A pointer to an integer to hold the index number of the row with name lname_str. If the routine is successful, *index_p contains the index number; otherwise, *index_p is undefined.

**Returns**  
The routine returns zero on success and nonzero if an error occurs.
CPXgetrowname

Category   Global Function
Definition File  cplex.h
Include Files  cplex.h

Synopsis
int CPXPUBLIC CPXgetrowname(CPXENVptr env, 
    CPXCLPptr lp, 
    char ** name, 
    char * namestore, 
    int storespace, 
    int * surplus_p, 
    int begin, 
    int end)

Description
The routine CPXgetrowname is used to access a range of row names or, equivalently, the constraint names of a CPLEX problem object. The beginning and end of the range, along with the length of the array in which the row names are to be returned, must be specified.

Example
status = CPXgetrowname (env, lp, cur_rowname, cur_rownamestore, 
    cur_storespace, &surplus, 0, 
    cur_numrows-1);

Parameters
env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.
name

Note: If the value of storespace is 0, then the negative of the value of *surplus_p returned indicates the total number of characters needed for the array namestore.
An array of pointers to the row names stored in the array `namestore`. This array must be of length at least \((\text{end} - \text{begin} + 1)\). The pointer to the name of row \(i\) is returned in `name[i - \text{begin}]`.

`namestore`

An array of characters where the specified row names are to be returned. May be NULL if `storespace` is 0.

`storespace`

An integer indicating the length of the array `namestore`. May be 0.

`surplus_p`

A pointer to an integer to contain the difference between `storespace` and the total amount of memory required to store the requested names. A nonnegative value of `*surplus_p` indicates that `storespace` was sufficient. A negative value indicates that it was insufficient and that the routine could not complete its task. In that case, `CPXgetrowname` returns the value `CPXERR_NEGATIVE_SURPLUS`, and the negative value of the variable `*surplus_p` indicates the amount of insufficient space in the array `namestore`.

`begin`

An integer indicating the beginning of the range of row names to be returned.

`end`

An integer indicating the end of the range of row names to be returned.

**Returns**

The routine returns zero on success and nonzero if an error occurs. The value `CPXERR_NEGATIVE_SURPLUS` indicates that insufficient space was available in the `namestore` array to hold the names.
CPXgetrows

Category  Global Function
Definition File  cplex.h
Include Files  cplex.h

Synopsis

int CPXPUBLIC CPXgetrows (CPXCENVptr env,
  CPXCLPptr lp,
  int * nzcnt_p,
  int * rmatbeg,
  int * rmatind,
  int * rmatval,
  int rmatspace,
  int * surplus_p,
  int begin,
  int end)

Description

The routine CPXgetrows is used to access a range of rows of the constraint matrix, not including the objective function or the bounds constraints on the variables of a CPLEX problem object. The beginning and end of the range, along with the length of the arrays in which the nonzero entries of these rows are to be returned, must be specified.

Note: If the value of rmatspace is 0 then the negative of the value of *surplus_p returned indicates the length needed for the arrays rmatval and rmatind.

Example

status = CPXgetrows (env, lp, &nzcnt, rmatbeg, rmatind, rmatval, rmatspace, &surplus, 0, cur_numrows-1);

Parameters

env
A pointer to the CPLEX environment as returned by the CPXopenCPLEX routine.

lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

nzcnt_p
A pointer to an integer to contain the number of nonzeros returned; that is, the true length of the arrays rmatind and rmatval.

**rmatbeg**

An array to contain indices indicating where each of the requested rows begins in the arrays rmatval and rmatind. Specifically, row i consists of the entries in rmatval and rmatind in the range from rmatbeg[i - begin] to rmatbeg[(i + 1) - begin]-1. (Row end consists of the entries from rmatbeg[end - begin] to *nzcnt_p-1.) This array must be of length at least((end - begin + 1).

**rmatind**

An array to contain the column indices of the entries of rmatval. May be NULL if rmatval is 0.

**rmatval**

An array to contain the nonzero entries of the specified rows. May be NULL if rmatval is 0.

**rmatspace**

An integer indicating the length of the arrays rmatind and rmatval. May be 0.

**surplus_p**

A pointer to an integer to contain the difference between rmatspace and the number of entries in each of the arrays rmatind and rmatval. A nonnegative value of *surplus_p indicates that the length of the arrays was sufficient. A negative value indicates that the length was insufficient and that the routine could not complete its task. In this case, the routine CPXgetrows returns the value CPXERR_NEGATIVE_SURPLUS, and the negative value of *surplus_p indicates the amount of insufficient space in the arrays.

**begin**

An integer indicating the beginning of the range of rows to be returned.

**end**

An integer indicating the end of the range of rows to be returned.

**Returns**

The routine returns zero on success and nonzero if an error occurs. The value CPXERR_NEGATIVE_SURPLUS indicates that insufficient space was available in the arrays rmatind and rmatval to hold the nonzero coefficients.
CPXgetsense

Category          Global Function
Definition File   cplex.h
Include Files    cplex.h
Synopsis          int CPXPUBLIC CPXgetsense(CPXENVptr env,
                              CPXCLPtr lp,
                              char * sense,
                              int begin,
                              int end)

Description       The routine CPXgetsense is used to access the sense for a range of constraints in a
                   CPLEX problem object. The beginning and end of the range must be specified.
                   Example
                   status = CPXgetsense (env, lp, sense, 0, cur_numrows-1);

                   | sense[i] | = 'L' | ≤ constraint |
                   | sense[i] | = 'E' | = constraint |
                   | sense[i] | = 'G' | ≥ constraint |
                   | sense[i] | = 'R' | ranged constraint |

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.
sense
An array where the specified constraint senses are to be returned. This array must be of
length at least (end-begin+1). The sense of constraint i is returned in sense[i - begin]. Possible values appear in the table.
begin
An integer indicating the beginning of the range of constraint senses to be returned.
end
An integer indicating the end of the range of constraint senses to be returned.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
**CPXgetub**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**
```c
int CPXPUBLIC CPXgetub(CPXENVptr env,
   CPXCLPptr lp,
   double * ub,
   int begin,
   int end)
```

**Description**
The routine CPXgetub is used to access a range of upper bounds on the variables of a CPLEX problem object. The beginning and end of the range must be specified.

**Example**
```c
status = CPXgetub(env, lp, ub, 0, cur_numcols-1);
```

**Parameters**
- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **lp**
  A pointer to a CPLEX problem object as returned by CPXcreateprob.
- **ub**
  An array where the specified upper bounds on the variables are to be returned. This array must be of length at least (end - begin+1). The upper bound of variable j is returned in ub[j-begin].
- **begin**
  An integer indicating the beginning of the range of upper bounds to be returned.
- **end**
  An integer indicating the end of the range of upper bounds to be returned.

**Returns**
The routine returns zero on success and nonzero if an error occurs.

**Group optim.cplex.callable.querymip**
The routines in the ILOG CPLEX Callable Library to query MIP problem data.
These routines to access information about a MIP problem object after it has been created can be used at any time, even after problem modifications.
**CPXgetctype**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c  
int CPXPUBLIC CPXgetctype(CPXCENVptr env,  
   CPXCLPptr lp,  
   char * xctype,  
   int begin,  
   int end)  
```

**Description**  
The routine `CPXgetctype` is used to access the types for a range of variables in a problem object. The beginning and end of the range must be specified.

**Example**  
```c  
status = CPXgetctype (env, lp, ctype, 0, cur_numcols-1);  
```

**See Also**  
`CPXcopyctype`

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- **lp**
  A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

- **xctype**
  An array where the specified types are to be returned. This array must be of length (`end-begin+1`). The type of variable `j` is returned in `ctype[j-begin]`. See the routine `CPXcopyctype` for a list of possible values for the variables in `ctype`.

- **begin**
  An integer indicating the beginning of the range of types to be returned

- **end**
  An integer indicating the end of the range of types to be returned.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXgetnumbin

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**
```c
int CPXPUBLIC CPXgetnumbin(CPXENVptr env, 
CPXCLPptr lp)
```

**Description**
The routine CPXgetnumbin is used to access the number of binary variables in a CPLEX problem object.

**Example**
```c
numbin = CPXgetnumbin (env, lp);
```

**Parameters**
- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **lp**
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

**Example**
```c
numbin = CPXgetnumbin (env, lp);
```

**Returns**
If the problem object or environment does not exist, CPXgetnumbin returns zero. Otherwise, it returns the number of binary variables in the problem object.
**CPXgetnumint**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXgetnumint(CPXENVptr env, CPXCLPptr lp)
```

**Description**  
The routine `CPXgetnumint` is used to access the number of general integer variables in a CPLEX problem object.

**Example**  
```c
numint = CPXgetnumint (env, lp);
```

**Parameters**  
- **env**  
A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- **lp**  
A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

**Example**  
```c
numint = CPXgetnumint (env, lp);
```

**Returns**  
If the problem object or environment does not exist, `CPXgetnumint` returns zero. Otherwise, it returns the number of general integer variables in the problem object.
**CPXgetnumsemicont**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**
```c
int CPXPUBLIC CPXgetnumsemicont (CPXCENVptr env, CPXCLPptr lp)
```

**Description**
The routine CPXgetnumsemicont is used to access the number of semi-continuous variables in a CPLEX problem object.

**Example**
```c
numsc = CPXgetnumsemicont (env, lp);
```

**Parameters**
- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **lp**
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

**Returns**
If the problem object or environment does not exist, CPXgetnumsemicont returns the value 0 (zero); otherwise, it returns the number of semi-continuous variables in the problem object.
**CPXgetnumsemiint**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXgetnumsemiint (CPXCENVptr env, CPXCLPptr lp)
```

**Description**  
The routine `CPXgetnumsemiint` is used to access the number of semi-integer variables in a CPLEX problem object.

**Example**  
```c
numsc = CPXgetnumsemiint (env, lp);
```

**Parameters**

- `env`  
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- `lp`  
  A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

**Returns**  
If the problem object or environment does not exist, `CPXgetnumsemiint` returns the value 0 (zero); otherwise, it returns the number of semi-integer variables in the problem object.
**CPXgetnumsos**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**
```c
int CPXPUBLIC CPXgetnumsos(CPXENVptr env,
                   CPXCLPptr lp)
```

**Description**
The routine `CPXgetnumsos` is used to access the number of Special Ordered Sets in a CPLEX problem object.

**Example**
```c
numsos = CPXgetnumsos (env, lp);
```

**Parameters**
- **env**
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
- **lp**
  A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

**Example**
```c
numsos = CPXgetnumsos (env, lp);
```

**Returns**
If the problem object or environment does not exist, or the problem is not a mixed integer problem, the routine returns the value 0; otherwise, it returns the number of Special Ordered Sets in the problem object.
## CPXgetorder

### Category
Global Function

### Definition File
cplex.h

### Include Files
cplex.h

### Synopsis

```c
int CPXPUBLIC CPXgetorder(CPXENVptr env,
    CPXCLPptr lp,
    int * cnt_p,
    int * indices,
    int * priority,
    int * direction,
    int ordspace,
    int * surplus_p)
```

### Description
The routine CPXgetorder is used to access all the MIP priority order information stored in a CPLEX problem object. A priority order is generated if there is no order and parameter CPX_PARAM_MIPORDTYPE is nonzero.

**Note:** If the value of ordspace is 0, then the negative of the value of *surplus_p returned indicates the length needed for the arrays indices, priority, and direction.

### Example

```c
status = CPXgetorder (env, lp, &listsize, indices, priority, direction, numcols, &surplus);
```

### Possible settings for direction

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_BRANCH_GLOBAL</td>
<td>0</td>
<td>Use global branching direction setting CPX_PARAM_BRDIR</td>
</tr>
<tr>
<td>CPX_BRANCH_DOWN</td>
<td>1</td>
<td>Branch down first on variable indices[k]</td>
</tr>
<tr>
<td>CPX_BRANCH_UP</td>
<td>2</td>
<td>Branch up first on variable indices[k]</td>
</tr>
</tbody>
</table>

### Parameters
- `env`
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp

A pointer to a CPLEX problem object as returned by CPXcreateprob.

cnt_p

A pointer to an integer to contain the number of order entries returned; i.e., the true length of the arrays indices, priority, and direction.

indices

An array where the indices of the variables in the order are to be returned. indices[k] is the index of the variable which is entry k in the order information.

priority

An array where the priority values are to be returned. The priority corresponding to the indices[k] is returned in priority[k]. May be NULL. If priority is not NULL, it must be of length at least ordspace.

direction

An array where the preferred branching directions are to be returned. The direction corresponding to indices[k] is returned in direction[k]. May be NULL. If direction is not NULL, it must be of length at least ordspace. Possible settings for direction[k] appear in the table.

ordspace

An integer indicating the length of the non-NULL arrays indices, priority, and direction. May be 0.

surplus_p

A pointer to an integer to contain the difference between ordspace and the number of entries in each of the arrays indices, priority, and direction. A nonnegative value of *surplus_p indicates that the length of the arrays was sufficient. A negative value indicates that the length was insufficient and that the routine could not complete its task. In this case, the routine CPXgetorder returns the value CPXERR_NEGATIVE_SURPLUS, and the negative value of *surplus_p indicates the amount of insufficient space in the arrays.

Returns

The routine returns zero on success and nonzero if an error occurs. The value CPXERR_NEGATIVE_SURPLUS indicates that insufficient space was available in the indices, priority, and direction arrays to hold the priority order information.
**CPXgetsos**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXgetsos(CPXENVptr env,  
                    CPXCLPptr lp,  
                    int * numsosnz_p,  
                    char * sostype,  
                    int * sospri,  
                    int * sosbeg,  
                    int * sosind,  
                    double * soswt,  
                    int sospace,  
                    int * surplus_p,  
                    int begin,  
                    int end)
```

**Description**

The routine `CPXgetsos` is used to access a range of Special Ordered Set (SOS) definitions stored in a CPLEX problem object. The beginning and end of the range, along with the length of the array in which the definitions are to be returned, must be provided.

**Note:** If the value of `sosspace` is 0 (zero), then the negative of the value of `*surplus_p` returned indicates the length needed for the arrays `sosind` and `soswt`.

**Example**

```c
status = CPXgetsos (env, lp, &numsosnz, sostype, sospri, sosbeg,  
                    sosind, soswt, sospace, 0, numsos-1);
```

**Parameters**

- **env**  
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- **lp**  
  A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

- **numsosnz_p**
A pointer to an integer to contain the number of set members returned; i.e., the true length of the arrays sosind and soswt.

sostype
An array to contain the types of the requested SOSs. The type of set k is returned in sostype[k-begin]. This array must be of length at least (end - begin + 1). Contains either CPX_TYPE_SOS1 ('1') or CPX_TYPE_SOS2 ('2'), for a type 1 or type 2 SOS respectively.

sospri
An array to contain the priorities of the SOSs. The priority of set i is returned in sospri[i-begin]. This array must be of length at least (end - begin + 1). May be NULL.

sosbeg
An array to contain indices indicating where each of the requested SOSs begins in the arrays sosind and soswt. Specifically, set k consists of the entries in sosind and soswt in the range from sosbeg[k-begin] to sosbeg[(k+1) - begin] - 1. (Set end consists of the entries from sosbeg[end - begin] to *num sosnz_p - 1.) This array must be of length at least (end - begin + 1).

sosind
An array to contain the variable indices of the SOS members. May be NULL if sosspace is 0.

soswt
An array to contain the reference values (weights) for SOS members. May be NULL if sosspace is 0. Weight soswt[k] corresponds to sosind[k].

sosspace
An integer indicating the length of the arrays sosind and soswt. May be 0.

surplus_p
A pointer to an integer to contain the difference between sosspace and the number of entries in each of the arrays sosind and soswt. A nonnegative value of *surplus_p indicates that the length of the arrays was sufficient. A negative value indicates that the length was insufficient and that the routine could not complete its task. In this case, the routine CPXgetsos returns the value CPXERR_NEGATIVE_SURPLUS, and the negative value of *surplus_p indicates the amount of insufficient space in the arrays.

begin
An integer indicating the beginning of the range of SOSs to be returned.

end
An integer indicating the end of the range of SOSs to be returned.

**Returns**

The routine returns zero on success and nonzero if an error occurs. The value CPXERR_NEGATIVE_SURPLUS indicates that insufficient space was available in the arrays sosind and soswt to hold the SOS definition.

**Group optim.cplex.callable.queryqp**

The routines in the ILOG CPLEX Callable Library to query QP problem data (that is, problems with a quadratic objective function).

**Global Functions Summary**

<table>
<thead>
<tr>
<th>Global Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXgetnumapnz</td>
</tr>
<tr>
<td>CPXgetnumquad</td>
</tr>
<tr>
<td>CPXgetqpcoef</td>
</tr>
<tr>
<td>CPXgetquad</td>
</tr>
</tbody>
</table>

**Description**

These routines to access information about a QP problem object after it has been created can be used at any time, even after problem modifications. That is, these routines access information about a problem object with a quadratic objective function.
CPXgetnumqpnz

Category        Global Function
Definition File  cplex.h
Include Files   cplex.h
Synopsis

int CPXPUBLIC CPXgetnumqpnz(CPXENVptr env, CPXCLPptr lp)

Description    The routine CPXgetnumqpnz returns the number of nonzeros in the Q matrix of a problem object.

Example

numqpnz = CPXgetnumqpnz (env, lp);

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

Returns
If successful, the routine returns the number of nonzeros in the Q matrix. If an error occurs, zero is returned.
CPXgetnumquad

Category          Global Function
Definition File   cplex.h
Include Files     cplex.h
Synopsis          int CPXPUBLIC CPXgetnumquad (CPXENVptr env, CPXCLPptr lp)

Description       The routine CPXgetnumquad returns the number of variables that have quadratic objective coefficients in a CPLEX problem object.
Example            numquad = CPXgetnumquad (env, lp);

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

Returns If successful, the routine returns the number of variables having quadratic coefficients. If an error occurs, 0 is returned.
CPXgetqpcoef

Category          Global Function
Definition File   cplex.h
Include Files     cplex.h
Synopsis          

    int CPXPUBLIC CPXgetqpcoef (CPXCENVptr env,
                       CPXCLPptr lp,
                       int rownum,
                       int colnum,
                       double * coef_p)

Description       The routine CPXgetqpcoef accesses the quadratic coefficient in the matrix Q of a
                   CPLEX problem object for the variable pair indexed by (rownum, colnum). The result
                   is stored in *coef_p.
                   
                   Example
                   
                   status = CPXgetqpcoef (env, lp, 10, 20, &coef);

Parameters

  env
  A pointer to the CPLEX environment as returned by CPXopenCLEX.

  lp
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

  rownum
  The first variable number (row number in Q).

  colnum
  The second variable number (column number in Q).

  coef_p
  A pointer to a double where the coefficient should be stored.

Returns          The routine returns zero on success and nonzero if an error occurs.
CPXgetquad

Category  Global Function
Definition File  cplex.h
Include Files  cplex.h

Synopsis

int CPXPUBLIC CPXgetquad(CPXENVptr env,
    CPXCLPptr lp,
    int * nzcnt_p,
    int * qmatbeg,
    int * qmatind,
    int * qmatval,
    int qmatspace,
    int * surplus_p,
    int begin,
    int end)

Description

The routine CPXgetquad is used to access a range of columns of the matrix Q of a model with a quadratic objective function. The beginning and end of the range, along with the length of the arrays in which the nonzero entries of these columns are to be returned, must be specified.

Specifically, column \( j \) consists of the entries in \( \text{qmatval} \) and \( \text{qmatind} \) in the range from \( \text{qmatbeg}[j - \text{begin}] \) to \( \text{qmatbeg}[(j + 1) - \text{begin}] - 1 \). (Column end consists of the entries from \( \text{qmatbeg}[\text{end} - \text{begin}] \) to \( \text{nzcnt}_p - 1 \).) This array must be of length at least \((\text{end} - \text{begin} + 1)\).

A nonnegative value of \( \text{surplus}_p \) indicates that the length of the arrays was sufficient. A negative value indicates that the length was insufficient and that the routine could not complete its task. In this case, CPXgetquad returns the value CPXERR_NEGATIVE_SURPLUS, and the negative value of \( \text{surplus}_p \) indicates the amount of insufficient space in the arrays.

Note: If the value of \( \text{qmatspace} \) is zero, the negative of the value of \( \text{surplus}_p \) returned indicates the length needed for the arrays \( \text{qmatind} \) and \( \text{qmatval} \).

Example

```c
status = CPXgetquad (env, lp, &nzcnt, qmatbeg, qmatind, 
    qmatval, qmatspace, &surplus, 0,
```
Parameters

- **env**: A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **lp**: A pointer to a CPLEX problem object as returned by CPXcreateprob.
- **nzcnt_p**: A pointer to an integer to contain the number of nonzeros returned; that is, the true length of the arrays `qmatind` and `qmatval`.
- **qmatbeg**: An array to contain indices indicating where each of the requested columns of Q begins in the arrays `qmatval` and `qmatind`.
- **qmatind**: An array to contain the row indices associated with the elements of `qmatval`. May be NULL if `cmatspace` is zero.
- **qmatval**: An array to contain the nonzero coefficients of the specified columns. May be NULL if `cmatspace` is zero.
- **cmatspace**: An integer indicating the length of the arrays `qmatind` and `qmatval`. May be zero.
- **surplus_p**: A pointer to an integer to contain the difference between `cmatspace` and the number of entries in each of the arrays `qmatind` and `qmatval`.
- **begin**: An integer indicating the beginning of the range of columns to be returned.
- **end**: An integer indicating the end of the range of columns to be returned.

Returns

The routine returns zero on success and nonzero if an error occurs.

Group optim.cplex.callable.querynetwork

The routines in the ILOG CPLEX Callable Library to query network problem data.
Description

These routines to access information about a network problem object after it has been created can be used at any time, even after problem modifications.
CPXNETgetarcindex

Category               Global Function
Definition File         cplex.h
Include Files          cplex.h
Synopsis
int CPXPUBLIC CPXNETgetarcindex(CPXENVptr env,
                                   CPXNETptr net,
                                   const char * lname_str,
                                   int * index_p)

Description
The routine CPXNETgetarcindex returns the index of the specified arc (in the network stored in a network problem object) in the integer pointed to by index_p.

Example
status = CPXNETgetarcindex (env, net, "from_a_to_b", &index);

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

net
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

lname_str
Name of the arc to look for.

index_p
A pointer to an integer to hold the arc index. If the routine is successful, *index_p contains the index number; otherwise, *index_p is undefined.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXNETgetarcname

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:
int CPXPUBLIC CPXNETgetarcname(CPXENVptr env, CPXNETptr net, char ** nnames, char * namestore, int namespc, int * surplus_p, int begin, int end)

Description:
The routine CPXNETgetarcname is used to access the names of a range of arcs in a network stored in a network problem object. The beginning and end of the range, along with the length of the array in which the arc names are to be returned, must be specified.

Example:
status = CPXNETgetarcname (env, net, nnames, namestore, namespc, &surplus, 0, narcs-1);

Parameters:

env: A pointer to the CPLEX environment as returned by CPXopenCPLEX.

net: A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

nnames: Where to copy pointers to arc names stored in the namestore array. The length of this array must be at least (end-begin+1). The pointer to the name of arc i is returned in nnames[i-begin].

namestore: Array of characters to which the specified arc names are to be copied. It may be NULL if namespc is 0.

namespc: Length of the namestore array.

surplus_p: A pointer to an integer where the number of surplus arcs in the range is stored.
CPXNETgetarcname

Pointer to an integer to which the difference between namespc and the number of characters required to store the requested names is returned. A nonnegative value indicates that namespc was sufficient. A negative value indicates that it was insufficient. In that case, CPXERR_NEGATIVE_SURPLUS is returned and the negative value of surplus_p indicates the amount of insufficient space in the array namestore.

begin

Index of the first arc for which a name is to be obtained.

end

Index of the last arc for which a name is to be obtained.

Returns

The routine returns zero on success and nonzero if an error occurs. The value CPXERR_NEGATIVE_SURPLUS indicates that insufficient space was available in the namestore array to hold the names.
CPXNETgetarcnodes

Category          Global Function
Definition File   cplex.h
Include Files     cplex.h
Synopsis          int CPXPUBLIC CPXNETgetarcnodes(CPXENVptr env,
                              CPXNETptr net,
                              int * fromnode,
                              int * tonode,
                              int begin,
                              int end)

Description       The routine CPXNETgetarcnodes is used to access the “from nodes” and “to nodes”
                   for a range of arcs in the network stored in a network problem object.

Example            status = CPXNETgetarcnodes (env, net, fromnode, tonode,
                              0, cur_narcs-1);

Parameters         env
                   A pointer to the CPLEX environment as returned by CPXopenCPLEX.
net
                   A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
fromnode
                   Array in which to write the “from node” indices of the requested arcs. If NULL is
                   passed, no “from node” indices are retrieved. Otherwise, the size of the array must be
                   (end-begin+1).
tonode
                   Array in which to write the “to node” indices of the requested arcs. If NULL is passed,
                   no “to node” indices are retrieved. Otherwise, the size of the array must be (end-
                   begin+1).
begin
                   Index of the first arc to get nodes for.
end
                   Index of the last arc to get nodes for.
Returns

The routine returns zero on success and nonzero if an error occurs.
CPXNETgetlb

Category                Global Function
Definition File          cplex.h
Include Files            cplex.h
Synopsis                 int CPXPUBLIC CPXNETgetlb(CPXCENVptr env,
                          CPXNETptr net,
                          double * low,
                          int begin,
                          int end)
Description              The routine CPXNETgetlb is used to access the lower capacity bounds for a range of arcs of the network stored in a network problem object.
Example                  status = CPXNETgetlb (env, net, low, 0, cur_narcs-1);
Parameters               env
                          A pointer to the CPLEX environment as returned by CPXopenCPLEX.
                          net
                          A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
                          low
                          Array in which to write the lower bound on the flow for the requested arcs. If NULL is passed, no lower bounds are retrieved. Otherwise, the size of the array must be (end-begin+1).
                          begin
                          Index of the first arc for which lower bounds are to be obtained.
                          end
                          Index of the last arc for which lower bounds are to be obtained.
Returns                  The routine returns zero on success and nonzero if an error occurs.
**CPXNETgetnodearcs**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**
```c
int CPXPUBLIC CPXNETgetnodearcs(CPXENVptr env, CPXCNETptr net, int * arccnt_p, int * arcbeg, int * arc, int arcspace, int * surplus_p, int begin, int end);
```

**Description**
The routine CPXNETgetnodearcs is used to access the arc indices incident to a range of nodes in the network stored in a network problem object.

**Example**
```c
status = CPXNETgetnodearcs (env, net, &arccnt, arcbeg, arc, arcspace, &surplus, begin, end);
```

**Parameters**

**env**
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

**net**
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

**arccnt_p**
A pointer to an integer to contain the total number of arc indices returned in the array arc.

**arcbeg**
An array to contain indices indicating where each of the requested arc lists start in array arc. Specifically, the list of arcs incident to node i (< end) consists of the indices in arcs in the range from arcbeg[i-begin] to arcbeg[(i+1)-begin]-1. The list of arcs incident to node end consists of the entries in arc in the range from arcbeg[end-begin] to *arccnt_p-1. This array must have a length of at least end-begin+1.
arc
An array to contain the arc indices for the arcs incident to the nodes in the specified range. May be NULL if arcspac is zero.

arcspac e
An integer indicating the length of the array arc. May be zero.

surplus_p
A pointer to an integer to contain the difference between arcspac e and the number of arcs incident to the nodes in the specified range. A nonnegative value indicates that arcspac e was sufficient. A negative value indicates that it was insufficient and that the routine could not complete its task. In that case, CPXERR_NEGATIVE_SURPLUS is returned and the negative value of surplus_p indicates the amount of insufficient space in the array arc.

begin
Index of the first node for which arcs are to be obtained.

end
Index of the last node for which arcs are to be obtained.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXNETgetnodeindex

Category          Global Function
Definition File   cplex.h
Include Files    cplex.h
Synopsis          int CPXPUBLIC CPXNETgetnodeindex(CPXENVptr env, 
                                             CPXNETptr net, 
                                             const char * lname_str, 
                                             int * index_p)
Description       The routine CPXNETgetnodeindex returns the index of the specified node (in the 
                   network stored in a network problem object) in the integer pointed to by index_p.
Example           status = CPXNETgetnodeindex (env, net, "root", &index);
Parameters        
                   env
                   A pointer to the CPLEX environment as returned by CPXopenCPLEX.
                   net
                   A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
                   lname_str
                   Name of the node to look for.
                   index_p
                   A pointer to an integer to hold the node index. If the routine is successful, *index_p 
                   contains the index number; otherwise, *index_p is undefined.
Returns           The routine returns zero on success and nonzero if an error occurs.
CPXNETgetnodename

Category Global Function

Definition File cplex.h

Include Files cplex.h

Synopsis

int CPXPUBLIC CPXNETgetnodename(CPXENVptr env,
                CPXNETptr net,
                char ** nnames,
                char * namestore,
                int namespc,
                int * surplus_p,
                int begin,
                int end)

Description

The routine CPXNETgetnodename is used to obtain the names of a range of nodes in a network stored in a network problem object. The beginning and end of the range, along with the length of the array in which the node names are to be returned, must be specified.

Example

status = CPXNETgetnodename (env, net, nnames, namestore, namespc,
                            &surplus, 0, nnodes-1);

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

net
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

nnames
Where to copy pointers to node names stored in the namestore array. The length of this array must be at least (end-begin+1). The pointer to the name of node i is returned in nnames[i-begin].

namestore
Array of characters to which the specified node names are to be copied. It may be NULL if namespc is 0.

namespc
Length of the namestore array.

surplus_p
Pointer to an integer in which the difference between namec and the number of characters required to store the requested names is returned. A nonnegative value indicates that namec was sufficient. A negative value indicates that it was insufficient. In that case, CPXERR_NEGATIVE_SURPLUS is returned and the negative value of surplus_p indicates the amount of insufficient space in the array namestore.

begin

Index of the first node for which a name is to be obtained.

end

Index of the last node for which a name is to be obtained.

Returns

The routine returns zero on success and nonzero if an error occurs. The value CPXERR_NEGATIVE_SURPLUS indicates that there was not enough space in the namestore array to hold the names.
CPXNETgetnumarcs

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:
int CPXPUBLIC CPXNETgetnumarcs(CPXENVptr env, CPXCNETptr net)

Description: The routine CPXNETgetnumarcs is used to access the number of arcs in a network stored in a network problem object.

Example:
cur_narcs = CPXNETgetnumarcs(env, net);

Parameters:
- env: A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- net: A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

Returns: The routine returns the number of network arcs stored in a network problem object. If an error occurs, 0 is returned and an error message is issued.
**CPXNETgetnumnodes**

**Category**    Global Function

**Definition File**  cplex.h

**Include Files**  cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXNETgetnumnodes(CPXENVptr env, CPXNETptr net)
```

**Description**  
The routine CPXNETgetnumnodes is used to access the number of nodes in a network stored in a network problem object.

**Example**
```c
cur_nnodes = CPXNETgetnumnodes (env, net);
```

**Parameters**
- **env**  
  A pointer to the CPLEX environment as returned by CFXopenCPLEX.
- **net**  
  A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

**Returns**  
The routine returns the number of network nodes stored in a network problem object. If an error occurs, 0 is returned and an error message is issued.
CPXNETgetobj

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXNETgetobj(CPXENVptr env,
   CPXCNETptr net,
   double * obj,
   int begin,
   int end)
```

Description:
The routine CPXNETgetobj is used to access the objective function values for a range of arcs in the network stored in a network problem object.

Example:

```c
status = CPXNETgetobj (env, net, obj, 0, cur_narcs-1);
```

Parameters:

- **env**: A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **net**: A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
- **obj**: Array in which to write the objective values for the requested range of arcs. If NULL is passed, no objective values are retrieved. Otherwise, obj must point to an array of size at least (end-begin+1).
- **begin**: Index of the first arc for which the objective value is to be obtained.
- **end**: Index of the last arc for which the objective value is to be obtained.

Returns:
The routine returns zero on success and nonzero if an error occurs.
CPXNETgetobjsen

Category          Global Function
Definition File   cplex.h
Include Files    cplex.h
Synopsis          int CPXPUBLIC CPXNETgetobjsen(CPXENVptr env, 
                 CPXNETptr net)
Description       The routine CPXNETgetobjsen returns the sense of the objective function, that is, 
                  maximization or minimization, of a network problem object.
Example           
                  objsen = CPXNETgetobjsen (env, net);
Parameters  
env
            A pointer to the CPLEX environment as returned by CPXopenCPLEX.
net
            A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
Returns          The value CPX_MAX (-1) is returned for a maximization problem; the value CPX_MIN 
                  (1) is returned for a minimization problem. In case of an error, the value zero is returned.
CPXNETgetprobname

Category            Global Function
Definition File     cplex.h
Include Files       cplex.h
Synopsis             int CPXPUBLIC CPXNETgetprobname(CPXENVptr env,
                                                        CPXNETptr net,
                                                        char * buf_str,
                                                        int bufspace,
                                                        int * surplus_p)
Description          The routine CPXNETgetprobname is used to access the name of the problem stored
                                                        in a network problem object.
Example              status = CPXNETgetprobname (env, net, name, namesize, &surplus);

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
net
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
buf_str
Buffer into which the problem name is copied.
bufspace
Size of the array buf_str in bytes.
surplus_p
Pointer to an integer in which the difference between bufspace and the number of
characters required to store the problem name is returned. A nonnegative value indicates
that bufspace was sufficient. A negative value indicates that it was insufficient. In that
case, CPXERR_NEGATIVE_SURPLUS is returned and the negative value of surplus_p indicates the amount of insufficient space in the array buf.

Returns
The routine returns zero on success and nonzero if an error occurs. The value
CPXERR_NEGATIVE_SURPLUS indicates that there was not enough space in the buf
array to hold the name.
CPXNETgetsupply

Category       Global Function
Definition File cplex.h
Include Files  cplex.h
Synopsis       int CPXPUBLIC CPXNETgetsupply(CPXENVptr env, CPXNETptr net, double * supply, int begin, int end)
Description    The routine CPXNETgetsupply is used to obtain supply values for a range of nodes in the network stored in a CPLEX network problem object.
Example        status = CPXNETgetsupply (env, net, supply, 0, CPXNETgetnumnodes (env, net) - 1);
Parameters     env
                A pointer to the CPLEX environment as returned by CPXopenCPLEX.
                net
                A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
                supply
                Place where requested supply values are copied. If NULL is passed, no supply values are copied. Otherwise, the array must be of length at least (end-begin+1).
                begin
                Index of the first node for which a supply value is to be obtained.
                end
                Index of the last node for which a supply value is to be obtained.
Returns        The routine returns zero on success and nonzero if an error occurs.
CPXNETgetub

Category: Global Function
Definition File: cplex.h
Include Files: cplex.h

Synopsis:
```c
int CPXPUBLIC CPXNETgetub(CPXENVptr env,
                        CPXNETptr net,
                        double * up,
                        int begin,
                        int end)
```

Description:
The routine CPXNETgetub is used to access the upper capacity bounds for a range of arcs in the network stored in a network problem object.

Example:
```c
status = CPXNETgetub (env, net, up, 0, cur_narcs-1);
```

Parameters:
- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **net**
  A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
- **up**
  Array in which to write the upper bound on the flow for the requested arcs. If NULL is passed, no upper bounds are retrieved. Otherwise, the array must be of size (end-begin+1).
- **begin**
  Index of the first arc for which upper bounds are to be obtained.
- **end**
  Index of the last arc for which upper bounds are to be obtained.

Returns:
The routine returns zero on success and nonzero if an error occurs.
Group optim.cplex.callable.manageparameters

The routines in the ILOG CPLEX Callable Library to manage parameters (that is, set parameters, get current values of parameters, and get information about parameters).

Global Functions Summary

<table>
<thead>
<tr>
<th>Global Functions Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXgetdblparam</td>
</tr>
<tr>
<td>CPXgetintparam</td>
</tr>
<tr>
<td>CPXgetparamname</td>
</tr>
<tr>
<td>CPXgetparamnum</td>
</tr>
<tr>
<td>CPXgetstrparam</td>
</tr>
<tr>
<td>CPXinfodb1param</td>
</tr>
<tr>
<td>CPXinfointparam</td>
</tr>
<tr>
<td>CPXinfostrparam</td>
</tr>
<tr>
<td>CPXreadcopyparam</td>
</tr>
<tr>
<td>CPXsetdblparam</td>
</tr>
<tr>
<td>CPXsetdefaults</td>
</tr>
<tr>
<td>CPXsetintparam</td>
</tr>
<tr>
<td>CPXsetstrparam</td>
</tr>
<tr>
<td>CPXwriteparam</td>
</tr>
</tbody>
</table>

Description

These routines are used to set parameters that control various aspects of ILOG CPLEX behavior and to find out current, default, and allowed values for parameters. For more information about parameters, see the reference manual *ILOG CPLEX Parameters*. 
CPXgetdblparam

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis: int CPXPUBLIC CPXgetdblparam(CPXENVptr env, int whichparam, double * value_p)

Description: The routine CPXgetdblparam is used to obtain the current value of a CPLEX parameter of type double.

The reference manual ILOG CPLEX Parameters provides a list of parameters with their types, options, and default values.

Example:

status = CPXgetdblparam (env, CPX_PARAM_TILIM, &curtilim);

Parameters:

- env
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- whichparam
  The symbolic constant (or reference number) of the parameter for which the value is to be obtained.
- value_p
  A pointer to a variable of type double to hold the current value of the CPLEX parameter.

Returns: The routine returns zero on success and nonzero if an error occurs.
CPXgetintparam

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXgetintparam(CPXENVptr env,
                               int whichparam,
                               int * value_p)
```

Description:

The routine `CPXgetintparam` is used to obtain the current value of a CPLEX parameter of type `int`.

The reference manual *ILOG CPLEX Parameter* provides a list of parameters with their types, options, and default values.

Example:

```c
status = CPXgetintparam (env, CPX_PARAM_PREIND, &curpreind);
```

Parameters:

- `env`
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- `whichparam`
  The symbolic constant (or reference number) of the parameter for which the value is to be obtained.

- `value_p`
  A pointer to an integer variable to hold the current value of the CPLEX parameter.

Returns:

The routine returns zero on success and nonzero if an error occurs.
**CPXgetparamname**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXgetparamname(CPXENVptr env,
        int whichparam,
        char * name_str)
```

**Description**  
The routine `CPXgetparamname` returns the name of a CPLEX parameter, given the symbolic constant (or reference number) for it.

The reference manual *ILOG CPLEX Parameters* provides a list of parameters with their types, options, and default values.

**Example**  
```c
status = CPXgetparamname (env, CPX_PARAM_ADVIND, param_string);
```

**Parameters**

- **env**  
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- **whichparam**  
  An integer indicating the symbolic constant (or reference number) of the desired parameter.

- **name_str**  
  A character array to receive the name of the selected parameter.

**Returns**  
The routine returns zero on success and nonzero if an error occurs.
**CPXgetparamnum**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXgetparamnum(CPXENVptr env,
    const char * name_str,
    int * whichparam_p)
```

**Description**  
The routine `CPXgetparamnum` returns the reference number of a CPLEX parameter, given a character string containing the name for it.

The reference manual *ILOG CPLEX Parameters* provides a list of parameters with their types, options, and default values.

**Example**  
```c
status = CPXgetparamnum (env, "CPX_PARAM_ADVIND", param_number);
```

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- **name_str**
  A character array containing the name of the target parameter.

- **whichparam_p**
  A pointer to an integer to receive the reference number.

**Returns**  
The routine returns zero on success and nonzero if an error occurs.
CPXgetstrparam

Category                 Global Function
Definition File          cplex.h
Include Files           cplex.h
Synopsis                 int CPXPUBLIC CPXgetstrparam(CPXENVptr env,
                                               int whichparam,
                                               char * value_str)
Description              The routine CPXgetstrparam is used to obtain the current value of a CPLEX string parameter.
Example                  status = CPXgetstrparam (env, CPX_PARAM_NODEFILEDIR, dirname);

Parameters              env
                        A pointer to the CPLEX environment as returned by CPXopenCPLEX.
                        whichparam
                        The symbolic constant (or reference number) of the parameter for which the value is to be obtained.
                        value_str
                        A pointer to a buffer of length at least CPX_STR_PARAM_MAX to hold the current value of the CPLEX parameter.

Returns                 The routine returns zero on success and nonzero if an error occurs.
CPXinfodblparam

Category  Global Function
Definition File  cplex.h
Include Files  cplex.h
Synopsis  

```c
int CPXPUBLIC CPXinfodblparam(CPXCENVptr env,
    int whichparam,
    double * defvalue_p,
    double * minvalue_p,
    double * maxvalue_p)
```

Description  The routine CPXinfodblparam is used to obtain the default, minimum, and maximum values of a CPLEX parameter of type double.

Note: Values of zero obtained for both the minimum and maximum values of a parameter of type double indicate that the parameter has no limit.

The reference manual ILOG CPLEX Parameters, provides a list of parameters with their types, options, and default values.

Example

```c
status = CPXinfodblparam (env, CPX_PARAM_TILIM, &default_tilim,
    &min_tilim, &max_tilim);
```

Parameters  

- **env**  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **whichparam**  The symbolic constant (or reference number) of the parameter value to be obtained.
- **defvalue_p**  A pointer to a variable of type double to hold the default value of the CPLEX parameter. May be NULL.
- **minvalue_p**  A pointer to a variable of type double to hold the minimum value of the CPLEX parameter. May be NULL.
maxvalue_p
A pointer to a variable of type double to hold the maximum value of the CPLEX parameter. May be NULL.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXinfointparam

Category       Global Function
Definition File cplex.h
Include Files  cplex.h

Synopsis       int CPXPUBLIC CPXinfointparam(CPXENVptr env,
                     int whichparam,
                     int * defvalue_p,
                     int * minvalue_p,
                     int * maxvalue_p)

Description    The routine CPXinfointparam is used to obtain the default, minimum, and
                maximum values of a CPLEX parameter of type int.
                The reference manual ILOG CPLEX Parameters provides a list of parameters with their
                types, options, and default values.

Example         status = CPXinfointparam (env, CPX_PARAM_PREIND, &default_preind,
                              &min_preind, &max_preind);

Parameters      env
                A pointer to the CPLEX environment as returned by CPXopenCPLEX.

                whichparam
                The symbolic constant (or reference number) of the parameter for which the value is to
                be obtained.

                defvalue_p
                A pointer to an integer variable to hold the default value of the CPLEX parameter. May
                be NULL.

                minvalue_p
                A pointer to an integer variable to hold the minimum value of the CPLEX parameter.
                May be NULL.

                maxvalue_p
                A pointer to an integer variable to hold the maximum value of the CPLEX parameter.
                May be NULL.

Returns        The routine returns zero on success and nonzero if an error occurs.
**CPXinfostrparam**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**
```c
int CPXPUBLIC CPXinfostrparam(CPXENVptr env,
                               int whichparam,
                               char * defvalue_str)
```

**Description**
The routine `CPXinfostrparam` is used to obtain the default value of a CPLEX string parameter.

**Example**
```c
status = CPXinfostrparam (env, CPX_PARAM_NODEFILEDIR, defdirname);
```

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- **whichparam**
  The symbolic constant (or reference number) of the parameter for which the default value is to be obtained.

- **defvalue_str**
  A pointer to a buffer of length at least `CPX_STR_PARAM_MAX` to hold the default value of the CPLEX parameter.

**Returns**
The routine returns zero on success and nonzero if an error occurs.
**CPXreadcopyparam**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXreadcopyparam(CPXENVptr env,  
                             const char * filename_str)
```

**Description**  
The routine CPXreadcopyparam reads parameter names and settings from the file indicated by `filename_str` and copies them into CPLEX.

This routine reads and copies files in the format created by **CPXwriteparam**. The file extension is `.prm`. The PRM format is documented in the reference manual *ILOG CPLEX File Formats*.

**Parameters**

`env`

A pointer to the CPLEX environment as returned by **CPXopenCPLEX**.

`filename_str`

Pointer to the file to read and copy into CPLEX.
**CPXsetdblparam**

**Category**   Global Function

**Definition File**   cplex.h

**Include Files**   cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXsetdblparam(CPXENVptr env, 
    int whichparam, 
    double newvalue)
```

**Description**

The routine `CPXsetdblparam` sets the value of a CPLEX parameter of type `double`. The reference manual *ILOG CPLEX Parameters* provides a list of parameters with their types, options, and default values.

**Example**

```c
status = CPXsetdblparam (env, CPX_PARAM_TILIM, 1000.0);
```

**Parameters**

- `env`  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
- `whichparam`  The symbolic constant (or reference number) of the parameter to change.
- `newvalue`  The new value of the parameter.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXsetdefaults

Category          Global Function
Definition File   cplex.h
Include Files     cplex.h
Synopsis          int CPXPUBLIC CPXsetdefaults(CPXENVptr env)
Description       The routine CPXsetdefaults resets all CPLEX parameters and settings to default values (with the exception of the log file).

Note: This routine also resets the CPLEX callback functions to NULL.

The reference manual ILOG CPLEX Parameters provides a list of parameters with their types, options, and default values.

Example

    status = CPXsetdefaults (env);

Parameters      env

A pointer to the CPLEX environment as returned by CPXopenCPLEX.

Returns         The routine returns zero on success and nonzero if an error occurs.
CPXsetintparam

Category          Global Function
Definition File   cplex.h
Include Files     cplex.h
Synopsis          int CPXPUBLIC CPXsetintparam(CPXENVptr env,
                              int whichparam,
                              int newvalue)
Description The routine CPXsetintparam sets the value of a CPLEX parameter of type int.
The reference manual ILOG CPLEX Parameters provides a list of parameters with their types, options, and default values.
Example

    status = CPXsetintparam (env, CPX_PARAM_SCRIND, CPX_ON);

See also lpex1.c in the CPLEX User's Manual.

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

whichparam
The symbolic constant (or reference number) of the parameter to change.

newvalue
The new value of the parameter.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXsetstrparam

Category Global Function

Definition File cplex.h

Include Files cplex.h

Synopsis

int CPXPUBLIC CPXsetstrparam(CPXENVptr env,
   int whichparam,
   const char * newvalue_str)

Description

The routine CPXsetstrparam sets the value of a CPLEX string parameter.

Example

   status = CPXsetstrparam (env, CPX_PARAM_WORKDIR, "mydir");

Parameters

   env

   A pointer to the CPLEX environment as returned by CPXopenCPLEX.

   whichparam

   The symbolic constant (or reference number) of the parameter to change.

   newvalue_str

   The new value of the parameter. The maximum length of newvalue_str, including
   the NULL terminator (the character '0' or char(0)), is CPX_STR_PARAM_MAX,
   defined in cplex.h. Setting newvalue_str to a string longer than this results in an
   error.

Returns

   The routine returns zero on success and nonzero if an error occurs.
CPXwriteparam

Category        Global Function
Definition File  cplex.h
Include Files   cplex.h
Synopsis
int CPXPUBLIC CPXwriteparam(CPXENVptr env,  
                          const char * filename_str)
Description     The routine CPXwriteparam writes the name and current setting of CPLEX parameters that are not at their default setting in the environment indicated by env.

This routine writes a file in a format suitable for reading by CPXreadcopyparam, so you can save current, nondefault parameter settings for re-use in a later session. The file extension is .prm. The PRM format is documented in the reference manual ILOG CPLEX File Formats.

Parameters
  env
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
  filename_str
  Pointer to the file to write.

Group optim.cplex.callable.modifyproblem

The routines in the ILOG CPLEX Callable Library to modify a problem created by CPXcreateprob.

<table>
<thead>
<tr>
<th>Global Functions Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXaddcols</td>
</tr>
<tr>
<td>CPXaddrows</td>
</tr>
<tr>
<td>CPXaddsos</td>
</tr>
<tr>
<td>CPXchgbdss</td>
</tr>
<tr>
<td>CPXchgcoefs</td>
</tr>
<tr>
<td>CPXchgcoefslist</td>
</tr>
<tr>
<td>CPXchgcolname</td>
</tr>
<tr>
<td>CPXchgobjjsen</td>
</tr>
</tbody>
</table>
Problem modification routines change a problem object after it has been created. The modifications that you can make are these:

- adding rows and columns to the constraint matrix,
- deleting rows and columns from the constraint matrix,
- changing the sense of the objective function,
- changing the value of coefficients in the constraint matrix,
- changing an objective or right-hand side coefficient,
- changing the bounds on a variable,
- changing the sense of a constraint,
- changing names of rows or columns.
**CPXaddcols**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXaddcols (CPXCENVptr env,  
CPXLPptr lp,  
int ccnt,  
int nzcnt,  
const double * obj,  
const int * cmatbeg,  
const int * cmatind,  
const double * cmatval,  
const double * lb,  
const double * ub,  
char ** colname)
```

**Description**  
The routine **CPXaddcols** adds columns to a specified CPLEX problem object. This routine may be called any time after a problem object is created via **CPXcreateprob**.

The routine **CPXaddcols** is very similar to the routine **CPXaddrows**. The primary difference is that **CPXaddcols** cannot add coefficients in rows that do not already exist (that is, in rows with index greater than the number returned by **CPXgetnumrows**); whereas **CPXaddrows** can add coefficients in columns with index greater than the value returned by **CPXgetnumcols**, by the use of the **ccnt** argument. (See the discussion of the **ccnt** argument for **CPXaddrows**.) Thus, **CPXaddcols** has no variable **rcnt** and no array **rowname**.

The routine **CPXnewrows** can be used to add empty rows before adding new columns via **CPXaddcols**.

The nonzero elements of every column must be stored in sequential locations in the array **cmatval** from position **cmatbeg[i]** to **cmatbeg[i+1]** (or from **cmatbeg[i]** to **nzcnt-1 if i=ccnt-1**). Each entry, **cmatind[i]**, indicates the row number of the corresponding coefficient, **cmatval[i]**. Unlike **CPXcopylp**, all columns must be contiguous, and **cmatbeg[0]** must be 0.

**Example**
```c
status = CPXaddcols (env, lp, ccnt, nzcnt, obj, cmatbeg,  
cmatind, cmatval, lb, ub, newcolname);
```

**Parameters**

* env
A pointer to the CPLEX environment as returned by the CPXopenCPLEX routine.

lp

A pointer to a CPLEX problem object as returned by CPXcreateprob.

ccnt

An integer that indicates the number of new columns being added to the constraint matrix.

nzcnt

An integer that indicates the number of nonzero constraint coefficients to be added to the constraint matrix.

obj

An array of length ccnt containing the objective function coefficients of the new variables. May be NULL, in which case, the objective coefficients of the new columns are set to 0.0.

cmatbeg

Array that specifies the nonzero elements of the columns being added.

cmatind

Array that specifies the nonzero elements of the columns being added.

cmatval

Array that specifies the nonzero elements of the columns being added. The format is similar to the format used to specify the constraint matrix in the routine CPXcopylp. (See description of matbeg, matcnt, matind, and matval in that routine).

lb

An array of length ccnt containing the lower bound on each of the new variables. Any lower bound that is set to a value less than or equal to that of the constant CPX_INFBOUND is treated as negative infinity. CPX_INFBOUND is defined in the header file cplex.h. May be NULL, in which case the lower bounds of the new columns are set to 0.0.

ub

An array of length ccnt containing the upper bound on each of the new variables. Any upper bound that is set to a value greater than or equal to that of the constant CPX_INFBOUND is treated as infinity. CPX_INFBOUND is defined in the header file cplex.h. May be NULL, in which case the upper bounds of the new columns are set to CPX_INFBOUND (positive infinity).

colname
An array of length `ccnt` containing pointers to character strings that specify the names of the new variables added to the problem object. May be NULL, in which case the new columns are assigned default names if the columns already resident in the CPLEX problem object have names; otherwise, no names are associated with the variables. If column names are passed to `CPXaddcols` but existing variables have no names assigned, default names are created for them.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXaddrows

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXaddrows (CPXCENVptr env,
    CPXLPptr lp,
    int ccnt,
    int rcnt,
    int nzcnt,
    const double * rhs,
    const char * sense,
    const int * rmatbeg,
    const int * rmatind,
    const double * rmatval,
    char ** colname,
    char ** rowname)
```

Description:
The routine CPXaddrows adds constraints to a specified CPLEX problem object. This routine may be called any time after a call to CPXcreateprob.

When you add a ranged row, CPXaddrows sets the corresponding range value to 0 (zero). Use the routine CPXchgrngval to change the range value.

Values of sense:

<table>
<thead>
<tr>
<th>sense[i]</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'L'</td>
<td>≤ constraint</td>
</tr>
<tr>
<td>'E'</td>
<td>= constraint</td>
</tr>
<tr>
<td>'G'</td>
<td>≥ constraint</td>
</tr>
<tr>
<td>'R'</td>
<td>ranged constraint</td>
</tr>
</tbody>
</table>

Note: The use of CPXaddrows as a way to add new columns is discouraged in favor of a direct call to CPXnewcols before calling CPXaddrows.

Example:

```c
status = CPXaddrows (env, lp, ccnt, rcnt, nzcnt, rhs,
    sense, rmatbeg, rmatind, rmatval,
```
newcolname, newrowname);

See also the example lpex3.c in the *ILOG CPLEX User's Manual* and in the standard distribution.

### Parameters

**env**

A pointer to the CPLEX environment as returned by CPXopenCPLEX.

**lp**

A pointer to a CPLEX problem object as returned by CPXcreateprob.

**ccnt**

An integer that indicates the number of new columns in the constraints being added to the constraint matrix. When new columns are added, they are given an objective coefficient of zero, a lower bound of zero, and an upper bound of CPX_INFBOUND.

**rcnt**

An integer that indicates the number of new rows to be added to the constraint matrix.

**nzcnt**

An integer that indicates the number of nonzero constraint coefficients to be added to the constraint matrix. This specifies the length of the arrays rmatind and rmatval.

**rhs**

An array of length rcnt containing the right-hand side term for each constraint to be added to the CPLEX problem object. May be NULL, in which case the new right-hand side values are set to 0.0.

**sense**

An array of length rcnt containing the sense of each constraint to be added to the CPLEX problem object. May be NULL, in which case the new constraints are created as equality constraints. Possible values of this argument appear in the table.

**rmatbeg**

An array used with rmatind and rmatval to define the rows to be added.

**rmatind**

An array used with rmatbeg and rmatval to define the rows to be added.

**rmatval**

An array used with rmatbeg and rmatind to define the rows to be added. The format is similar to the format used to describe the constraint matrix in the routine CPXcopylp (see description of matbeg, matcnt, matind, and matval in that routine), but the nonzero coefficients are grouped by row instead of column in the array rmatval. The
nonzero elements of every row must be stored in sequential locations in this array from position rmatbeg[i] to rmatbeg[i+1]-1 (or from rmatbeg[i] to nzcnt -1 if i=rcnt-1). Each entry, rmatind[i], indicates the column index of the corresponding coefficient, rmatval[i]. Unlike CPXcopylp, all rows must be contiguous, and rmatbeg[0] must be 0.

colname
An array of length ccnt containing pointers to character strings that represent the names of the new columns added to the CPLEX problem object, or equivalently, the new variable names. May be NULL, in which case the new columns are assigned default names if the columns already resident in the CPLEX problem object have names; otherwise, no names are associated with the variables. If column names are passed to CPXaddrows but existing variables have no names assigned, default names are created for them.

rowname
An array containing pointers to character strings that represent the names of the new rows, or equivalently, the constraint names. May be NULL, in which case the new rows are assigned default names if the rows already resident in the CPLEX problem object have names; otherwise, no names are associated with the constraints. If row names are passed to CPXaddrows but existing constraints have no names assigned, default names are created for them.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXaddsos

Category Global Function
Definition File cplex.h
Include Files cplex.h
Synopsis
```
int CPXPUBLIC CPXaddsos (CPXCENVptr env,
    CPXLPptr lp,
    int numsos,
    int numsosnz,
    const char * sostype,
    const int * sospri,
    const int * sosbeg,
    const int * sosind,
    const double * soswt)
```

Description
The routine CPXaddsos adds information about a Special Ordered Set (SOS) to a problem object of type CPXPROB_MILP, CPXPROB_MIQP, or CPXPROB_MIQCP. The problem may already contain SOS information.

### Table 1: Values of elements of sostype

<table>
<thead>
<tr>
<th>CPX_TYPE_SOS1</th>
<th>'1'</th>
<th>Type 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_TYPE_SOS2</td>
<td>'2'</td>
<td>Type 2</td>
</tr>
</tbody>
</table>

The arrays sosbeg, sosind, and soswts follow the same conventions as similar arrays in other routines of the Callable Library. For \( j < \text{numsos}-1 \) the indices of set \( j \) must be stored in sosind[sosbeg[j]],..., sosind[sosbeg[j+1]-1] and the weights in soswt[sosbeg[j]],..., soswt[sosbeg[j+1]-1]. For the last set, \( j = \text{numsos}-1 \), the indices must be stored in sosind[sosbeg[\text{numsos}-1]],..., sosind[\text{numsosnz}-1] and the corresponding weights in soswt[sosbeg[\text{numsos}-1]],..., soswt[\text{numsosnz}-1]. Hence, the length of sosbeg must be at least numsos, while the lengths of sosind and soswt must be at least numsosnz.

Example

```
status = CPXaddsos (env, lp, numsos, numsosnz, sostype, sospri, 
    sosbeg, sosind, soswt);
```

Parameters
- **env**
  A pointer to the CPLEX environment as returned by CPXopenCplex.
CPXaddsos

lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

numsos
The number of sets to be added to existing SOS sets, if any.

numzosnz
The total number of members in all of the sets to be added to existing SOS sets, if any.

sostype
An array containing SOS type information for the sets to be added. According to Table 1, sostype[i] specifies the SOS type of set i. The length of this array must be at least numsos.

sospri
An array containing priority values for each set to be added. sospri[i] specifies the priority for set i, and may take any nonnegative value. This array may be NULL; otherwise, its length must be at least numsos.

sosbeg
An array that with sosind and soswt defines the weights for the sets to be added.

sosind
An array that with sosbeg and soswt defines the weights of the sets to be added.

soswt
An array that with sosbeg and sosind defines the indices and weights for the sets to be added. The indices of each set must be stored in sequential locations in sosind. The weights of each set must be stored in sequential locations in soswt. The array sosbeg[j] containing the index of the beginning of set j. The weights must be unique within each set.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXchgbds

Category Global Function

Definition File cplex.h

Include Files cplex.h

Synopsis

```c
int CPXPUBLIC CPXchgbds (CPXCENVptr env,
            CPXLPptr lp,
            int cnt,
            const int * indices,
            const char * lu,
            const double * bd)
```

Description

The routine CPXchgbds is used to change the upper or lower bounds on a set of variables of a problem. Several bounds can be changed at once, with each bound specified by the index of the variable with which it is associated. The value of a variable can be fixed at one value by setting the upper and lower bounds to the same value.

Example

```c
status = CPXchgbds (env, lp, cnt, indices, lu, bd);
```

<table>
<thead>
<tr>
<th>Values of lu indicating lower or upper bound in indices[j]</th>
</tr>
</thead>
<tbody>
<tr>
<td>lu[j]</td>
</tr>
<tr>
<td>lu[j]</td>
</tr>
<tr>
<td>lu[j]</td>
</tr>
</tbody>
</table>

Parameters

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **lp**
  A pointer to a CPLEX problem object as returned by CPXcreateprob.
- **cnt**
  An integer that indicates the total number of bounds to be changed, and thus specifies the length of the arrays indices, lu, and bd.
- **indices**
An array of length \( \text{cnt} \) containing the numeric indices of the columns corresponding to the variables for which bounds are to be changed.

\( \text{lu} \)

An array of length \( \text{cnt} \) containing characters that indicate whether the corresponding entry in the array \( \text{bd} \) specifies the lower or upper bound on column \( \text{indices}[j] \). Possible values appear in the table.

\( \text{bd} \)

An array of length \( \text{cnt} \) containing the new values of the upper or lower bounds of the variables present in \( \text{indices} \).

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXchgcoef

Category      Global Function
Definition File cplex.h
Include Files cplex.h

Synopsis      

```
int CPXPUBLIC CPXchgcoef (CPXCENVptr env,
        CPXLPptr lp,
        int i,
        int j,
        double newvalue)
```

Description      The routine CPXchgcoef is used to change a single coefficient in the constraint matrix, linear objective coefficients, right-hand side or ranges of a CPLEX problem object. The coefficient is specified using its coordinates in the constraint matrix. When changing matrix coefficients from zero to nonzero values, be sure that the corresponding row and column indices exist in the problem, so that 

\[-1 \leq i < \text{CPXgetnumrows}(env, lp)\]  
\[-2 \leq j < \text{CPXgetnumcols}(env, lp)\]

Example      

```
status = CPXchgcoef (env, lp, 10, 15, 23.2);
```

See Also      CPXchgobj, CPXchgrhs, CPXchgrngval

Parameters      

```
  env
    A pointer to the CPLEX environment as returned by CPXopenCPLEX.

  lp
    A pointer to a CPLEX problem object as returned by CPXcreateprob.

  i
    An integer that indicates the numeric index of the row in which the coefficient is located. The linear objective row is referenced with \(i = -1\).

  j
    An integer that indicates the numeric index of the column in which the coefficient is located. The RHS column is referenced with \(j = -1\). The range value column is referenced with \(j = -2\). If \(j = -2\) is specified and row \(i\) is not a ranged row, an error status is returned.

  newvalue
```
The new value for the coefficient being changed.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXchgcoeflist

Category    Global Function
Definition File    cplex.h
Include Files    cplex.h
Synopsis    int CPXPUBLIC CPXchgcoeflist (CPXCENVptr env, CPXLPptr ip, int numcoefs, const int * rowlist, const int * collist, const double * vallist)

Description    The routine CPXchgcoeflist is used to change a list of matrix coefficients of a CPLEX problem object. The list is prepared as a set of triples (i, j, value), where i is the row index, j is the column index, and value is the new value. The list may be in any order.

Note: The corresponding rows and columns must already exist in the CPLEX problem object.
This routine cannot be used to change objective, right-hand side, range, or bound coefficients.
Duplicate entries, that is, two triplets with identical i and j, are not allowed.

Example    status = CPXchgcoeflist (env, ip, numcoefs, rowlist, collist, vallist);

Parameters    env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
ip
A pointer to a CPLEX problem object as returned by CPXcreateprob.
numcoefs
The number of coefficients to change, or, equivalently, the length of the arrays rowlist, collist, and vallist.
rowlist
An array of length numcoefs that with collist and vallist indicates the coefficients to change.

collist
An array of length numcoefs that with rowlist and vallist indicates the coefficients to change.

vallist
An array of length numcoefs that with rowlist and collist indicates the coefficients to change. The entries rowlist[k], collist[k], and vallist[k] indicate that the matrix coefficient in row rowlist[k] and column collist[k] should be changed to the value vallist[k].

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXchgcolname

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**
```c
int CPXPUBLIC CPXchgcolname(CPXENVptr env,
                           CPXLPptr lp,
                           int cnt,
                           const int * indices,
                           char ** newname)
```

**Description**
The routine CPXchgcolname changes the names of variables in a CPLEX problem object. If this routine is performed on a problem object with no variable names, default names are created before the change is made.

**See Also**
CPXdelnames

**Parameters**
- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **lp**
  A pointer to a CPLEX problem object as returned by CPXcreateprob.
- **cnt**
  An integer that indicates the total number of variable names to be changed. Thus cnt specifies the length of the arrays indices and newname.
- **indices**
  An array of length cnt containing the numeric indices of the variables for which the names are to be changed.
- **newname**
  An array of length cnt containing the strings of the new variable names for the columns specified in indices.

**Returns**
The routine returns zero on success and nonzero if an error occurs.
CPXchcname

Category          Global Function
Definition File   cplex.h
Include Files     cplex.h

Synopsis
int CPXPUBLIC CPXchcname(CPXCENVptr env,
                          CPXLPptr lp,
                          int key,
                          int ij,
                          const char * newname_str)

Description
The routine CPXchcname changes the name of a constraint or the name of a variable in a CPLEX problem object. If this routine is performed on a problem object with no row or column names, default names are created before the change is made.

Example
status = CPXchcname (env, lp, 'c', 10, "name10");

Values of key

<table>
<thead>
<tr>
<th>key</th>
<th>change</th>
</tr>
</thead>
<tbody>
<tr>
<td>'r'</td>
<td>row name</td>
</tr>
<tr>
<td>'c'</td>
<td>column name</td>
</tr>
</tbody>
</table>

See Also
CPXdelnames

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

key
A character to indicate whether a row name or a column name should be changed. Possible values appear in the table.

ij
An integer that indicates the numeric index of the column or row for which the name is to be changed.
newname_str

A pointer to a character string containing the new name.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXchgobjsen

Category  Global Function
Definition File  cplex.h
Include Files  cplex.h
Synopsis  void CPXPUBLIC CPXchgobjsen(CPXENVptr env, CPXLPptr lp, int maxormin)
Description  The routine CPXchgobjsen is used to change the sense of the optimization for a problem, to maximization or minimization.

Note: For problems with a quadratic objective function, changing the objective sense may make the problem unsolvable. Further changes to the quadratic coefficients may then be required to restore the convexity (concavity) of a minimization (maximization) problem.

Values of maxormin

<table>
<thead>
<tr>
<th>CPX_MIN</th>
<th>CPX_MAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>(-1)</td>
</tr>
<tr>
<td>new sense is minimize</td>
<td>new sense is maximize</td>
</tr>
</tbody>
</table>

Example

```c
CPXchgobjsen (env, lp, CPX_MAX);
```

Parameters

- **env**  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **lp**  A pointer to a CPLEX problem object as returned by CPXcreateprob.
- **maxormin**  An integer that indicates the new sense of the problem.

Returns  This routine does not return a result.
CPXchgprobname

Category               Global Function
Definition File        cplex.h
Include Files          cplex.h
Synopsis

int CPXPUBLIC CPXchgprobname(CPXENVptr env,
                             CPXLPptr lp,
                             const char * probname_str)

Description
The routine CPXchgprobname is used to change the name of the current problem.

Example

    status = CPXchgprobname (env, lp, probname);

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

probname_str
The new name of the problem.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXchgprobtype

Category  Global Function
Definition File  cplex.h
Include Files  cplex.h
Synopsis  int CPXPUBLIC CPXchgprobtype(CPXENVptr env, CPXLPptr lp, int type)
Description  The routine CPXchgprobtype is used to change the current problem to a related problem. The problem types that can be used appear in the table.

Table 1: Problem Types

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CPXPROB_LP</td>
<td>Linear program, no ctype or quadratic data stored.</td>
</tr>
<tr>
<td>1</td>
<td>CPXPROB_MILP</td>
<td>Problem with ctype information.</td>
</tr>
<tr>
<td>3</td>
<td>CPXPROB_FIXEDMILP</td>
<td>Problem with ctype information, integer variables fixed.</td>
</tr>
<tr>
<td>5</td>
<td>CPXPROB_QP</td>
<td>Problem with quadratic data stored.</td>
</tr>
<tr>
<td>6</td>
<td>CPXPROB_ZEROEDQP</td>
<td>Problem with quadratic data stored, ignoring Q.</td>
</tr>
<tr>
<td>7</td>
<td>CPXPROB_MIQP</td>
<td>Problem with quadratic data and ctype information.</td>
</tr>
<tr>
<td>8</td>
<td>CPXPROB_FIXEDMIQP</td>
<td>Problem with quadratic data and ctype information, integer variables fixed.</td>
</tr>
<tr>
<td>10</td>
<td>CPXPROB_QCP</td>
<td>Problem with quadratic constraints.</td>
</tr>
<tr>
<td>11</td>
<td>CPXPROB_MIQCP</td>
<td>Problem with quadratic constraints and ctype information.</td>
</tr>
</tbody>
</table>

A mixed integer problem (CPXPROB_MILP or CPXPROB_MIQP) can be changed to a problem (CPXPROB_FIXEDMILP or CPXPROB_FIXEDMIQP) where bounds on integer variables are fixed to the values attained in the integer solution. A mixed integer
problem (or its related fixed type) can also be changed to a linear program or a quadratic program (CPXPROB_LP or CPXPROB_QP), which causes any existing ctype values to be permanently discarded from the problem object.

The original mixed integer problem can be recovered from the fixed problem. If the current problem type is CPXPROB_FIXEDMILP or CPXPROB_FIXEDMIQP, any calls to problem modification routines fail. To modify the problem object, the problem type should be changed to CPXPROB_MILP or CPXPROB_MIQP.

Changing a problem from type CPXPROB_LP to CPXPROB_MILP or from type CPXPROB_QP to CPXPROB_MIQP causes a ctype array to be created such that all variables are considered continuous. Changing a problem from type CPXPROB_MILP to CPXPROB_MIQP causes an empty quadratic matrix to be created such that the objective is quadratic with \( Q = 0 \). A problem of type CPXPROB_MILP or CPXPROB_MIQP can be solved only by the routine CPXmipopt.

A quadratic programming problem (CPXPROB_QP) can be changed to a linear program (CPXPROB_LP), causing any existing quadratic information to be permanently discarded from the problem object. Changing a problem from type CPXPROB_LP to CPXPROB_QP causes an empty quadratic matrix to be created such that the problem is quadratic with the matrix \( Q = 0 \).

**Example**

```c
status = CPXchgprobtype (env, lp, CPXPROB_MILP);
```

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **lp**
  A pointer to a CPLEX LP problem object as returned by CPXcreateprob.

- **type**
  An integer indicating the desired problem type. See the previous discussion for possible values.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
**CPXchgpqcoef**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXchgpqcoef (CPXCENVptr env,
       CPXLPptr lp,
       int i,
       int j,
       double newvalue)
```

**Description**  
The routine `CPXchgpqcoef` changes the coefficient in the quadratic objective of a quadratic problem (QP) corresponding to the variable pair \((i, j)\) to the value `newvalue`. If \(i\) is not equal to \(j\), both \(Q(i, j)\) and \(Q(j, i)\) are changed to `newvalue`.

**Example**  
```c
status = CPXchgpqcoef (env, lp, 10, 12, 82.5);
```

**Parameters**  
- **env**  
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
- **lp**  
  A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
- **i**  
  An integer that indicates the first variable number (row number in \(Q\)).
- **j**  
  An integer that indicates the second variable number (column number in \(Q\)).
- **newvalue**  
  The new coefficient value.

**Returns**  
The routine returns zero on success and nonzero if an error occurs.
**CPXchgrhs**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXchgrhs (CPXCENVptr env,
   CPXLPptr lp,
   int cnt,
   const int * indices,
   const double * values)
```

**Description**  
The routine CPXchgrhs is used to change the right-hand side coefficients of a set of linear constraints in the CPLEX problem object.

**Example**  
```c
status = CPXchgrhs (env, lp, cnt, indices, values);
```

**Parameters**

- **env**  
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **lp**  
A pointer to a CPLEX problem object as returned by CPXcreateprob.

- **cnt**  
An integer that indicates the total number of right-hand side coefficients to be changed, and thus specifies the length of the arrays indices and values.

- **indices**  
An array of length cnt containing the numeric indices of the rows corresponding to the linear constraints for which right-hand side coefficients are to be changed.

- **values**  
An array of length cnt containing the new values of the right-hand side coefficients of the linear constraints present in indices.

**Returns**  
The routine returns zero on success and nonzero if an error occurs.
CPXchgrngval

Category                  Global Function
Definition File           cplex.h
Include Files            cplex.h

Synopsis

    int CPXPUBLIC CPXchgrngval (CPXCENVptr env,
                               CPXLPptr lp,
                               int cnt,
                               const int * indices,
                               const double * values)

Description

    The routine CPXchgrngval is used to change the range coefficients of a set of linear constraints in the CPLEX problem object.

Example

    status = CPXchgrngval (env, lp, cnt, indices, values);

Parameters

    env
    A pointer to the CPLEX environment as returned by CPXopenCPLEX.
    lp
    A pointer to a CPLEX problem object as returned by CPXcreateprob.
    cnt
    An integer that indicates the total number of range coefficients to be changed, and thus specifies the length of the arrays indices and values.
    indices
    An array of length cnt containing the numeric indices of the rows corresponding to the linear constraints for which range coefficients are to be changed.
    values
    An array of length cnt containing the new values of the range coefficients of the linear constraints present in indices.

Returns

    The routine returns zero on success and nonzero if an error occurs.
CPXchgrowname

Category       Global Function
Definition File cplex.h
Include Files  cplex.h
Synopsis       int CPXPUBLIC CPXchgrowname(CPXENVptr env,
                         CPXLPptr lp,
                         int cnt,
                         const int * indices,
                         char ** newname)

Description    This routine changes the names of linear constraints in a CPLEX problem object. If this routine is performed on a problem object with no constraint names, default names are created before the change is made.

Example         status = CPXchgrowname (env, lp, cnt, indices, values);

See Also        CPXdelnames

Parameters      env
                 A pointer to the CPLEX environment as returned by CPXopenCPLEX.

                 lp
                 A pointer to a CPLEX problem object as returned by CPXcreateprob.

                 cnt
                 An integer that indicates the total number of linear constraint names to be changed, and thus specifies the length of the arrays indices and newname.

                 indices
                 An array of length cnt containing the numeric indices of the linear constraints for which the names are to be changed.

                 newname
                 An array of length cnt containing the strings of the new names for the linear constraints specified in indices.

Returns         The routine returns zero on success and nonzero if an error occurs.
**CPXchgsense**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXchgsense(CPXENVptr env,  
CPXLPptr lp,  
int cnt,  
const int * indices,  
const char * sense)
```

**Description**  
The routine **CPXchgsense** is used to change the sense of a set of linear constraints of a CPLEX problem object. When changing the sense of a row to ranged, **CPXchgsense** sets the corresponding range value to 0 (zero). The routine **CPXchgrngval** can then be used to change the range value.

**Example**  
```c
status = CPXchgsense (env, lp, cnt, indices, sense);
```

**Values of sense**

<table>
<thead>
<tr>
<th>sense[i]</th>
<th>The new sense is</th>
</tr>
</thead>
<tbody>
<tr>
<td>'L'</td>
<td>≤</td>
</tr>
<tr>
<td>'E'</td>
<td>=</td>
</tr>
<tr>
<td>'G'</td>
<td>≥</td>
</tr>
<tr>
<td>'R'</td>
<td>The constraint is ranged</td>
</tr>
</tbody>
</table>

**Parameters**

- `env`  
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- `lp`  
  A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

- `cnt`  
  An integer that indicates the total number of linear constraints to be changed, and thus represents the length of the arrays `indices` and `sense`.

- `indices`  
  An array of length `cnt` containing the numeric indices of the rows corresponding to the linear constraints which are to have their senses changed.
sense

An array of length cnt containing characters that indicate the new sense of the linear constraints specified in indices. Possible values appear in the table.

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXdelcols

Category: Global Function
Definition File: cplex.h

Synopsis:
```c
int CPXPUBLIC CPXdelcols(CPXENVptr env,
   CPXLPptr lp,
   int begin,
   int end)
```

Description:
The routine CPXdelcols is used to delete all the columns in a specified range. The range is specified using a lower and an upper index that represent the first and last column to be deleted, respectively. The indices of the columns following those deleted are decreased by the number of columns deleted.

Example:
```c
status = CPXdelcols (env, lp, 10, 20);
```

Parameters:
- `env`: A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- `lp`: A pointer to a CPLEX problem object as returned by CPXcreateprob.
- `begin`: An integer that indicates the numeric index of the first column to be deleted.
- `end`: An integer that indicates the numeric index of the last column to be deleted.

Returns:
The routine returns zero on success and nonzero if an error occurs.
CPXdelnames

Category: Global Function

Definition File: cplex.h

Synopsis: int CPXPUBLIC CPXdelnames (CPXCENVptr env, CPXLPptr lp)

Description: The routine CPXdelnames removes all names that have been previously assigned to rows and columns. The memory that was used by those names is released.

Names can be assigned to rows and columns in a variety of ways, and this routine allows them to be removed. For example, if the problem is read from a file in LP or MPS format, names are also read from the file. Names can be assigned by the user by calling one of the routines CPXchgrowname, CPXchgcolname, or CPXchgname.

Example:

CPXdelnames (env, lp);

Parameters:

- env
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- lp
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

Returns:

The routine returns zero on success and nonzero if an error occurs.
CPXdelrows

Category: Global Function
Definition File: cplex.h
Include Files: cplex.h

Synopsis:
```c
int CPXPUBLIC CPXdelrows (CPXCENVptr env,
    CPXLPtr lp,
    int begin,
    int end)
```

Description:
The routine CPXdelrows deletes a range of rows. The range is specified using a lower and upper index that represent the first and last row to be deleted, respectively. The indices of the rows following those deleted are decreased by the number of deleted rows.

Example:
```c
status = CPXdelrows (env, lp, 10, 20);
```

Parameters:
- `env`
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- `lp`
  A pointer to a CPLEX problem object as returned by CPXcreateprob.
- `begin`
  An integer that indicates the numeric index of the first row to be deleted.
- `end`
  An integer that indicates the numeric index of the last row to be deleted.

Returns:
The routine returns zero on success and nonzero if an error occurs.
**CPXdelsetcols**

**Category**  Global Function

**Definition File**  cplex.h

**Include Files**  cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXdelsetcols(CPXENVptr env,
    CPXLPptr lp,
    int * delstat)
```

**Description**  
The routine CPXdelsetcols is used to delete a set of columns from a CPLEX problem object. Unlike the routine CPXdelcols, CPXdelsetcols does not require the columns to be in a contiguous range. After the deletion occurs, the remaining columns are indexed consecutively starting at 0, and in the same order as before the deletion.

**Example**  
```c
status = CPXdelsetcols (env, lp, delstat);
```

**Parameters**

- **env**  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **lp**  A pointer to a CPLEX problem object as returned by CPXcreateprob.
- **delstat**  An array indicating the columns to be deleted. The routine CPXdelsetcols deletes each column j for which delstat[j] = 1. The deletion of columns results in a renumbering of the remaining columns. After termination, delstat[j] is either -1 for columns that have been deleted or the new index number that has been assigned to the remaining columns.

**Returns**  
The routine returns zero on success and nonzero if an error occurs.

**Note:**  The delstat array must have at least CPXgetnumcols(env, lp) elements.
CPXdelsetrows

Category                        Global Function
Definition File                 cplex.h
Include Files                   cplex.h
Synopsis

int CPXPUBLIC CPXdelsetrows(CPXENVptr env,
                           CPXLPtr lp,
                           int * delstat)

Description
The routine CPXdelsetrows deletes a set of rows. Unlike the routine CPXdelrows, CPXdelsetrows does not require the rows to be in a contiguous range. After the deletion occurs, the remaining rows are indexed consecutively starting at 0, and in the same order as before the deletion.

Note: The delstat array must have at least CPXgetnumrows(env, lp) elements.

Example

status = CPXdelsetrows (env, lp, delstat);

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

delstat
An array indicating the rows to be deleted. The routine CPXdelsetrows deletes each row i for which delstat[i] = 1. The deletion of rows results in a renumbering of the remaining rows. After termination, delstat[i] is either -1 for rows that have been deleted or the new index number that has been assigned to the remaining rows.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXdelsetsos

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXdelsetsos(CPXENVptr env, 
   CPXLPptr lp, 
   int * delset)
```

Description:
The routine `CPXdelsetsos` is used to delete a group of special ordered sets (SOSs) from a CPLEX problem object.

**Example**:

```c
status = CPXdelsetsos(env, lp, delstat);
```

Parameters:

- `env`
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- `lp`
  A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

- `delset`
  An array indicating the SOSs to be deleted. The routine `CPXdelsetsos` deletes each SOS $j$ for which $\text{delstat}[j] = 1$. The deletion of SOSs results in a renumbering of the remaining SOSs. After termination, $\text{delstat}[j]$ is either -1 for SOSs that have been deleted or the new index number that has been assigned to the remaining SOSs.

**Note:** The `delstat` array must have at least $\text{CPXgetnumsos}(env, lp)$ elements.

Example:

```c
status = CPXdelsetsos(env, lp, delstat);
```
status = CPXdelsetsos (env, lp, delstat);

**Returns**

The routine returns zero on success and nonzero if an error occurs.
**CPXnewcols**

**Category** Global Function

**Definition File** cplex.h

**Include Files** cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXnewcols(CPXENVptr env, 
CPXLPptr lp, 
int ccnt, 
const double * obj, 
const double * lb, 
const double * ub, 
const char * xctype, 
char ** colname)
```

**Description**

The routine CPXnewcols adds empty columns to a specified CPLEX problem object. This routine may be called any time after a call to CPXcreateprob.

For each column, the user can specify the objective coefficient, the lower and upper bounds, the variable type, and name of the variable. The added columns are indexed to put them at the end of the problem. Thus, if ccnt columns are added to a problem object already having k columns, the new columns have indices k, k+1, ... k+ccnt-1. The constraint coefficients in the new columns are zero; the constraint coefficients can be changed with calls to CPXchgcoef, CPXchgcoeflist or CPXaddrows.

The routine CPXnewcols is very similar to the routine CPXnewrows. It can be used to add variables to a problem object without specifying the matrix coefficients.

### Types of new variables: values of xctype[j]

<table>
<thead>
<tr>
<th>CPX_CONTINUOUS</th>
<th>'C'</th>
<th>continuous variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_BINARY</td>
<td>'B'</td>
<td>binary variable</td>
</tr>
<tr>
<td>CPX_INTEGER</td>
<td>'I'</td>
<td>general integer variable</td>
</tr>
<tr>
<td>CPX_SEMCINT</td>
<td>'S'</td>
<td>semi-continuous variable</td>
</tr>
<tr>
<td>CPX_SEMIINT</td>
<td>'N'</td>
<td>semi-integer variable</td>
</tr>
</tbody>
</table>

**Example**

```c
status = CPXnewcols (env, lp, ccnt, obj, lb, ub, NULL, NULL);
```

See also the example lpex8.c in the *ILOG CPLEX User’s Manual* and in the standard distribution.
Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

ccnt
An integer that indicates the number of new variables being added to the problem object.

obj
An array of length ccnt containing the objective function coefficients of the new variables. This array may be NULL, in which case the new objective coefficients are all set to 0.

lb
An array of length ccnt containing the lower bound on each of the new variables. Any lower bound that is set to a value less than or equal to that of the constant -CPX_INFBOUND is treated as -?. CPX_INFBOUND is defined in the header file cplex.h. This array may be NULL, in which case the new lower bounds are all set to 0 (zero).

ub
An array of length ccnt containing the upper bound on each of the new variables. Any upper bound that is set to a value greater than or equal to that of the constant CPX_INFBOUND is treated as ?. CPX_INFBOUND is defined in the header file cplex.h. This array may be NULL, in which case the new upper bounds are all set to CPX_INFBOUND.

xctype
An array of length ccnt containing the type of each of the new variables. Possible values appear in the table. This array may be NULL, in which case the new variables are created as continuous type.

colname
An array of length ccnt containing pointers to character strings that specify the names of the new variables added to the problem object. May be NULL, in which case the new columns are assigned default names if the columns already resident in the problem object have names; otherwise, no names are associated with the variables. If column names are passed to CPXnewcols but existing variables have no names assigned, default names are created for the existing variables.

Returns
The routine returns zero on success and nonzero if an error occurs.
## CPXnewrows

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXnewrows(CPXENVptr env,  
    CPXLPptr lp,  
    int rcnt,  
    const double * rhs,  
    const char * sense,  
    const double * rngval,  
    char ** rowname)
```

**Description**  
The routine `CPXnewrows` adds empty constraints to a specified CPLEX problem object. This routine may be called any time after a call to `CPXcreateprob`.

For each row, the user can specify the sense, right-hand side value, range value and name of the constraint. The added rows are indexed to put them at the end of the problem. Thus, if `rcnt` rows are added to a problem object already having `k` rows, the new rows have indices `k, k+1, ..., k+rcnt-1`. The constraint coefficients in the new rows are zero; the constraint coefficients can be changed with calls to `CPXchgcoef`, `CPXchgcoeflist` or `CPXaddcols`.

**Table 1: Settings for elements of the array sense**

<table>
<thead>
<tr>
<th><code>sense[i]</code></th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>='L'</code></td>
<td></td>
<td>≤ constraint</td>
</tr>
<tr>
<td><code>='E'</code></td>
<td></td>
<td>= constraint</td>
</tr>
<tr>
<td><code>='G'</code></td>
<td></td>
<td>≥ constraint</td>
</tr>
<tr>
<td><code>='R'</code></td>
<td></td>
<td>ranged constraint</td>
</tr>
</tbody>
</table>

**Example**
```c
closure = CPXnewrows (env, lp, rcnt, rhs, sense, NULL, newrowname);
```

See also the example `lpex1.c` in the *ILOG CPLEX User's Manual* and in the standard distribution.

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
- **lp**
A pointer to a CPLEX problem object as returned by CPXcreateprob.

rcnt
An integer that indicates the number of new rows to be added to the problem object.

rhs
An array of length rcnt containing the right-hand side term for each constraint to be added to the problem object. May be NULL, in which case the right-hand side terms are set to 0.0 for the new constraints.

sense
An array of length rcnt containing the sense of each constraint to be added to the problem object. This array may be NULL, in which case the sense of each constraint is set to 'E'. The values of the elements of this array appear in Table 1.

rngval
An array of length rcnt containing the range values for the new constraints. If a new constraint has sense[i] = 'R', the value of constraint i can be between rhs[i] and rhs[i] + rngval[i]. May be NULL, in which case the range values are all set to zero.

rowname
An array of length rcnt containing pointers to character strings that represent the names of the new rows, or equivalently, the constraint names. May be NULL, in which case the new rows are assigned default names if the rows already resident in the problem object have names; otherwise, no names are associated with the constraints. If row names are passed to CPXnewrows but existing constraints have no names assigned, default names are created for the existing constraints.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXprechgobj

Category       Global Function
Definition File cplex.h
Include Files  cplex.h
Synopsis       

```
int CPXPUBLIC CPXprechgobj(CPXENVptr env, 
    CPXLPptr lp, 
    int cnt, 
    const int * indices, 
    const double * values)
```

Description

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXprechgobj is used to change the objective function coefficients of an LP problem object and its associated presolved LP problem object. Note that the CPLEX parameter CPX_PARAM_REDUCE must be set to CPX_PREREDUCE_PRIMALONLY (1) or CPX_PREREDUCE_NOPRIMALORDUAL (0) at the time of the presolve in order to change objective coefficients and preserve the presolved problem. This routine should be used in place of CPXchgbobj when it is desired to preserve the presolved problem.

The arguments and operation of CPXprechgobj are the same as those of CPXchgbobj. The objective coefficient changes are applied to both the original LP problem object and the associated preserved LP problem object.

**Example**

```
status = CPXprechgobj (env, lp, objcnt, objind, objval);
```

See also the example adpreex1.c in the standard distribution.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
Group optim.cplex.callable.modifynetwork

The routines in the ILOG CPLEX Callable Library to modify a network.

<table>
<thead>
<tr>
<th>Global Functions Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXNETchgarname</td>
</tr>
<tr>
<td>CPXNETchgarcnodes</td>
</tr>
<tr>
<td>CPXNETchgbd</td>
</tr>
<tr>
<td>CPXNETchgname</td>
</tr>
<tr>
<td>CPXNETchnodename</td>
</tr>
<tr>
<td>CPXNETchobj</td>
</tr>
<tr>
<td>CPXNETchgojsen</td>
</tr>
<tr>
<td>CPXNETchgsupply</td>
</tr>
<tr>
<td>CPXNETdelarcs</td>
</tr>
<tr>
<td>CPXNETdelnodes</td>
</tr>
<tr>
<td>CPXNETdelset</td>
</tr>
</tbody>
</table>

Description
After you have created a network problem object, use these routines to modify it.
CPXNETchgarcname

Category            Global Function
Definition File      cplex.h
Include Files        cplex.h
Synopsis             int CPXPUBLIC CPXNETchgarcname(CPXENVptr env,
                                 CPXNETptr net,
                                 int cnt,
                                 const int * indices,
                                 char ** newname)

Description          This routine CPXNETchgarcname changes the names of a set of arcs in
                      the network stored in a network problem object.
Example               status = CPXNETchgarcname (env, net, 10, indices, newname);

Parameters
env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

net
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

cnt
An integer that indicates the total number of arc names to be changed. Thus cnt
specifies the length of the arrays indices and anames.

indices
An array of length cnt containing the numeric indices of the arcs for which the names
are to be changed.

newname
An array of length cnt containing the strings of the new arc names specified in
indices.

Returns               The routine returns zero on success and nonzero if an error occurs.


**CPXNETchgarcnodes**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```
int CPXPUBLIC CPXNETchgarcnodes(CPXCENVptr env,
    CPXNETptr net,
    int cnt,
    const int * indices,
    const int * fromnode,
    const int * tonode)
```

**Description**  
The routine CPXNETchgarcnodes can be used to change the nodes for a set of arcs in the network stored in a network problem object.

Any solution information stored in the problem object is lost.

**Example**  
```
status = CPXNETchgarcs (env, net, cnt, indices, newfrom, newto);
```

**Parameters**  

- **env**  
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **net**  
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

- **cnt**  
Number of arcs to change.

- **indices**  
An array of arc indices that indicate the arcs to be changed. This array must have a length of at least cnt. All indices must be in the range \([0, \text{narcs}-1]\).

- **fromnode**  
An array of indices of the “from node” for which the arcs are to be changed. All indices must be in the range \([0, \text{nnodes}-1]\). The size of the array must be at least cnt.

- **tonode**  
An array of indices of the “to node” for which the arcs are to be changed. All indices must be in the range \([0, \text{nnodes}-1]\). The size of the array must be at least cnt.
**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXNETchgbds

Category  Global Function
Definition File  cplex.h
Include Files  cplex.h

Synopsis

```c
int CPXPUBLIC CPXNETchgbds (CPXCENVptr env,
                           CPXNETptr net,
                           int cnt,
                           const int * indices,
                           const char * lu,
                           const double * bd)
```

Description

The routine CPXNETchgbds is used to change the upper, lower, or both bounds on the flow for a set of arcs in the network stored in a network problem object. The flow value of an arc can be fixed to a value by setting both bounds to that value.

Any solution information stored in the problem object is lost.

Example

```c
status = CPXNETchgbds (env, net, cnt, index, lu, bd);
```

**Indicators to change lower, upper bounds of flows through arcs**

<table>
<thead>
<tr>
<th>lu[i]</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'L'</td>
<td>The lower bound of arc index[i] is changed to bd[i]</td>
</tr>
<tr>
<td>'U'</td>
<td>The upper bound of arc index[i] is changed to bd[i]</td>
</tr>
<tr>
<td>'B'</td>
<td>Both bounds of arc index[i] are changed to bd[i]</td>
</tr>
</tbody>
</table>

Parameters

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **net**
  A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
- **cnt**
  Number of bounds to change.
- **indices**
An array of arc indices that indicate the bounds to be changed. This array must have a length of at least \texttt{cnt}. All indices must be in the range \([0, \texttt{narcs}-1]\).

\texttt{lu}

An array indicating which bounds to change. This array must have a length of at least \texttt{cnt}. The indicators appear in the table.

\texttt{bd}

An array of bound values. This array must have a length of at least \texttt{cnt}. Values greater than or equal to \texttt{CPX\_INFBOUND} and less than or equal to \texttt{-CPX\_INFBOUND} are considered infinity or -infinity, respectively.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
**CPXNETchgname**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXNETchgname(CPXENVptr env,  
   CPXNETptr net,  
   int key,  
   int vindex,  
   const char * name_str)
```

**Description**  
The routine CPXNETchgname changes the name of a node or an arc in the network stored in a network problem object.

**Values of key in CPXNETchgname**

<table>
<thead>
<tr>
<th>key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'a'</td>
<td>Indicates the arc name is to be changed.</td>
</tr>
<tr>
<td>'n'</td>
<td>Indicates the node name is to be changed.</td>
</tr>
</tbody>
</table>

**Example**

```c
status = CPXNETchgname (env, net, 'a', 10, "arc10");
```

**Parameters**

- **env**
  
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **net**
  
  A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

- **key**
  
  A character to indicate whether an arc name should be changed, or a node name should be changed.

- **vindex**
  
  The index of the arc or node whose name is to be changed.

- **name_str**
  
  The new name for the arc or node.
Returns

The routine returns zero on success and nonzero if an error occurs.
CPXNETchgnodename

**Category**  Global Function

**Definition File**  cplex.h

**Include Files**  cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXNETchgnodename (CPXCENVptr env,
                                   CPXNETptr net,
                                   int cnt,
                                   const int * indices,
                                   char ** newname)
```

**Description**

The routine CPXNETchgnodename changes the names of a set of nodes in the network stored in a network problem object.

**Example**

```c
status = CPXNETchgnodename (env, net, 10, indices, newname);
```

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX

- **net**
  A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

- **cnt**
  An integer that indicates the total number of node names to be changed. Thus `cnt` specifies the length of the arrays `indices` and `newname`.

- **indices**
  An array of length `cnt` containing the numeric indices of the nodes for which the names are to be changed.

- **newname**
  An array of length `cnt` containing the strings of the new node names specified in `indices`.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXNETchgobj

Category: Global Function
Definition File: cplex.h
Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXNETchgobj(CPXENVptr env,
                          CPXNETptr net,
                          int cnt,
                          const int * indices,
                          const double * obj)
```

Description:
The routine `CPXNETchgobj` is used to change the objective values for a set of arcs in the network stored in a network problem object. Any solution information stored in the problem object is lost.

Example:

```c
status = CPXNETchgobj (env, net, cnt, indices, newobj);
```

Parameters:

- `env`: A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
- `net`: A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.
- `cnt`: Number of arcs for which the objective values are to be changed.
- `indices`: An array of indices that indicate the arcs for which the objective values are to be changed. This array must have a length of at least `cnt`. The indices must be in the range [0, `narcs-1`].
- `obj`: An array of the new objective values for the arcs. This array must have a length of at least `cnt`.

Returns:
The routine returns zero on success and nonzero if an error occurs.
CPXNETchgobjsen

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXNETchgobjsen(CPXENVptr env, 
    CPXNETptr net, 
    int maxormin)
```

Description:

The routine CPXNETchgobjsen is used to change the sense of the network problem to a minimization or maximization problem.

Any solution information stored in the problem object is lost.

**Changed optimization sense in a network problem**

<table>
<thead>
<tr>
<th>CPX_MAX</th>
<th>For a maximization problem.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_MIN</td>
<td>For a minimization problem.</td>
</tr>
</tbody>
</table>

Example:

```c
status = CPXNETchgobjsen (env, net, CPX_MAX);
```

Parameters:

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **net**
  A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

- **maxormin**
  New optimization sense for the network problem. The possible values are in the table.

Returns:

The routine returns zero on success and nonzero if an error occurs.
CPXNETchgsupply

Category Global Function

Definition File cplex.h

Include Files cplex.h

Synopsis

```c
int CPXPUBLIC CPXNETchgsupply(CPXENVptr env,
    CPXNETptr net,
    int cnt,
    const int * indices,
    const double * supply)
```

Description

The routine CPXNETchgsupply is used to change supply values for a set of nodes in the network stored in a network problem object.

Any solution information stored in the problem object is lost.

Example

```c
status = CPXNETchgsupply (env, net, cnt, indices, supply);
```

Parameters

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **net**
  A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

- **cnt**
  An integer indicating the number of nodes for which the objective values are to be changed.

- **indices**
  An array of indices that indicate the nodes for which the supply values are to be changed. This array must have a length of at least cnt. The indices must be in the range [0, nnodes-1].

- **supply**
  An array to contain the new supply values. This array must have a length of at least cnt.

Returns

The routine returns zero on success and nonzero if an error occurs.
**CPXNETdelarcs**

**Category**       Global Function  
**Definition File**  cplex.h  
**Include Files**   cplex.h  

**Synopsis**  

```c
int CPXPUBLIC CPXNETdelarcs(CPXENVptr env, 
   CPXNETptr net, 
   int begin, 
   int end)
```

**Description**  

The routine `CPXNETdelarcs` is used to remove a range of arcs from the network stored in a network problem object. The remaining arcs are renumbered starting at zero; their order is preserved. If removing arcs disconnects some nodes from the rest of the network, the disconnected nodes remain part of the network.

Any solution information stored in the problem object is lost.

**Example**  

```c
status = CPXNETdelarcs(env, net, 10, 20);
```

**Parameters**  

- **env**  
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
- **net**  
  A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.
- **begin**  
  Index of the first arc to be deleted.
- **end**  
  Index of the last arc to be deleted.

**Returns**  

The routine returns zero on success and nonzero if an error occurs.
CPXNETdelnodes

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis: int CPXPUBLIC CPXNETdelnodes(CPXENVptr env,
                  CPXNETptr net,
                  int begin,
                  int end)

Description: The routine CPXNETdelnodes is used to remove a range of nodes from the network stored in a network problem object. The remaining nodes are renumbered starting at zero; their order is preserved. All arcs incident to the nodes that are deleted are also deleted from the network.

Any solution information stored in the problem object is lost.

Example

    status = CPXNETdelnodes (env, net, 10, 20);

Parameters:

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

net
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

begin
Index of the first node to be deleted.

end
Index of the last node to be deleted.

Returns: The routine returns zero on success and nonzero if an error occurs.
CPXNETdelset

Category          Global Function
Definition File   cplex.h
Include Files     cplex.h
Synopsis
int CPXPUBLIC CPXNETdelset(CPXENVptr env,
            CPXNETptr net,
            int * whichnodes,
            int * whicharcs)

Description
The routine CPXNETdelset is used to delete a set of nodes and arcs from the network stored in a network problem object. The remaining nodes and arcs are renumbered starting at zero; their order is preserved.

Any solution information stored in the problem object is lost.

Example
status = CPXNETdelset (env, net, whichnodes, whicharcs);

Parameters
env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

net
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

whichnodes
Array of size at least CPXNETgetnumnodes that indicates the nodes to be deleted. If whichnodes[i] == 1, the node is deleted. For every node deleted, all arcs incident to it are deleted as well. After termination, whichnode[j] indicates either the position to which node with index j before deletion has been moved or, -1 if the node has been deleted. If NULL is passed, no nodes are deleted.

whicharcs
Array indicating the arc to be deleted. Every arc i in the network with whicharcs[i] == 1 is deleted. After termination, whicharc[j] indicates either the position to which arc with index j before deletion has been moved or, -1 if the arc has been deleted. This array also contains the deletions due to removed nodes. If NULL is passed, the only arcs deleted are those that are incident to nodes that have been deleted.

Returns
The routine returns zero on success and nonzero if an error occurs.
Group optim.cplex.callable.readfiles

The routines in the ILOG CPLEX Callable Library to read files.

**Global Functions Summary**

<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXNETreadcopybase</td>
</tr>
<tr>
<td>CPXreadcopybase</td>
</tr>
<tr>
<td>CPXreadcopymipstart</td>
</tr>
<tr>
<td>CPXreadcopyorder</td>
</tr>
<tr>
<td>CPXreadcopyprob</td>
</tr>
<tr>
<td>CPXreadcopyqp</td>
</tr>
<tr>
<td>CPXreadcopyrqp</td>
</tr>
<tr>
<td>CPXreadcopytree</td>
</tr>
<tr>
<td>CPXreadcopyvec</td>
</tr>
</tbody>
</table>

**Description**

Use these routines to read data from system files. ILOG CPLEX can read problem files stored in a variety of formats. For more information about the file formats, see the reference manual *ILOG CPLEX File Formats.*
CPXNETreadcopybase

Category Global Function
Definition File cplex.h
Include Files cplex.h
Synopsis
int CPXPUBLIC CPXNETreadcopybase(CPXENVptr env,
    CPXNETptr net,
    const char * filename_str)
Description
The routine CPXNETreadcopybase reads a basis file in BAS format and copies the
basis to a network problem object. If no arc or node names are available for the problem
object when reading the basis file, default names are assumed. Any basis that may have
been created or saved in the problem object is replaced.
Example
status = CPXNETreadcopybase (env, net, "netbasis.bas");
Parameters
env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
net
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
filename_str
Name of the basis file to read.
Returns
The routine returns zero on success and nonzero if an error occurs.
CPXreadcopybase

Category: Global Function
Definition File: cplex.h
Include Files: cplex.h

Synopsis:
```c
int CPXPUBLIC CPXreadcopybase(CPXENVptr env,
                               CPXLPptr lp,
                               const char * filename_str)
```

Description:
The routine `CPXreadcopybase` reads a basis from a BAS file, and copies that basis into a CPLEX problem object. The parameter `CPX_PARAM_ADVIND` must be set to 1 (one), its default value, in order for the basis to be used for starting a subsequent optimization.

Example:
```c
status = CPXreadcopybase (env, lp, "myprob.bas");
```

Parameters:
- `env`:
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
- `lp`:
  A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
- `filename_str`:
  The name of the file from which the basis should be read.

Returns:
The routine returns zero on success and nonzero if an error occurs.
CPXreadcopymipstart

Category  Global Function
Definition File  cplex.h
Include Files  cplex.h
Synopsis  

\[
\text{int CPXPUBLIC CPXreadcopymipstart}(\text{CPXCENVptr env, CPXLPptr lp, const char * filename_str})
\]

Description  The routine CPXreadcopymipstart reads a MST file and copies the MIP start information into a CPLEX problem object. The parameter CPX_PARAM_MIPSTART must be set to CPX_ON, in order for the MIP start information to be used for starting a subsequent optimization. The default value for the parameter is CPX_OFF.

Example  

\[
\text{status = CPXreadcopymipstart}(\text{env, lp, "myprob.mst"});}
\]

Parameters  

- env
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- lp
  A pointer to a CPLEX problem object as returned by CPXcreateprob.
- filename_str
  A string containing the name of the MST file.

Returns  The routine returns zero on success and nonzero if an error occurs.
**CPXreadcopyorder**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXreadcopyorder(CPXENVptr env,
                             CPXLPptr lp,
                             const char * filename_str)
```

**Description**  
The routine **CPXreadcopyorder** reads an ORD file and copies the priority order information into a CPLEX problem object. The parameter **CPX_PARAM_MIPORDIND** must be set to **CPX_ON** (its default value), in order for the priority order to be used for starting a subsequent optimization.

**Example**  
```c
status = CPXreadcopyorder (env, lp, "myprob.ord");
```

**Parameters**  

- **env**  
  A pointer to the CPLEX environment as returned by **CPXopenCPLEX**.

- **lp**  
  A pointer to a CPLEX problem object as returned by **CPXcreateprob**.

- **filename_str**  
  The name of the file from which the priority order and start values should be read.

**Returns**  
The routine returns zero on success and nonzero if an error occurs.
CPXreadcopyprob

Category    Global Function
Definition File    cplex.h
Include Files    cplex.h
Synopsis    int CPXPUBLIC CPXreadcopyprob(CPXENVptr env,
            CPXLPptr lp,
            const char * filename_str,
            const char * filetype_str)

Description    The routine CPXreadcopyprob reads an MPS, LP or SAV file into an existing
                CPLEX problem object. Any existing data associated with the problem object is
                destroyed. The problem can then be optimized via any one of the optimization routines.
                To determine the contents of the data, CPLEX query routines can be used.

                The type of the file may be specified with the filetype argument. When the
                filetype argument is NULL, the end of the file name is checked for one of the strings
                .lp, .lp.gz, .lp.z, .mps, .mps.gz, .mps.z, .sav, .sav.gz, or .sav.z.

                If one of these strings is present, filetype is set accordingly. If filetype is NULL
                and none of these strings is at the end of the file name, the routine automatically detects
                the type of the file by examining the first few bytes of the file.

                If the file name ends in .gz or .z the file is read as a compressed file. Thus, a file name
                ending in .sav is read as a SAV format file, while a file name ending in .sav.gz is
                read as a compressed SAV format file.

                If the file name does not end with a recognized type, CPLEX attempts to auto-detect the
                file type.

                Values of filetype_str

                | Value | Description   |
                |-------|--------------|
                | "SAV" | Use SAV format |
                | "MPS" | Use MPS format |
                | "LP"  | Use LP format |

Example

    status = CPXreadcopyprob (env, lp, "myprob.mps", NULL);

See also the example lpex2.c in the ILOG CPLEX User's Manual and in the standard
distribution.
Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

filename_str
The name of the file from which the problem should be read.

filetype_str
A case insensitive string containing the type of the file, one of the strings in the table. May be NULL, in which case the file type is inferred from the last characters of the file name.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXreadcopyqp

Category                           Global Function
Definition File                    cplex.h
Include Files                      cplex.h
Synopsis                           int CPXPUBLIC CPXreadcopyqp(CPXENVptr env,
                                  CPXLPptr lp,
                                  const char * filename_str)
Description
The routine CPXreadcopyqp reads in the QP file specified by the argument filename and copies the quadratic matrix data into the CPLEX problem object. This routine changes the problem type to QP if it is not already a QP. It may not be used on a problem whose type is MIP.

Example
status = CPXreadcopyqp (env, lp, "myprob.qp");

Parameters
env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.
filename_str
The name of the file from which the QP information should be read.

Returns
The routine returns zero on success and nonzero if an error occurs.
**CPXreadcopysos**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXreadcopysos(CPXENVptr env,
   CPXLPptr lp,
   const char * filename_str)
```

**Description**  
The routine `CPXreadcopysos` reads in SOS data from an SOS file, and copies the data into a MIP problem object. This routine changes the problem type to MIP, if it is not already a MIP.

**Example**  
```c
status = CPXreadcopysos (env, lp, "myprob.sos");
```

**Parameters**
- **env**  
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
- **lp**  
  A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
- **filename_str**  
  The name of the file from which the SOS information should be read.

**Returns**  
The routine returns zero on success and nonzero if an error occurs.
CPXreadcopytree

Category          Global Function
Definition File   cplex.h
Include Files     cplex.h
Synopsis          int CPXPUBLIC CPXreadcopytree(CPXENVptr env, 
                                      CPXLPptr lp, 
                                      const char * filename_str)

Description       The routine CPXreadcopytree is used to read branch & cut progress information 
                   from a prior run, contained in a TRE format file, into a CPLEX problem object. The 
                   parameter CPX_PARAM_ADVIND must be set to CPX_ON (its default value), in order 
                   for the tree to be used for starting a subsequent optimization.

Example            status = CPXreadcopytree (env, lp, "myprob.tre");

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.
filename_str
The name of the file from which the tree information should be read.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXreadcopyvec

Category  Global Function
Definition File  cplex.h
Include Files  cplex.h
Synopsis  
```
int CPXPUBLIC CPXreadcopyvec(CPXENVptr env, 
   CPXLPptr lp, 
   const char * filename_str)
```
Description  The routine CPXreadcopyvec reads Barrier solution information from a VEC format file and copies this information into a CPLEX problem object. This routine is typically used to initiate a crossover from the Barrier solution. The parameter CPX_PARAM_ADVIND must be set to 1 (one), its default value, in order for the vector file to take effect for starting a crossover.

Example  
```
status = CPXreadcopyvec (env, lp, "myprob.vec");
```

Parameters  
- `env`  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- `lp`  A pointer to a CPLEX problem object as returned by CPXcreateprob.
- `filename_str`  The name of the VEC file to be read.

Returns  The routine returns zero on success and nonzero if an error occurs.

Group optim.cplex.callable.readnetworkfiles

The routines in the ILOG CPLEX Callable Library to read network files.

<table>
<thead>
<tr>
<th>Global Functions Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXNETcreateprob</td>
</tr>
<tr>
<td>CPXNETreadcopybase</td>
</tr>
</tbody>
</table>
Description

Use these routines to read data from system files into a network problem object.
CPXNETcreateprob

Category  
Global Function

Definition File  
cplex.h

Include Files  
cplex.h

Synopsis  
CPXNETptr CPXNETcreateprob(CPXENVptr env,  
   int * status_p,  
   const char * name_str)

Description  
The routine CPXNETcreateprob constructs a new network problem object. The new object contains a minimization problem for a network with 0 nodes and 0 arcs. Other network problem data can be copied to a network with one of the routines CPXNETaddnodes, CPXNETaddarcs, CPXNETcopynet, CPXNETextract, or CPXNETreadcopyprob.

Example  
CPXNETptr net = CPXNETcreateprob (env, &status, "mynet");

See Also  
CPXNETaddnodes, CPXNETaddarcs, CPXNETcopynet, CPXNETextract, CPXNETreadcopyprob

Parameters  
env  
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

status_p  
A pointer to an integer used to return any error code produced by this routine.

name_str  
Name of the network to be created.

Returns  
If the operation is successful, CPXNETcreateprob returns the newly constructed network problem object; if not, it returns either NULL or a nonzero value to indicate an error. In case of an error, the value pointed to by status_p contains an integer indicating the cause of the error.
**CPXNETreadcopybase**

**Category** Global Function

**Definition File** cplex.h

**Include Files** cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXNETreadcopybase (CPXCENVptr env, 
    CPXNETptr net, 
    const char * filename_str)
```

**Description**

The routine **CPXNETreadcopybase** reads a basis file in BAS format and copies the basis to a network problem object. If no arc or node names are available for the problem object when reading the basis file, default names are assumed. Any basis that may have been created or saved in the problem object is replaced.

**Example**

```c
status = CPXNETreadcopybase (env, net, "netbasis.bas");
```

**Parameters**

- `env` A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- `net` A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
- `filename_str` Name of the basis file to read.

**Returns**

The routine returns zero on success and nonzero if an error occurs.

**Group optim.cplex.callable.writefiles**

The routines in the ILOG CPLEX Callable Library to write files.

<table>
<thead>
<tr>
<th>Global Functions Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXdperwrite</td>
</tr>
<tr>
<td>CPXdualwrite</td>
</tr>
<tr>
<td>CPXembwrite</td>
</tr>
</tbody>
</table>

---

**ILOG CPLEX Callable Library 9.0 Reference Manual** 816
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXliswrite</td>
<td>These routines write a problem object or, after the problem has been optimized, they write the optimal basis or solution report to a file.</td>
</tr>
<tr>
<td>CPXmbasewrite</td>
<td></td>
</tr>
<tr>
<td>CPXmstwrite</td>
<td></td>
</tr>
<tr>
<td>CPXNETbasewrite</td>
<td></td>
</tr>
<tr>
<td>CPXNETwriteprob</td>
<td></td>
</tr>
<tr>
<td>CPXordwrite</td>
<td></td>
</tr>
<tr>
<td>CPXpperwrite</td>
<td></td>
</tr>
<tr>
<td>CPXpreslvwrite</td>
<td></td>
</tr>
<tr>
<td>CPXqgwrite</td>
<td></td>
</tr>
<tr>
<td>CPXsoswrite</td>
<td></td>
</tr>
<tr>
<td>CPXtreewrite</td>
<td></td>
</tr>
<tr>
<td>CPXvecwrite</td>
<td></td>
</tr>
<tr>
<td>CPXwriteprob</td>
<td></td>
</tr>
<tr>
<td>CPXwritesol</td>
<td></td>
</tr>
</tbody>
</table>
CPXNETbasewrite

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXNETbasewrite(CPXENVptr env,
                              CPXNETptr net,
                              const char * filename_str)
```

Description:
The routine `CPXNETbasewrite` writes the current basis stored in a network problem object to a file in BAS format. If no arc or node names are available for the problem object, default names are used.

Example:

```c
status = CPXNETbasewrite (env, net, "netbasis.bas");
```

Parameters:

- `env`: A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
- `net`: A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.
- `filename_str`: Name of the basis file to write.

Returns:
The routine returns zero on success and nonzero if an error occurs.
CPXNETwriteprob

Category Global Function

Definition File cplex.h

Include Files cplex.h

Synopsis int CPXPUBLIC CPXNETwriteprob(CPXENVptr env, CPXCNETptr net, const char * filename_str, const char * format_str)

Description The routine CPXNETwriteprob writes the network stored in a network problem object to a file. This can be done in CPLEX (.net) or DIMACS (.min) network file format (MIN) or as the LP representation of the network in any of the LP formats (.lp, .mps, or .sav).

If the file name ends with .gz, a compressed file is written.

File extensions for network files

<table>
<thead>
<tr>
<th>net</th>
<th>for CPLEX network format</th>
</tr>
</thead>
<tbody>
<tr>
<td>min</td>
<td>for DIMACS network format</td>
</tr>
<tr>
<td>lp</td>
<td>for LP format of LP formulation</td>
</tr>
<tr>
<td>mps</td>
<td>for MPS format of LP formulation</td>
</tr>
<tr>
<td>sav</td>
<td>for SAV format of LP formulation</td>
</tr>
</tbody>
</table>

Example

status = CPXNETwriteprob (env, net, "network.net", NULL);

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

net
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

filename_str
Name of the network file to write, where the file extension specifies the file format unless overridden by the format argument. If the file name ends with .gz a compressed file is written in accordance with the selected file type.

format_str
File format to generate. Possible values appear in the table. If NULL is passed, the format is inferred from the file name.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXdperwrite

Category          Global Function
Definition File   cplex.h
Include Files     cplex.h
Synopsis          int CPXPUBLIC CPXdperwrite(CPXENVptr env,
                                        CPXLPptr lp,           
                                        const char * filename_str, 
                                        double epsilon)

Description       When solving degenerate linear programs with the dual simplex method, CPLEX may
                   initiate a perturbation of the objective function of the problem in order to improve
                   performance. The routine CPXdperwrite writes a similarly perturbed problem to a
                   binary SAV format file.

Example

    status = CPXdperwrite (env, lp, "myprob.dpe", epsilon);

Parameters

    env
    A pointer to the CPLEX environment as returned by CPXopenCPLEX.

    lp
    A pointer to a CPLEX problem object as returned by CPXcreateprob.

    filename_str
    A character string containing the name of the file to which the perturbed LP problem
    should be written.

    epsilon
    The perturbation constant.

Returns

    The routine returns zero on success and nonzero if an error occurs.
CPXdualwrite

Category: Global Function
Definition File: cplex.h
Include Files: cplex.h

Synopsis

```c
int CPXPUBLIC CPXdualwrite (CPXCENVptr env,
                           CPXCLPptr lp,
                           const char * filename_str,
                           double * objshift_p)
```

Description

The routine CPXdualwrite is used to write a dual formulation of the current CPLEX problem object. MPS format is used. This function can only be applied to a linear program; it generates an error for other problem types.

**Note:** Any fixed variables in the primal are removed before the dual problem is written to a file. Each fixed variable with a nonzero objective coefficient causes the objective value to shift. As a result, if fixed variables are present, the optimal objective obtained from solving the dual problem created using CPXdualwrite may not be the same as the optimal objective of the primal problem. The objshift_p parameter can be used to reconcile this difference.

Example

```c
status = CPXdualwrite (env, lp, "myfile.dua", &objshift);
```

Parameters

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **lp**
  A pointer to a CPLEX problem object as returned by CPXcreateprob.
- **filename_str**
  A character string containing the name of the file to which the dual problem should be written.
- **objshift_p**
  A pointer to a double where the objective value shift is stored.
A pointer to a variable of type double to hold the change in the objective function resulting from the removal of fixed variables in the primal.

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXembwrite

Category Global Function

Definition File cplex.h

Include Files cplex.h

Synopsis

#include <cplex.h>

int CPXPUBLIC CPXembwrite (CPXCENVptr env, 
CPXLPptr lp, 
const char * filename_str)

Description

The routine CPXembwrite is used to write out the network embedded in the selected problem object. MPS format is used. The specific network extracted depends on the current setting of the CPX_PARAM_NETFIND parameter.

Example

status = CPXembwrite (env, lp, "myfile.emb");

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

filename_str
A character string containing the name of the file to which the embedded network should be written.

Example

status = CPXembwrite (env, lp, "myfile.emb");

Returns

The routine returns zero on success and nonzero if an error occurs.
**CPXiiswrite**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXiiswrite(CPXENVptr env,
   CPXLPptr lp,
   const char * filename_str)
```

**Description**  
The routine `CPXiiswrite` is used to write an LP format file containing the rows and columns in the Irreducibly Inconsistent Set (IIS) of an infeasible LP. Note that the infeasibility must previously have been detected by a simplex optimizer. If an IIS has already been obtained using the same method as is currently specified by the IIS algorithm indicator `CPXPARAM_IISIND`, `CPXiiswrite` simply writes an LP format file containing the existing IIS. If no IIS has been computed, or the IIS algorithm differs from the one used to compute the available IIS, `CPXiiswrite` first computes an IIS, then writes its LP format representation out.

**Example**

```c
status = CPXiiswrite (env, lp, "myprob.iis");
```

**Parameters**

- **env**  
A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- **lp**  
A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

- **filename_str**  
A character string containing the name of the file to which the IIS should be written.

**Returns**  
The routine returns zero on success and nonzero if an error occurs.
**CPXmbasewrite**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**
```c
int CPXPUBLIC CPXmbasewrite(CPXENVptr env,
                           CPXCLPptr lp,
                           const char * filename_str)
```

**Description**
The routine **CPXmbasewrite** is used to write the most current basis associated with a CPLEX problem object to a file. The file is saved in BAS format which corresponds to the industry standard MPS insert format for bases.

When **CPXmbasewrite** is invoked, the current basis is written to a file. This routine does not remove the basis from the problem object.

**Example**
```c
status = CPXmbasewrite (env, lp, "myprob.bas");
```

**Parameters**
- **env**
  A pointer to the CPLEX environment as returned by **CPXopenCPLEX**.
- **lp**
  A pointer to the CPLEX problem object as returned by **CPXcreateprob**.
- **filename_str**
  A character string containing the name of the file to which the basis should be written.

**Returns**
The routine returns zero on success and nonzero if an error occurs.
**CPXmstwrite**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXmstwrite(CPXENVptr env,
   CPXCptr lp,
   const char * filename_str)
```

**Description**  
The routine CPXmstwrite is used to write a MIP start to an .mst file.

**Parameters**

- `env`  
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- `lp`  
A pointer to the CPLEX problem object as returned by CPXcreateprob.

- `filename_str`  
A character string containing the name of the file to which the MIP start information should be written.
CPXordwrite

Category     Global Function

Definition File  cplex.h

Include Files  cplex.h

Synopsis     int CPXPUBLIC CPXordwrite(CPXENVptr env,
                          CPXCLPptr lp,
                          const char * filename_str)

Description  The routine CPXordwrite is used to write a priority order to an ORD file. If a priority order has been associated with the CPLEX problem object, or the parameter CPX_PARAM_MIPORDTYPE is nonzero, or a MIP feasible solution exists, this routine writes the priority order into a file.

Example

    status = CPXordwrite (env, lp, "myfile.ord");

See also the example mipex3.c in the standard distribution.

Parameters  

    env
    A pointer to the CPLEX environment as returned by CPXopenCPLEX.

    lp
    A pointer to a CPLEX problem object as returned by CPXcreateprob.

    filename_str
    A character string containing the name of the file to which the ORD information should be written.

Returns     The routine returns zero on success and nonzero if an error occurs.
CPXpperrwrite

Category          Global Function
Definition File   cplex.h
Include Files     cplex.h

Synopsis
int CPXPUBLIC CPXpperrwrite (CPXCENVptr env,
                            CPXLPptr lp,
                            const char * filename_str,
                            double epsilon)

Description
When solving degenerate linear programs with the primal simplex method, CPLEX may
initiate a perturbation of the bounds of the problem in order to improve performance.
The routine CPXpperrwrite writes a similarly perturbed problem to a binary SAV
format file.

Example
status = CPXpperrwrite (env, lp, "myprob.ppe", epsilon);

Parameters
env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.
filename_str
A character string containing the name of the file to which the perturbed problem should be written.
epsilon
The perturbation constant.

Returns
The routine returns zero on success and nonzero if an error occurs.
**CPXpreslvwrite**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**
```c
#include <cplex.h>

int CPXPUBLIC CPXpreslvwrite(CPXENVptr env,
                              CPXLPptr lp,
                              const char * filename_str,
                              double * objoff_p)
```

**Description**
The routine `CPXpreslvwrite` is used to write a presolved version of the problem to a file. The file is saved in binary format, and can be read using the routine `CPXreadcopyprob`.

- **Note:** Reductions done by the CPLEX presolve algorithms can cause the objective value to shift. As a result, the optimal objective obtained from solving the presolved problem created using `CPXpreslvwrite` may not be the same as the optimal objective of the original problem. The `objoff_p` parameter can be used to reconcile this difference.

**Example**
```c
status = CPXpreslvwrite (env, lp, "myfile.pre", &objoff);
```

**Parameters**
- **env**
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
- **lp**
  A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
- **filename_str**
  A character string containing the name of the file to which the presolved problem should be written.
- **objoff_p**
  A pointer to a double precision variable that is used to hold the objective value difference between the original problem and the presolved problem. That is: \( \text{original objective value} = (*\text{objoff}_p) + \text{presolved objective value} \)
Returns

The routine returns zero on success and nonzero if an error occurs.
CPXqpwrite

Category  Global Function
Definition File  cplex.h
Include Files  cplex.h

Synopsis  

```c
int CPXPUBLIC CPXqpwrite(CPXENVptr env,
     CPXCLPptr lp,
     const char * filename_str)
```

Description  The routine CPXqpwrite is used to write a QP format file corresponding to the quadratic coefficient matrix of the problem object. This format is described in detail in the reference manual *ILOG CPLEX File Formats*.

Example  

```c
status = CPXqpwrite (env, lp, "myprob.qp");
```

Parameters  

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **lp**
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

- **filename_str**
  A character string containing the name of the file to which the QP information should be written.

Returns  The routine returns zero on success and nonzero if an error occurs.
CPXsoswrite

Category Global Function
Definition File cplex.h
Include Files cplex.h
Synopsis
int CPXPUBLIC CPXsoswrite (CPXCENVptr env,
          CPXCLPptr lp,
          const char * filename_str)
Description
The routine CPXsoswrite is used to write SOS information for a problem into an
SOS format file.
Example
    status = CPXsoswrite (env, lp, "myprob.sos");

See also the example mipex3.c in the standard distribution.

Parameters
    env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
    lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.
    filename_str
A character string containing the name of the file to which the SOS information should
be written.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXtreewrite

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis: int CPXPUBLIC CPXtreewrite(CPXENVptr env, CPXCLPptr lp, const char * filename_str)

Description: The routine CPXtreewrite is used to save a TRE file containing information about the branch & cut path or “tree” for the current problem.

Example:

status = CPXtreewrite (env, lp, "myprob.tre");

Parameters:

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

filename_str
A character string containing the name of the file to which the tree information is to be written.

Returns: The routine returns zero on success and nonzero if an error occurs.
CPXvecwrite

Category Global Function

Definition File cplex.h

Include Files cplex.h

Synopsis

int CPXPUBLIC CPXvecwrite (CPXCENVptr env, CPXCLPptr lp, const char * filename_str)

Description

The routine CPXvecwrite is used to write solution information from a Barrier optimization (without crossover) into a VEC format file. The VEC file can later be read back into CPLEX for a crossover.

Example

status = CPXvecwrite (env, lp, "myprob.vec");

Parameters

env

A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp

A pointer to a CPLEX problem object as returned by CPXcreateprob.

filename_str

A character string containing the name of the file to which the solution information is to be written.

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXwriteprob

Category Global Function
Definition File cplex.h
Include Files cplex.h
Synopsis

int CPXPUBLIC CPXwriteprob (CPXCENVptr env,
                           CPXCLPptr lp,
                           const char * filename_str,
                           const char * filetype_str)

Description
The routine CPXwriteprob is used to write the CPLEX problem object to a file in one of the formats in the table. These formats are documented in the reference manual ILOG CPLEX File Formats and examples of their use appear in the ILOG CPLEX User’s Manual.

File formats

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAV</td>
<td>Binary matrix and basis file</td>
</tr>
<tr>
<td>MPS</td>
<td>MPS format</td>
</tr>
<tr>
<td>LP</td>
<td>CPLEX LP format</td>
</tr>
<tr>
<td>REN</td>
<td>MPS format, with all names changed to generic names</td>
</tr>
<tr>
<td>RMP</td>
<td>MPS format, with all names changed to generic names</td>
</tr>
<tr>
<td>RLP</td>
<td>LP format, with all names changed to generic names</td>
</tr>
</tbody>
</table>

When this routine is invoked, the current problem is written to a file. If the file name ends with .gz, a compressed file is written.

Example

    status = CPXwriteprob (env, lp, "myprob.sav", NULL);

See also the example lpex1.c in the ILOG CPLEX User’s Manual and in the standard distribution.

Parameters

    env

A pointer to the CPLEX environment as returned by CPXopenCPLEX.
lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

filename_str
A character string containing the name of the file to which the problem is to be written, unless otherwise specified with the filetype argument. If the file name ends with .gz, a compressed file is written in accordance with the selected file type.

filetype_str
A character string containing the type of the file, which can be one of the values in the table. May be NULL, in which case the type is inferred from the file name. The string is not case sensitive.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXwritesol

Category          Global Function
Definition File     cplex.h
Include Files      cplex.h
Synopsis
int CPXPUBLIC CPXwritesol(CPXENVptr env,  
                     CPXCptr lp,  
                     const char * filename_str,  
                     const char * filetype_str)

Description
The routine CPXwritesol is used to write a solution file for the selected CPLEX problem object. The routine can write files in:
- TXT format, which is the ASCII format
- BIN format, which is the binary format.


Values of filetype_str

| 'TXT'    | ASCII solution file |
| 'BIN'    | Binary solution file |

Example

status = CPXwritesol (env, lp, "myfile.txt", NULL);

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp
A pointer to a CPLEX problem object as returned by CPXcreateprob.

filename_str
A character string containing the name of the file to which the solution should be written.

filetype_str

| 'TXT'    | ASCII solution file |
| 'BIN'    | Binary solution file |
A character string containing the type of the file. The type of the file can be one of the values in the table. May be NULL, in which case the type is inferred from the characters following the last period in the filename. The string is not case sensitive.

**Returns**

The routine returns zero on success and nonzero if an error occurs.

### Group optim.cplex.callable.writenetworkfiles

The routines in the ILOG CPLEX Callable Library to write network files.

<table>
<thead>
<tr>
<th>Global Functions Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXNETbasewrite</td>
</tr>
<tr>
<td>CPXNETwriteprob</td>
</tr>
</tbody>
</table>

**Description**

These routines write a network problem object or, after the network problem has been optimized, they write the optimal basis or solution report to a file.
CPXNETbasewrite

Category  Global Function
Definition File  cplex.h
Include Files  cplex.h
Synopsis  int CPXPUBLIC CPXNETbasewrite(CPXCENVptr env,
                                    CPXCNETptr net, 
                                    const char * filename_str)
Description  The routine CPXNETbasewrite writes the current basis stored in a network problem object to a file in BAS format. If no arc or node names are available for the problem object, default names are used.

Example  
          status = CPXNETbasewrite (env, net, "netbasis.bas");

Parameters  
env  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
net  A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
filename_str  Name of the basis file to write.

Returns  The routine returns zero on success and nonzero if an error occurs.
CPXNETwriteprob

Category              Global Function
Definition File       cplex.h
Include Files         cplex.h
Synopsis              int CPXPUBLIC CPXNETwriteprob(CPXENVptr env,
                              CPXNETptr net,
                              const char * filename_str,
                              const char * format_str)

Description           The routine CPXNETwriteprob writes the network stored in a network problem object to a file. This can be done in CPLEX (.net) or DIMACS (.min) network file format (MIN) or as the LP representation of the network in any of the LP formats (.lp, .mps, or .sav).
                        If the file name ends with .gz, a compressed file is written.

                        **File extensions for network files**
                        | net     | for CPLEX network format |
                        | min     | for DIMACS network format |
                        | lp      | for LP format of LP formulation |
                        | mps     | for MPS format of LP formulation |
                        | sav     | for SAV format of LP formulation |

Example               status = CPXNETwriteprob (env, net, "network.net", NULL);

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

net
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

filename_str
Name of the network file to write, where the file extension specifies the file format unless overridden by the format argument. If the file name ends with .gz a compressed file is written in accordance with the selected file type.

format_str
File format to generate. Possible values appear in the table. If NULL is passed, the format is inferred from the file name.

**Returns**
The routine returns zero on success and nonzero if an error occurs.

**Group optim.cplex.callable.network**
The network routines in the ILOG CPLEX Callable Library.

### Global Functions Summary

<table>
<thead>
<tr>
<th>Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXNETaddarcs</td>
</tr>
<tr>
<td>CPXNETaddnodes</td>
</tr>
<tr>
<td>CPXNETbegintwrite</td>
</tr>
<tr>
<td>CPXNETcheckcopynet</td>
</tr>
<tr>
<td>CPXNETchgarcname</td>
</tr>
<tr>
<td>CPXNETchgarcnodes</td>
</tr>
<tr>
<td>CPXNETchgbds</td>
</tr>
<tr>
<td>CPXNETchgname</td>
</tr>
<tr>
<td>CPXNETchgnodename</td>
</tr>
<tr>
<td>CPXNETchgoj</td>
</tr>
<tr>
<td>CPXNETchgojbsen</td>
</tr>
<tr>
<td>CPXNETchgsupply</td>
</tr>
<tr>
<td>CPXNETcopybase</td>
</tr>
<tr>
<td>CPXNETcopynet</td>
</tr>
<tr>
<td>CPXNETcreateprob</td>
</tr>
<tr>
<td>CPXNEDelarcs</td>
</tr>
<tr>
<td>CPXNEDelnodes</td>
</tr>
<tr>
<td>CPXNEDelslip</td>
</tr>
<tr>
<td>CPXNETextract</td>
</tr>
<tr>
<td>CPXNETfreeprob</td>
</tr>
<tr>
<td>CPXNETgetarcindex</td>
</tr>
<tr>
<td>CPXNETgetarcname</td>
</tr>
<tr>
<td>CPXNETgetarcnodes</td>
</tr>
<tr>
<td>CPXNETgetbase</td>
</tr>
<tr>
<td>CPXNETgetdj</td>
</tr>
<tr>
<td>CPXNETgetitcst</td>
</tr>
<tr>
<td>CPXNETgetlb</td>
</tr>
<tr>
<td>CPXNETgetnodearcs</td>
</tr>
<tr>
<td>CPXNETgetnodeindex</td>
</tr>
<tr>
<td>CPXNETgetnodename</td>
</tr>
</tbody>
</table>
If part of your problem is structured as a network, then you may want to consider calling the ILOG CPLEX Network Optimizer. This optimizer may have a positive impact on performance. There are two alternative ways of calling the network optimizer:

- If your problem is an LP where a large part is a network structure, you may call the network optimizer for the populated LP object.
- If your entire problem consists of a network flow, you should consider creating a network object instead of an LP object. Then populate it, and solve it with the network optimizer. This alternative generally yields the best performance because it does not incur the overhead of LP data structures. This option is only available for the ILOG CPLEX Callable Library.

For more about formulating a problem in this way and applying the network optimizer in your application, see this topic in the *ILOG CPLEX User's Manual*. 

<table>
<thead>
<tr>
<th>CPXNETgetnumarcs</th>
<th>CPXNETgetnumnodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXNETgetobj</td>
<td>CPXNETgetobjsen</td>
</tr>
<tr>
<td>CPXNETgetobjval</td>
<td>CPXNETgetphaselcnt</td>
</tr>
<tr>
<td>CPXNETgetpi</td>
<td>CPXNETgetprobi</td>
</tr>
<tr>
<td>CPXNETgetslack</td>
<td>CPXNETgetstat</td>
</tr>
<tr>
<td>CPXNETgetsupply</td>
<td>CPXNETgetub</td>
</tr>
<tr>
<td>CPXNETgetx</td>
<td>CPXNETprimopt</td>
</tr>
<tr>
<td>CPXNETreadcopybase</td>
<td>CPXNETisolininfo</td>
</tr>
<tr>
<td>CPXNETsolution</td>
<td>CPXNETwriteprob</td>
</tr>
</tbody>
</table>
CPXNETaddarcs

Category  Global Function
Definition File  cplex.h
Include Files  cplex.h
Synopsis  
```
int CPXPUBLIC CPXNETaddarcs(CPXENVptr env,
   CPXNETptr net,
   int narcs,
   const int * fromnode,
   const int * tonode,
   const double * low,
   const double * up,
   const double * obj,
   char ** anames)
```
Description  The routine CPXNETaddarcs adds new arcs to the network stored in a network problem object.
Example  
```
status = CPXNETaddarcs (env, net, narcs, fromnode, tonode, NULL,
   NULL, obj, NULL);
```
See Also  CPXNETgetnumnodes
Parameters

- `env`
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- `net`
  A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
- `narcs`
  Number of arcs to be added.
- `fromnode`
  Array of indices of the “from node” for the arcs to be added. All the indices must be greater than or equal to 0. If a node index is greater than or equal to the number of nodes currently in the network (see CPXNETgetnumnodes) new nodes are created implicitly with default supply values 0. The size of the fromnode array must be at least `narcs`.
- `tonode`
Array of indices of the “to node” for the arcs to be added. All the indices must be greater than or equal to 0. If a node index is greater than or equal to the number of nodes currently in the network (see CPXNETgetnumnodes) new nodes are created implicitly with default supply values 0. The size of the tonode array must be at least \( \text{narcs} \).

**low**

Pointer to an array of lower bounds on the flow through added arcs. If NULL is passed, all lower bounds default to 0 (zero). Otherwise, the size of the array must be at least \( \text{narcs} \). Values less than or equal to \(-\text{CPX\_INFBOUND}\) are considered as -\(\infty\).

**up**

Pointer to an array of upper bounds on the flow of added arcs. If NULL is passed, all upper bounds default to \(\text{CPX\_INFBOUND}\). Otherwise, the size of the array must be at least \( \text{narcs} \). Values greater than or equal to \(\text{CPX\_INFBOUND}\) are considered as \(\infty\).

**obj**

Pointer to an array of objective values for the added arcs. If NULL is passed, all objective values default to 0. Otherwise, the size of the array must be at least \( \text{narcs} \).

**anames**

Pointer to an array of names for added arcs. If NULL is passed and the existing arcs have names, default names are assigned to the added arcs. If NULL is passed and the existing arcs have no names, the new arcs are assigned no names. Otherwise, the size of the array must be at least \( \text{narcs} \) and every name in the array must be a string terminating in 0. If the existing arcs have no names and \( \text{anames} \) is not NULL, default names are assigned to the existing arcs.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
**CPXNETaddnodes**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**
```c
int CPXPUBLIC CPXNETaddnodes (CPXCENVptr env,
    CPXNETptr net,
    int nnodes,
    const double * supply,
    char ** name)
```

**Description**
The routine `CPXNETaddnodes` adds new nodes to the network stored in a network problem object.

**Example**
```c
    status = CPXNETaddnodes (env, net, nnodes, supply, NULL);
```

**Parameters**

- **env**
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **net**
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

- **nnodes**
Number of nodes to add.

- **supply**
Supply values for the added nodes. If NULL is passed, all supplies defaults to 0 (zero). Otherwise, the size of the array must be at least `nnodes`.

- **name**
Pointer to an array of names for added nodes. If NULL is passed and the existing nodes have names, default names are assigned to the added nodes. If NULL is passed but the existing nodes have no names, the new nodes are assigned no names. Otherwise, the size of the array must be at least `nnodes` and every name in the array must be a string terminating in 0. If the existing nodes have no names and `nnames` is not NULL, default names are assigned to the existing nodes.

**Returns**
The routine returns zero on success and nonzero if an error occurs.
CPXNETbasewrite

Category       Global Function
Definition File cplex.h
Include Files  cplex.h
Synopsis

int CPXPUBLIC CPXNETbasewrite(CPXENVptr env,
                               CPXCNETptr net,
                               const char * filename_str)

Description
The routine CPXNETbasewrite writes the current basis stored in a network problem object to a file in BAS format. If no arc or node names are available for the problem object, default names are used.

Example

status = CPXNETbasewrite (env, net, "netbasis.bas");

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

net
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

filename_str
Name of the basis file to write.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXNETcheckcopynet

Category       Global Function
Definition File cplex.h
Include Files  cplex.h

Synopsis

int CPXPUBLIC CPXNETcheckcopynet (CPXCENVptr env, 
                   CPXNETptr net, 
                   int objsen, 
                   int nnodes, 
                   const double * supply, 
                   char ** nnames, 
                   int narcs, 
                   const int * fromnode, 
                   const int * tonode, 
                   const double * low, 
                   const double * up, 
                   const double * obj, 
                   char ** aname)

Description

The routine CPXNETcheckcopynet performs a consistency check on the argument passed to the routine CPXNETcopynet.

The CPXNETcheckcopynet routine has the same argument list as the CPXNETcopynet routine.

Example

status = CPXNETcheckcopynet (env, net, CPX_MAX, nnodes, supply, 
                   nnames, narcs, fromnode, tonode, 
                   lb, ub, obj, anames);

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXNETchgarcname

Category                  Global Function
Definition File           cplex.h
Include Files             cplex.h
Synopsis                  int CPXPUBLIC CPXNETchgarcname(CPXENVptr env,
                           CPXNETptr net,
                           int cnt,
                           const int * indices,
                           char ** newname)
Description               This routine CPXNETchgarcname changes the names of a set of arcs in the network stored in a network problem object.
Example                   status = CPXNETchgarcname (env, net, 10, indices, newname);
Parameters
  env                      A pointer to the CPLEX environment as returned by CPXopenCPLEX.
  net                      A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
  cnt                      An integer that indicates the total number of arc names to be changed. Thus cnt specifies the length of the arrays indices and anames.
  indices                  An array of length cnt containing the numeric indices of the arcs for which the names are to be changed.
  newname                  An array of length cnt containing the strings of the new arc names specified in indices.
Returns                   The routine returns zero on success and nonzero if an error occurs.
CPXNETchgarcnodes

Category Global Function
Definition File cplex.h
Include Files cplex.h

Synopsis
int CPXPUBLIC CPXNETchgarcnodes(CPXENVptr env,
CPXNETptr net,
int cnt,
const int * indices,
const int * fromnode,
const int * tonode)

Description
The routine CPXNETchgarcnodes can be used to change the nodes for a set of arcs in
the network stored in a network problem object.
Any solution information stored in the problem object is lost.

Example
status = CPXNETchgarcs (env, net, cnt, indices, newfrom, newto);

Parameters
env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
net
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
cnt
Number of arcs to change.
indices
An array of arc indices that indicate the arcs to be changed. This array must have a length
of at least cnt. All indices must be in the range [0, narcs-1].
fromnode
An array of indices of the “from node” for which the arcs are to be changed. All indices
must be in the range [0, nnodes-1]. The size of the array must be at least cnt.
tonode
An array of indices of the “to node” for which the arcs are to be changed. All indices
must be in the range [0, nnodes-1]. The size of the array must be at least cnt.
**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXNETchgbds

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXNETchgbds (CPXCENVptr env,
    CPXNETptr net,
    int cnt,
    const int * indices,
    const char * lu,
    const double * bd)
```

**Description**
The routine CPXNETchgbds is used to change the upper, lower, or both bounds on the flow for a set of arcs in the network stored in a network problem object. The flow value of an arc can be fixed to a value by setting both bounds to that value.

Any solution information stored in the problem object is lost.

**Example**

```c
status = CPXNETchgbds (env, net, cnt, index, lu, bd);
```

### Indicators to change lower, upper bounds of flows through arcs

<table>
<thead>
<tr>
<th>lu[i]</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'L'</td>
<td>The lower bound of arc index[i] is changed to bd[i]</td>
</tr>
<tr>
<td>'U'</td>
<td>The upper bound of arc index[i] is changed to bd[i]</td>
</tr>
<tr>
<td>'B'</td>
<td>Both bounds of arc index[i] are changed to bd[i]</td>
</tr>
</tbody>
</table>

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **net**
  A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

- **cnt**
  Number of bounds to change.

- **indices**
  Indicators to change lower, upper bounds of flows through arcs.
An array of arc indices that indicate the bounds to be changed. This array must have a length of at least \( \text{cnt} \). All indices must be in the range \([0, \ n\text{arcs}-1]\).

\( lu \)

An array indicating which bounds to change. This array must have a length of at least \( \text{cnt} \). The indicators appear in the table.

\( bd \)

An array of bound values. This array must have a length of at least \( \text{cnt} \). Values greater than or equal to \( \text{CPX}_\text{INFBOUND} \) and less than or equal to \( -\text{CPX}_\text{INFBOUND} \) are considered infinity or -infinity, respectively.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
**CPXNETchgname**

**Category**  Global Function

**Definition File**  cplex.h

**Include Files**  cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXNETchgname(CPXENVptr env,
CPXNETptr net,
int key,
int vindex,
const char * name_str)
```

**Description**

The routine `CPXNETchgname` changes the name of a node or an arc in the network stored in a network problem object.

**Values of key in CPXNETchgname**

<table>
<thead>
<tr>
<th>key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'a'</td>
<td>Indicates the arc name is to be changed.</td>
</tr>
<tr>
<td>'n'</td>
<td>Indicates the node name is to be changed.</td>
</tr>
</tbody>
</table>

**Example**

```c
status = CPXNETchgname (env, net, 'a', 10, "arc10");
```

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
- **net**
  A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.
- **key**
  A character to indicate whether an arc name should be changed, or a node name should be changed.
- **vindex**
  The index of the arc or node whose name is to be changed.
- **name_str**
  The new name for the arc or node.
Returns

The routine returns zero on success and nonzero if an error occurs.
**CPXNETchgnodename**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**
```c
int CPXPUBLIC CPXNETchgnodename (CPXCENVptr env,
                             CPXNETptr net,
                             int cnt,
                             const int * indices,
                             char ** newname)
```

**Description**
The routine `CPXNETchgnodename` changes the names of a set of nodes in the network stored in a network problem object.

**Example**
```c
status = CPXNETchgnodename (env, net, 10, indices, newname);
```

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`

- **net**
  A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`

- **cnt**
  An integer that indicates the total number of node names to be changed. Thus `cnt` specifies the length of the arrays `indices` and `newname`.

- **indices**
  An array of length `cnt` containing the numeric indices of the nodes for which the names are to be changed.

- **newname**
  An array of length `cnt` containing the strings of the new node names specified in `indices`.

**Returns**
The routine returns zero on success and nonzero if an error occurs.
CPXNETchgobj

Category          Global Function
Definition File   cplex.h
Include Files    cplex.h
Synopsis          int CPXPUBLIC CPXNETchgobj (CPXCENVptr env, 
                  CPXNETptr net, 
                  int cnt, 
                  const int * indices, 
                  const double * obj)

Description       The routine CPXNETchgobj is used to change the objective values for a set of arcs in 
                   the network stored in a network problem object. 
                   Any solution information stored in the problem object is lost.
                   Example
                   status = CPXNETchgobj (env, net, cnt, indices, newobj);

Parameters

  env                  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
  net                  A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
  cnt                  Number of arcs for which the objective values are to be changed.
  indices              An array of indices that indicate the arcs for which the objective values are to be 
                        changed. This array must have a length of at least cnt. The indices must be in the range 
                        [0, narcs-1].
  obj                   An array of the new objective values for the arcs. This array must have a length of at least 
                        cnt.

Returns

  The routine returns zero on success and nonzero if an error occurs.
CPXNETchgobjsen

Category             Global Function
Definition File      cplex.h
Include Files        cplex.h
Synopsis             int CPXPUBLIC CPXNETchgobjsen(CPXENVptr env,
                          CPXNETptr net,
                          int maxormin)

Description          The routine CPXNETchgobjsen is used to change the sense of the network problem
to a minimization or maximization problem.
                      Any solution information stored in the problem object is lost.

Changed optimization sense in a network problem

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_MAX</td>
<td>For a maximization problem.</td>
</tr>
<tr>
<td>CPX_MIN</td>
<td>For a minimization problem.</td>
</tr>
</tbody>
</table>

Example

status = CPXNETchgobjsen (env, net, CPX_MAX);

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

net
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

maxormin
New optimization sense for the network problem. The possible values are in the table.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXNETchgsupply

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXNETchgsupply (CPXCENVptr env,
                              CPXNETptr net,
                              int cnt,
                              const int * indices,
                              const double * supply)
```

Description:
The routine CPXNETchgsupply is used to change supply values for a set of nodes in
the network stored in a network problem object.
Any solution information stored in the problem object is lost.

Example:

```c
status = CPXNETchgsupply (env, net, cnt, indices, supply);
```

Parameters:

- `env`
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- `net`
  A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
- `cnt`
  An integer indicating the number of nodes for which the objective values are to be
  changed.
- `indices`
  An array of indices that indicate the nodes for which the supply values are to be changed.
  This array must have a length of at least `cnt`. The indices must be in the range [0, nnodes-1].
- `supply`
  An array to contain the new supply values. This array must have a length of at least `cnt`.

Returns:
The routine returns zero on success and nonzero if an error occurs.
CPXNETcopybase

Category          Global Function
Definition File   cplex.h
Include Files     cplex.h
Synopsis          int CPXPUBLIC CPXNETcopybase(CPXENVptr env,
                             CPXNETptr net,
                             const int * astat,
                             const int * nstat)

Description       The routine CPXNETcopybase can be used to set the network basis for a network problem object. It is not necessary to load a basis prior to optimizing a problem, but a very good starting basis may increase the speed of optimization significantly. A copied basis does not need to be feasible to be used by the network optimizer. Any solution information stored in the problem object is lost.

Example           status = CPXNETcopybase (env, net, arc_stat, node_stat);

Parameters        env

Table 1: Status of arcs in astat

<table>
<thead>
<tr>
<th>CPX_BASIC</th>
<th>if the arc is to be basic</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_AT_LOWER</td>
<td>if the arc is to be nonbasic and its flow is on the lower bound</td>
</tr>
<tr>
<td>CPX_AT_UPPER</td>
<td>if the arc is to be nonbasic and its flow is on the upper bound</td>
</tr>
<tr>
<td>CPX_FREE_SUPER</td>
<td>if the arc is to be nonbasic but is free. In this case its flow is set to 0</td>
</tr>
</tbody>
</table>

Table 2: Status of artificial arcs in nstat

<table>
<thead>
<tr>
<th>CPX_BASIC</th>
<th>if the arc is to be basic</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_AT_LOWER</td>
<td>if the arc is to be nonbasic and its flow is set to 0</td>
</tr>
</tbody>
</table>
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

net

A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

astat

Array of status values for network arcs. Each arc needs to be assigned one of the values in Table 1.

nstat

Array of status values for artificial arcs from each node to the root node. Each artificial arc needs to be assigned one of the values in Table 2. At least one of the artificial arcs must be assigned the status CPX_BASiC for a network basis.

Returns

The routine returns zero on success and nonzero if an error occurs.
**CPXNETcopynet**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXNETcopynet (CPXCENVptr env,
    CPXNETptr net,
    int objsen,
    int nnodes,
    const double * supply,
    char ** nnames,
    int narcs,
    const int * fromnode,
    const int * tonode,
    const double * low,
    const double * up,
    const double * obj,
    char ** anames)
```

**Description**
The routine `CPXNETcopynet` copies a network to a network object, overriding any other network saved in the object. The network to be copied is specified by providing:

- the objective sense
- number of nodes
- supply values for each node
- names for each node
- number of arcs
- indices of the “from nodes” (or, equivalently, the tail nodes) for each arc
- indices of the “to nodes” (or, equivalently, the head nodes) for each arc
- lower and upper bounds on flow through each arc
- cost for flow through each arc

The arcs are numbered according to the order given in the `fromnode` and `tonode` arrays. Some of the parameters are optional and replaced by default values if NULL is passed for them.

**Example**
status = CPXNETcopynet (env, net, CPX_MAX, nnodes, supply, NULL, 
narcs, fromnode, tonode, NULL, NULL, obj, 
NULL);

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

net
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

objs
Optimization sense of the network to be copied. It may take values CPX_MAX for a maximization problem or CPX_MIN for a minimization problem.

nnodes
Number of nodes to be copied to the network object.

supply
Supply values for the nodes. If NULL is passed all supply values default to 0 (zero). Otherwise, the size of the array must be at least nnodes.

names
Pointer to an array of names for the nodes. If NULL is passed, no names are assigned to the nodes. Otherwise, the size of the array must be at least nnodes and every name in the array must be a string terminating in 0.

narcs
Number of arcs to be copied to the network object.

fromnode
The array of indices in each arc's "from node." The indices must be in the range [0, nnodes-1]. The size of the array must be at least narcs.

tonode
The array of indices in each arc's "to node." The indices must be in the range [0, nnodes-1]. The size of the array must be at least narcs.

low
Pointer to an array of lower bounds on the flow through arcs. If NULL is passed, all lower bounds default to 0 (zero). Otherwise, the size of the array must be at least narcs. Values less than or equal to -CPX_INFBOUND are considered -infinity.
up
Pointer to an array of upper bounds on the flow through arcs. If NULL is passed, all lower bounds default to CPX_INFBOUND. Otherwise, the size of the array must be at least narcs. Values greater than or equal to CPX_INFBOUND are considered infinity.

obj
Pointer to an array of objective values for flow through arcs. If NULL is passed, all objective values default to 0 (zero). Otherwise, the size of the array must be at least narcs.

anames
Pointer to an array of names for the arcs. If NULL is passed, no names are assigned to the nodes. Otherwise, the size of the array must be at least narcs, and every name in the array must be a string terminating in 0 (zero).

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXNETcreateprob

Category                      Global Function
Definition File               cplex.h
Include Files                 cplex.h
Synopsis
CPXNETptr CPXNETcreateprob(CPXENVptr env,
int * status_p,
const char * name_str)

Description
The routine CPXNETcreateprob constructs a new network problem object. The new object contains a minimization problem for a network with 0 nodes and 0 arcs. Other network problem data can be copied to a network with one of the routines CPXNETaddnodes, CPXNETaddarcs, CPXNETcopynet, CPXNETextract, or CPXNETreadcopyprob.

Example
CPXNETptr net = CPXNETcreateprob (env, &status, "mynet");

See Also
CPXNETaddnodes, CPXNETaddarcs, CPXNETcopynet, CPXNETextract, CPXNETreadcopyprob

Parameters
env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
status_p
A pointer to an integer used to return any error code produced by this routine.
name_str
Name of the network to be created.

Returns
If the operation is successful, CPXNETcreateprob returns the newly constructed network problem object; if not, it returns either NULL or a nonzero value to indicate an error. In case of an error, the value pointed to by status_p contains an integer indicating the cause of the error.
CPXNETdelarcs

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXNETdelarcs(CPXENVptr env,
                            CPXNETptr net,
                            int begin,
                            int end)
```

Description:
The routine CPXNETdelarcs is used to remove a range of arcs from the network stored in a network problem object. The remaining arcs are renumbered starting at zero; their order is preserved. If removing arcs disconnects some nodes from the rest of the network, the disconnected nodes remain part of the network.

Any solution information stored in the problem object is lost.

Example:

```c
status = CPXNETdelarcs (env, net, 10, 20);
```

Parameters:

- `env`
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- `net`
  A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

- `begin`
  Index of the first arc to be deleted.

- `end`
  Index of the last arc to be deleted.

Returns:
The routine returns zero on success and nonzero if an error occurs.
CPXNETdelnodes

Category          Global Function
Definition File   cplex.h
Include Files    cplex.h
Synopsis          int CPXPUBLIC CPXNETdelnodes(CPXENVptr env,
                CPXNETptr net,
                int begin,
                int end)

Description       The routine CPXNETdelnodes is used to remove a range of nodes from the network
                   stored in a network problem object. The remaining nodes are renumbered starting at
                   zero; their order is preserved. All arcs incident to the nodes that are deleted are also
                   deleted from the network.
                   Any solution information stored in the problem object is lost.

Example

                   status = CPXNETdelnodes (env, net, 10, 20);

Parameters

                   env
                   A pointer to the CPLEX environment as returned by CPXopenCPLEX.
                   net
                   A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
                   begin
                   Index of the first node to be deleted.
                   end
                   Index of the last node to be deleted.

Returns          The routine returns zero on success and nonzero if an error occurs.
CPXNETdelset

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXNETdelset (CPXCENVptr env,
CPXNETptr net,
int * whichnodes,
int * whicharcs)
```

**Description**
The routine `CPXNETdelset` is used to delete a set of nodes and arcs from the network stored in a network problem object. The remaining nodes and arcs are renumbered starting at zero; their order is preserved.

Any solution information stored in the problem object is lost.

**Example**

```c
status = CPXNETdelset (env, net, whichnodes, whicharcs);
```

**Parameters**

- `env`
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- `net`
  A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.

- `whichnodes`
  Array of size at least `CPXNETgetnumnodes` that indicates the nodes to be deleted. If `whichnodes[i] == 1`, the node is deleted. For every node deleted, all arcs incident to it are deleted as well. After termination, `whichnode[j]` indicates either the position to which node with index `j` before deletion has been moved or, -1 if the node has been deleted. If NULL is passed, no nodes are deleted.

- `whicharcs`
  Array indicating the arc to be deleted. Every arc `i` in the network with `whicharcs[i] == 1` is deleted. After termination, `whicharc[j]` indicates either the position to which arc with index `j` before deletion has been moved or, -1 if the arc has been deleted. This array also contains the deletions due to removed nodes. If NULL is passed, the only arcs deleted are those that are incident to nodes that have been deleted.

**Returns**
The routine returns zero on success and nonzero if an error occurs.
CPXNETextract

**Category**          Global Function

**Definition File**    cplex.h

**Include Files**      cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXNETextract (CPXCENVptr env,
                             CPXNETptr net,
                             CPXCLPptr lp,
                             int * colmap,
                             int * rowmap)
```

**Description**

The routine CPXNETextract finds an embedded network in the LP stored in a CPLEX problem object and copies it as a network to the network problem object, `net`. The extraction algorithm is controlled by the environment parameter `CPX_PARAM_NETFIND`.

If the CPLEX problem object has a basis, an attempt is made to copy the basis to the network object. However, this may fail if the statuses corresponding to the rows and columns of the subnetworks do not form a basis. Even if the entire LP is a network, it may not be possible to load the basis to the network object if none of the slack or artificial variables are basic.

The size of `colmap` must be at least `CPXnumcols(env, lp) + CPXnumrows(env, lp)`.

The size of `rowmap` must be at least `CPXnumrows(env, lp) + 1`.

**Example**

```c
status = CPXNETextract (env, net, lp, colmap, rowmap);
```

**Parameters**

`env`

A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

`net`

A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.

`lp`

A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

`colmap`
If not NULL, after completion colmap[i] contains the index of the LP column that has been mapped to arc i. If colmap[i] < 0, arc i corresponds to the slack variable for row -colmap[i]-1.

The size of colmap must be at least CPXnumcols(env, lp) + CPXnumrows(env, lp).

rowmap

If not NULL, after completion rowmap[i] contains the index of the LP row that has been mapped to node i. If colmap[i] < 0, node i is a dummy node that has no corresponding row in the LP.

The size of rowmap must be least CPXnumrows(env, lp) + 1.

Example

```c
status = CPXNETextract (env, net, lp, colmap, rowmap);
```

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXNETfreeprob

Category                      Global Function
Definition File               cplex.h
Include Files                 cplex.h
Synopsis
int CPXPUBLIC CPXNETfreeprob(CPXENVptr env,
                                   CPXNETptr * net_p)

Description
The routine CPXNETfreeprob deletes the network problem object pointed to by net_p. This also deletes all network problem data and solution data stored in the network problem object.

Example
CPXNETfreeprob (env, &net);

Parameters
env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
net_p
CPLEX network problem object to be deleted.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXNETgetarcindex

Category  Global Function

Definition File  cplex.h

Include Files  cplex.h

Synopsis  

```c
int CPXPUBLIC CPXNETgetarcindex(CPXENVptr env,
                              CPXNETptr net,
                              const char * lname_str,
                              int * index_p)
```

Description  The routine CPXNETgetarcindex returns the index of the specified arc (in the network stored in a network problem object) in the integer pointed to by `index_p`.

Example  

```c
status = CPXNETgetarcindex (env, net, "from_a_to_b", &index);
```

Parameters  

- **env**  
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- **net**  
  A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.

- **lname_str**  
  Name of the arc to look for.

- **index_p**  
  A pointer to an integer to hold the arc index. If the routine is successful, `index_p` contains the index number; otherwise, `index_p` is undefined.

Returns  The routine returns zero on success and nonzero if an error occurs.
**CPXNETgetarcname**

**Category**  Global Function

**Definition File**  cplex.h

**Include Files**  cplex.h

**Synopsis**  
```
int CPXPUBLIC CPXNETgetarcname(CPXCENVptr env,
    CPXCNETptr net,
    char ** nnames,
    char * namestore,
    int namespc,
    int * surplus_p,
    int begin,
    int end)
```

**Description**  
The routine CPXNETgetarcname is used to access the names of a range of arcs in a network stored in a network problem object. The beginning and end of the range, along with the length of the array in which the arc names are to be returned, must be specified.

**Example**  
```
status = CPXNETgetarcname (env, net, nnames, namestore, namespc,
    &surplus, 0, narcs-1);
```

**Parameters**  

- **env**  
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **net**  
  A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

- **nnames**  
  Where to copy pointers to arc names stored in the namestore array. The length of this array must be at least (end-begin+1). The pointer to the name of arc i is returned in nnames[i-begin].

- **namestore**  
  Array of characters to which the specified arc names are to be copied. It may be NULL if namespc is 0.

- **namespc**  
  Length of the namestore array.

- **surplus_p**  
  A pointer to the surplus value array.
Pointer to an integer to which the difference between namespc and the number of characters required to store the requested names is returned. A nonnegative value indicates that namespc was sufficient. A negative value indicates that it was insufficient. In that case, CPXERR_NEGATIVE_SURPLUS is returned and the negative value of surplus_p indicates the amount of insufficient space in the array namestore.

begin

Index of the first arc for which a name is to be obtained.

end

Index of the last arc for which a name is to be obtained.

Returns

The routine returns zero on success and nonzero if an error occurs. The value CPXERR_NEGATIVE_SURPLUS indicates that insufficient space was available in the namestore array to hold the names.
CPXNETgetarcnodes

Category          Global Function
Definition File   cplex.h
Include Files    cplex.h

Synopsis
int CPXPUBLIC CPXNETgetarcnodes(CPXENVptr env,
                                 CPXNETptr net,
                                 int * fromnode,
                                 int * tonode,
                                 int begin,
                                 int end)

Description
The routine CPXNETgetarcnodes is used to access the “from nodes” and “to nodes”
for a range of arcs in the network stored in a network problem object.

Example
status = CPXNETgetarcnodes (env, net, fromnode, tonode,
                            0, cur_narcs-1);

Parameters
env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
net
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
fromnode
Array in which to write the “from node” indices of the requested arcs. If NULL is
passed, no “from node” indices are retrieved. Otherwise, the size of the array must be
(end-begin+1).
tonode
Array in which to write the “to node” indices of the requested arcs. If NULL is passed,
no “to node” indices are retrieved. Otherwise, the size of the array must be (end–
begin+1).
begin
Index of the first arc to get nodes for.
end
Index of the last arc to get nodes for.
**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXNETgetbase

Category Global Function
Definition File cplex.h
Include Files cplex.h
Synopsis

int CPXPUBLIC CPXNETgetbase (CPXCENVptr env,
CPXCNETptr net,
int * astat,
int * nstat)

Description
The routine CPXNETgetbase is used to access the network basis for a network problem object. Either of the arguments astat or nstat may be NULL.
For this function to succeed, a solution must exist for the problem object.

Example

status = CPXNETgetbase (env, net, astat, nstat);

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCplex.

net
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

Table 1: Status codes of network arcs

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_BASIC</td>
<td>If the arc is basic.</td>
</tr>
<tr>
<td>CPX_AT_LOWER</td>
<td>If the arc is nonbasic and its flow is on the lower bound.</td>
</tr>
<tr>
<td>CPX_AT_UPPER</td>
<td>If the arc is nonbasic and its flow is on the upper bound.</td>
</tr>
<tr>
<td>CPX_FREE_SUPER</td>
<td>If the arc is nonbasic but is free. In this case its flow is 0.</td>
</tr>
</tbody>
</table>

Table 2: Status of artificial arcs

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_BASIC</td>
<td>If the arc is basic.</td>
</tr>
<tr>
<td>CPX_AT_LOWER</td>
<td>If the arc is nonbasic and its flow is on the lower bound.</td>
</tr>
</tbody>
</table>
astat

An array in which the statuses for network arcs are to be written. After termination, 
astat[i] contains the status assigned to arc i of the network stored in net. The 
status may be one of the values in Table 1. If NULL is passed, no arc statuses are copied. 
Otherwise, astat must be an array of a size that is at least CPXNETgetnumarcs.

nstat

An array in which the statuses for artificial arcs from each node to the root node are to be 
written. After termination, nstat[i] contains the status assigned to the artificial arc 
from node i to the root node of the network stored in net. The status may be one of 
values in Table 2. If NULL is passed, no node statuses are copied. Otherwise, nstat 
must be an array of a size that is at least CPXNETgetnumnodes.

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXNETgetdj

Category Global Function

Definition File cplex.h

Include Files cplex.h

Synopsis

```
int CPXPUBLIC CPXNETgetdj(CPXENVptr env,
        CPXNETptr net,
        double * dj,
        int begin,
        int end)
```

Description

The routine CPXNETgetdj is used to access reduced costs for a range of arcs of the network stored in a network problem object.

For this function to succeed, a solution must exist for the problem object. If the solution is not feasible (CPXNETsolninfo returns 0 in argument pfeasind_p), the reduced costs are computed with respect to an objective function that penalizes infeasibilities.

Example

```
status = CPXNETgetdj (env, net, dj, 10, 20);
```

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

net
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

dj
Array in which to write requested reduced costs. If NULL is passed, no reduced cost values are returned. Otherwise, dj must point to an array of size at least (end-begin+1).

begin
Index of the first arc for which a reduced cost value is to be obtained.

end
Index of the last arc for which a reduced cost value is to be obtained.

Example

```
status = CPXNETgetdj (env, net, dj, 10, 20);
```
**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXNETgetitcnt

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis: int CPXPUBLIC CPXNETgetitcnt(CPXENVptr env, CPXNETptr net)

Description: The routine CPXNETgetitcnt accesses the total number of network simplex iterations for a previous call to CPXNETprimopt, for a network problem object.

Example:

itcnt = CPXNETgetitcnt (env, net);

Parameters:

- env: A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- net: A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

Returns: Returns the total number of network simplex iterations for the last call to CPXNETprimopt, for a network problem object. If CPXNETprimopt has not been called, zero is returned. If an error occurs, -1 is returned and an error message is issued.
CPXNETgetlb

Category   Global Function
Definition File   cplex.h
Include Files   cplex.h
Synopsis
int CPXPUBLIC CPXNETgetlb (CPXCENVptr env,
          CPXCNETptr net,
          double * low,
          int begin,
          int end)
Description
The routine CPXNETgetlb is used to access the lower capacity bounds for a range of arcs of the network stored in a network problem object.
Example
status = CPXNETgetlb (env, net, low, 0, cur_narcs-1);
Parameters
env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
net
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
low
Array in which to write the lower bound on the flow for the requested arcs. If NULL is passed, no lower bounds are retrieved. Otherwise, the size of the array must be (end-begin+1).
begin
Index of the first arc for which lower bounds are to be obtained.
end
Index of the last arc for which lower bounds are to be obtained.
Returns
The routine returns zero on success and nonzero if an error occurs.
CPXNETgetnodearcs

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**
```c
int CPXPUBLIC CPXNETgetnodearcs(CPXENVptr env,
CPXNETptr net,
int * arccnt_p,
int * arcbeg,
int * arc,
int arcspace,
int * surplus_p,
int begin,
int end)
```

**Description**
The routine CPXNETgetnodearcs is used to access the arc indices incident to a range of nodes in the network stored in a network problem object.

**Example**
```c
status = CPXNETgetnodearcs (env, net, &arccnt, arcbeg, arc,
arcspace, &surplus, begin, end);
```

**Parameters**

**env**
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

**net**
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

**arccnt_p**
A pointer to an integer to contain the total number of arc indices returned in the array arc.

**arcbeg**
An array to contain indices indicating where each of the requested arc lists start in array arc. Specifically, the list of arcs incident to node i (< end) consists of the indices in arcs in the range from arcbeg[i-begin] to arcbeg[(i+1)-begin]-1. The list of arcs incident to node end consists of the entries in arc in the range from arcbeg[begin-end] to *arccnt_p-1. This array must have a length of at least end-begin+1.
arc

An array to contain the arc indices for the arcs incident to the nodes in the specified range. May be NULL if arcspace is zero.

arcspace

An integer indicating the length of the array arc. May be zero.

surplus_p

A pointer to an integer to contain the difference between arcspace and the number of arcs incident to the nodes in the specified range. A nonnegative value indicates that arcspace was sufficient. A negative value indicates that it was insufficient and that the routine could not complete its task. In that case, CPXERR_NEGATIVE_SURPLUS is returned and the negative value of surplus_p indicates the amount of insufficient space in the array arc.

begin

Index of the first node for which arcs are to be obtained.

end

Index of the last node for which arcs are to be obtained.

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXNETgetnodeindex

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:
```c
int CPXPUBLIC CPXNETgetnodeindex(CPXENVptr env,
                                 CPXCNETptr net,
                                 const char * lname_str,
                                 int * index_p)
```

Description:
The routine CPXNETgetnodeindex returns the index of the specified node (in the network stored in a network problem object) in the integer pointed to by index_p.

Example:
```c
status = CPXNETgetnodeindex (env, net, "root", &index);
```

Parameters:
- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **net**
  A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
- **lname_str**
  Name of the node to look for.
- **index_p**
  A pointer to an integer to hold the node index. If the routine is successful, *index_p* contains the index number; otherwise, *index_p* is undefined.

Returns:
The routine returns zero on success and nonzero if an error occurs.
CPXNETgetnodename

Category Global Function

Definition File cplex.h

Include Files cplex.h

Synopsis

```c
int CPXPUBLIC CPXNETgetnodename(CPXENVptr env,
    CPXNETptr net,
    char ** nnames,
    char * namestore,
    int namespc,
    int * surplus_p,
    int begin,
    int end)
```

Description

The routine CPXNETgetnodename is used to obtain the names of a range of nodes in a network stored in a network problem object. The beginning and end of the range, along with the length of the array in which the node names are to be returned, must be specified.

Example

```c
status = CPXNETgetnodename (env, net, nnames, namestore, namespc, &surplus, 0, nnodes-1);
```

Parameters

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **net**
  A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

- **nnames**
  Where to copy pointers to node names stored in the namestore array. The length of this array must be at least (end-begin+1). The pointer to the name of node \(i\) is returned in nnames[\(i\)-begin].

- **namestore**
  Array of characters to which the specified node names are to be copied. It may be NULL if namespc is 0.

- **namespc**
  Length of the namestore array.

- **surplus_p**
Pointer to an integer in which the difference between namespc and the number of characters required to store the requested names is returned. A nonnegative value indicates that namespc was sufficient. A negative value indicates that it was insufficient. In that case, CPXERR_NEGATIVE_SURPLUS is returned and the negative value of surplus_p indicates the amount of insufficient space in the array namestore.

\begin{itemize}
  \item \textit{begin}
  \begin{itemize}
    \item Index of the first node for which a name is to be obtained.
  \end{itemize}
  \item \textit{end}
  \begin{itemize}
    \item Index of the last node for which a name is to be obtained.
  \end{itemize}
\end{itemize}

**Returns**

The routine returns zero on success and nonzero if an error occurs. The value CPXERR_NEGATIVE_SURPLUS indicates that there was not enough space in the namestore array to hold the names.
**CPXNETgetnumarcs**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXNETgetnumarcs(CPXENVptr env, CPXNETptr net)
```

**Description**
The routine `CPXNETgetnumarcs` is used to access the number of arcs in a network stored in a network problem object.

**Example**

```c
cur_narcs = CPXNETgetnumarcs(env, net);
```

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- **net**
  A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.

**Returns**
The routine returns the number of network arcs stored in a network problem object. If an error occurs, 0 is returned and an error message is issued.
CPXNETgetnumnodes

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXNETgetnumnodes(CPXENVptr env, 
                                CPXNETptr net)
```

**Description**
The routine `CPXNETgetnumnodes` is used to access the number of nodes in a network stored in a network problem object.

**Example**

```c
cur_nnodes = CPXNETgetnumnodes (env, net);
```

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- **net**
  A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.

**Returns**
The routine returns the number of network nodes stored in a network problem object. If an error occurs, 0 is returned and an error message is issued.
CPXNETgetobj

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXNETgetobj(CPXENVptr env,
                           CPXCNETptr net,
                           double * obj,
                           int begin,
                           int end)
```

Description:
The routine CPXNETgetobj is used to access the objective function values for a range of arcs in the network stored in a network problem object.

Example:

```c
status = CPXNETgetobj (env, net, obj, 0, cur_narcs-1);
```

Parameters:

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **net**
  A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

- **obj**
  Array in which to write the objective values for the requested range of arcs. If NULL is passed, no objective values are retrieved. Otherwise, obj must point to an array of size at least (end-begin+1).

- **begin**
  Index of the first arc for which the objective value is to be obtained.

- **end**
  Index of the last arc for which the objective value is to be obtained.

Returns:
The routine returns zero on success and nonzero if an error occurs.
**CPXNETgetobjsen**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXNETgetobjsen(CPXCENVptr env, 
                             CPXCNETptr net)
```

**Description**  
The routine `CPXNETgetobjsen` returns the sense of the objective function, that is, maximization or minimization, of a network problem object.

**Example**  
```c
objsen = CPXNETgetobjsen (env, net);
```

**Parameters**

- **env**  
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

- **net**  
  A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.

**Returns**  
The value `CPX_MAX` (-1) is returned for a maximization problem; the value `CPX_MIN` (1) is returned for a minimization problem. In case of an error, the value zero is returned.
CPXNETgetobjval

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXNETgetobjval(CPXENVptr env,
                               CPXNETptr net,
                               double * objval_p)
```

**Description**  
The routine `CPXNETgetobjval` returns the objective value of the solution stored in a network problem object.

If the current solution is not feasible, the value returned depends on the setting of the parameter CPX_PARAM_NETDISPLAY. If this parameter is set to CPXNET_PENALIZED_OBJECTIVE (2), an objective function value is reported that includes penalty contributions for arcs on which the flow at termination violated the flow bounds on that arc.

**Example**  
```c
status = CPXNETgetobjval (env, net, &objval);
```

**Parameters**

- **env**  
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **net**  
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

- **objval_p**  
Pointer to where the objective value is written. If NULL is passed, no objective value is returned.

**Returns**  
The routine returns zero on success and nonzero if an error occurs.
CPXNETgetphase1cnt

Category    Global Function
Definition File    cplex.h
Include Files    cplex.h
Synopsis

int CPXPUBLIC CPXNETgetphase1cnt (CPXCENVptr env, CPXNETptr net)

Description
The routine CPXNETgetphase1cnt returns the number of phase 1 network simplex
iterations for a previous call to CPXNETprimopt.

Example

phaset1cnt = CPXNETgetphase1cnt (env, net);

Parameters

env
A pointer to the CPLEX environment as returned by CFXopenCPLEX.

net
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

Returns
Returns the total number of phase 1 network simplex iterations for the last call to
CPXNETprimopt, for a CPXNETptr object. If CPXNETprimopt has not been
called, zero is returned. In an error occurs, -1 is returned and an error message is issued.
**CPXNETgetpi**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXNETgetpi(CPXCENVptr env,
                           CPXCNETptr net,
                           double * pi,
                           int begin,
                           int end)
```

**Description**  
The routine **CPXNETgetpi** is used to access dual values for a range of nodes in the network stored in a network problem object.

For this function to succeed, a solution must exist for the problem object.

**Example**  
```c
status = CPXNETgetpi (env, net, pi, 10, 20);
```

**Parameters**  
- `env`  
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- `net`  
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

- `pi`  
Array in which to write solution dual values for requested nodes. If NULL is passed, no data is returned. Otherwise, `pi` must point to an array of size at least `(end-begin+1)`.

- `begin`  
Index of the first node for which the dual value is to be obtained.

- `end`  
Index of the last node for which the dual value is to be obtained.

**Returns**  
The routine returns zero on success and nonzero if an error occurs.
**CPXNETgetprobname**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**
```c
int CPXPUBLIC CPXNETgetprobname(CPXENVptr env,
                                 CPXNETptr net,
                                 char * buf_str,
                                 int bufspace,
                                 int * surplus_p)
```

**Description**
The routine CPXNETgetprobname is used to access the name of the problem stored in a network problem object.

**Example**
```c
status = CPXNETgetprobname (env, net, name, namesize, &surplus);
```

**Parameters**
- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **net**
  A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
- **buf_str**
  Buffer into which the problem name is copied.
- **bufspace**
  Size of the array buf_str in bytes.
- **surplus_p**
  Pointer to an integer in which the difference between bufspace and the number of characters required to store the problem name is returned. A nonnegative value indicates that bufspace was sufficient. A negative value indicates that it was insufficient. In that case, CPXERR_NEGATIVE_SURPLUS is returned and the negative value of surplus_p indicates the amount of insufficient space in the array buf.

**Returns**
The routine returns zero on success and nonzero if an error occurs. The value CPXERR_NEGATIVE_SURPLUS indicates that there was not enough space in the buf array to hold the name.
**CPXNETgetslack**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXNETgetslack(CPXENVptr env,
                              CPXCNETptr net,
                              double * slack,
                              int begin,
                              int end)
```

**Description**  
The routine **CPXNETgetslack** is used to access slack values or, equivalently, violations of supplies/demands for a range of nodes in the network stored in a network problem object.

For this function to succeed, a solution must exist for the problem object.

**Example**  
```c
status = CPXNETgetslack (env, net, slack, 10, 20);
```

**Parameters**

- **env**  
A pointer to the CPLEX environment as returned by **CPXopenCPLEX**.

- **net**  
A pointer to a CPLEX network problem object as returned by **CPXNETcreateprob**.

- **slack**  
Array in which to write solution slack variables for requested nodes. If NULL is passed, no data is returned. Otherwise, slack must point to an array of size at least \( (end - begin + 1) \).

- **begin**  
Index of the first node for which a slack value is to be obtained.

- **end**  
Index of the last node for which a slack value is to be obtained.

**Returns**  
The routine returns zero on success and nonzero if an error occurs.
CPXNETgetstat

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis: int CPXPUBLIC CPXNETgetstat(CPXENVptr env, CPXCNETptr net)

Description: The routine CPXNETgetstat returns the solution status for a network problem object.

Example:

netstatus = CPXNETgetstat(env, net);

Parameters:

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

net
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

Returns:
If no solution is available for the network problem object, CPXNETgetstat returns 0 (zero). When a solution exists, the possible return values are:

<table>
<thead>
<tr>
<th>CPX_STAT_OPTIMAL</th>
<th>Optimal solution found.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_STAT_UNBOUNDED</td>
<td>Problem has an unbounded ray.</td>
</tr>
<tr>
<td>CPX_STAT_INFEASIBLE</td>
<td>Problem is infeasible.</td>
</tr>
<tr>
<td>CPX_STAT_INFOrUNB</td>
<td>Problem is infeasible or unbounded.</td>
</tr>
<tr>
<td>CPX_STAT_ABORT_IT_LIM</td>
<td>Aborted due to iteration limit.</td>
</tr>
<tr>
<td>CPX_STAT_ABORT_TIME_LIM</td>
<td>Aborted due to time limit.</td>
</tr>
<tr>
<td>CPX_STAT_ABORT_USER</td>
<td>Aborted on user request.</td>
</tr>
</tbody>
</table>
CPXNETgetsupply

Category          Global Function
Definition File   cplex.h
Include Files    cplex.h
Synopsis
int CPXPUBLIC CPXNETgetsupply (CPXCENVptr env,
                              CPXNETptr net,
                              double * supply,
                              int begin,
                              int end)

Description
The routine CPXNETgetsupply is used to obtain supply values for a range of nodes in the network stored in a CPLEX network problem object.

Example
status = CPXNETgetsupply (env, net, supply,
0, CPXNETgetnumnodes (env, net) - 1);

Parameters
env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
net
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
supply
Place where requested supply values are copied. If NULL is passed, no supply values are copied. Otherwise, the array must be of length at least (end-begin+1).
begin
Index of the first node for which a supply value is to be obtained.
end
Index of the last node for which a supply value is to be obtained.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXNETgetub

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXNETgetub(CPXENVptr env,
                           CPXCNETptr net,
                           double * up,
                           int begin,
                           int end)
```

Description:
The routine CPXNETgetub is used to access the upper capacity bounds for a range of arcs in the network stored in a network problem object.

Example:

```c
status = CPXNETgetub (env, net, up, 0, cur_narcs-1);
```

Parameters:

- **env**: A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **net**: A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
- **up**: Array in which to write the upper bound on the flow for the requested arcs. If NULL is passed, no upper bounds are retrieved. Otherwise, the array must be of size (end-begin+1).
- **begin**: Index of the first arc for which upper bounds are to be obtained.
- **end**: Index of the last arc for which upper bounds are to be obtained.

Returns:
The routine returns zero on success and nonzero if an error occurs.
CPXNETgetx

Category          Global Function
Definition File   cplex.h
Include Files     cplex.h
Synopsis

    int CPXPUBLIC CPXNETgetx(CPXENVptr env,
                          CPXCNETptr net,
                          double * x,
                          int begin,
                          int end)

Description

    The routine CPXNETgetx is used to access solution values or, equivalently, flow values
    for a range of arcs stored in a network problem object.
    For this routine to succeed, a solution must exist for the network problem object.

Example

    status = CPXNETgetx(env, net, x, 10, 20);

Parameters

    env
    A pointer to the CPLEX environment as returned by CPXopenCPLEX.
    net
    A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
    x
    Array in which to write solution (or flow) values for requested arcs. If NULL is passed,
    no solution vector is returned. Otherwise, x must point to an array of size at least (end-
    begin+1).
    begin
    Index of the first arc for which a solution (or flow) value is to be obtained.
    end
    Index of the last arc for which a solution (or flow) value is to be obtained.

Returns

    The routine returns zero on success and nonzero if an error occurs.
**CPXNETprimopt**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXNETprimopt(CPXENVptr env,  
                          CPXNETptr net)
```

**Description**  
The routine `CPXNETprimopt` can be called after a network problem has been copied to a network problem object, to find a solution to that problem using the primal network simplex method. When this function is called, the CPLEX primal network algorithm attempts to optimize the problem. The results of the optimization are recorded in the problem object and can be retrieved by calling the appropriate solution functions for that object.

**Example**  
```c
status = CPXNETprimopt (env, net);
```

See also the examples `netex1.c` and `netex2.c` in the standard distribution of the product.

**Parameters**  
- **env**  
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
- **net**  
  A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.

**Returns**  
The routine returns zero unless an error occurred during the optimization. Examples of errors include exhausting available memory (CPXERR_NO_MEMORY) or encountering invalid data in the CPLEX problem object (CPXERR_NO_PROBLEM). Exceeding a user-specified CPLEX limit, or proving the model infeasible or unbounded, are not considered errors. Note that a zero return value does not necessarily mean that a solution exists. Use query routines `CPXNETsolninfo`, `CPXNETgetstat`, and `CPXNETsolution` to obtain further information about the status of the optimization.
CPXNETreadcopybase

Category  Global Function
Definition File  cplex.h
Include Files  cplex.h
Synopsis  
```
int CPXPUBLIC CPXNETreadcopybase(CPXENVptr env, 
cPXNETptr net, 
const char * filename_str)
```
Description  
The routine CPXNETreadcopybase reads a basis file in BAS format and copies the basis to a network problem object. If no arc or node names are available for the problem object when reading the basis file, default names are assumed. Any basis that may have been created or saved in the problem object is replaced.
Example  
```
status = CPXNETreadcopybase (env, net, "netbasis.bas");
```
Parameters  
- env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- net
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
- filename_str
Name of the basis file to read.
Returns  
The routine returns zero on success and nonzero if an error occurs.
CPXNETsolninfo

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXNETsolninfo(CPXENVptr env,
                               CPXNETptr net,
                               int * pfeasind_p,
                               int * dfeasind_p)
```

Description:
The routine CPXNETsolninfo is used to access solution information computed by a previous call to CPXNETprimopt. The solution values are maintained in the object as long as no changes are applied to it with one of the CPXNETchg..., CPXNETcopy... or CPXNETadd... functions.

The arguments to CPXNETsolninfo are pointers to locations where data are to be written. The returned values indicate what is known about the primal and dual feasibility of the current solution. If either piece of information represented by an argument to CPXNETsolninfo is not required, a NULL pointer can be passed for that argument.

Example:

```c
status = CPXNETsolninfo (env, lp, &pfeasind, &dfeasind);
```

Parameters:

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **net**
  A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

- **pfeasind_p**
  A pointer to an integer variables indicating whether the current solution is known to be primal feasible. Note that a false return value does not necessarily mean that the solution is not feasible. It simply means that the relevant algorithm was not able to conclude that it was feasible when it terminated.

- **dfeasind_p**
  A pointer to an integer variables indicating whether the current solution is known to be dual feasible. Note that a false return value does not necessarily mean that the solution is not feasible. It simply means that the relevant algorithm was not able to conclude that it was feasible when it terminated.
Returns

The routine returns zero on success and nonzero if an error occurs.
CPXNETsolution

Category  
Global Function

Definition File  
cplex.h

Include Files  
cplex.h

Synopsis  
int CPXPUBLIC CPXNETsolution(CPXENVptr env,  
CPXNETptr net,  
int * netstat_p,  
double * objval_p,  
double * x,  
double * pi,  
double * slack,  
double * dj)

Description  
The routine CPXNETsolution accesses solution values for a network problem object  
computed by a previous call to CPXNETprimopt for that object. The solution values  
are maintained in the object as long as no changes are applied to it with one of the  
CPXNETchg...,CPXNETcopy... or CPXNETadd... functions. Whether or not  
a solution exists can be determined by CPXNETsolninfo.

The arguments to CPXNETsolution are pointers to locations where data is to be  
written. Such data includes the solution status, the value of the objective function,  
primal, dual and slack values and the reduced costs.

Although all the above data exists after a successful call to CPXNETprimopt, it is  
possible that the user only needs a subset of the available data. Thus, if any part of the  
solution represented by an argument to CPXNETsolution is not required, a NULL  
pointer can be passed for that argument.

Example  
status = CPXNETsolution (env, net, &netstatus, &objval, x, pi,  
slack, dj);

Parameters  
env  
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

net  
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

netstat_p
Pointer to which the solution status is to be written. The specific values that
*netstat_p can take and their meanings are the same as the return values
documented for CPXNETgetstat.

objval_p

Pointer to which the objective value is to be written. If NULL is passed, no objective
value is returned. If the solution status is one of the CPX_STAT_ABORT codes, the value
returned depends on the setting of parameter CPX_PARAM_NETDISPLAY. If this
parameter is set to 2, objective function values that are penalized for infeasible flows are
used to compute the objective value of the solution. Otherwise, the true objective
function values are used.

x

Array to which the solution (flow) vector is to be written. If NULL is passed, no solution
vector is returned. Otherwise, x must point to an array of size at least that returned by
CPXNETgetnumarcs.

pi

Array to which the dual values are to be written. If NULL is passed, no dual values are
returned. Otherwise, pi must point to an array of size at least that returned by
CPXNETgetnumnodes.

slack

Array to which the slack values (violations of supplies/demands) are to be written. If
NULL is passed, no slack values are returned. Otherwise, slack must point to an array
of size at least that returned by CPXNETgetnumnodes.

dj

Array to which the reduced cost values are to be written. If NULL is passed, no reduced
cost values are returned. Otherwise, dj must point to an array of size at least that
returned by CPXNETgetnumarcs.

Returns

If a solution exists, it returns zero; if not, it returns nonzero to indicate an error.
CPXNETwriteprob

Category       Global Function
Definition File cplex.h
Include Files  cplex.h
Synopsis        int CPXPUBLIC CPXNETwriteprob(CPXENVptr env, 
                  CPXNETptr net, 
                  const char * filename_str, 
                  const char * format_str)

Description     The routine CPXNETwriteprob writes the network stored in a network problem object to a file. This can be done in CPLEX (.net) or DIMACS (.min) network file format (MIN) or as the LP representation of the network in any of the LP formats (.lp, .mps, or .sav).

If the file name ends with .gz, a compressed file is written.

<table>
<thead>
<tr>
<th>File extensions for network files</th>
</tr>
</thead>
<tbody>
<tr>
<td>net                        for CPLEX network format</td>
</tr>
<tr>
<td>min                        for DIMACS network format</td>
</tr>
<tr>
<td>lp                         for LP format of LP formulation</td>
</tr>
<tr>
<td>mps                        for MPS format of LP formulation</td>
</tr>
<tr>
<td>sav                        for SAV format of LP formulation</td>
</tr>
</tbody>
</table>

Example

    status = CPXNETwriteprob (env, net, "network.net", NULL);

Parameters

    env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

    net
A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.

    filename_str
Name of the network file to write, where the file extension specifies the file format unless overridden by the format argument. If the file name ends with .gz a compressed file is written in accordance with the selected file type.

    format_str
File format to generate. Possible values appear in the table. If NULL is passed, the format is inferred from the file name.

**Returns**
The routine returns zero on success and nonzero if an error occurs.

**Group optim.cplex.callable.util**
The general utilities in the ILOG CPLEX Callable Library.

<table>
<thead>
<tr>
<th>Global Functions Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXcloseCPLEX</td>
</tr>
<tr>
<td>CPXopenCPLEX</td>
</tr>
<tr>
<td>CPXversion</td>
</tr>
</tbody>
</table>

**Description**
These utilities initialize and close the ILOG CPLEX environment.
CPXcloseCPLEX

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXcloseCPLEX(CPXENVptr * env_p)
```

**Description**  
This routine frees all of the data structures associated with CPLEX and releases the license. It should be the last CPLEX routine called in any Callable Library application.

**Example**  
```c
status = CPXcloseCPLEX (&env);
```

See also lpex1.c in the *CPLEX User's Manual*.

**Parameters**  
`env_p`  
A pointer to a variable holding the pointer to the CPLEX environment as returned by CPXopenCPLEX.

**Returns**  
The routine returns zero on success and nonzero if an error occurs.
CPXopenCPLEX

Category              Global Function
Definition File       cplex.h
Include Files         cplex.h
Synopsis              CPXENVptr CPXPUBLIC CPXopenCPLEX(int * status_p)
Description
The routine CPXopenCPLEX initializes a CPLEX environment when accessing a license for CPLEX and works only if the computer is licensed for Callable Library use. The routine CPXopenCPLEX must be the first CPLEX routine called. The routine returns a pointer to a CPLEX environment. This pointer is used as a parameter to every other non-advanced CPLEX routine (except CPXmsg).

Example

```c
env = CPXopenCPLEX (&status);
```


Parameters

status_p
A pointer to an integer, where an error code is placed by this routine.

Returns
A pointer to the CPLEX environment. If an error occurs (including licensing problems), the value NULL is returned. The reason for the error is returned in the variable *status_p. If the routine is successful, then *status_p is 0 (zero).
CPXversion

Category Global Function
Definition File cplex.h
Synopsis CPXCHARptr CPXPUBLIC CPXversion(CPXENVptr env)
Description The routine CPXversion returns a pointer to a string indicating the version of the CPLEX library linked with the application. The caller should not change the string returned by this function.
Example
printf("CPLEX version is\n
Parameters
env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
Returns The routine returns NULL if the environment does not exist and the pointer to a string otherwise.

Group optim.cplex.callable.portability
The portability routines in the ILOG CPLEX Callable Library.

<table>
<thead>
<tr>
<th>Global Functions Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXfclose</td>
</tr>
<tr>
<td>CPXfopen</td>
</tr>
<tr>
<td>CPXfputs</td>
</tr>
<tr>
<td>CPXmsgstr</td>
</tr>
<tr>
<td>CPXputenv</td>
</tr>
<tr>
<td>CPXstrcpy</td>
</tr>
<tr>
<td>CPXstrlen</td>
</tr>
</tbody>
</table>

Description Portability routines are needed for Windows platforms. They may also be used on UNIX platforms.
CPXfclose

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis: int CPXPUBLIC CPXfclose(CPXFILEptr stream)

Description: The routine CPXfclose closes files that are used in conjunction with the routines CPXaddfpdest, CPXdelfpdest, and CPXsetlogfile. It is used in the same way as the standard C library function fclose. Files that are opened with the routine CPXfopen must be closed with the routine CPXfclose.

Example:

CPXfclose (fp);


Parameters:

stream
A pointer to a file opened by the routine CPXfopen.

Returns: This routine returns zero on success and nonzero if a failure occurs. The syntax is identical to the standard C library routine fclose.
CPXfopen

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
CPXFILEptr CPXPUBLIC CPXfopen(const char * filename_str,  
    const char * type_str)

**Description**  
The routine CPXfopen is used to open files to be used in conjunction with the routines CPXaddfpdest, CPXdelfpdest and CPXsetlogfile. It has the same arguments as the standard C library function fopen.

**Example**  

```c
fp = CPXfopen ("mylog.log", "w");
```

See also lpex5.c in the *ILOG CPLEX User's Manual*.

**Parameters**

- **filename_str**  
A pointer to a character string that contains the name of the file to be opened.

- **type_str**  
A pointer to a character string, containing characters according to the syntax of the standard C function fopen.

**Returns**  
The routine returns a pointer to an object representing an open file, or NULL if the file could not be opened. A CPXFILEptr is analogous to FILE *type in C language.
CPXfputs

Category       Global Function
Definition File  cplex.h
Include Files   cplex.h
Synopsis        int CPXPUBLIC CPXfputs(const char * s_str, CPXFILEptr stream)
Description     The routine CPXfputs can be used to write output to a file opened with CPXfopen. The purpose of this routine is to allow user-defined output in a file to be interspersed with the output created by using the routines CPXaddfpdest or CPXsetlogfile. The syntax of CPXfputs is the same as the standard C library function fputs.
Example         CPXfputs (*Solved first problem.

Parameters      s_str
                A pointer to a string to be output to the file.

                stream
                A pointer to a file opened by the routine CPXfopen.

Returns         This routine returns a nonnegative value if successful. Otherwise, it returns the system constant EOF (end of file).
**CPXmsgstr**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**
```c
int CPXPUBLIC CPXmsgstr(CPXCHANNELptr channel, const char * msg_str)
```

**Description**
The routine `CPXmsgstr` sends a character string to a CPLEX message channel. It is provided as an alternative to `CPXmsg`, which due to its variable-length argument list, cannot be used in some environments, such as Visual Basic.

**Example**
```
CPXmsgstr (p, q);
```

**Parameters**
- **channel**
The pointer to the channel receiving the message.
- **msg_str**
A pointer to a string that should be sent to the message channel.

**Returns**
The routine returns the number of characters in the string `msg`. 
**CPXputenv**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXputenv(const char * envsetting_str)
```

**Description**  
The routine `CPXputenv` sets an environment variable to be used by CPLEX. Use it instead of the standard C Library `putenv` function to make sure your application ports properly to Windows. Be sure to allocate the memory dynamically for the string passed to `CPXputenv`.

As with the `putenv` routine, the address of the character string goes directly into the environment. Therefore, the memory identified by the pointer must remain active throughout the remaining parts of the application where CPLEX runs. Since global or static variables are not thread safe, ILOG recommends dynamic memory allocation of the `envsetting` string.

**Example**

```c
char *envstr = NULL;
envstr = (char *) malloc (256);
if ( envstr != NULL ) {
    strcpy (envstr,
            "ILOG_LICENSE_FILE=c:\myapp\license\access.ilm");
    CPXputenv (envstr);
}
```

**Parameters**  
`envsetting_str`  
A string containing an environment variable assignment. This argument typically sets the `ILOG_LICENSE_FILE` environment variable that customizes the location of the license key.

**Returns**  
The routine returns 0 (zero) when it executes successfully and -1 when it fails.
**CPXstrcpy**

**Category** 
Global Function

**Definition File** 
cplex.h

**Synopsis** 
CPXCHARptr CPXPUBLIC CPXstrcpy(char * s1_str, const char * s2_str)

**Description** 
The routine CPXstrcpy is used to copy strings. It is exactly the same as the standard C library routine strcpy. This routine is provided so that strings passed to the message function routines (see CPXaddfuncdest) can be copied by languages that do not allow dereferencing of pointers (for example, older versions of Visual Basic).

**Example**

CPXstrcpy (p, q);

**Parameters**

- **s1_str**  
  A pointer to the string to hold the copy of the string pointed to by src.

- **s2_str**  
  A pointer to a string to be copied to dest.

**Returns**  
The routine returns a pointer to the string being copied to.
CPXstrlen

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis: int CPXPUBLIC CPXstrlen(const char * s_str)

Description: The routine CPXstrlen determines the length of a string. It is exactly the same as the standard C library routine strlen. This routine is provided so that strings passed to the message function routines (see CPXaddfuncdest) can be analyzed by languages that do not allow dereferencing of pointers (for example, older versions of Visual Basic).

Example:

len = CPXstrlen (p);

Parameters:

s_str
A pointer to a character string.

Returns:
The routine returns the length of the string.

Group optim.cplex.callable.debug

The ILOG CPLEX Callable Library routines for debugging data.

<table>
<thead>
<tr>
<th>Global Functions Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXcheckaddcols</td>
</tr>
<tr>
<td>CPXcheckaddrrows</td>
</tr>
<tr>
<td>CPXcheckchqcoeflist</td>
</tr>
<tr>
<td>CPXcheckcotypetype</td>
</tr>
<tr>
<td>CPXcheckcopylp</td>
</tr>
<tr>
<td>CPXcheckcopylpnames</td>
</tr>
<tr>
<td>CPXcheckcopyqsep</td>
</tr>
<tr>
<td>CPXcheckcopyquad</td>
</tr>
<tr>
<td>CPXcheckcopyzos</td>
</tr>
</tbody>
</table>
**CPXcheckvals**

**Description**

These routines help you validate data in your problem to verify that you are solving the problem you intend.
CPXcheckaddcols

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXcheckaddcols(CPXENVptr env,
   CPXCLPptr lp,
   int ccnt,
   int nzcnt,
   const double * obj,
   const int * cmatbeg,
   const int * cmatind,
   const double * cmatval,
   const double * lb,
   const double * ub,
   char ** colname)
```

Description:

The routine `CPXcheckaddcols` validates the arguments of the corresponding `CPXaddcols` routine. This data checking routine is found in source format in the file `check.c` which is provided with the standard CPLEX distribution. To call this routine, you must compile and link `check.c` with your program as well as the CPLEX Callable Library.

The `CPXcheckaddcols` routine has the same argument list as the `CPXaddcols` routine. The second argument, `lp`, is technically a pointer to a constant LP object of type `CPXCLPptr` rather than type `CPXLPtr`, as this routine will not modify the model. For most user applications, this distinction is unimportant.

Example:

```c
status = CPXcheckaddcols (env, lp, ccnt, nzcnt, obj, cmatbeg,
   cmatind, cmatval, lb, ub, newcolname);
```

Returns:

The routine returns nonzero if it detects an error in the data; it returns zero if it does not detect any data errors.
CPXcheckaddrows

Category  Global Function
Definition File  cplex.h
Include Files  cplex.h
Synopsis  
int CPXPUBLIC CPXcheckaddrows (CPXENVptr env,  
   CPXCLPptr lp,  
   int ccnt,  
   int rcnt,  
   int nzcnt,  
   const double * rhs,  
   const char * sense,  
   const int * rmatbeg,  
   const int * rmatind,  
   const double * rmatval,  
   char ** colname,  
   char ** rowname)

Description  
The routine CPXcheckaddrows validates the arguments of the corresponding  
CPXaddrows routine. This data checking routine is found in source format in the file  
check.c which is provided with the standard CPLEX distribution. To call this routine,  
you must compile and link check.c with your program as well as the CPLEX  
Callable Library.

The CPXcheckaddrows routine has the same argument list as the CPXaddrows  
routine. The second argument, lp, is technically a pointer to a constant LP object of  
type CPXCLPptr rather than type CPXLPptr, as this routine will not modify the  
model. For most user applications, this distinction is unimportant.

Example  

status = CPXcheckaddrows (env, lp, ccnt, rcnt, nzcnt, rhs,  
sense, rmatbeg, rmatind, rmatval,  
   newcolname, newrowname);

Returns  
The routine returns nonzero if it detects an error in the data; it returns zero if it does not  
detect any data errors.
CPXcheckchgcoeflist

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**
```c
int CPXPUBLIC CPXcheckchgcoeflist (CPXCENVptr env,
                                CPXCLPptr lp,
                                int numcoefs,
                                const int * rowlist,
                                const int * collist,
                                const double * vallist)
```

**Description**
The routine CPXcheckchgcoeflist validates the arguments of the corresponding CPXchgcoeflist routine. This data checking routine is found in source format in the file check.c which is provided with the standard CPLEX distribution. To call this routine, you must compile and link check.c with your program as well as the CPLEX Callable Library.

The CPXcheckchgcoeflist routine has the same argument list as the CPXchgcoeflist routine. The second argument, lp, is technically a pointer to a constant LP object of type CPXCLPptr rather than type CPXLPtr, as this routine will not modify the model. For most user applications, this distinction is unimportant.

**Example**
```c
status = CPXcheckchgcoeflist (env, lp, numcoefs, rowlist,
                               collist, vallist);
```

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **lp**
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

- **numcoefs**
  The number of coefficients to check, or, equivalently, the length of the arrays rowlist, collist, and vallist.

- **rowlist**
  An array of length numcoefs that with collist and vallist indicates the coefficients to check.

- **collist**
  A pointer to the column indices.
An array of length numcoefs that with rowlist and vallist indicates the coefficients to check.

vallist

An array of length numcoefs that with rowlist and collist indicates the coefficients to change. The entries rowlist[k], collist[k], and vallist[k] indicate that the matrix coefficient in row rowlist[k] and column collist[k] should be checked with respect to the value vallist[k].

Returns

The routine returns nonzero if it detects an error in the data; it returns zero if it does not detect any data errors.
CPXcheckcopyctype

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXcheckcopyctype(CPXENVptr env,
    CPXCLPptr lp,
    const char * xtype)
```

Description:

The routine **CPXcheckcopyctype** validates the arguments of the corresponding **CPXcopyctype** routine. This data checking routine is found in source format in the file *check.c* which is provided with the standard CPLEX distribution. To call this routine, you must compile and link *check.c* with your program as well as the CPLEX Callable Library.

The **CPXcheckcopyctype** routine has the same argument list as the **CPXcopyctype** routine. The second argument, *lp*, is technically a pointer to a constant LP object of type `CPXCLPptr` rather than type `CPXLPptr`, as this routine will not modify the model. For most user applications, this distinction is unimportant.

Example:

```c
status = CPXcheckcopyctype (env, lp, xtype);
```

Returns:

The routine returns nonzero if it detects an error in the data; it returns zero if it does not detect any data errors.
**Synopsis**

```c
int CPXPUBLIC CPXcheckcopylp(CPXENVptr env, CPXCLPptr lp, int numcols, int numrows, int objsen, const double * obj, const double * rhs, const char * sense, const int * matbeg, const int * matcnt, const int * matind, const double * matval, const double * lb, const double * ub, const double * rngval)
```

**Description**

The routine CPXcheckcopylp validates the arguments of the corresponding CPXcopylp routine. This data checking routine is found in source format in the file check.c which is provided with the standard CPLEX distribution. To call this routine, you must compile and link check.c with your program as well as the CPLEX Callable Library.

The CPXcheckcopylp routine has the same argument list as the CPXcopylp routine. The second argument, lp, is technically a pointer to a constant LP object of type CPXCLPptr rather than type CPXLPptr, as this routine will not modify the model. For most user applications, this distinction is unimportant.

**Example**

```c
status = CPXcheckcopylp (env, lp, numcols, numrows, objsen, obj, rhs, sense, matbeg, matcnt, matind, matval, lb, ub, rngval);
```

**Returns**

The routine returns nonzero if it detects an error in the data; it returns zero if it does not detect any data errors.
CPXcheckcopylpwnames

Category     Global Function

Definition File     cplex.h

Include Files     cplex.h

Synopsis

```c
int CPXPUBLIC CPXcheckcopylpwnames (CPXCENVptr env,
    CPXCLPptr lp,
    int numcols,
    int numrows,
    int objsen,
    const double * obj,
    const double * rhs,
    const char * sense,
    const int * matbeg,
    const int * matcnt,
    const int * matind,
    const double * matval,
    const double * lb,
    const double * ub,
    const double * rngval,
    char ** colname,
    char ** rowname)
```

Description

The routine CPXcheckcopylpwnames validates the arguments of the corresponding CPXcopylpwnames routine. This data checking routine is found in source format in the file check.c which is provided with the standard CPLEX distribution. To call this routine, you must compile and link check.c with your application as well as the CPLEX Callable Library.

The routine CPXcheckcopylpwnames has the same argument list as the routine CPXcopylpwnames. The second argument, lp, is technically a pointer to a constant LP object of type CPXCLPptr rather than type CPXLPtr, as this routine will not modify the model. For most user applications, this distinction is unimportant.

Example

```c
status = CPXcheckcopylpwnames (env,
    lp,
    numcols,
    numrows,
    objsen,
    obj,
    rhs,
    sense,
    matbeg,
    matcnt,
```
CPXcheckcopylpwnames

matind,
matval,
lb,
ub,
rngval,
colname,
cowname);

Returns

The routine returns nonzero if it detects an error in the data; it returns zero if it does not detect any data errors.
CPXcheckcopyqpsep

Category   Global Function

Definition File   cplex.h

Include Files   cplex.h

Synopsis   int CPXPUBLIC CPXcheckcopyqpsep(CPXENVptr env, CPXCLPptr lp, const double * qsepvec)

Description   The routine CPXcheckcopyqpsep validates the argument of the corresponding CPXcopyqpsep routine. This data checking routine is found in source format in the file check.c which is provided with the standard CPLEX distribution. To call this routine, you must compile and link check.c with your program as well as the CPLEX Callable Library.

The CPXcheckcopyqpsep routine has the same argument list as the CPXcopyqpsep routine. The second argument, lp, is technically a pointer to a constant LP object of type CPXCLPptr rather than type CPXLPptr, as this routine will not modify the model. For most user applications, this distinction is unimportant.

Example

status = CPXcheckcopyqpsep (env, lp, qsepvec);

Returns   The routine returns nonzero if it detects an error in the data; it returns zero if it does not detect any data errors.
CPXcheckcopyquad

Category  Global Function
Definition File  cplex.h
Include Files  cplex.h

Synopsis

int CPXPUBLIC CPXcheckcopyquad(CPXCENVptr env,
    CPXCLPptr lp,
    const int * qmatbeg,
    const int * qmatcnt,
    const int * qmatind,
    const double * qmatval)

Description

The routine CPXcheckcopyquad validates the arguments of the corresponding CPXcopyquad routine. This data checking routine is found in source format in the file check.c which is provided with the standard CPLEX distribution. To call this routine, you must compile and link check.c with your program as well as the CPLEX Callable Library.

The CPXcheckcopyquad routine has the same argument list as the CPXcopyquad routine. The second argument, lp, is technically a pointer to a constant LP object of type CPXCLPptr rather than type CPXLPptr, as this routine will not modify the model. For most user applications, this distinction is unimportant.

Example

status = CPXcheckcopyquad (env, lp, qmatbeg, qmatcnt,
    qmatind, qmatval);

Returns

The routine returns nonzero if it detects an error in the data; it returns zero if it does not detect any data errors.
CPXcheckcopySos

Category	Global Function
Definition File
cplex.h
Include Files
cplex.h
Synopsis
int CPXPUBLIC CPXcheckcopySos(CPXENVptr env,
    CPXCLPptr lp,
    int numsos,
    int numsosnz,
    const char * sostype,
    const int * sospri,
    const int * sosbeg,
    const int * sosind,
    const double * soswt)

Description
The routine CPXcheckcopySos validates the arguments of the corresponding
CPXcopySos routine. This data checking routine is found in source format in the file
check.c which is provided with the standard CPLEX distribution. To call this routine,
you must compile and link check.c with your program as well as the CPLEX
Callable Library.

The CPXcheckcopySos routine has the same argument list as the CPXcopySos
routine. The second argument, lp, is technically a pointer to a constant LP object
of type CPXCLPptr rather than type CPXLPptr, as this routine will not modify
the model. For most user applications, this distinction is unimportant.

Example

status = CPXcheckcopySos (env, lp, numsos, numsosnz, sostype, sospri,
    sosbeg, sosind, soswt);

Returns
The routine returns nonzero if it detects an error in the data; it returns zero if it does not
detect any data errors.
CPXcheckvals

Category Global Function

Definition File cplex.h

Include Files cplex.h

Synopsis

```c
int CPXPUBLIC CPXcheckvals(CPXENVptr env,
    CPXCLPptr lp,
    int cnt,
    const int * rowind,
    const int * colind,
    const double * values)
```

Description

The routine CPXcheckvals checks an array of indices and a corresponding array of values for input errors. The routine is useful for validating the arguments of problem modification routines such as CPXchgcoeflist, CPXchgbds, CPXchgobj, and CPXchgrhs. This data checking routine is found in source format in the file check.c which is provided with the standard CPLEX distribution. To call this routine, you must compile and link check.c with your program as well as the CPLEX Callable Library.

Example

Consider the following call to CPXchgobj:

```c
status = CPXchgobj (env, lp, cnt, indices, values);
```

The parameters to this routine could be checked with a call to CPXcheckvals like this:

```c
status = CPXcheckvals (env, lp, cnt, NULL, indices, values);
```

Parameters

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **lp**
  A pointer to a CPLEX problem object as returned by CPXcreateprob.

- **cnt**
  The length of the indices and values arrays to be examined.

- **rowind**
  An array containing row indices. May be NULL.
colind
An array containing column indices. May be NULL.

values
An array of values. May be NULL.

Returns
The routine returns zero on success and nonzero if an error occurs.

Group optim.cplex.callable.message
The ILOG CPLEX Callable Library routines for managing messages.

<table>
<thead>
<tr>
<th>Global Functions Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXaddchannel</td>
</tr>
<tr>
<td>CPXaddfpdest</td>
</tr>
<tr>
<td>CPXaddfuncdest</td>
</tr>
<tr>
<td>CPXdelchannel</td>
</tr>
<tr>
<td>CPXdelfpdest</td>
</tr>
<tr>
<td>CPXdelfuncdest</td>
</tr>
<tr>
<td>CPXdisconnectchannel</td>
</tr>
<tr>
<td>CPXflushchannel</td>
</tr>
<tr>
<td>CPXflushstdchannels</td>
</tr>
<tr>
<td>CPXgetchannels</td>
</tr>
<tr>
<td>CPXgeterrorstring</td>
</tr>
<tr>
<td>CPXgetlogfile</td>
</tr>
<tr>
<td>CPXmsg</td>
</tr>
<tr>
<td>CPXsetlogfile</td>
</tr>
</tbody>
</table>

Description
These routines make it possible for your application to control which messages from ILOG CPLEX appear on screen, which are sent to files. They also provide support for you to create your own messages.
**CPXaddchannel**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
CPXCHANNELptr CPXPUBLIC CPXaddchannel(CPXENVptr env)

**Description**  
The routine CPXaddchannel instantiates a new channel object.

**Example**  
mychannel = CPXaddchannel (env);

See also lpex5.c in the CPLEX User's Manual.

**Parameters**  
env

A pointer to the CPLEX environment as returned by CPXopenCPLEX.

**Returns**  
If successful, CPXaddchannel returns a pointer to the new channel object; otherwise, it returns NULL.
**CPXaddfpdest**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXaddfpdest (CPXCENVptr env,  
CPXCHANNELptr channel,  
CPXFILEptr fileptr)
```

**Description**  
The routine `CPXaddfpdest` adds a file to the list of message destinations for a channel. The destination list for all CPLEX-defined channels is initially empty.

**Example**
```
CPXaddfpdest (env, mychannel, fileptr);
```

See `lpex5.c` in the *CPLEX User's Manual*.

**Parameters**

- **env**  
  A pointer to the CPLEX environment as returned by `CPXopenCplex`.

- **channel**  
  A pointer to the channel for which destinations are to be added.

- **fileptr**  
  A pointer to the file to be added to the destination list. Before calling this routine, obtain this pointer with a call to `CPXfopen`.

**Returns**  
The routine returns zero on success and nonzero if an error occurs.
CPXaddfuncdest

Category  Global Function
Definition File  cplex.h
Include Files  cplex.h
Synopsis  
```c
int CPXPUBLIC CPXaddfuncdest(CPXENVptr env, 
    CPXCHANNELptr channel, 
    void * handle, 
    void(CPXPUBLIC *msgfunction)(void *, const char *) )
```
Description  The routine CPXaddfuncdest adds a function msgfunction to the message destination list for a channel. This routine allows users to “trap” messages instead of printing them. That is, when a message is sent to the channel, each destination that was added to the message destination list by CPXaddfuncdest calls its associated message.

To illustrate, consider an application in which a developer wishes to trap CPLEX error messages and display them in a dialog box that prompts the user for an action. Use CPXaddfuncdest to add the address of a function to the list of message destinations associated with the cpxerror channel. Then write the msgfunction routine. It must contain the code that controls the dialog box. When CPXmsg is called with cpxerror as its first argument, it calls the msgfunction routine, which can then display the error message.

```
Example

void msgfunction (void *handle, char *msg_string)
{
    FILE *fp;
    fp = (FILE *)handle;
    fprintf (fp, "\%s", msg_string);
}
status = CPXaddfuncdest (env, mychannel, fileptr, msgfunction);
```

Parameters

---

**Note:** The argument handle is a generic pointer that can be used to hold information needed by the msgfunction routine to avoid making such information global to all routines.
env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

channel
A pointer to the channel to which the function destination is to be added.

handle
A void pointer that can be used to pass arbitrary information into msgfunction.

msgfunction
A pointer to the function to be called when a message is sent to a channel.

See Also
CPXdelfuncdest

Returns
This routine returns zero on success and nonzero if an error occurs. Failure occurs when msgfunction is not in the message destination list or the channel does not exist.
CPXdelchannel

Category       Global Function
Definition File cplex.h
Include Files  cplex.h
Synopsis       void CPXPUBLIC CPXdelchannel(CPXENVptr env,
                         CPXCHANNELptr * channel_p)
Description    The routine CPXdelchannel flushes all message destinations for a channel, clears the message destination list, and frees the memory allocated to the channel. On completion, the pointer to the channel is set to NULL.

Example
CPXdelchannel (env, &mychannel);

See also lpex5.c in the ILOG CPLEX User's Manual.

Parameters

env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

channel_p
A pointer to the pointer to the channel containing the message destinations to be flushed, cleared, and destroyed.

Returns
This routine does not have a return value.
CPXdelfpdest

Category: Global Function
Definition File: cplex.h
Include Files: cplex.h

Synopsis:
```c
int CPXPUBLIC CPXdelfpdest (CPXCENVptr env,
   CPXCHANNELptr channel,
   CPXFILEptr fileptr)
```

Description:
The routine CPXdelfpdest removes a file from the list of message destinations for a channel. Failure occurs when the channel does not exist or the file pointer is not in the message destination list.

Example:
```c
CPXdelfpdest (env, mychannel, fileptr);
```


Parameters:
- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **channel**
  The pointer to the channel for which destinations are to be deleted.
- **fileptr**
  A CPXFILEptr for the file to be removed from the destination list.

Returns:
The routines return zero on success and nonzero if an error occurs.
**CPXdelfuncdest**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXdelfuncdest(CPXENVptr env,  
CPXCHANNELptr channel,  
void * handle,  
void(CPXPUBLIC *msgfunction)(void *, const char *) )
```

**Description**  
The routine **CPXdelfuncdest** removes the function **msgfunction** from the list of message destinations associated with a channel. Use **CPXdelfuncdest** to remove functions that were added to the list using **CPXaddfuncdest**.

To illustrate, consider an application in which a developer wishes to trap CPLEX error messages and display them in a dialog box that prompts the user for an action. Use **CPXaddfuncdest** to add the address of a function to the list of message destinations associated with the **cpxerror** channel. Then write the **msgfunction** routine. It must contain the code that controls the dialog box. When **CPXmsg** is called with **cpxerror** as its first argument, it calls the **msgfunction** routine, which then displays the error message.

---

**Note:** The handle parameter is a generic pointer that can be used to hold information needed by the **msgfunction** routine to avoid making such information global to all routines.

**Example**

```c
void msgfunction (void *handle, char *msg_string)  
{  
    FILE *fp;  
    fp = (FILE *)handle;  
    fprintf (fp, "%s", msg_string);  
}  
status = CPXdelfuncdest (env, mychannel, fileptr, msgfunction);
```

**Parameters**

- **env**
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

channel
The pointer to the channel to which the function destination is to be added.

handle
A void pointer that can be used in the msgfunction routine to direct the message to a file, the screen, or a memory location.

msgfunction
The pointer to the function to be called when a message is sent to a channel. For details about this callback function, see CPXaddfuncdest.

See Also
CPXaddfuncdest

Returns
The routines return zero on success and nonzero if an error occurs. Failure occurs when msgfunction is not in the message destination list or the channel does not exist.
**CPXdisconnectchannel**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
void CPXPUBLIC CPXdisconnectchannel(CPXENVptr env, CPXCHANNELptr channel)

**Description**  
The routine CPXdisconnectchannel flushes all message destinations associated with a channel and clears the corresponding message destination list.

**Example**  
CPXdisconnectchannel (env, mychannel);

**Parameters**

- **env**  
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **channel**  
A pointer to the channel containing the message destinations to be flushed and cleared.

**Returns**  
This routine does not have a return value.
CPXflushchannel

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis: void CPXPUBLIC CPXflushchannel(CPXENVptr env, CPXCHANNELptr channel)

Description: The routine CPXflushchannel flushes (outputs and clears the buffers of) all message destinations for a channel. Use this routine in cases when it is important to have output written to disk immediately after it is generated. For most applications this routine need not be used.

Example:

CPXflushchannel (env, mychannel);

Parameters:

- **env**: A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **channel**: A pointer to the channel containing the message destinations to be flushed.

Returns: This routine does not return a value.
**CPXflushstdchannels**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXflushstdchannels (CPXCENVptr env)
```

**Description**  
The routine `CPXflushstdchannels` flushes the output buffers of the four standard channels `cpxresults`, `cpxwarning`, `cpxerror`, and `cpxlog`. Use this routine where it is important to see all of the output created by CPLEX either on the screen or in a disk file without calling `CPXflushchannel` for each of the four channels.

**Example**  
```c
status = CPXflushstdchannels (env);
```

**Parameters**  

- **env**  
A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

**Returns**  
The routine returns zero on success and nonzero if an error occurs.
CPXgetchannels

Category          Global Function
Definition File   cplex.h
Include Files    cplex.h
Synopsis          int CPXPUBLIC CPXgetchannels(CPXENVptr env,
                        CPXCHANNELptr * cpxresults_p,
                        CPXCHANNELptr * cpxwarning_p,
                        CPXCHANNELptr * cpxerror_p,
                        CPXCHANNELptr * cpxlog_p)
Description       The routine CPXgetchannels obtains pointers to the four default channels created when CPXopenCPLEX is called. To manipulate the messages for any of these channels, this routine must be called.
Example

        status = CPXgetchannels (env, &cpxresults, &cpxwarning,
                        &cpxerror, &cpxlog);

See also lpex5.c in the ILOG CPLEX User's Manual.

Parameters

        env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.

cpxresults_p
A pointer to a variable of type CPXCHANNELptr to hold the address of the channel corresponding to cpxresults. May be NULL.

cpxwarning_p
A pointer to a variable of type CPXCHANNELptr to hold the address of the channel corresponding to cpxwarning. May be NULL.

cpxerror_p
A pointer to a variable of type CPXCHANNELptr to hold the address of the channel corresponding to cpxerror. May be NULL.

cpxlog_p
A pointer to a variable of type CPXCHANNELptr to hold the address of the channel corresponding to cpxlog. May be NULL.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXgeterrorstring

Category: Global Function

Definition File: cplex.h

Synopsis: CPXCCHARptr CPXPUBLIC CPXgeterrorstring(CPXENVptr env, int errcode, char * buffer_str)

Description: The routine CPXgeterrorstring returns an error message string corresponding to an error code. Error codes are returned by CPLEX routines when an error occurs.

Note: This routine allows the CPLEX environment parameter to be NULL so that errors caused by the routine CPXopenCPLEX can be translated.

Example:

```c
char *errstr;
errstr = CPXgeterrorstring (env, errcode, buffer);
if ( errstr != NULL ) {
    printf ("%s

; }
else {  
    printf ("CPLEX Error %d: Unknown error code. errcode);  
 }
```

Parameters:

- `env`: A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- `errcode`: The error code to be translated.
- `buffer_str`: A character string buffer. This buffer must be at least 510 characters to hold the error string.

Returns: This routine returns a pointer to the parameter `buffer` if the string does exist. In that case, `buffer` contains the error message string. It returns NULL if the error code does not have a corresponding string.
**CPXgetlogfile**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXgetlogfile(CPXENVptr env,
                           CPXFILEptr * logfile_p)
```

**Description**
The routine CPXgetlogfile accesses the log file to which messages from all four CPLEX-defined channels are written.

**Example**

```c
status = CPXgetlogfile (env, &logfile);
```

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by CPXopenPLEX.

- **logfile_p**
  The address of a CPXFILEptr variable. This routine sets logfile_p to be the file pointer for the current log file.

**Returns**
The routine returns zero on success and nonzero if an error occurs.
CPXmsg

Category  Global Function

Definition File cplex.h

Include Files cplex.h

Synopsis

```c
int CPXPUBVARARGS CPXmsg(CPXCHANNELptr channel,
    const char * format,
    ...)
```

Description

The routine CPXmsg writes a message to a specified channel. Like the C function printf, it takes a variable number of arguments comprising the message to be written. The list of variables specified after the format string should be at least as long as the number of format codes in the format. The format string and variables are processed by the C library function vsprintf or a substitute on systems that do not have the vsprintf function.

The formatted string is limited to 1024 characters, and is usually output with the C function fputs to each output destination in the output destination list for a channel, except when a function has been specified by the routine CPXaddfuncdest as a destination.

The CPLEX Callable Library uses CPXmsg for all message output. The CPXmsg routine may also be used in applications to send messages to either CPLEX-defined or user-defined channels.

**Note:** CPXmsg is the only non-advanced CPLEX routine not requiring the CPLEX environment parameter.

Example

```c
CPXmsg (mychannel, "The objective value was %f.
```

See lpex5.c in the CPLEX User’s Manual.

Parameters

- **channel**
  
  The pointer to the channel receiving the message.

- **format**
The format string controlling the message output. This string is used in a way identical to the format string in a `printf` statement.

**Returns**

At completion, `CPXmsg` returns the number of characters in the formatted result string.
CPXsetlogfile

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis: 

```c
int CPXPUBLIC CPXsetlogfile(CPXENVptr env, CPXFILEptr lfile)
```

Description: The routine CPXsetlogfile modifies the log file to which messages from all four CPLEX-defined channels are written.

**Note:** A call to CPXsetlogfile is equivalent to directing output from the cpxresults, cpxwarning, cpxerror and cpxlog message channels to a single file.

Example:

```c
status = CPXsetlogfile (env, logfile);
```

Parameters:

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **lfile**
  A CPXFILEptr to the log file. This routine sets lfile to be the file pointer for the current log file. A NULL pointer may be passed if no log file is desired. NULL is the default value. Before calling this routine, obtain this pointer with a call to CPXfopen.

Returns: The routine returns zero on success and nonzero if an error occurs.

Group optim.cplex.callablecallbacks

The ILOG CPLEX Callable Library routines for managing callbacks.
## Global Functions Summary

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXbranchcallbackbranchbds</td>
<td>Branch callback function to branch over variables</td>
</tr>
<tr>
<td>CPXbranchcallbackbranchcst</td>
<td>Branch callback function to branch over constraints</td>
</tr>
<tr>
<td>CPXbranchcallbackbranchgeneral</td>
<td>Branch callback function to branch over general constraints</td>
</tr>
<tr>
<td>CPXgetbranchcallbackfunc</td>
<td>Get function for the branch callback function</td>
</tr>
<tr>
<td>CPXgetcallbackctype</td>
<td>Get function for the callback type</td>
</tr>
<tr>
<td>CPXgetcallbackgloballb</td>
<td>Get function for the global lower bound</td>
</tr>
<tr>
<td>CPXgetcallbackglobalub</td>
<td>Get function for the global upper bound</td>
</tr>
<tr>
<td>CPXgetcallbackincumbent</td>
<td>Get function for the incumbent solution</td>
</tr>
<tr>
<td>CPXgetcallbackinfo</td>
<td>Get function for the callback information</td>
</tr>
<tr>
<td>CPXgetcallbacklp</td>
<td>Get function for the linear program information</td>
</tr>
<tr>
<td>CPXgetcallbacknodeintfeas</td>
<td>Get function for the node integer feasibility</td>
</tr>
<tr>
<td>CPXgetcallbacknodelp</td>
<td>Get function for the node linear program information</td>
</tr>
<tr>
<td>CPXgetcallbacknodeobjval</td>
<td>Get function for the node objective value</td>
</tr>
<tr>
<td>CPXgetcallbacknodestat</td>
<td>Get function for the node status</td>
</tr>
<tr>
<td>CPXgetcallbacknodeub</td>
<td>Get function for the node upper bound</td>
</tr>
<tr>
<td>CPXgetcallbackorder</td>
<td>Get function for the callback order</td>
</tr>
<tr>
<td>CPXgetcallbackpseudocosts</td>
<td>Get function for the pseudocosts</td>
</tr>
<tr>
<td>CPXgetcutcallbackfunc</td>
<td>Get function for the cut callback function</td>
</tr>
<tr>
<td>CPXgetdeletenodecallbackfunc</td>
<td>Get function for the delete node callback function</td>
</tr>
<tr>
<td>CPXgetheuristiccallbackfunc</td>
<td>Get function for the heuristic callback function</td>
</tr>
<tr>
<td>CPXgetincumbentcallbackfunc</td>
<td>Get function for the incumbent callback function</td>
</tr>
<tr>
<td>CPXgetlpcallbackfunc</td>
<td>Get function for the linear callback function</td>
</tr>
<tr>
<td>CPXgetmipcallbackfunc</td>
<td>Get function for the MIP callback function</td>
</tr>
<tr>
<td>CPXgetnetcallbackfunc</td>
<td>Get function for the network callback function</td>
</tr>
<tr>
<td>CPXgetsolvecallbackfunc</td>
<td>Get function for the solve callback function</td>
</tr>
<tr>
<td>CPXsetbranchcallbackfunc</td>
<td>Set function for the branch callback function</td>
</tr>
<tr>
<td>CPXsetcutcallbackfunc</td>
<td>Set function for the cut callback function</td>
</tr>
<tr>
<td>CPXsetdeletenodecallbackfunc</td>
<td>Set function for the delete node callback function</td>
</tr>
<tr>
<td>CPXsetheuristiccallbackfunc</td>
<td>Set function for the heuristic callback function</td>
</tr>
<tr>
<td>CPXsetincumbentcallbackfunc</td>
<td>Set function for the incumbent callback function</td>
</tr>
<tr>
<td>CPXsetqipcallbackfunc</td>
<td>Set function for the QP callback function</td>
</tr>
<tr>
<td>CPXsetmipcallbackfunc</td>
<td>Set function for the MIP callback function</td>
</tr>
<tr>
<td>CPXsetnetcallbackfunc</td>
<td>Set function for the network callback function</td>
</tr>
</tbody>
</table>
**Description**

These callback routines, also known as interrupt routines, make it possible for you to define your own functions and for your application to call those functions to interrupt and resume optimization. You can also use callbacks to access progress information while the optimization is in process.
**CPXbranchcallbackbranchbds**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXbranchcallbackbranchbds(CPXENVptr env,
 void * cbdata,
 int wherefrom,
 double nodeest,
 int cnt,
 const int * indices,
 const char * lu,
 const int * bd,
 void * userhandle,
 int * seqnum_p)
```

**Description**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXbranchcallbackbranchbds specifies the branches to be taken from the current node. It may be called only from within a user-written branch callback function.

Branch variables are in terms of the original problem if the parameter CPX_PARAM_MIPCBREDLP is set to CPX_OFF before the call to CPXmipopt that calls the callback. Otherwise, branch variables are in terms of the presolved problem.

**Parameters**

**env**

A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

**cbdata**

A pointer passed to the user-written callback. This argument must be the value of cbdata passed to the user-written callback.

**wherefrom**

An integer value that indicates where the user-written callback was called from. This argument must be the value of wherefrom passed to the user-written callback.
nodeest

A double that indicates the value of the node estimate for the node to be created with this branch. The node estimate is used to select nodes from the branch & cut tree with certain values of the NodeSel parameter.

cnt

An integer. The integer indicates the number of bound changes that are specified in the arrays indices, lu, and bd.

indices

An array. Together with lu and bd, this array defines the bound changes for the branch. The entry indices[i] is the index for the variable.

lu

An array. Together with indices and bd, this array defines the bound changes for each of the created nodes. The entry lu[i] is one of the three possible values indicating which bound to change: L for lower bound, U for upper bound, or B for both bounds.

bd

An array. Together with indices and lu, this array defines the bound changes for each of the created nodes. The entry bd[i] indicates the new value of the bound.

userhandle

A pointer to user private data that should be associated with the node created by this branch. May be NULL.

seqnum_p

A pointer to an integer. On return, that integer will contain the sequence number that CPLEX has assigned to the node created from this branch. The sequence number may be used to select this node in later calls to the node callback.

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXbranchcallbackbranchconstraints

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXbranchcallbackbranchconstraints(CPXCENVptr env,
    void * cbdata,
    int wherefrom,
    double nodeest,
    int rcnt,
    int nzcnt,
    const double * rhs,
    const char * sense,
    const int * rmatbeg,
    const int * rmatind,
    const double * rmatval,
    void * userhandle,
    int * seqnum_p)
```

**Description**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine **CPXbranchcallbackbranchconstraints** specifies the branches to be taken from the current node when the branch is specified by adding one or more constraints to the node problem. It may be called only from within a user-written branch callback function.

Constraints are in terms of the original problem if the parameter **CPX_PARAM_MIPCBREDLP** is set to **CPX_OFF** before the call to **CPXmipopt** that calls the callback. Otherwise, constraints are in terms of the presolved problem.

**Table 1: Values of sense[i]**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>less than or equal to constraint</td>
</tr>
<tr>
<td>E</td>
<td>equal to constraint</td>
</tr>
<tr>
<td>G</td>
<td>greater than or equal to constraint</td>
</tr>
</tbody>
</table>
Parameters

- env
  A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

- cbdata
  A pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.

- wherefrom
  An integer value that indicates where the user-written callback was called from. This parameter must be the value of wherefrom passed to the user-written callback.

- nodeest
  A double that indicates the value of the node estimate for the node to be created with this branch. The node estimate is used to select nodes from the branch & cut tree with certain values of the NodeSel parameter.

- rcnt
  An integer that indicates the number of constraints for the branch.

- nzcnt
  An integer that indicates the number of nonzero constraint coefficients for the branch. This specifies the length of the arrays rmatind and rmatval.

- rhs
  An array of length rcnt containing the right-hand side term for each constraint for the branch.

- sense
  An array of length rcnt containing the sense of each constraint to be added for the branch. Values of the sense appear in Table 1.

- rmatbeg
  An array that with rmatind and rmatval defines the constraints for the branch.

- rmatind
  An array that with rmatbeg and rmatval defines the constraints for the branch.

- rmatval
  An array that with rmatbeg and rmatind defines the constraints for the branch. The format is similar to the format used to describe the constraint matrix in the routine CPXaddrows. Every row must be stored in sequential locations in this array from position rmatbeg[i] to rmatbeg[i+1]-1 (or from rmatbeg[i] to nzcnt -1 if i=rcnt-1). Each entry, rmatind[i], indicates the column index of the corresponding coefficient, rmatval[i]. All rows must be contiguous, and rmatbeg[0] must be 0.
userhandle

A pointer to user private data that should be associated with the node created by this branch. May be NULL.

seqnum_p

A pointer to an integer that, on return, will contain the sequence number that CPLEX has assigned to the node created from this branch. The sequence number may be used to select this node in later calls to the node callback.

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXbranchcallbackbranchgeneral

Category                  Global Function
Definition File           cplex.h
Include Files             cplex.h
Synopsis
int CPXPUBLIC CPXbranchcallbackbranchgeneral(CPXENVptr env,
   void * cbdata,
   int wherefrom,
   double nodeest,
   int varcnt,
   const int * varind,
   const char * varlu,
   const int * varbd,
   int rcnt,
   int nzcnt,
   const double * rhs,
   const char * sense,
   const int * rmatbeg,
   const int * rmatind,
   const double * rmatval,
   void * userhandle,
   int * seqnum_p)

Description

Note: This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXbranchcallbackbranchgeneral specifies the branches to be taken from the current node when the branch includes variable bound changes and additional constraints. It may be called only from within a user-written branch callback function.

Branch variables are in terms of the original problem if the parameter CPX_PARAM_MIPCBREDLP is set to CPX_OFF before the call to CPXmipopt that calls the callback. Otherwise, branch variables are in terms of the presolved problem.
Parameters

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbdata
A pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.

wherefrom
An integer value that indicates where the user-written callback was called from. This parameter must be the value of wherefrom passed to the user-written callback.

nodeest
A double that indicates the value of the node estimate for the node to be created with this branch. The node estimate is used to select nodes from the branch & cut tree with certain values of the NodeSel parameter.

varcnt
An integer that indicates the number of bound changes that are specified in the arrays varind, varlu, and varbd.

varind
Together with varlu and varbd, this array defines the bound changes for the branch. The entry varind[i] is the index for the variable.

varlu
Together with varind and varbd, this array defines the bound changes for the branch. The entry varlu[i] is one of three possible values indicating which bound to change. Those values appear in Table 1.

varbd

---

Table 1: Values of varlu[i]

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>change the lower bound</td>
</tr>
<tr>
<td>U</td>
<td>change the upper bound</td>
</tr>
<tr>
<td>B</td>
<td>change both upper and lower bounds</td>
</tr>
</tbody>
</table>

Table 2: Values of sense[i]

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>less than or equal to constraint</td>
</tr>
<tr>
<td>E</td>
<td>equal to constraint</td>
</tr>
<tr>
<td>G</td>
<td>greater than or equal to constraint</td>
</tr>
</tbody>
</table>
Together with \texttt{varind} and \texttt{varlu}, this array defines the bound changes for the branch. The entry \texttt{varbd[i]} indicates the new value of the bound.

\texttt{rcnt}

An integer that indicates the number of constraints for the branch.

\texttt{nzcnt}

An integer that indicates the number of nonzero constraint coefficients for the branch. This specifies the length of the arrays \texttt{rmatind} and \texttt{rmatval}.

\texttt{rhs}

An array of length \texttt{rcnt} containing the right-hand side term for each constraint for the branch.

\texttt{sense}

An array of length \texttt{rcnt} containing the sense of each constraint to be added for the branch. Possible values appear in Table 2.

\texttt{rmatbeg}

An array that with \texttt{rmatbeg} and \texttt{rmatind} defines the constraints for the branch.

\texttt{rmatind}

An array that with \texttt{rmatbeg} and \texttt{rmatind} defines the constraints for the branch.

\texttt{rmatval}

An array that with \texttt{rmatbeg} and \texttt{rmatind} defines the constraints for the branch. The format is similar to the format used to describe the constraint matrix in the routine \texttt{CPXaddrows}. Every row must be stored in sequential locations in this array from position \texttt{rmatbeg[i]} to \texttt{rmatbeg[i+1]-1} (or from \texttt{rmatbeg[i]} to \texttt{nzcnt -1} if \texttt{i=rcnt-1}). Each entry, \texttt{rmatind[i]}, indicates the column index of the corresponding coefficient, \texttt{rmatval[i]}. All rows must be contiguous, and \texttt{rmatbeg[0]} must be 0.

\texttt{userhandle}

A pointer to user private data that should be associated with the node created by this branch. May be NULL.

\texttt{seqnum_p}

A pointer to an integer that, on return, will contain the sequence number that CPLEX has assigned to the node created from this branch. The sequence number may be used to select this node in later calls to the node callback.

\textbf{Returns}

The routine returns zero on success and nonzero if an error occurs.
**CPXgetbranchcallbackfunc**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
void CPXPUBLIC CPXgetbranchcallbackfunc(CPXENVptr env,  
int (CPXPUBLIC **branchcallback_p)(CALLBACK_BRANCH_ARGS),  
void ** cbhandle_p)

**Description**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetbranchcallbackfunc accesses the user-written callback routine to be called during MIP optimization after a branch has been selected but before the branch is carried out. ILOG CPLEX uses the callback routine to change its branch selection.

**Example**

```c
CPXgetbranchcallbackfunc(env, &current_callback, 
                         &current_handle);
```

See also **Advanced MIP Control Interface** in the *ILOG CPLEX User's Manual*.

**Parameters**

- **env**  
  A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

- **branchcallback_p**  
  The address of the pointer to the current user-written branch callback. If no callback has been set, the returned pointer evaluates to NULL.

- **cbhandle_p**  
  The address of a variable to hold the user's private pointer.
Callback description

```c
int callback (CPXCENVptr env,
               void       *cbdata,
               int        wherefrom,
               void       *cbhandle,
               int        type,
               int        sos,
               int        nodecnt,
               int        bdcnt,
               double     *nodeest,
               int        *nodebeg,
               int        *indices,
               char       *lu,
               int        *bd,
               int        *useraction_p);
```

The call to the branch callback occurs after a branch has been selected but before the branch is carried out. This function is written by the user. On entry to the callback, the ILOG CPLEX-selected branch is defined in the arguments. The arguments to the callback specify a list of changes to make to the bounds of variables when child nodes are created. One, two, or zero child nodes can be created, so one, two, or zero lists of changes are specified in the arguments. The first branch specified is considered first. The callback is called with zero lists of bound changes when the solution at the node is integer feasible.

Custom branching strategies can be implemented by calling the CPLEX function CPXbranchcallbackbranchbds and setting the useraction variable to CPX_CALLBACK_SET. Then CPLEX will carry out these branches instead of the CPLEX-selected branches.

Branch variables are in terms of the original problem if the parameter CPX_PARAM_MIPCBREDLP is set to CPX_OFF before the call to CPXmipopt that calls the callback. Otherwise, branch variables are in terms of the presolved problem.

Callback return value

The callback returns zero on success and nonzero if an error occurs.

Callback arguments

env

A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbdata
A pointer passed from the optimization routine to the user-written callback that identifies the problem being optimized. The only purpose of this pointer is to pass it to the callback information routines.

wherefrom

An integer value indicating where in the optimization this function was called. It will have the value CPX_CALLBACK_MIP_BRANCH.

cbhandle

A pointer to user-private data.

int type

An integer that indicates the type of branch. This table summarizes possible values.

<table>
<thead>
<tr>
<th>Symbolic Constant</th>
<th>Value</th>
<th>Branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_TYPE_VAR</td>
<td>0</td>
<td>variable branch</td>
</tr>
<tr>
<td>CPX_TYPE_SOS1</td>
<td>1</td>
<td>SOS1 branch</td>
</tr>
<tr>
<td>CPX_TYPE_SOS2</td>
<td>2</td>
<td>SOS2 branch</td>
</tr>
<tr>
<td>CPX_TYPE_USER</td>
<td>X</td>
<td>user-defined</td>
</tr>
</tbody>
</table>

sos

An integer that indicates the special ordered set (SOS) used for this branch. A value of -1 indicates that this branch is not an SOS-type branch.

nodecnt

An integer that indicates the number of nodes CPLEX will create from this branch. Possible values are:
- 0 (zero), or
- 1, or
- 2.

If the argument is 0, the node will be fathomed unless user-specified branches are made; that is, no child nodes are created and the node itself is discarded.

bdcnt

An integer that indicates the number of bound changes defined in the arrays indices, lu, and bd that define the CPLEX-selected branch.

nodeest
An array with `nodecnt` entries that contains estimates of the integer objective-function value that will be attained from the created node.

`nodebeg`  
An array with `nodecnt` entries. The i-th entry is the index into the arrays `indices`, `lu`, and `bd` of the first bound changed for the i-th node.

`indices`  
Together with `lu` and `bd`, this array defines the bound changes for each of the created nodes. The entry `indices[i]` is the index for the variable.

`lu`  
Together with `indices` and `bd`, this array defines the bound changes for each of the created nodes. The entry `lu[i]` is one of the three possible values indicating which bound to change:
- **L** for lower bound, or
- **U** for upper bound, or
- **B** for both bounds.

`bd`  
Together with `indices` and `lu`, this array defines the bound changes for each of the created nodes. The entry `bd[i]` indicates the new value of the bound.

`useraction_p`  
A pointer to an integer indicating the action for ILOG CPLEX to take at the completion of the user callback. The table summarizes the possible actions.

### Actions to be Taken After a User-Written Branch Callback

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CPX_CALLBACK_DEFAULT</td>
<td>Use CPLEX-selected branch</td>
</tr>
<tr>
<td>1</td>
<td>CPX_CALLBACK_FAIL</td>
<td>Exit optimization</td>
</tr>
<tr>
<td>2</td>
<td>CPX_CALLBACK_SET</td>
<td>Use user-selected branch, as defined by calls to <code>CPXbranchcallbackbranchbds</code></td>
</tr>
<tr>
<td>3</td>
<td>CPX_CALLBACK_NO_SPACE</td>
<td>Allocate more space and call callback again</td>
</tr>
</tbody>
</table>

**See Also**  
[CPXsetbranchcallbackfunc](#)

**Returns**  
This routine does not return a result.
**CPXgetcallbackctype**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXgetcallbackctype(CPXCENVptr env,
    void * cbdata,
    int wherefrom,
    char * xctype,
    int begin,
    int end)
```

**Description**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetcallbackctype is used to get the ctypes for the MIP problem from within a user-written callback during MIP optimization. The values are from the original problem if CPX_PARAM_MIPCBREDLP is set to CPX_OFF. Otherwise, they are from the presolved problem.

This routine may be called only when the value of the wherefrom argument is one of the following:

- CPX_CALLBACK_MIP,
- CPX_CALLBACK_MIP_BRANCH,
- CPX_CALLBACK_MIP_INCUMBENT,
- CPX_CALLBACK_MIP_NODE,
- CPX_CALLBACK_MIP_HEURISTIC,
- CPX_CALLBACK_MIP_SOLVE, or
- CPX_CALLBACK_MIP_CUT.

**Example**
status = CPXgetcallbackctype (env, cbdata, wherefrom, prectype, 0, precols-1);

**Parameters**

- **env**
  A pointer to the CPLEX environment, as returned by CPXopenCPLEX.
- **cbdata**
  The pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.
- **wherefrom**
  An integer value indicating from where the user-written callback was called. The parameter must be the value of wherefrom passed to the user-written callback.
- **xctype**
  An array where the ctype values for the MIP problem will be returned. The array must be of length at least (end - begin + 1). If successful, xctype[0] through xctype[end-begin] contain the variable types.
- **begin**
  An integer indicating the beginning of the range of ctype values to be returned.
- **end**
  An integer indicating the end of the range of ctype values to be returned.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
**CPXgetcallbackgloballb**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXgetcallbackgloballb(CPXENVptr env,  
    void * cbdata,  
    int wherefrom,  
    double * lb,  
    int begin,  
    int end)
```

**Description**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetcallbackgloballb is used to get the best known global lower bound values during MIP optimization from within a user-written callback. The values are from the original problem if CPX_PARAM_MIPCBREDLP is set to CPX_OFF, otherwise they are from the presolved problem.

This routine may be called only when the value of the `wherefrom` argument is one of the following:

- `CPX_CALLBACK_MIP`,
- `CPX_CALLBACK_MIP_BRANCH`,
- `CPX_CALLBACK_MIP_INCUMBENT`,
- `CPX_CALLBACK_MIP_NODE`,
- `CPX_CALLBACK_MIP_HEURISTIC`,
- `CPX_CALLBACK_MIP_SOLVE`, or
- `CPX_CALLBACK_MIP_CUT`.

**Example**
status = CPXgetcallbackglobalb (env, cbdata, wherefrom, 
        glb, 0, cols-1);

Parameters

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbdata
The pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.

wherefrom
An integer value indicating from where the user-written callback was called. The parameter must be the value of wherefrom passed to the user-written callback.

lb
An array to receive the values of the global lower bound values. This array must be of length at least (end - begin + 1). If successful, lb[0] through lb[end-begin] contain the global lower bound values.

begin
An integer indicating the beginning of the range of lower bound values to be returned.

end
An integer indicating the end of the range of lower bound values to be returned.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXgetcallbackglobalub

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXgetcallbackglobalub(CPXENVptr env,
    void * cbdata,
    int wherefrom,
    double * ub,
    int begin,
    int end)
```

Description:

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetcallbackglobalub is used to get the best known global upper bound values during MIP optimization from within a user-written callback. The values are from the original problem if CPX_PARAM_MIPCBREDLP is set to CPX_OFF, otherwise they are from the presolved problem.

This routine may be called only when the value of the wherefrom argument is one of the following:

- CPX CALLBACK_MIP,
- CPX CALLBACK_MIP_BRANCH,
- CPX CALLBACK_MIP_INCUMBENT,
- CPX CALLBACK_MIP_NODE,
- CPX CALLBACK_MIP_HEURISTIC,
- CPX CALLBACK_MIP_SOLVE, or
- CPX CALLBACK_MIP_CUT.

Example
status = CPXgetcallbackglobalub (env, cbdata, wherefrom, 
    gub, 0, cols-1);

Parameters

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbdata
The pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.

wherefrom
An integer value indicating from where the user-written callback was called. The parameter must be the value of wherefrom passed to the user-written callback.

ub
An array to receive the values of the global upper bound values. This array must be of length at least (end - begin + 1). If successful, ub[0] through ub[end-begin] contain the global upper bound values.

begin
An integer indicating the beginning of the range of upper bound values to be returned.

end
An integer indicating the end of the range of upper bound values to be returned.

Returns
The routine returns zero on success and nonzero if an error occurs.
**CPXgetcallbackincumbent**

**Category**     Global Function

**Definition File**     cplex.h

**Include Files**     cplex.h

**Synopsis**
```c
int CPXPUBLIC CPXgetcallbackincumbent(CPXENVptr env,
                                      void * cbdata,
                                      int wherefrom,
                                      double * x,
                                      int begin,
                                      int end)
```

**Description**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetcallbackincumbent is used to get the incumbent values during MIP optimization from within a user-written callback. The values are from the original problem if CPX_PARAM_MIPCBREDLP is set to CPX_OFF. Otherwise, they are from the presolved problem.

This routine may be called only when the value of the wherefrom argument is one of the following:

- CPX_CALLBACK_MIP
- CPX_CALLBACK_MIP_BRANCH
- CPX_CALLBACK_MIP_INCUMBENT
- CPX_CALLBACK_MIP_NODE
- CPX_CALLBACK_MIP_HEURISTIC
- CPX_CALLBACK_MIP_SOLVE, or
- CPX_CALLBACK_MIP_CUT

**Example**
status = CPXgetcallbackincumbent (env, cbdata, wherefrom,
    bestx, 0, cols-1);

Parameters

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbdata
The pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.

wherefrom
An integer value indicating from where the user-written callback was called. The parameter must be the value of wherefrom passed to the user-written callback.

x
An array to receive the values of the incumbent (best available) integer solution. This array must be of length at least (end - begin + 1). If successful, x[0] through x[end-begin] contain the incumbent values.

begin
An integer indicating the beginning of the range of incumbent values to be returned.

end
An integer indicating the end of the range of incumbent values to be returned.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXgetcallbackinfo

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXgetcallbackinfo(CPXENVptr env,
    void * cbdata,
    int wherefrom,
    int whichinfo,
    void * result_p)
```

Description:

The routine CPXgetcallbackinfo is used to access information about the current optimization process from within a user-written callback function.

**Note:** This routine is the only one that can access optimization status information from within a non-advanced user-written callback function. It is also the only Callable Library routine that may be called from within a non-advanced user-written callback function, and in fact, may only be called from the callback function.

Parameters:

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **cbdata**
  The cbdata pointer passed to the user-written callback function. The parameter cbdata MUST be the value of cbdata passed to the user-written callback function.

- **wherefrom**
  An integer value indicating the optimization algorithm from which the user-written callback function was called. The parameter wherefrom MUST be the value of wherefrom passed to the user-written callback function. See CPXgetlpcallbackfunc, CPXgetmipcallbackfunc, and CPXgetnetcallbackfunc for possible values of wherefrom and their meaning.

- **whichinfo**
  A code indicating the type of information that is to be accessed. See CPXgetcallbackinfo for possible values of whichinfo and their meaning.
An integer value indicating the specific information that should be returned by `CPXgetcallbackinfo` to the result argument. Values for `whichinfo`, the type of the information returned into `*result_p`, plus a description appear in the table.

**result_p**

A generic pointer to a variable of type `double` or `int`, dependent on the value of `whichinfo`, as documented in the following tables.

### For LP algorithms:

<table>
<thead>
<tr>
<th><code>whichinfo</code></th>
<th><code>type of *result_p</code></th>
<th><strong>description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CPX_CALLBACK_INFO_PRIMAL_OBJ</code></td>
<td>double</td>
<td>primal objective value</td>
</tr>
<tr>
<td><code>CPX_CALLBACK_INFO_DUAL_OBJ</code></td>
<td>double</td>
<td>dual objective value</td>
</tr>
<tr>
<td><code>CPX_CALLBACK_INFO_PRIMAL_INFMEAS</code></td>
<td>double</td>
<td>measure of primal infeasibility</td>
</tr>
<tr>
<td><code>CPX_CALLBACK_INFO_DUAL_INFMEAS</code></td>
<td>double</td>
<td>measure of dual infeasibility</td>
</tr>
<tr>
<td><code>CPX_CALLBACK_INFO_PRIMAL_FEAS</code></td>
<td>int</td>
<td>1 if primal feasible, 0 if not</td>
</tr>
<tr>
<td><code>CPX_CALLBACK_INFO_DUAL_FEAS</code></td>
<td>int</td>
<td>1 if dual feasible, 0 if not</td>
</tr>
<tr>
<td><code>CPX_CALLBACK_INFO_ITCOUNT</code></td>
<td>int</td>
<td>iteration count</td>
</tr>
<tr>
<td><code>CPX_CALLBACK_INFO_CROSSOVER_PPUSH</code></td>
<td>int</td>
<td>primal push crossover itn. count</td>
</tr>
<tr>
<td><code>CPX_CALLBACK_INFO_CROSSOVER_PEXCH</code></td>
<td>int</td>
<td>primal exchange crossover itn. count</td>
</tr>
<tr>
<td><code>CPX_CALLBACK_INFO_CROSSOVER_DPPUSH</code></td>
<td>int</td>
<td>dual push crossover itn. count</td>
</tr>
<tr>
<td><code>CPX_CALLBACK_INFO_CROSSOVER_DEXCH</code></td>
<td>int</td>
<td>dual exchange crossover itn. count</td>
</tr>
<tr>
<td><code>CPX_CALLBACK_INFO_USERPROBLEM</code></td>
<td><code>CPXCLPptr</code></td>
<td>returns pointer to original user problem; available for primal, dual, barrier, mip</td>
</tr>
</tbody>
</table>

### For Network algorithms:

<table>
<thead>
<tr>
<th><code>whichinfo</code></th>
<th><code>type of *result_p</code></th>
<th><strong>description</strong></th>
</tr>
</thead>
</table>
### For Presolve algorithms:

<table>
<thead>
<tr>
<th>whichinfo</th>
<th>type of *result_p</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_CALLBACK_INFO_PRESOLVE_ROWSGONE</td>
<td>int</td>
<td>number of rows eliminated</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_PRESOLVE_COLSGONE</td>
<td>int</td>
<td>number of columns eliminated</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_PRESOLVE_AGGSUBST</td>
<td>int</td>
<td>number of aggregator substitutions</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_PRESOLVE_COEFFS</td>
<td>int</td>
<td>number of modified coefficients</td>
</tr>
</tbody>
</table>

### For MIP algorithms:

<table>
<thead>
<tr>
<th>whichinfo</th>
<th>type of *result_p</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_CALLBACK_INFO_BEST_INTEGER</td>
<td>double</td>
<td>obj. value of best integer solution</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_BEST_REMAINING</td>
<td>double</td>
<td>obj. value of best remaining node</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_NODE_COUNT</td>
<td>int</td>
<td>total number of nodes solved</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_NODES_LEFT</td>
<td>int</td>
<td>number of remaining nodes</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_MIP_ITERATIONS</td>
<td>int</td>
<td>total number of MIP iterations</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_MIP_FEAS</td>
<td>int</td>
<td>returns 1 if feasible solution exists; otherwise, 0</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_CUTOFF</td>
<td>double</td>
<td>updated cutoff value</td>
</tr>
</tbody>
</table>
### Example

See `lpex4.c` in the *CPLEX User's Manual*.

<table>
<thead>
<tr>
<th>CPX_CALLBACK_INFO_CLIQUE_COUNT</th>
<th>int</th>
<th>number of clique cuts added</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_CALLBACK_INFO_COVER_COUNT</td>
<td>int</td>
<td>number of cover cuts added</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_DISJCOUNT</td>
<td>int</td>
<td>number of disjunctive cuts added</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_FLOWCOVER_COUNT</td>
<td>int</td>
<td>number of flow cover cuts added</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_FLOWPATH_COUNT</td>
<td>int</td>
<td>number of flow path cuts added</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_FRACUT_COUNT</td>
<td>int</td>
<td>number of Gomory fractional cuts added</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_GUBCOVER_COUNT</td>
<td>int</td>
<td>number of GUB cover cuts added</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_IMPLEMENTED_COUNT</td>
<td>int</td>
<td>number of implied bound cuts added</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_MIRCOUNT</td>
<td>int</td>
<td>number of mixed integer rounding cuts added</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_USERPROBLEM</td>
<td>CPXCLPtr</td>
<td>returns pointer to original user problem; available for primal, dual, barrier, MIP</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_PROBE_PHASE</td>
<td>int</td>
<td>current phase of probing (0-3)</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_PROBE_PROGRESS</td>
<td>double</td>
<td>fraction of probing phase completed (0.0-1.0)</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_FRACUT_PROGRESS</td>
<td>double</td>
<td>fraction of Gomory cut generation for the pass completed (0.0 - 1.0)</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_DISJCOUNT_PROGRESS</td>
<td>double</td>
<td>fraction of disjunctive cut generation for the pass completed (0.0 - 1.0)</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_FLOWMIR_PROGRESS</td>
<td>double</td>
<td>fraction of flow cover and MIR cut generation for the pass completed (0.0 - 1.0)</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_MYTHREAD</td>
<td>int</td>
<td>identifier of the parallel thread making this call</td>
</tr>
<tr>
<td>CPX_CALLBACK_INFO_USER_THREADS</td>
<td>int</td>
<td>total number of parallel threads currently running</td>
</tr>
</tbody>
</table>
Suppose you want to know the objective value on each iteration for a graphical user display. In addition, if primal simplex is not feasible after 1000 iterations, you want to stop the optimization. The function `mycallback` is a callback function to do this.

```c
int mycallback (CPXCENVptr env, void *cbdata, int wherefrom,
                void *cbhandle)
{
    int itcount;
    double objval;
    int ispfeas;
    int status = 0;
    if ( wherefrom == CPX_CALLBACK_PRIMAL ) {
        status = CPXgetcallbackinfo (env, cbdata, wherefrom,
                                      CPX_CALLBACK_INFO_PRIMAL_FEAS,
                                      &ispfeas);
        if ( status ) {
            fprintf (stderr,"error %d in CPXgetcallbackinfo
                       status = 1;
            goto TERMINATE;
        }
        if ( ispfeas ) {
            status = CPXgetcallbackinfo (env, cbdata, wherefrom,
                                          CPX_CALLBACK_INFO_PRIMAL_OBJ,
                                          &objval) )
            if ( status ) {
                fprintf (stderr,"error %d in CPXgetcallbackinfo
                        status);)
                status = 1;
                goto TERMINATE;
            }
            /* Do some graphics with the value of objval */
        }
        else {
            status = CPXgetcallbackinfo (env, cbdata, wherefrom,
                                          CPX_CALLBACK_INFO_ITCOUNT, &itcount);
            if ( status ) {
                fprintf (stderr,"error %d in CPXgetcallbackinfo
                        status = 1;
                goto TERMINATE;
            }
            if ( itcount > 1000 )  status = 1;
        }
    }
    TERMINATE:
    return (status);
}
```
Returns

The routine returns zero on success and nonzero if an error occurs. If nonzero, the requested value may not be available for the specific optimization algorithm. For example, the dual objective is not available from primal simplex.
CPXgetcallbacklp

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXgetcallbacklp(CPXCENVptr env, 
                               void * cbdata, 
                               int wherefrom, 
                               CPXCLPptr * lp_p)
```

**Description**

- **Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetcallbacklp is used to get the pointer to the MIP problem that is in use when the user-written callback function is called. It is the original MIP if CPX_PARAM_MIPCBREDLP is set to CPX_OFF; otherwise, it is the presolved MIP. In contrast, the function CPXgetcallbacknodelp returns a pointer to the node subproblem, which is an LP. Generally, this pointer may be used only in CPLEX Callable Library query routines, such as CPXsolution or CPXgetrows.

This routine may be called only when the value of the wherefrom argument is one of the following:

- CPX_CALLBACK_MIP,
- CPX_CALLBACK_MIP_BRANCH,
- CPX_CALLBACK_MIP_INCUMBENT,
- CPX_CALLBACK_MIP_NODE,
- CPX_CALLBACK_MIP_HEURISTIC,
- CPX_CALLBACK_MIP_SOLVE, or
- CPX_CALLBACK_MIP_CUT.

**Example**
status = CPXgetcallbacklp (env, cbdata, wherefrom, &origlp);

See also admipex1.c, admipex2.c, and admipex3.c in the standard distribution.

**Parameters**

- **env**
  A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

- **cbdata**
  The pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.

- **wherefrom**
  An integer value indicating from where the user-written callback was called. The parameter must be the value of wherefrom passed to the user-written callback.

- **lp_p**
  A pointer to a variable of type CPXLPPtr to receive the pointer to the LP problem object, which is a MIP.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXgetcallbacknodeintfeas

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:
```c
int CPXPUBLIC CPXgetcallbacknodeintfeas(CPXCENVptr env,
void * cbdata,
int wherefrom,
int * feas,
int begin,
int end)
```

Description:

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetcallbacknodeintfeas is used to get indicators for each variable of whether or not the variable is integer feasible in the node subproblem. It can be used in a user-written callback during MIP optimization. The indicators are from the original problem if CPX_PARAM_MIPCBREDLP is set to CPX_OFF. Otherwise, they are from the presolved problem.

**Example**
```c
status = CPXgetcallbacknodeintfeas(env, cbdata, wherefrom,
feas, 0, cols-1);
```

See admipex1.c and admipex2.c in the standard distribution.

This routine may be called only when the value of the wherefrom argument is one of the following:
- CPX_CALLBACK_MIP,
- CPX_CALLBACK_MIP_BRANCH,
- CPX_CALLBACK_MIP_INCUMBENT,
- CPX_CALLBACK_MIP_NODE.
Parameters

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbdata
The pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.

wherefrom
An integer value indicating from where the user-written callback was called. The parameter must be the value of wherefrom passed to the user-written callback.

feas
An array to receive an indicator of feasibility for the node subproblem. This array must be of length at least (end - begin + 1). If successful, feas[0] through feas[end-begin] will contain the indicators. The indicators of feasibility for a node of the subproblem appear in the table.

begin
An integer indicating the beginning of the range of feasibility indicators to be returned.

end
An integer indicating the end of the range of feasibility indicators to be returned.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXgetcallbacknodelb

Category  Global Function
Definition File  cplex.h
Include Files  cplex.h
Synopsis
```
int CPXPUBLIC CPXgetcallbacknodelb(CPXCENVptr env,
     void * cbdata,
     int wherefrom,
     double * lb,
     int begin,
     int end)
```

Description

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetcallbacknodelb is used to get the lower bound values for the subproblem at the current node during MIP optimization from within a user-written callback. The values are from the original problem if CPX_PARAM_MIPCBREDLP is set to CPX_OFF; otherwise, they are from the presolved problem.

This routine may be called only when the value of the wherefrom argument is one of the following:
- CPX_CALLBACK_MIP,
- CPX_CALLBACK_MIP_BRANCH,
- CPX_CALLBACK_MIP_INCUMBENT,
- CPX_CALLBACK_MIP_NODE,
- CPX_CALLBACK_MIP_HEURISTIC,
- CPX_CALLBACK_MIP_SOLVE, or
- CPX_CALLBACK_MIP_CUT.

Example
status = CPXgetcallbacknodelb (env, cbdata, wherefrom,
        lb, 0, cols-1);

Parameters

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbdata
The pointer passed to the user-written callback. This parameter must be the value of
cbdata passed to the user-written callback.

wherefrom
An integer value indicating from where the user-written callback was called. The parameter must be the value of
wherefrom passed to the user-written callback.

lb
An array to receive the values of the lower bound values. This array must be of length at
least (end - begin + 1). If successful, lb[0] through lb[end-begin] contain
the lower bound values for the current subproblem.

begin
An integer indicating the beginning of the range of lower bounds to be returned.

end
An integer indicating the end of the range of lower bounds to be returned.

Returns
The routine returns zero on success and nonzero if an error occurs.
**CPXgetcallbacknodep**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**
```c
int CPXPUBLIC CPXgetcallbacknodep(CPXCENVptr env,
    void * cbdata,
    int wherefrom,
    CPXLPptr * nodelp_p)
```

**Description**

The routine **CPXgetcallbacknodep** accesses the **lp** pointer indicating the currently defined linear programming subproblem (LP) from within user-written callbacks. Generally, this pointer may be used only in ILOG CPLEX Callable Library query routines, such as **CPXsolution** or **CPXgetrows**.

**Example**
```
status = CPXgetcallbacknodep (env, cbdata, &nodelp);
```

See also the example **admipex1.c** and **admipex6.c** in the standard distribution.

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.
When the `wherefrom` argument has the value `CPX_CALLBACK_MIP_SOLVE`, the subproblem pointer may also be used in ILOG CPLEX optimization routines.

**Note:** *Any modification to the subproblem may result in corruption of the problem and of the ILOG CPLEX environment.*

**Parameters**

`env`  
A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.

`cbdata`  
The `cbdata` pointer passed to the user-written callback. This parameter must be the value of `cbdata` passed to the user-written callback.

`wherefrom`  
An integer value indicating where the user-written callback was called from. This parameter must be the value of the `wherefrom` passed to the user-written callback.

`nodelp_p`  
The `lp` pointer indicating the current subproblem. If no subproblem is defined, the pointer is set to NULL.

**Returns**

The routine returns zero on success and nonzero if an error occurs. A nonzero return value may mean that the requested value is not available.
CPXgetcallbacknodeobjval

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXgetcallbacknodeobjval (CPXCENVptr env,
    void * cbdata,
    int wherefrom,
    double * objval_p)
```

Description:

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetcallbacknodeobjval is used to get the objective value for the subproblem at the current node during MIP optimization from within a user-written callback.

This routine may be called only when the value of the `wherefrom` argument is one of the following:

- `CPX_CALLBACK_MIP`
- `CPX_CALLBACK_MIP_BRANCH`
- `CPX_CALLBACK_MIP_INCUMBENT`
- `CPX_CALLBACK_MIP_NODE`
- `CPX_CALLBACK_MIP_HEURISTIC`, or
- `CPX_CALLBACK_MIP_CUT`.

Example:

```c
status = CPXgetcallbacknodeobjval (env, cbdata, wherefrom, 
    &objval);
```

See also `admipex1.c` and `admipex3.c` in the standard distribution.
Parameters

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbdta
The pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.

wherefrom
An integer value indicating from where the user-written callback was called. The parameter must be the value of wherefrom passed to the user-written callback.

objval_p
A pointer to a variable of type double where the objective value of the node subproblem is to be stored.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXgetcallbacknodestat

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```
int CPXPUBLIC CPXgetcallbacknodestat (CPXENVptr env,
                                 void * cbdata,
                                 int wherefrom,
                                 int * nodestat_p);
```

Description:

The routine **CPXgetcallbacknodestat** is used to get the optimization status of the
subproblem at the current node from within a user-written callback during MIP optimization.

The optimization status will be either optimal or unbounded. An unbounded status can occur when some of the constraints are being treated as lazy constraints. When the node status is unbounded, then the function **CPXgetcallbacknodex** returns a ray that can be used to decide which lazy constraints need to be added to the subproblem.

This routine may be called only when the value of the **wherefrom** argument is **CPX CALLBACK MIP CUT**.

Example:

```
status = CPXgetcallbacknodestat (env, cbdata, wherefrom,
                                 &nodestatus);
```

Parameters:

- **env**
  
  A pointer to the CPLEX environment, as returned by **CPXopenCPLEX**.

- **cbdata**
  
  The pointer passed to the user-written callback. This parameter must be the value of **cbdata** passed to the user-written callback.

Note: This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.
wherefrom

An integer value indicating from where the user-written callback was called. The parameter must be the value of wherefrom passed to the user-written callback.

nodestat_p

A pointer to an integer where the node subproblem optimization status is to be returned. The values of *nodestat_p may be CPX_STAT_OPTIMAL or CPX_STAT_UNBOUNDED.

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXgetcallbacknodeub

Category: Global Function
Definition File: cplex.h
Include Files: cplex.h
Synopsis:
```c
int CPXPUBLIC CPXgetcallbacknodeub(CPXCENVptr env,
                    void * cbdata,
                    int wherefrom,
                    double * ub,
                    int begin,
                    int end)
```

Description:

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetcallbacknodeub is used to get the upper bound values for the subproblem at the current node during MIP optimization from within a user-written callback. The values are from the original problem if CPX_PARAM_MIPCBREDLP is set to CPX_OFF; otherwise, they are from the presolved problem.

This routine may be called only when the value of the wherefrom argument is one of the following:

- CPX_CALLBACK_MIP,
- CPX_CALLBACK_MIP_BRANCH,
- CPX_CALLBACK_MIP_INCUMBENT,
- CPX_CALLBACK_MIP_NODE,
- CPX_CALLBACK_MIP_HEURISTIC,
- CPX_CALLBACK_MIP_SOLVE, or
- CPX_CALLBACK_MIP_CUT.

Example
status = CPXgetcallbacknodeub (env, cbdata, wherefrom, 
                      ub, 0, cols-1);

Parameters

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbdata
The pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.

wherefrom
An integer value indicating from where the user-written callback was called. The parameter must be the value of wherefrom passed to the user-written callback.

ub
An array to receive the values of the upper bound values. This array must be of length at least (end - begin + 1). If successful, ub[0] through ub[end-begin] contain the upper bound values for the current subproblem.

begin
An integer indicating the beginning of the range of upper bound values to be returned.

end
An integer indicating the end of the range of upper bound values to be returned.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXgetcallbacknodex

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXgetcallbacknodex(CPXENVptr env,
void * cbdata,
int wherefrom,
double * x,
int begin,
int end)
```

Description:

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetcallbacknodex is used to get the primal variable (x) values for the subproblem at the current node during MIP optimization from within a user-written callback. The values are from the original problem if CPX_PARAM_MIPCBREDLP is set to CPX_OFF; otherwise, they are from the presolved problem.

This routine may be called only when the value of the wherefrom argument is one of the following:

- CPX_CALLBACK_MIP
- CPX_CALLBACK_MIP_BRANCH
- CPX_CALLBACK_MIP_INCUMBENT
- CPX_CALLBACK_MIP_NODE
- CPX_CALLBACK_MIP_HEURISTIC, or
- CPX_CALLBACK_MIP_CUT.

Example:

```c
status = CPXgetcallbacknodex (env, cbdata, wherefrom,
                           nodex, 0, cols-1);
```
See also admipex1.c, admipex3.c, and admipex5.c in the standard distribution.

**Parameters**

- **env**
  A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

- **cbdata**
  The pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.

- **wherefrom**
  An integer value indicating from where the user-written callback was called. The parameter must be the value of wherefrom passed to the user-written callback.

- **x**
  An array to receive the values of the primal variables for the node subproblem. This array must be of length at least \((end - begin + 1)\). If successful, \(x[0]\) through \(x[end-begin]\) contain the primal values.

- **begin**
  An integer indicating the beginning of the range of primal variable values for the node subproblem to be returned.

- **end**
  An integer indicating the end of the range of primal variable values for the node subproblem to be returned.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
**CPXgetcallbackorder**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXgetcallbackorder(CPXENVptr env,
   void * cbdata,
   int wherefrom,
   int * priority,
   int * direction,
   int begin,
   int end)
```

**Description**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetcallbackorder is used to get MIP priority order information during MIP optimization from within a user-written callback. The values are from the original problem if CPX_PARAM_MIPCBREDLP is set to CPX_OFF. Otherwise, they are from the presolved problem.

This routine may be called only when the value of the wherefrom argument is one of the following values:

- CPX_CALLBACK_MIP,
- CPX_CALLBACK_MIP_BRANCH,
- CPX_CALLBACK_MIP_INCUMBENT,
- CPX_CALLBACK_MIP_NODE,
- CPX_CALLBACK_MIP_HEURISTIC,
- CPX_CALLBACK_MIP_SOLVE, or
- CPX_CALLBACK_MIP_CUT.

**Example**
status = CPXgetcallbackorder (env, cbdata, wherefrom,
priority, NULL, 0, cols-1);

Branching direction

<table>
<thead>
<tr>
<th>Branching direction</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_BRANCH_GLOBAL</td>
<td>0</td>
<td>use global branching direction setting</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CPX_PARAM_BRDIR</td>
</tr>
<tr>
<td>CPX_BRANCH_DOWN</td>
<td>-1</td>
<td>branch down first on variable j+begin</td>
</tr>
<tr>
<td>CPX_BRANCH_UP</td>
<td>1</td>
<td>branch up first on variable j+begin</td>
</tr>
</tbody>
</table>

Parameters

`env`
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

`cbdata`
The pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.

`wherefrom`
An integer value indicating from where the user-written callback was called. The parameter must be the value of wherefrom passed to the user-written callback.

`priority`
An array where the priority values are to be returned. This array must be of length at least (end - begin + 1). If successful, priority[0] through priority[end-begin] contain the priority order values. May be NULL. The value of direction[j] will be a value from the table of branching directions.

`begin`
An integer indicating the beginning of the range of priority order information to be returned.

`end`
An integer indicating the end of the range of priority order information to be returned.

Returns
The routine returns zero on success and nonzero if an error occurs.
**Synopsis**

```c
int CPXPUBLIC CPXgetcallbackpseudocosts(CPXCENVptr env,
    void * cbdata,
    int wherefrom,
    double * uppc,
    double * downpc,
    int begin,
    int end)
```

**Description**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetcallbackpseudocosts is used to get the pseudo-cost values during MIP optimization from within a user-written callback. The values are from the original problem if CPX_PARAM_MIPCBREDLP is set to CPX_OFF. Otherwise, they are from the presolved problem.

**Note:** When pseudo-costs are retrieved for the original problem variables, pseudo-costs are zero for variables that have been removed from the problem, since they are never used for branching.

This routine may be called only when the value of the wherefrom argument is one of the following:

- CPX_CALLBACK_MIP,
- CPX_CALLBACK_MIP_BRANCH,
- CPX_CALLBACK_MIP_INCUMBENT,
- CPX_CALLBACK_MIP_NODE,
◆ CPX_CALLBACK_MIP_HEURISTIC,
◆ CPX_CALLBACK_MIP_SOLVE, or
◆ CPX_CALLBACK_MIP_CUT.

Example

```c
status = CPXgetcallbackpseudocosts (env, cbdata, wherefrom,
                                   upcost, downcost,
                                   j, k);
```

Parameters

- **env**
  A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

- **cbdata**
  The pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.

- **wherefrom**
  An integer value indicating from where the user-written callback was called. The parameter must be the value of wherefrom passed to the user-written callback.

- **uppc**
  An array to receive the values of up pseudo-costs. This array must be of length at least (end - begin + 1). If successful, uppc[0] through uppc[end-begin] will contain the up pseudo-costs. May be NULL.

- **downpc**
  An array to receive the values of the down pseudo-costs. This array must be of length at least (end - begin + 1). If successful, downpc[0] through downpc[end-begin] will contain the down pseudo-costs. May be NULL.

- **begin**
  An integer indicating the beginning of the range of pseudo-costs to be returned.

- **end**
  An integer indicating the end of the range of pseudo-costs to be returned.

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXgetcutcallbackfunc

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
void CPXPUBLIC CPXgetcutcallbackfunc(CPXCENVptr env,
    int(CPXPUBLIC **cutcallback_p)(CALLBACK_CUT_ARGS),
    void ** cbhandle_p)
```

**Description**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetcutcallbackfunc` accesses the user-written callback for adding cuts. The user-written callback is called by ILOG CPLEX during MIP branch & cut for every node that has an LP optimal solution with objective value below the cutoff and that is integer infeasible. The callback routine adds globally valid cuts to the LP subproblem.

**Example**

```c
CPXgetcutcallbackfunc(env, &current_cutfunc, &current_data);
```

See also *Advanced MIP Control Interface* in the *ILOG CPLEX User's Manual*.

For documentation of callback arguments, see the routine `CPXsetcutcallbackfunc`.

**Parameters**

- `env`  
  A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.

- `cutcallback_p`  
  The address of the pointer to the current user-written cut callback. If no callback has been set, the pointer evaluates to NULL.
cbhandle_p
The address of a variable to hold the user's private pointer.

See Also  CPXcutcallbackadd, CPXsetcutcallbackfunc

Returns  This routine does not return a result.
CPXgetdeletenodecallbackfunc

Category: Global Function
Definition File: cplex.h
Include Files: cplex.h
Synopsis:
void CPXPUBLIC CPXgetdeletenodecallbackfunc(CPXCENVptr env,
void(CPXPUBLIC **deletecallback_p)(CALLBACK_DELETENODE_ARGS) ,
void ** cbhandle_p)

Description:

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetdeletenodecallbackfunc accesses the user-written callback to be called during MIP optimization when a node is to be deleted. Nodes are deleted when a branch is carried out from that node, when the node relaxation is infeasible, or when the node relaxation objective value is worse than the cutoff. This callback can be used to delete user data associated with a node.

Example:

```c
CPXgetdeletenodecallbackfunc(env,
    &current_callback,
    &current_cbdata);
```

See also Advanced MIP Control Interface in the ILOG CPLEX User's Manual.

For documentation of callback arguments, see the routine CPXsetdeletenodecallbackfunc.

Parameters:

- **env**
  - A pointer to the CPLEX environment, as returned by CPXopenCPLEX.
- **deletenodecallback_p**
The address of the pointer to the current user-written delete-node callback. If no callback has been set, the pointer evaluates to NULL.

`cbhandle_p`

The address of a variable to hold the user's private pointer.

See Also

`CPXsetdeletenodecallbackfunc`, `CPXbranchcallbackbranchbds`, `CPXbranchcallbackbranchconstraints`, `CPXbranchcallbackbranchgeneral`

Returns

This routine does not return a result.
CPXgetheuristiccallbackfunc

Category          Global Function
Definition File   cplex.h
Include Files     cplex.h
Synopsis

void CPXPUBLIC CPXgetheuristiccallbackfunc(CPXENVptr env,
                                          int(CPXPUBLIC **heuristiccallback_p)(CALLBACK_HEURISTIC_ARGS),
                                          void ** cbhandle_p)

Description

Note: This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetheuristiccallbackfunc accesses the user-written callback to be called by ILOG CPLEX during MIP optimization after the subproblem has been solved to optimality. That callback is not called when the subproblem is infeasible or cut off. The callback supplies ILOG CPLEX with heuristically-derived integer solutions.

Example

CPXgetheuristiccallbackfunc(env, &current_callback,
                           &current_handle);

See also Advanced MIP Control Interface in the ILOG CPLEX User's Manual.
For documentation of callback arguments, see the routine CPXsetheuristiccallbackfunc.

Parameters

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

heuristiccallback_p
The address of the pointer to the current user-written heuristic callback. If no callback has been set, the pointer evaluates to NULL.

cbhandle_p
The address of a variable to hold the user's private pointer.

**See Also**

CPXsetheuristiccallbackfunc

**Returns**

This routine does not return a result.
CPXgetincumbentcallbackfunc

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis: void CPXPUBLIC CPXgetincumbentcallbackfunc(CPXCENVptr env, int(CPXPUBLIC **incumbentcallback_p)(CALLBACK_INCUMBENT_ARGS) , void ** cbhandle_p)

Description:

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetincumbentcallbackfunc accesses the user-written callback to be called by CPLEX during MIP optimization after an integer solution has been found but before this solution replaces the incumbent. This callback can be used to discard solutions that do not meet criteria beyond that of the mixed integer programming formulation.

Example:

CPXgetincumbentcallbackfunc(env, &current_incumbentcallback, &current_handle);

See also Advanced MIP Control Interface in the ILOG CPLEX User's Manual.

For documentation of callback arguments, see the routine CPXsetincumbentcallbackfunc.

Parameters:

- **env**
  A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

- **incumbentcallback_p**
  The address of the pointer to the current user-written incumbent callback. If no callback has been set, the pointer evaluates to NULL.
<table>
<thead>
<tr>
<th><strong>cbhandle_p</strong></th>
<th>The address of a variable to hold the user's private pointer.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>See Also</strong></td>
<td>CPXsetincumbentcallbackfunc</td>
</tr>
<tr>
<td><strong>Returns</strong></td>
<td>This routine does not return a result.</td>
</tr>
</tbody>
</table>
CPXgetlpcallbackfunc

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**
```c
int CPXPUBLIC CPXgetlpcallbackfunc(CPXENVptr env,
    int(CPXPUBLIC **callback_p)(CPXENVptr, void *, int, void *) ,
    void ** cbhandle_p)
```

**Description**
The routine CPXgetlpcallbackfunc is used to access the user-written callback routine to be called after each iteration during the optimization of a linear or quadratic program, and also periodically during the CPLEX presolve algorithm.

**Callback description**
```c
int callback (CPXENVptr env,
    void   *cbdata,
    int    wherefrom,
    void   *cbhandle);
```

This is the user-written callback routine.

**Callback return value**
A nonzero terminates the optimization.

**Callback arguments**

*env*
A pointer to the CPLEX environment that was passed into the associated optimization routine.

*cbdata*
A pointer passed from the optimization routine to the user-written callback function that identifies the LP problem being optimized. The only purpose for the cbdata pointer is to pass it to the routine CPXgetcallbackinfo.

*wherefrom*
An integer value indicating which optimization algorithm the user-written callback function was called from. Possible values and their meaning appear in the table.
cbhandle
Pointer to user private data, as passed to CPXsetlpcallbackfunc.

Parameters
env
A pointer to the CPLEX environment as returned by CPXopenCPLEX.
callback_p
The address of the pointer to the current user-written callback function. If no callback function has been set, the pointer evaluates to NULL.

cbhandle_p
The address of a variable to hold the user’s private pointer.

Example

status = CPXgetlpcallbackfunc (env, mycallback, NULL);

See Also  CPXgetcallbackinfo

Returns
The routine returns zero on success and nonzero if an error occurs.
**CPXgetmipcallbackfunc**

**Category**  
Global Function

**Definition File**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXgetmipcallbackfunc(CPXENVptr env,
      int(CPXPUBLIC **callback_p)(CPXENVptr, void *, int, void *) ,
      void ** cbhandle_p)
```

**Description**  
The routine `CPXgetmipcallbackfunc` is used to access the user-written callback routine to be called prior to solving each subproblem in the branch & cut tree during the optimization of a mixed integer program. This routine works in the same way as the routine `CPXgetlpcallbackfunc`. It enables the user to create a separate callback function to be called during the solution of mixed integer programming problems. The prototype for the callback function is identical to that of `CPXgetlpcallbackfunc`.

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
- **callback_p**
  The address of the pointer to the current user-written callback function. If no callback function has been set, the pointer evaluates to NULL.
- **cbhandle_p**
  The address of a variable to hold the user's private pointer.

**Example**

```c
status = CPXgetmipcallbackfunc (env, mycallback, NULL);
```

**Callback description**

```c
int callback (CPXENVptr env,  
    void *cbdata,  
    int wherefrom,  
    void *cbhandle);
```

This is the user-written callback routine.
Callback return value
A nonzero terminates the optimization.

Callback arguments
env
A pointer to the CPLEX environment that was passed into the associated optimization routine.
cbdata
A pointer passed from the optimization routine to the user-written callback function that identifies the LP problem being optimized. The only purpose for the cbdata pointer is to pass it to the routine CPXgetcallbackinfo.
wherefrom
An integer value indicating from which optimization algorithm the user-written callback function was called. Possible values and their meaning appear in this table.

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>CPX_CALLBACK_MIP</td>
<td>From mipopt</td>
</tr>
<tr>
<td>107</td>
<td>CPX_CALLBACK_MIP_PROBE</td>
<td>From probing or clique merging</td>
</tr>
<tr>
<td>108</td>
<td>CPX_CALLBACK_MIP_FRACCU T</td>
<td>From Gomory fractional cuts</td>
</tr>
<tr>
<td>109</td>
<td>CPX_CALLBACK_MIP_DISJCU T</td>
<td>From disjunctive cuts</td>
</tr>
<tr>
<td>110</td>
<td>CPX_CALLBACK_MIP_FLOWMI R</td>
<td>From Mixed Integer Rounding cuts</td>
</tr>
</tbody>
</table>

cbhandle
Pointer to user private data, as passed to CPXsetmipcallbackfunc.

See Also
CPXgetcallbackinfo

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXgetnetcallbackfunc

Category: Global Function
Definition File: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXgetnetcallbackfunc(CPXENVptr env,
    int(CPXPUBLIC **callback_p)(CPXENVptr, void *, int, void *) ,
    void ** cbhandle_p)
```

Description:

The `CPXgetnetcallbackfunc` is used to access the user-written callback routine to be called each time a log message is issued during the optimization of a network problem. If the display log is turned off, the callback routine is still called.

This routine works in the same way as the routine `CPXgetlpcallbackfunc`. It enables the user to create a separate callback function to be called during the solution of a network problem. The prototype for the callback function is identical to that of `CPXgetlpcallbackfunc`.

Callback description:

```c
int callback (CPXENVptr env,
    void *cbdata,
    int wherefrom,
    void *cbhandle);
```

This is the user-written callback routine.

Callback return value:

A nonzero terminates the optimization.

Callback arguments:

`env`

A pointer to the CPLEX environment that was passed into the associated optimization routine.

`cbdata`

A pointer passed from the optimization routine to the user-written callback function that identifies the problem being optimized. The only purpose for the `cbdata` pointer is to pass it to the routine `CPXgetcallbackinfo`.

`wherefrom`

A pointer passed from the optimization routine to the user-written callback function.
An integer value indicating which optimization algorithm the user-written callback function was called from. Possible values and their meaning appear in the table.

<table>
<thead>
<tr>
<th>Value?</th>
<th>Symbolic Constant?</th>
<th>Meaning?</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>CPX_CALLBACK_NETWORK</td>
<td>From network simplex?</td>
</tr>
</tbody>
</table>

**cbhandle**

Pointer to user private data, as passed to `CPXsetlpcallbackfunc`.

**Parameters**

**env**

A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

**callback**

The address of the pointer to the current user-written callback function. If no callback function has been set, the pointer evaluates to NULL.

**cbhandle_p**

The address of a variable to hold the private pointer of the user.

**Example**

```c
status = CPXgetnetcallbackfunc (env, mycallback, NULL);
```

**See Also**

`CPXgetcallbackinfo`

**Returns**

A nonzero terminates the optimization.
null
The address of a variable to hold the user's private pointer.

**Returns**

This routine does not return a result.
CPXgetsolvecallbackfunc

### Category
Global Function

### Definition File
cplex.h

### Include Files
cplex.h

### Synopsis
```c
void CPXPUBLIC CPXgetsolvecallbackfunc(CPXENVptr env,
                                       int (CPXPUBLIC **solvecallback_p)(CALLBACK_SOLVE_ARGS) ,
                                       void ** cbhandle_p)
```

### Description

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetsolvecallbackfunc accesses the user-written callback to be called during MIP optimization to optimize the subproblem.

**Example**

```c
CPXgetsolvecallbackfunc(env, &current_callback,
                        &current_cbdata);
```

See also Advanced MIP Control Interface in the ILOG CPLEX User's Manual.

For documentation of callback arguments, see the routine CPXsetsolvecallbackfunc.

### Parameters

- **env**
  A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

- **solvecallback_p**
  The address of the pointer to the current user-written solve callback. If no callback has been set, the pointer evaluates to NULL.

- **cbhandle_p**
  The address of a variable to hold the user's private pointer.
See Also

CPXgetcallbacknodelp, CPXsetsolvecallbackfunc

Returns

This routine does not return a result.
CPXsetbranchcallbackfunc

Category       Global Function
Definition File cplex.h
Include Files  cplex.h
Synopsis

int CPXPUBLIC CPXsetbranchcallbackfunc(CPXENVptr env,
                                     int (CPXPUBLIC *branchcallback)(CALLBACK_BRANCH_ARGS) ,
                                     void * cbhandle)

Description

Note: This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXsetbranchcallbackfunc sets and modifies the user-written callback routine to be called after a branch has been selected but before the branch is carried out during MIP optimization. In the callback routine, the CPLEX-selected branch can be changed to a user-selected branch.

Example

status = CPXsetbranchcallbackfunc (env, mybranchfunc, mydata);

See also the example admipex1.c in the standard distribution.

Parameters

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.
branchcallback
A pointer to a user-written branch callback. If the callback is set to NULL, no callback can be called during optimization.
cbhandle
A pointer to user private data. This pointer is passed to the callback.
Callback description

```c
int callback (CPXCENVptr env,
    void    *cbdata,
    int     workaround,
    void    *cbhandle,
    int     type,
    int     sos,
    int     nodecnt,
    int     bdcnt,
    double  *nodeest,
    int     *nodebeg,
    int     *indices,
    char    *lu,
    int     *bd,
    int     *useraction_p);
```

The call to the branch callback occurs after a branch has been selected but before the branch is carried out. This function is written by the user. On entry to the callback, the ILOG CPLEX-selected branch is defined in the arguments. The arguments to the callback specify a list of changes to make to the bounds of variables when child nodes are created. One, two, or zero child nodes can be created, so one, two, or zero lists of changes are specified in the arguments. The first branch specified is considered first. The callback is called with zero lists of bound changes when the solution at the node is integer feasible. ILOG CPLEX occasionally elects to branch by changing a number of variables bounds or by adding constraints to the node subproblem; the branch type is then CPX_TYPE_ANY. The details of the constraints added for a CPX_TYPE_ANY branch are not available to the user.

Custom branching strategies can be implemented by calling the CPLEX routine CPXbranchcallbackbranchbds, CPXbranchcallbackbranchconstraints, or CPXbranchcallbackbranchgeneral and setting the useraction variable to CPX_CALLBACK_SET. Then CPLEX will carry out these branches instead of the CPLEX-selected branches.

Branch variables are in terms of the original problem if the parameter CPX_PARAM_MIPCBREDLP is set to CPX_OFF before the call to CPXmipopt that calls the callback. Otherwise, branch variables are in terms of the presolved problem.

Callback return value

The callback returns zero on success and nonzero if an error occurs.

Callback arguments

- `env`
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbdata
A pointer passed from the optimization routine to the user-written callback that identifies the problem being optimized. The only purpose of this pointer is to pass it to the callback information routines.

wherefrom
An integer value indicating where in the optimization this function was called. It will have the value CPX_CALLBACK_MIP_BRANCH.

cbhandle
A pointer to user-private data.

int type
An integer that indicates the type of branch. This table summarizes possible values.

<table>
<thead>
<tr>
<th>Symbolic Constant</th>
<th>Value</th>
<th>Branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_TYPE_VAR</td>
<td>'0'</td>
<td>variable branch</td>
</tr>
<tr>
<td>CPX_TYPE_SOS1</td>
<td>'1'</td>
<td>SOS1 branch</td>
</tr>
<tr>
<td>CPX_TYPE_SOS2</td>
<td>'2'</td>
<td>SOS2 branch</td>
</tr>
<tr>
<td>CPX_TYPE_ANY</td>
<td>'A'</td>
<td>multiple bound changes and/or constraints will be used for branching</td>
</tr>
</tbody>
</table>

sos
An integer that indicates the special ordered set (SOS) used for this branch. A value of −1 indicates that this branch is not an SOS-type branch.

nodecnt
An integer that indicates the number of nodes CPLEX will create from this branch. Possible values are:
- 0 (zero), or
- 1, or
- 2.

If the argument is 0, the node will be fathomed unless user-specified branches are made; that is, no child nodes are created and the node itself is discarded.

bdcnt
An integer that indicates the number of bound changes defined in the arrays indices, lu, and bd that define the CPLEX-selected branch.

nodeest
An array with nodecnt entries that contains estimates of the integer objective-function value that will be attained from the created node.

nodebeg
An array with nodecnt entries. The i-th entry is the index into the arrays indices, lu, and bd of the first bound changed for the i-th node.

indices
Together with lu and bd, this array defines the bound changes for each of the created nodes. The entry indices[i] is the index for the variable.

lu
Together with indices and bd, this array defines the bound changes for each of the created nodes. The entry lu[i] is one of the three possible values indicating which bound to change:
   ◆ 'L' for lower bound, or
   ◆ 'U' for upper bound, or
   ◆ 'B' for both bounds.

bd
Together with indices and lu, this array defines the bound changes for each of the created nodes. The entry bd[i] indicates the new value of the bound.

useraction_p
A pointer to an integer indicating the action for ILOG CPLEX to take at the completion of the user callback. The table summarizes the possible actions.

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CPX_CALLBACK_DEFAULT</td>
<td>Use CPLEX-selected branch</td>
</tr>
<tr>
<td>1</td>
<td>CPX_CALLBACK_FAIL</td>
<td>Exit optimization</td>
</tr>
<tr>
<td>2</td>
<td>CPX_CALLBACK_SET</td>
<td>Use user-selected branch, as defined by calls to CPXbranchcallbackbranch bds</td>
</tr>
</tbody>
</table>

Returns
The routine returns zero on success and nonzero if an error occurs.
**CPXsetbranchnosolncallbackfunc**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXsetbranchnosolncallbackfunc(CPXENVptr env,
   int (CPXPUBLIC *branchnosolncallback)(CALLBACK_BRANCH_ARGS),
   void * cbhandle)
```

**Description**

*Note:* This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXsetbranchnosolncallbackfunc` sets the callback function that will be called instead of the branch callback when there is a failure due to situations as iteration limit being reached, unboundedness being detected, numeric difficulties being encountered, while the node LP is being solved. In consequence of the failure, whether the node is feasible or infeasible cannot be known and thus CPLEX routines such as `CPXsolution` may fail. In this situation, CPLEX will attempt to fix some variables and continue.

These conditions are rare (except when the user has set a very low iteration limit), so it is acceptable to let CPLEX follow its default action in these cases.
**CPXsetcutcallbackfunc**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXsetcutcallbackfunc(CPXENVptr env,
    int(CPXPUBLIC *cutcallback)(CALLBACK_CUT_ARGS) ,
    void * cbhandle)
```

**Description**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXsetcutcallbackfunc sets and modifies the user-written callback for adding cuts. The user-written callback is called by ILOG CPLEX during MIP branch & cut for every node that has an LP optimal solution with objective value below the cutoff and is integer infeasible. The callback routine adds globally valid cuts to the LP subproblem. The cut may be for the original problem if the parameter CPX_PARAM_MIPCBREDLP was set to CPX_OFF before the call to CPXmipopt that calls the callback. Otherwise, the cut is for the presolved problem.

Within the user-written cut callback, the routine CPXgetcallbacknodelp and other query routines from the Callable Library access information about the subproblem. The routines CPXgetcallbacknodeintfeas and CPXgetcallbackssosinfo examines the status of integer entities.

The routine CPXcutcallbackadd adds cuts to the problem. Cuts added to the problem are first put into a cut pool, so they are not present in the subproblem LP until after the user-written cut callback is finished.

Any cuts that are duplicates of cuts already in the subproblem are not added to the subproblem. Cuts that are added remain part of all subsequent subproblems; there is no cut deletion.

If cuts have been added, the subproblem is re-solved and evaluated, and if the LP solution is still integer infeasible and not cut off, the cut callback is called again.
If the problem has names, user-added cuts have names of the form Xnumber where number is a sequence number among all cuts generated.

The parameter CPX_PARAM_REDUCE must be set to CPX_PREREDUCE_PRIMALONLY (1) or CPX_PREREDUCE_NOPRIMALORDUAL (0) if the constraints to be added in the callback are lazy constraints, that is, not implied by the constraints in the constraint matrix. The parameter CPX_PARAM_PRELINEAR must be set to 0 if the constraints to be added are in terms of the original problem and the constraints are valid cutting planes.

Example

```c
status = CPXsetcutcallbackfunc(env, mycutfunc, mydata);
```

See also the example admipex5.c in the standard distribution.

Parameters

- **env**
  A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

- **cutcallback**
  The pointer to the current user-written cut callback. If no callback has been set, the pointer evaluates to NULL.

- **cbhandle**
  A pointer to user private data. This pointer is passed to the user-written cut callback.

Callback description

```c
int callback (CPXENVptr env,
              void     *cbdata,
              int       wherefrom,
              void     *cbhandle,
              int       *useraction_p);
```

ILOG CPLEX calls the cut callback when the LP subproblem for a node has an optimal solution with objective value below the cutoff and is integer infeasible.

Callback return value

The callback returns zero on success and nonzero if an error occurs.

Callback arguments

- **env**
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbdata

A pointer passed from the optimization routine to the user-written callback that identifies the problem being optimized. The only purpose of this pointer is to pass it to the callback information routines.

wherefrom

An integer value indicating where in the optimization this function was called. It has the value CPX_CALLBACK_MIP_CUT.

cbhandle

A pointer to user private data.

useraction_p

A pointer to an integer indicating the action for ILOG CPLEX to take at the completion of the user callback. The table summarizes possible actions.

### Actions to be Taken After a User-Written Cut Callback

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CPX_CALLBACK_DEFAULT</td>
<td>Use cuts as added</td>
</tr>
<tr>
<td>1</td>
<td>CPX_CALLBACK_FAIL</td>
<td>Exit optimization</td>
</tr>
<tr>
<td>2</td>
<td>CPX_CALLBACK_SET</td>
<td>Use cuts as added</td>
</tr>
</tbody>
</table>

### Returns

The routine returns zero on success and nonzero if an error occurs.
**CPXsetdeletenodecallbackfunc**

**Category**  
Global Function

**Definition File**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXsetdeletenodecallbackfunc(CPXENVptr env,  
void(CPXPUBLIC *deletecallback)(CALLBACK_DELETENODE_ARGS),  
void * cbhandle)
```

**Description**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXsetdeletenodecallbackfunc` sets and modifies the user-written callback to be called during MIP optimization when a node is to be deleted. Nodes are deleted in these circumstances:

- when a branch is carried out from that node, or
- when the node relaxation is infeasible, or
- when the node relaxation objective value is worse than the cutoff.

**Example**

```c
status = CPXsetdeletenodecallbackfunc (env,  
    mybranchfunc,  
    mydata);
```

See also the example admipex1.c in the standard distribution.

**Parameters**

- **env**  
  A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
- **deletecallback**  
  A pointer to a user-written branch callback. If the callback is set to NULL, no callback is called during optimization.
cbhandle
A pointer to user private data. This pointer is passed to the callback.

Callback description

```c
int callback (CPXENVptr env,
             void  *cbdata,
             int       wherefrom,
             void  *cbhandle,
             int       seqnum,
             void  *handle);
```

The call to the delete node callback routine occurs during MIP optimization when a node is to be deleted.

The main purpose of the callback is to provide an opportunity to free any user data associated with the node, thus preventing memory leaks.

Callback return value
The callback returns zero on success and nonzero if an error occurs.

Callback arguments

`env`
A pointer to the CPLEX environment, as returned by one of the CPXopenCPLEX routines.

`cbdata`
A pointer passed from the optimization routine to the user-written callback that identifies the problem being optimized. The only purpose of this pointer is to pass it to the callback information routines.

`wherefrom`
An integer value indicating where in the optimization this function was called. It will have the value CFX_CALLBACK_MIP_DELETENODE.

`cbhandle`
A pointer to user private data.

`seqnum`
The sequence number of the node that is being deleted.

`handle`
A pointer to the user private data that was assigned to the node when it was created with one of the callback branching routines:

- CPXbranchcallbackbranchbds, or
- CPXbranchcallbackbranchconstraints, or
- CPXbranchcallbackbranchgeneral.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXsetheuristiccallbackfunc

Category                      Global Function
Definition File               cplex.h
Include Files                 cplex.h
Synopsis                      

Description

Note: This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXsetheuristiccallbackfunc sets or modifies the user-written callback to be called by ILOG CPLEX during MIP optimization after the subproblem has been solved to optimality. That callback is not called when the subproblem is infeasible or cut off. The callback supplies ILOG CPLEX with heuristically-derived integer solutions.

If a linear program must be solved as part of a heuristic callback, make a copy of the node LP and solve the copy, not the CPLEX node LP.

Example

status = CPXsetheuristiccallbackfunc(env, myheuristicfunc, mydata);

See also the example admipex2.c in the standard distribution.

Parameters

env

A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

heuristiccallback

A pointer to a user-written heuristic callback. If this callback is set to NULL, no callback is called during optimization.

cbblehandle
A pointer to the user's private data. This pointer is passed to the callback.

Callback description

```c
int callback (CPXENVptr env,
              void      *cbdata,
              int       wherefrom,
              void      *cbhandle,
              double    *objval_p,
              double    *x,
              int       *checkfeas_p,
              int       *useraction_p);
```

The call to the heuristic callback occurs after an optimal solution to the subproblem has been obtained. The user can provide that solution to start a heuristic for finding an integer solution. The integer solution provided to ILOG CPLEX replaces the incumbent if it has a better objective value. The basis that is saved as part of the incumbent is the optimal basis from the subproblem; it may not be a good basis for starting optimization of the fixed problem.

The integer solution returned to CPLEX is for the original problem if the parameter CPX_PARAM_MIPCBREDLP was set to CPX_OFF before the call to CPXmipopt that calls the callback. Otherwise, it is for the presolved problem.

Callback return value

The callback returns zero on success and nonzero if an error occurs.

Callback arguments

- **env**
  A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

- **cbdata**
  A pointer passed from the optimization routine to the user-written callback to identify the problem being optimized. The only purpose of the cbdata pointer is to pass it to the callback information routines.

- **wherefrom**
  An integer value indicating at which point in the optimization this function was called. It has the value CPX_CALLBACK_MIP_HEURISTIC for the heuristic callback.

- **cbhandle**
  A pointer to user private data.

- **objval_p**
A pointer to a variable that on entry contains the optimal objective value of the subproblem and on return contains the objective value of the integer solution found, if any.

x

An array that on entry contains primal solution values for the subproblem and on return contains solution values for the integer solution found, if any.

checkfeas_p

A pointer to an integer that indicates whether or not ILOG CPLEX should check the returned integer solution for integer feasibility. The solution is checked if checkfeas_p is nonzero. When the solution is checked and found to be integer infeasible, it is discarded, and optimization continues.

useraction_p

A pointer to an integer to contain the indicator for the action to be taken on completion of the user callback. The table summarizes the possible values.

### Actions to be Taken after a User-Written Heuristic Callback

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CPX_CALLBACK_DEFAULT</td>
<td>No solution found</td>
</tr>
<tr>
<td>1</td>
<td>CPX_CALLBACK_FAIL</td>
<td>Exit optimization</td>
</tr>
<tr>
<td>2</td>
<td>CPX_CALLBACK_SET</td>
<td>Use user solution as indicated in return values</td>
</tr>
</tbody>
</table>

### Returns

The routine returns zero on success and nonzero if an error occurs.
CPXsetincumbentcallbackfunc

Category       Global Function
Definition File cplex.h
Include Files  cplex.h
Synopsis       int CPXPUBLIC CPXsetincumbentcallbackfunc (CPXENVptr env,
               int (CPXPUBLIC *incumbentallback)(CALLBACK_INCUMBENT_ARGS),
               void * cbhandle)

Description

Note: This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXsetincumbentcallbackfunc sets and modifies the user-written incumbent callback routine to be called when an integer solution has been found but before this solution replaces the incumbent. This callback can be used to discard solutions that do not meet criteria beyond that of the mixed integer programming formulation.

Variables are in terms of the original problem if the parameter CPX_PARAM_MIPCBREDLP is set to CPX_OFF before the call to CPXmipopt that calls the callback. Otherwise, variables are in terms of the presolved problem.

Example

status = CPXsetincumbentcallbackfunc (env, myincumbentcheck, mydata);

See also Advanced MIP Control Interface in the ILOG CPLEX User's Manual.

Parameters

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

incumbentcallback
A pointer to a user-written incumbent callback. If the callback is set to NULL, no callback can be called during optimization.
cbhandle
A pointer to user private data. This pointer is passed to the callback.

Callback description

```c
int callback (CPXENVptr env,
            void     *cbdata,
            int       wherefrom,
            void     *cbhandle,
            double   objval,
            double   *x,
            int      *isfeas_p,
            int      *useraction_p);
```

The incumbent callback is called when CPLEX has found an integer solution, but before this solution replaces the incumbent integer solution.

Variables are in terms of the original problem if the parameter CPX_PARAM_MIPCBREDLP is set to CPX_OFF before the call to CPXmipopt that calls the callback. Otherwise, variables are in terms of the presolved problem.

Callback return value
The callback returns zero on success and nonzero if an error occurs.

Callback arguments

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbdata
A pointer passed from the optimization routine to the user-written callback that identifies the problem being optimized. The only purpose of this pointer is to pass it to the callback information routines.

wherefrom
An integer value indicating where in the optimization this function was called. It will have the value CPX_CALLBACK_MIP_BRANCH.

cbhandle
A pointer to user private data.

objval
A variable that contains the objective value of the integer solution.

x
An array that contains primal solution values for the integer solution.

isfeas_p

A pointer to an integer variable that indicates whether or not CPLEX should use the integer solution specified in \( x \) to replace the current incumbent. A nonzero value indicates that the incumbent should be replaced by \( x \); a zero value indicates that it should not.

useraction_p

A pointer to an integer to contain the indicator for the action to be taken on completion of the user callback. The table summarizes the possible values.

### Actions to be Taken after a User-Written Incumbent Callback

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CPX_CALLBACK_DEFAULT</td>
<td>Proceed with optimization</td>
</tr>
<tr>
<td>1</td>
<td>CPX_CALLBACK_FAIL</td>
<td>Exit optimization</td>
</tr>
<tr>
<td>2</td>
<td>CPX_CALLBACK_SET</td>
<td>Proceed with optimization</td>
</tr>
</tbody>
</table>

See Also

CPXgetincumbentcallbackfunc

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXsetlpcallbackfunc

Category                  Global Function
Definition File           cplex.h

Synopsis                  int CPXPUBLIC CPXsetlpcallbackfunc(CPXENVptr env,
                                      int(CPXPUBLIC *callback)(CPXCENVptr, void *, int, void *),
                                      void * cbhandle)

Description               The routine CPXsetlpcallbackfunc is used to modify the user-written callback
                           routine to be called after each iteration during the optimization of a linear program, and
                           also periodically during the CPLEX presolve algorithm.

Callback description       int callback (CPXENVptr env,
                            void *cbdata,
                            int wherefrom,
                            void * cbhandle);

This is the user-written callback routine.

Callback return value      A nonzero terminates the optimization.

Callback arguments         env
                           A pointer to the CPLEX environment that was passed into the associated optimization
                           routine.

                           cbdata
                           A pointer passed from the optimization routine to the user-written callback function that
                           identifies the problem being optimized. The only purpose for the cbdata pointer is to pass
                           it to the routine CPXgetcallbackinfo.

                           wherefrom
                           An integer value indicating from which optimization algorithm the user-written callback
                           function was called. Possible values and their meaning appear in the table below.
cbhandle

Pointer to user private data, as passed to CPXsetlpcallbackfunc.

Parameters

env

A pointer to the CPLEX environment as returned by CPXopenCPLEX.

myfunc

A pointer to a user-written callback function. Setting callback to NULL prevents any callback function from being called during optimization. The call to callback occurs after every iteration during optimization and periodically during the CPLEX presolve algorithms. This function is written by the user, and is prototyped as documented here.

cbhandle

A pointer to user private data. This pointer is passed to the callback function.

Example

status = CPXsetlpcallbackfunc (env, myfunc, NULL);

See Also

CPXgetcallbackinfo, CPXsetmipcallbackfunc, CPXsetnetcallbackfunc

Returns

The routine returns zero on success and nonzero if an error occurs.


**CPXsetmipcallbackfunc**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXsetmipcallbackfunc(CPXENVptr env,
    int(CPXPUBLIC *callback)(CPXENVptr, void *, int, void *),
    void * cbhandle)
```

**Description**
The routine CPXsetmipcallbackfunc is used to set the user-written callback routine to be called prior to solving each subproblem in the branch & cut tree, including the root node, during the optimization of a mixed integer program and during some cut generation routines. This routine works in the same way as the routine CPXsetlpcallbackfunc. It enables the user to create a separate callback function to be called during the solution of mixed integer programming problems. The prototype for the callback function is identical to that of CPXsetlpcallbackfunc.

**Example**

```
status = CPXsetmipcallbackfunc (env, mycallback, NULL);
```

**Parameters**

- **env**
A pointer to the CPLEX environment, as returned by one of the CPXopenCPLEX routines.

- **callback**
A pointer to a user-written callback function. Setting callback to NULL will prevent any callback function from being called during optimization. The call to callback will occur after every node during optimization and periodically during the presolve algorithms. This function must be written by the user. Its prototype is explained in the Callback description.

- **cbhandle**
A pointer to user private data. This pointer will be passed to the callback function.

**Callback description**

```c
int callback (CPXENVptr env,
    void *cbdata,
    int wherefrom,
```
void *cbhandle);

This is the user-written callback routine.

**Callback return value**
A nonzero terminates the optimization.

**Callback arguments**

- **env**
  A pointer to the CPLEX environment that was passed into the associated optimization routine.

- **cbdata**
  A pointer passed from the optimization routine to the user-written callback function that identifies the problem being optimized. The only purpose for the `cbdata` pointer is to pass it to the routine `CPXgetcallbackinfo`.

- **wherefrom**
  An integer value indicating from which optimization algorithm the user-written callback function was called. Possible values and their meaning appear in the table.

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>CPX_CALLBACK_MIP</td>
<td>From mipopt</td>
</tr>
<tr>
<td>107</td>
<td>CPX_CALLBACK_MIP_PROBE</td>
<td>From probing or clique merging</td>
</tr>
<tr>
<td>108</td>
<td>CPX_CALLBACK_MIP_FRACCU</td>
<td>From Gomory fractional cuts</td>
</tr>
<tr>
<td>109</td>
<td>CPX_CALLBACK_MIP_DISJCU</td>
<td>From disjunctive cuts</td>
</tr>
<tr>
<td>110</td>
<td>CPX_CALLBACK_MIP_FLOWMI</td>
<td>From Mixed Integer Rounding cuts</td>
</tr>
</tbody>
</table>

**cbhandle**
A pointer to user private data as passed to `CPXsetmipcallbackfunc`.

**See Also**
`CPXgetcallbackinfo`, `CPXsetlpcallbackfunc`, `CPXsetnetcallbackfunc`

**Returns**
The routine returns zero on success and nonzero if an error occurs.
CPXsetnetcallbackfunc

Category Global Function

Definition File cplex.h

Include Files cplex.h

Synopsis

```c
int CPXPUBLIC CPXsetnetcallbackfunc(CPXENVptr env,
    int(CPXPUBLIC *callback)(CPXENVptr, void *, int, void *),
    void * cbhandle)
```

Description

The routine CPXsetnetcallbackfunc is used to set the user-written callback routine to be called each time a log message is issued during the optimization of a network program. If the display log is turned off, the callback routine will still be called. This routine works in the same way as the routine CPXsetlpcallbackfunc. It enables the user to create a separate callback function to be called during the solution of a network problem. The prototype for the callback function is identical to that of CPXsetlpcallbackfunc.

Callback description

```c
int callback (CPXENVptr env,
    void    *cbdata,
    int      wherefrom,
    void    *cbhandle);
```

This is the user-written callback routine.

Callback return value

A nonzero terminates the optimization.

Callback arguments

env

A pointer to the CPLEX environment that was passed into the associated optimization routine.

cbdata

A pointer passed from the optimization routine to the user-written callback function that identifies the problem being optimized. The only purpose for the cbdata pointer is to pass it to the routine CPXgetcallbackinfo.

wherefrom
An integer value indicating from which optimization algorithm the user-written callback function was called. Possible values and their meaning appear in the table.

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>CPX_CALLBACK_NETWORK</td>
<td>From network simplex</td>
</tr>
</tbody>
</table>

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.

- **callback**
  A pointer to a user-written callback function. Setting callback to NULL prevents any callback function from being called during optimization. The call to callback occurs after every log message is issued during optimization and periodically during the CPLEX presolve algorithms. This function is written by the user.

- **cbhandle**
  A pointer to user private data. This pointer is passed to the callback function.

**Example**

```c
status = CPXsetnetcallbackfunc (env, myfunc, NULL);
```

**See Also**

- CPXgetcallbackinfo, CPXsetlpcallbackfunc,
- CPXsetmipcallbackfunc

**Returns**

If the operation is successful, the routine returns zero; if not, it returns nonzero to indicate an error.
CPXsetnodecallbackfunc

Category: Global Function
Definition File: cplex.h
Include Files: cplex.h

Synopsis:
int CPXPUBLIC CPXsetnodecallbackfunc(CPXENVptr env,
               int (CPXPUBLIC *nodecallback) (CALLBACK_NODE_ARGS),
               void * cbhandle)

Description:
Note: This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXsetnodecallbackfunc sets and modifies the user-written callback to be called during MIP optimization after ILOG CPLEX has selected a node to explore, but before this exploration is carried out. The callback routine can change the node selected by ILOG CPLEX to a node selected by the user.

Example:
status = CPXgetnodecallbackfunc(env, mynodefunc, mydata);

See also the example admipex1.c in the standard distribution.

Parameters:
env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.
nodecallback
A pointer to the current user-written node callback. If no callback has been set, the pointer evaluates to NULL.
cbhandle
A pointer to user private data. This pointer is passed to the user-written node callback.

Callback description
int callback (CPXENVptr env,
    void   *cbdata,
    int     wherefrom,
    void   *cbhandle,
    int   *nodeindex_p,
    int   *useraction_p);

ILOG CPLEX calls the node callback after selecting the next node to explore. The user can choose another node by setting the argument values of the callback.

Callback return value
The callback returns zero on success and nonzero if an error occurs.

Callback arguments

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbdata
A pointer passed from the optimization routine to the user-written callback that identifies the problem being optimized. The only purpose of this pointer is to pass it to the callback information routines.

wherefrom
An integer value indicating where in the optimization this function was called. It has the value CPX_CALLBACK_MIP_NODE.

cbhandle
A pointer to user private data.

nodeindex_p
A pointer to an integer that indicates the node number of the user-selected node. The node selected by ILOG CPLEX is node number 0 (zero). Other nodes are numbered relative to their position in the tree, and this number changes with each tree operation. The unchanging identifier for a node is its sequence number. To access the sequence number of a node, use the ILOG CPLEX Callable Library routine CPXgetcallbacknodeinfo. An error results if a user attempts to select a node that has been moved to a node file. (See the ILOG CPLEX User’s Manual for more information about node files.)

useraction_p
A pointer to an integer indicating the action to be taken on completion of the user callback. The table summarizes the possible actions.
Returns

The routine returns zero on success and nonzero if an error occurs.

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CPX_CALLBACK_DEFAULT</td>
<td>Use ILOG CPLEX-selected node</td>
</tr>
<tr>
<td>1</td>
<td>CPX_CALLBACK_FAIL</td>
<td>Exit optimization</td>
</tr>
<tr>
<td>2</td>
<td>CPX_CALLBACK_SET</td>
<td>Use user-selected node as defined in returned values</td>
</tr>
</tbody>
</table>
CPXsetsolvecallbackfunc

Category                  Global Function
Definition File           cplex.h
Include Files            cplex.h
Synopsis                  

```
int CPXPUBLIC CPXsetsolvecallbackfunc(CPXENVptr env,
      int (CPXPUBLIC *solvecallback)(CALLBACK_SOLVE_ARGS),
      void * cbhandle)
```

Description

```
Note: This is an advanced routine. Advanced routines typically demand a profound
understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher
risk of incorrect behavior in your application, behavior that can be difficult to
debug. Therefore, ILOG encourages you to consider carefully whether you can
accomplish the same task by means of other Callable Library routines instead.
```

The routine CPXsetsolvecallbackfunc sets and modifies the user-written
callback to be called during MIP optimization to optimize the subproblem.

Example

```
status = CPXsetsolvecallbackfunc(env, mysolvefunc, mydata);
```

See also the example admipex1.c in the standard distribution.

Parameters

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.
solvecallback
A pointer to a user-written solve callback. If the callback is set to NULL, no callback
is called during optimization.
cbhandle
A pointer to user private data. This pointer is passed to the callback.

Callback description

```
int callback (CPXENVptr env,
      void *cbdata,
```
ILOG CPLEX calls the solve callback before ILOG CPLEX solves the subproblem defined by the current node. The user can choose to solve the subproblem in the solve callback instead by setting the user action parameter of the callback. The optimization that the user provides to solve the subproblem must provide a CPLEX solution. That is, the Callable Library routine CPXgetstat must return a nonzero value. The user may access the lp pointer of the subproblem with the Callable Library routine CPXgetcallbacknodelp.

Callback return value
The callback returns zero on success and nonzero if an error occurs.

Callback arguments

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbdata
A pointer passed from the optimization routine to the user-written callback that identifies the problem being optimized. The only purpose of this pointer is to pass it to the callback information routines.

wherefrom
An integer value indicating where in the optimization this function was called. It will have the value CPX_CALLBACK_MIP_SOLVE.

cbhandle
A pointer to user private data.

useraction_p
A pointer to an integer indicating the to be taken on completion of the user callback. Table 11 summarizes the possible actions.

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CPX_CALLBACK_DEFAULT</td>
<td>Use ILOG CPLEX subproblem optimizer</td>
</tr>
<tr>
<td>1</td>
<td>CPX_CALLBACK_FAIL</td>
<td>Exit optimization</td>
</tr>
</tbody>
</table>

Actions to be Taken after a User-Written Solve Callback
Actions to be Taken after a User-Written Solve Callback

| 2 | CPX_CALLBACK_SET | The subproblem has been solved in the callback |

Returns

The routine returns zero on success and nonzero if an error occurs.

Group optim.cplex.errorcodes

The Callable Library macros that define error codes, their symbolic constants, their short message strings, and their explanations. There is a key to the symbols in the short message strings after the table.

<table>
<thead>
<tr>
<th>Macros Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXERR_ABORT_CONDITION_NO</td>
</tr>
<tr>
<td>CPXERR_ABORT_STRONGBRANCH</td>
</tr>
<tr>
<td>CPXERR_ADJ_SIGN_QUAD</td>
</tr>
<tr>
<td>CPXERR_ADJ_SIGN_SENSE</td>
</tr>
<tr>
<td>CPXERR_ADJ_SIGNS</td>
</tr>
<tr>
<td>CPXERR_ARC_INDEX_RANGE</td>
</tr>
<tr>
<td>CPXERR_ARRAY_BAD_SOS_TYPE</td>
</tr>
<tr>
<td>CPXERR_ARRAY_NOT_ASCENDING</td>
</tr>
<tr>
<td>CPXERR_ARRAY_TOO_LONG</td>
</tr>
<tr>
<td>CPXERR_BAD_ARGUMENT</td>
</tr>
<tr>
<td>CPXERR_BAD_BOUND_SENSE</td>
</tr>
<tr>
<td>CPXERR_BAD_BOUND_TYPE</td>
</tr>
<tr>
<td>CPXERR_BAD_CHAR</td>
</tr>
<tr>
<td>CPXERR_BAD_CTYPE</td>
</tr>
<tr>
<td>CPXERR_BAD_DIRECTION</td>
</tr>
<tr>
<td>CPXERR_BAD_EXPO_RANGE</td>
</tr>
<tr>
<td>Error Code</td>
</tr>
<tr>
<td>-----------------------------</td>
</tr>
<tr>
<td>CPXERR_BAD_EXPONENT</td>
</tr>
<tr>
<td>CPXERR_BAD_FILETYPE</td>
</tr>
<tr>
<td>CPXERR_BAD_ID</td>
</tr>
<tr>
<td>CPXERR_BAD_INDICATOR</td>
</tr>
<tr>
<td>CPXERR_BAD_LUB</td>
</tr>
<tr>
<td>CPXERR_BAD_METHOD</td>
</tr>
<tr>
<td>CPXERR_BAD_NUMBER</td>
</tr>
<tr>
<td>CPXERR_BAD_OBJ_SENSE</td>
</tr>
<tr>
<td>CPXERR_BAD_PARAM_NAME</td>
</tr>
<tr>
<td>CPXERR_BAD_PARAM_NUM</td>
</tr>
<tr>
<td>CPXERR_BAD_PIVOT</td>
</tr>
<tr>
<td>CPXERR_BAD_PRIORITY</td>
</tr>
<tr>
<td>CPXERR_BAD_PROB_TYPE</td>
</tr>
<tr>
<td>CPXERR_BAD_ROW_ID</td>
</tr>
<tr>
<td>CPXERR_BAD_SECTION_BOUNDS</td>
</tr>
<tr>
<td>CPXERR_BAD_SECTION_ENDATA</td>
</tr>
<tr>
<td>CPXERR_BAD_SECTION_QMATRIX</td>
</tr>
<tr>
<td>CPXERR_BAD_SENSE</td>
</tr>
<tr>
<td>CPXERR_BAD_SOS_TYPE</td>
</tr>
<tr>
<td>CPXERR_BAD_STATUS</td>
</tr>
<tr>
<td>CPXERR_BAS_FILE_SHORT</td>
</tr>
</tbody>
</table>
CPX.ERR_BAS_FILE_SIZE  1555 %d %s basic variable(s).
CPX.ERR_BOUNDS_INFES  1100 Bounds on '%s' contradictory.
CPX.ERR_BOUNDS_INT    3011 Non-integral %s bound value %g for integer column '%s'.
CPX.ERR_CALLBACK      1006 Error during callback.
CPX.ERR_CANT_CLOSE_CHILD 1021 Cannot close a child environment.
CPX.ERR_CHILD_OF_CHILD 1019 Cannot clone a cloned environment.
CPX.ERR_COL_INDEX_RANGE 1201 Column index %d out of range.
CPX.ERR_COL_REPEAT_PRINT 1478 %d Column repeats messages not printed.
CPX.ERR_COL_REPEATS    1446 Column '%s' repeats.
CPX.ERR_COL_ROW_REPEATS 1443 Column '%s' has repeated row '%s'.
CPX.ERR_COL_UNKNOWN    1449 Line %d: '%s' is not a column name.
CPX.ERR_COUNT_OVERLAP  1228 Count entry %d indicates overlapping entries.
CPX.ERR_COUNT_RANGE    1227 Count entry %d negative or larger than allowed.
CPX.ERR_DBL_MAX        1233 Numeric entry %d is larger than allowed maximum of %g.
CPX.ERR_DECOMPRESSION  1027 Decompression of unpresolved model failed.
CPX.ERR_DUP_ENTRY      1222 Duplicate entry.
CPX.ERR_EXTRA_BV_BOUND 1456 Line %d: 'BV' bound type illegal when prior bound given.
CPX.ERR_EXTRA_FR_BOUND 1455 Line %d: 'FR' bound type illegal when prior bound given.
CPX.ERR_EXTRA_FX_BOUND 1454 Line %d: 'FX' bound type illegal when prior bound given.
CPX.ERR_EXTRA_INTEND   1481 Line %d: 'INTEND' found while not reading integers.
CPX.ERR_EXTRA_INTORG   1480 Line %d: 'INTORG' found while reading integers.
CPX.ERR_EXTRA_SOSEND   1483 Line %d: 'SOSEND' found while not reading a SOS.
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXERR_EXTRA_SOSORG</td>
<td>1482 Line %d: 'SOSORG' found while reading a SOS.</td>
</tr>
<tr>
<td>CPXERR_FAIL_OPEN_READ</td>
<td>1423 Could not open file '%s' for reading.</td>
</tr>
<tr>
<td>CPXERR_FAIL_OPEN_WRITE</td>
<td>1422 Could not open file '%s' for writing.</td>
</tr>
<tr>
<td>CPXERR_FILE_ENTRIES</td>
<td>1553 Line %d: Wrong number of entries.</td>
</tr>
<tr>
<td>CPXERR_FILE_FORMAT</td>
<td>1563 File '%s' has an incompatible format. Try setting reverse flag.</td>
</tr>
<tr>
<td>CPXERR_IIS_DEFAULT</td>
<td>1706 Unable to compute an IIS.</td>
</tr>
<tr>
<td>CPXERR_IIS_FEAS</td>
<td>1703 Problem is feasible; no IIS available.</td>
</tr>
<tr>
<td>CPXERR_IIS_NO_BASIC</td>
<td>1707 Infeasibility Finder requires a basic solution.</td>
</tr>
<tr>
<td>CPXERR_IIS_NO_INFO</td>
<td>1701 No IIS information exists.</td>
</tr>
<tr>
<td>CPXERR_IIS_NO_LOAD</td>
<td>1709 Failed to load IIS subproblem.</td>
</tr>
<tr>
<td>CPXERR_IIS_NO_SOLN</td>
<td>1702 No solution exists; infeasibility analysis not available.</td>
</tr>
<tr>
<td>CPXERR_IIS_NOT_INFEAS</td>
<td>1704 Problem may be feasible; no IIS available.</td>
</tr>
<tr>
<td>CPXERR_IIS_NUM_BEST</td>
<td>1713 Unable to optimize IIS subproblem.</td>
</tr>
<tr>
<td>CPXERR_IIS_OPT_INFEAS</td>
<td>1705 Problem optimal with unscaled infeasibilities. No IIS available.</td>
</tr>
<tr>
<td>CPXERR_IIS_SUB_IT_LIM</td>
<td>1711 Iteration limit reached on IIS subproblem.</td>
</tr>
<tr>
<td>CPXERR_IIS_SUB_OBJ_LIM</td>
<td>1710 Objective limit reached on IIS subproblem.</td>
</tr>
<tr>
<td>CPXERR_IIS_SUB_TIME_LIM</td>
<td>1712 Time limit reached on IIS subproblem.</td>
</tr>
<tr>
<td>CPXERR_INDEX_NOT_BASIC</td>
<td>1251 Index must correspond to a basic variable.</td>
</tr>
<tr>
<td>CPXERR_INDEX_RANGE</td>
<td>1200 Index is outside range of valid values.</td>
</tr>
<tr>
<td>CPXERR_INDEX_RANGE_HIGH</td>
<td>1206 %s: 'end' value %d is greater than %d.</td>
</tr>
<tr>
<td>CPXERR_INDEX_RANGE_LOW</td>
<td>1205 %s: 'begin' value %d is less than %d.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CPXERR_INT_TOO_BIG</td>
<td>3018 Magnitude of variable %s: %g exceeds integer limit %d.</td>
</tr>
<tr>
<td>CPXERR_INT_TOO_BIG_INPUT</td>
<td>1463 Line %d: Magnitude exceeds integer limit %d.</td>
</tr>
<tr>
<td>CPXERR_INVALID_NUMBER</td>
<td>1650 Number not representable in exponential notation.</td>
</tr>
<tr>
<td>CPXERR_LIMITS_TOO_BIG</td>
<td>1012 Problem size limits too large.</td>
</tr>
<tr>
<td>CPXERR_LO_BOUND_REPEATS</td>
<td>1459 Line %d: Repeated lower bound.</td>
</tr>
<tr>
<td>CPXERR_MISS_SOS_TYPE</td>
<td>3301 Line %d: Missing SOS type.</td>
</tr>
<tr>
<td>CPXERR_MSG_NO_CHANNEL</td>
<td>1051 No channel pointer supplied to message routine.</td>
</tr>
<tr>
<td>CPXERR_MSG_NO_FILEPTR</td>
<td>1052 No file pointer found for message routine.</td>
</tr>
<tr>
<td>CPXERR_MSG_NO_FUNCTION</td>
<td>1053 No function pointer found for message routine.</td>
</tr>
<tr>
<td>CPXERR_NAME_CREATION</td>
<td>1209 Unable to create default names.</td>
</tr>
<tr>
<td>CPXERR_NAME_NOT_FOUND</td>
<td>1210 Name not found.</td>
</tr>
<tr>
<td>CPXERR_NAME_TOO_LONG</td>
<td>1464 Line %d: Identifier/name too long to process.</td>
</tr>
<tr>
<td>CPXERR_NAN</td>
<td>1225 Numeric entry %d is not a double precision number (NAN).</td>
</tr>
<tr>
<td>CPXERR_NEED_OPT_SOLN</td>
<td>1252 Optimal solution required.</td>
</tr>
<tr>
<td>CPXERR_NEGATIVE_SURPLUS</td>
<td>1207 Insufficient array length.</td>
</tr>
<tr>
<td>CPXERR_NET_DATA</td>
<td>1530 Inconsistent network file.</td>
</tr>
<tr>
<td>CPXERR_NET_FILE_SHORT</td>
<td>1538 Unexpected end of network file.</td>
</tr>
<tr>
<td>CPXERR_NO_BARRIER_SOLN</td>
<td>1223 No barrier solution exists.</td>
</tr>
<tr>
<td>CPXERR_NO_BASIC_SOLN</td>
<td>1261 No basic solution exists.</td>
</tr>
<tr>
<td>CPXERR_NO_BASIS</td>
<td>1262 No basis exists.</td>
</tr>
<tr>
<td>CPXERR_NO_BOUND_SENSE</td>
<td>1621 Line %d: No bound sense.</td>
</tr>
<tr>
<td>CPXERR_NO_BOUND_TYPE</td>
<td>1460 Line %d: Bound type missing.</td>
</tr>
<tr>
<td>CPXERR_NO_COLUMNS_SECTION</td>
<td>1472 Line %d: No COLUMNS section.</td>
</tr>
<tr>
<td>CPXERR_NO_DUAL_SOLN</td>
<td>1232 No dual solution exists.</td>
</tr>
<tr>
<td>CPXERR_NO_ENDATA</td>
<td>1552 ENDATA missing.</td>
</tr>
<tr>
<td>CPXERR_NO_ENVIRONMENT</td>
<td>1002 No environment.</td>
</tr>
<tr>
<td>CPXERR_NO_FILENAME</td>
<td>1421 File name not specified.</td>
</tr>
<tr>
<td>CPXERR_NO_ID</td>
<td>1616 Line %d: Expected identifier, found '%c'.</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>CPXERR_NO_ID_FIRST</td>
<td>1609 Line %d: Expected identifier first.</td>
</tr>
<tr>
<td>CPXERR_NO_INT_SOLN</td>
<td>3017 No integer feasible solution exists.</td>
</tr>
<tr>
<td>CPXERR_NO_INT_X</td>
<td>3023 Integer feasible solution values are unavailable.</td>
</tr>
<tr>
<td>CPXERR_NO_LU_FACTOR</td>
<td>1258 No LU factorization exists.</td>
</tr>
<tr>
<td>CPXERR_NO_MEMORY</td>
<td>1001 Out of memory.</td>
</tr>
<tr>
<td>CPXERR_NO_MIPSTART</td>
<td>3020 No MIP start exists.</td>
</tr>
<tr>
<td>CPXERR_NO_NAME_SECTION</td>
<td>1441 Line %d: No NAME section.</td>
</tr>
<tr>
<td>CPXERR_NO_NAMES</td>
<td>1219 No names exist.</td>
</tr>
<tr>
<td>CPXERR_NO_NORMS</td>
<td>1264 No norms available.</td>
</tr>
<tr>
<td>CPXERR_NO_NUMBER</td>
<td>1615 Line %d: Expected number, found '%c'.</td>
</tr>
<tr>
<td>CPXERR_NO_NUMBER_BOUND</td>
<td>1623 Line %d: Missing bound number.</td>
</tr>
<tr>
<td>CPXERR_NO_NUMBER_FIRST</td>
<td>1611 Line %d: Expected number first.</td>
</tr>
<tr>
<td>CPXERR_NO_OBJ_SENSE</td>
<td>1436 Max or Min missing.</td>
</tr>
<tr>
<td>CPXERR_NO_OBJECTIVE</td>
<td>1476 Line %d: No objective row found.</td>
</tr>
<tr>
<td>CPXERR_NO_OP_OR_SENSE</td>
<td>1608 Line %d: Expected '+' or '-' or sense, found '%c'.</td>
</tr>
<tr>
<td>CPXERR_NO_OPERATOR</td>
<td>1607 Line %d: Expected '+' or '-', found '%c'.</td>
</tr>
<tr>
<td>CPXERR_NO_ORDER</td>
<td>3016 No priority order exists.</td>
</tr>
<tr>
<td>CPXERR_NO_PROBLEM</td>
<td>1009 No problem exists.</td>
</tr>
<tr>
<td>CPXERR_NO_QMATRIX_SECTION</td>
<td>1461 Line %d: No QMATRIX section.</td>
</tr>
<tr>
<td>CPXERR_NO_QP_OPERATOR</td>
<td>1614 Line %d: Expected '^' or '*'.</td>
</tr>
<tr>
<td>CPXERR_NO_QUAD_EXP</td>
<td>1612 Line %d: Expected quadratic exponent.</td>
</tr>
<tr>
<td>CPXERR_NO_RHS_COEFF</td>
<td>1610 Line %d: Expected RHS coefficient.</td>
</tr>
<tr>
<td>CPXERR_NO_RHS_IN_OBJ</td>
<td>1211 rhs has no coefficient in obj.</td>
</tr>
<tr>
<td>CPXERR_NO_RNGVAL</td>
<td>1216 No range values.</td>
</tr>
<tr>
<td>CPXERR_NO_ROW_NAME</td>
<td>1486 Line %d: No row name.</td>
</tr>
<tr>
<td>CPXERR_NO_ROW_SENSE</td>
<td>1453 Line %d: No row sense.</td>
</tr>
<tr>
<td>----------------------------</td>
<td>------------------------------------------------------</td>
</tr>
<tr>
<td>CPXERR_NO_ROWS_SECTION</td>
<td>1471 Line %d: No ROWS section.</td>
</tr>
<tr>
<td>CPXERR_NO_SENSIT</td>
<td>1260 Sensitivity analysis not available for current status.</td>
</tr>
<tr>
<td>CPXERR_NO_SOLN</td>
<td>1217 No solution exists.</td>
</tr>
<tr>
<td>CPXERR_NO_SOS</td>
<td>3015 No user-defined SOSs exist.</td>
</tr>
<tr>
<td>CPXERR_NO_SOS_SEPARATOR</td>
<td>1627 Expected ‘:’, found ‘%c’.</td>
</tr>
<tr>
<td>CPXERR_NO_TREE</td>
<td>3412 Current problem has no tree.</td>
</tr>
<tr>
<td>CPXERR_NO_VECTOR_SOLN</td>
<td>1556 Vector solution does not exist.</td>
</tr>
<tr>
<td>CPXERR_NODE_INDEX_RANGE</td>
<td>1230 Node index %d out of range.</td>
</tr>
<tr>
<td>CPXERR_NODE_ON_DISK</td>
<td>3504 No callback info on disk/compressed nodes.</td>
</tr>
<tr>
<td>CPXERR_NOT_DUAL_UNBOUNDED</td>
<td>1265 Dual unbounded solution required.</td>
</tr>
<tr>
<td>CPXERR_NOT_FIXED</td>
<td>1221 Only fixed variables are pivoted out.</td>
</tr>
<tr>
<td>CPXERR_NOT_FOR_MIP</td>
<td>1017 Not available for mixed-integer problems.</td>
</tr>
<tr>
<td>CPXERR_NOT_FOR_QCP</td>
<td>1031 Not available for QCP.</td>
</tr>
<tr>
<td>CPXERR_NOT_FOR_QP</td>
<td>1018 Not available for quadratic programs.</td>
</tr>
<tr>
<td>CPXERR_NOT_MILPCLASS</td>
<td>1024 Not a MILP, relaxed MILP, or fixed MILP.</td>
</tr>
<tr>
<td>CPXERR_NOT_MIN_COST_FLOW</td>
<td>1531 Not a min-cost flow problem.</td>
</tr>
<tr>
<td>CPXERR_NOT_MIP</td>
<td>3003 Not a mixed-integer problem.</td>
</tr>
<tr>
<td>CPXERR_NOT_MIQPCCLASS</td>
<td>1029 Not a MIQP or fixed MIQP.</td>
</tr>
<tr>
<td>CPXERR_NOT_ONE_PROBLEM</td>
<td>1023 Not a single problem, relaxed or fixed.</td>
</tr>
<tr>
<td>CPXERR_NOT_QP</td>
<td>5004 Not a quadratic program.</td>
</tr>
<tr>
<td>CPXERR_NOT_QPCCLASS</td>
<td>1025 Not a QP or zeroed QP.</td>
</tr>
<tr>
<td>CPXERR_NOT_SAV_FILE</td>
<td>1560 File ‘%s’ is not a SAV file.</td>
</tr>
<tr>
<td>CPXERR_NOT_UNBOUNDED</td>
<td>1254 Unbounded solution required.</td>
</tr>
<tr>
<td>CPXERR_NULL_NAME</td>
<td>1224 Null pointer %d in name array.</td>
</tr>
<tr>
<td>CPXERR_NULL_POINTER</td>
<td>1004 Null pointer for required data.</td>
</tr>
<tr>
<td>CPXERR_ORDER_BAD_DIRECTION</td>
<td>3007 Illegal direction entry %d.</td>
</tr>
<tr>
<td>CPXERR_PARAM_TOO_BIG</td>
<td>1015 Parameter value too big.</td>
</tr>
<tr>
<td>CPXERR_PARAM_TOO_SMALL</td>
<td>1014 Parameter value too small.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>CPXERR_PRESLV_ABORT</td>
<td>1106 Aborted during presolve.</td>
</tr>
<tr>
<td>CPXERR_PRESLV_BAD_PARAM</td>
<td>1102 Bad presolve parameter setting.</td>
</tr>
<tr>
<td>CPXERR_PRESLV_BASIS_MEM</td>
<td>1107 Not enough memory to build basis for original LP.</td>
</tr>
<tr>
<td>CPXERR_PRESLV_COPYORDER</td>
<td>1109 Can't copy priority order info from original MIP.</td>
</tr>
<tr>
<td>CPXERR_PRESLV_COPYSOS</td>
<td>1108 Can't copy SOS info from original MIP.</td>
</tr>
<tr>
<td>CPXERR_PRESLV_CRUSHFORM</td>
<td>1121 Can't crush solution form.</td>
</tr>
<tr>
<td>CPXERR_PRESLV_DUAL</td>
<td>1119 The feature is not available for solving dual formulation.</td>
</tr>
<tr>
<td>CPXERR_PRESLV_FAIL_BASIS</td>
<td>1114 Could not load unpresolved basis for original LP.</td>
</tr>
<tr>
<td>CPXERR_PRESLV_INF</td>
<td>1117 Presolve determines problem is infeasible.</td>
</tr>
<tr>
<td>CPXERR_PRESLV_INFOrUNBD</td>
<td>1101 Presolve determines problem is infeasible or unbounded.</td>
</tr>
<tr>
<td>CPXERR_PRESLV_NO_BASIS</td>
<td>1115 Failed to find basis in presolved LP.</td>
</tr>
<tr>
<td>CPXERR_PRESLV_NO_PROB</td>
<td>1103 No presolved problem created.</td>
</tr>
<tr>
<td>CPXERR_PRESLV_SOLN_MIP</td>
<td>1110 Not enough memory to recover solution for original MIP.</td>
</tr>
<tr>
<td>CPXERR_PRESLV_SOLN_QP</td>
<td>1111 Not enough memory to compute solution to original QP.</td>
</tr>
<tr>
<td>CPXERR_PRESLV_START_LP</td>
<td>1112 Not enough memory to build start for original LP.</td>
</tr>
<tr>
<td>CPXERR_PRESLV_UNBD</td>
<td>1118 Presolve determines problem is unbounded.</td>
</tr>
<tr>
<td>CPXERR_PRESLV_UNCRUSHFORM</td>
<td>1120 Can't uncrush solution form.</td>
</tr>
<tr>
<td>CPXERR_PRIIND</td>
<td>1257 Incorrect usage of pricing indicator.</td>
</tr>
<tr>
<td>CPXERR_PRM_DATA</td>
<td>1660 Line %d: Not enough entries.</td>
</tr>
<tr>
<td>CPXERR_PRM_HEADER</td>
<td>1661 Line %d: Missing or invalid header.</td>
</tr>
<tr>
<td>CPXERR_PTHREAD_CREATE</td>
<td>3603 Could not create thread.</td>
</tr>
<tr>
<td>CPXERR_PTHREAD_MUTEX_INIT</td>
<td>3601 Could not initialize mutex.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Message Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CPXERR_Q_DIVISOR</td>
<td>1619 Line %d: Missing or incorrect divisor for Q terms.</td>
</tr>
<tr>
<td>CPXERR_Q_DUP_ENTRY</td>
<td>5011 Duplicate entry for pair &quot;%s&quot; and '%s'.</td>
</tr>
<tr>
<td>CPXERR_Q_NOT_INDEF</td>
<td>5014 Q is not indefinite.</td>
</tr>
<tr>
<td>CPXERR_Q_NOT_POS_DEF</td>
<td>5002 Q is not positive semi-definite.</td>
</tr>
<tr>
<td>CPXERR_Q_NOT_SYMMETRIC</td>
<td>5012 Q is not symmetric.</td>
</tr>
<tr>
<td>CPXERR_QCP_SENSE_FILE</td>
<td>1437 Line %d: Equality sense is not allowed for quadratic constraints.</td>
</tr>
<tr>
<td>CPXERR_QUAD_EXP_NOT_2</td>
<td>1613 Line %d: Quadratic exponent must be 2.</td>
</tr>
<tr>
<td>CPXERR_QUAD_IN_ROW</td>
<td>1605 Line %d: Quadratic term in lazy constraint or user cut.</td>
</tr>
<tr>
<td>CPXERR_RANGE_SECTION_ORDER</td>
<td>1474 Line %d: 'RANGES' section out of order.</td>
</tr>
<tr>
<td>CPXERR_RESTRICTED_VERSION</td>
<td>1016 Promotional version. Problem size limits exceeded.</td>
</tr>
<tr>
<td>CPXERR_RHS_IN_OBJ</td>
<td>1603 Line %d: RHS sense in objective.</td>
</tr>
<tr>
<td>CPXERR_RIM_REPEATS</td>
<td>1447 Line %d: %s '%s' repeats.</td>
</tr>
<tr>
<td>CPXERR_RIM_ROW_REPEATS</td>
<td>1444 %s '%s' has repeated row '%s'.</td>
</tr>
<tr>
<td>CPXERR_RIMNZ_REPEATS</td>
<td>1479 Line %d: %s %s repeats.</td>
</tr>
<tr>
<td>CPXERR_ROW_INDEX_RANGE</td>
<td>1203 Row index %d out of range.</td>
</tr>
<tr>
<td>CPXERR_ROW_REPEAT_PRINT</td>
<td>1477 %d Row repeats messages not printed.</td>
</tr>
<tr>
<td>CPXERR_ROW_REPEATS</td>
<td>1445 Row '%s' repeats.</td>
</tr>
<tr>
<td>CPXERR_ROW_UNKNOWN</td>
<td>1448 Line %d: '%s' is not a row name.</td>
</tr>
<tr>
<td>CPXERR_SAV_FILE_DATA</td>
<td>1561 Not enough data in SAV file.</td>
</tr>
<tr>
<td>CPXERR_SAV_FILE_WRITE</td>
<td>1562 Unable to write SAV file to disk.</td>
</tr>
<tr>
<td>CPXERR_SBASE_ILLEGAL</td>
<td>1554 Superbases are not allowed.</td>
</tr>
<tr>
<td>CPXERR_SBASE_INCOMPAT</td>
<td>1255 Incompatible with superbasis.</td>
</tr>
<tr>
<td>CPXERR_SEMI_BDS</td>
<td>3022 Infinite bds on semi-continuous or semi-integer variable %d.</td>
</tr>
<tr>
<td>CPXERR_SINGULAR</td>
<td>1256 Basis singular.</td>
</tr>
<tr>
<td>CPXERR_STR_PARAM_TOO_LONG</td>
<td>1026 String parameter is too long.</td>
</tr>
<tr>
<td>CPXERR_SUBPROB_SOLVE</td>
<td>3019 Failure to solve MIP subproblem.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CPXERR_TILIM_CONDITION_NO</td>
<td>1268 Time limit reached in computing condition number.</td>
</tr>
<tr>
<td>CPXERR_TILIM_STRONGBRANCH</td>
<td>1266 Time limit reached in strong branching.</td>
</tr>
<tr>
<td>CPXERR_TOO_MANY_COEFFS</td>
<td>1433 Too many coefficients.</td>
</tr>
<tr>
<td>CPXERR_TOO_MANY_COLS</td>
<td>1432 Too many columns.</td>
</tr>
<tr>
<td>CPXERR_TOO_MANY_RIMNZ</td>
<td>1485 Too many rim nonzeros.</td>
</tr>
<tr>
<td>CPXERR_TOO_MANY_RIMS</td>
<td>1484 Too many rim vectors.</td>
</tr>
<tr>
<td>CPXERR_TOO_MANY.Rows</td>
<td>1431 Too many rows.</td>
</tr>
<tr>
<td>CPXERR_TOO_MANY_THREADS</td>
<td>1020 Thread limit exceeded.</td>
</tr>
<tr>
<td>CPXERR_TRE_FILE_COLS</td>
<td>3405 Tree in file %s does not match the current problem. Number of variables in TRE file: %d. Number of variables in current problem: %d.</td>
</tr>
<tr>
<td>CPXERR_TRE_FILE_DATA</td>
<td>3401 Not enough data in TRE file.</td>
</tr>
<tr>
<td>CPXERR_TRE_FILE_FORMAT</td>
<td>3414 File '%s' has an incompatible format.</td>
</tr>
<tr>
<td>CPXERR_TRE_FILE_INTS</td>
<td>3407 Tree in file %s does not match the current problem. Number of integer variables in TRE file: %d. Number of integer variables in current problem: %d.</td>
</tr>
<tr>
<td>CPXERR_TRE_FILE_NONZ</td>
<td>3408 Tree in file %s does not match the current problem. Number of nonzeros in TRE file: %d. Number of nonzeros in current problem: %d.</td>
</tr>
<tr>
<td>CPXERR_TRE_FILE_OBJ</td>
<td>3404 Objective names in TRE file %s do not match current problem.</td>
</tr>
<tr>
<td>CPXERR_TRE_FILE_OBJSEN</td>
<td>3415 Objective sense in TRE file %s does match current problem.</td>
</tr>
<tr>
<td>CPXERR_TRE_FILE_PRESOLVE</td>
<td>3410 Presolve setting for %s in file %s does not match the current setting.</td>
</tr>
<tr>
<td>CPXERR_TRE_FILE_ROWS</td>
<td>3406 Tree in file %s does not match the current problem. Number of constraints in TRE file: %d. Number of constraints in current problem: %d.</td>
</tr>
<tr>
<td>CPXERR_TRE_FILE_TYPES</td>
<td>3409 Different type in TRE file %s for variable '%s'.</td>
</tr>
</tbody>
</table>
Description

Each error code, such as 1616, is associated with a symbolic constant, such as CPXERR_NO_ID, and a short message string, such as Line %d: Expected identifier, found '%c'.

In the short message strings, the following symbols occur:

%d means a number, such as a line number
%s means a string, such as a file name, variable name, or other
%c means a character, such as a letter or arithmetic operator

Click the symbolic constant in the table to go to a longer explanation of an error code.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXERR_TRE_FILE_VERSION</td>
<td>3403 Can't read TRE file written by CPLEX %s (Current version is %s).</td>
</tr>
<tr>
<td>CPXERR_TRE_FILE_WRITE</td>
<td>3402 Error on write to TRE file.</td>
</tr>
<tr>
<td>CPXERR_TREE_MEMORY_LIMIT</td>
<td>3413 Tree memory limit exceeded.</td>
</tr>
<tr>
<td>CPXERR_UNIQUE_WEIGHTS</td>
<td>3010 Set does not have unique weights.</td>
</tr>
<tr>
<td>CPXERR_UP_BOUND_REPEATS</td>
<td>1458 Line %d: Repeated upper bound.</td>
</tr>
<tr>
<td>CPXERR_WORK_FILE_READ</td>
<td>1802 Failure on temporary file read.</td>
</tr>
<tr>
<td>CPXERR_WORK_FILE_WRITE</td>
<td>1803 Failure on temporary file write.</td>
</tr>
</tbody>
</table>
CPXERR_ABORT_CONDITION_NO

Category Macro

Synopsis CPXERR_ABORT_CONDITION_NO()

Summary 1269 Condition number computation aborted.

Description The computation of Kappa was terminated early, due to a user interrupt or other reason.
CPXERR_ABORT_STRONGBRANCH

Category: Macro

Synopsis: CPXERR_ABORT_STRONGBRANCH()

Summary: 1263 Strong branching aborted.

Description: Strong branching, for variable selection, could not proceed because a subproblem optimization was aborted.
CPXERR_ADJ_SIGNS

Category  Macro

Synopsis  CPXERR_ADJ_SIGNS()

Summary  1602 Lines %d,%d: Adjacent signs.

Description  The previous line ended with a + or - so the next line must start with a variable name rather than an operator.
CPXERR_ADJ_SIGN_QUAD

Category  Macro
Synopsis    CPXERR_ADJ_SIGN_QUAD()
Summary    1606 Lines %d,%d: Adjacent sign and quadratic character.
Description The previous line ended with a + or - so the subsequent line must start with a variable name rather than an one of the reserved quadratic characters []*^.
**CPXERR_ADJ_SIGN_SENSE**

**Category**    Macro

**Synopsis**    `CPXERR_ADJ_SIGN_SENSE()`

**Summary**    1604 Lines %d,%d: Adjacent sign and sense.

**Description**    A sense indicator erroneously follows an arithmetic operator.
CPXERR_ARC_INDEX_RANGE

<table>
<thead>
<tr>
<th>Category</th>
<th>Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synopsis</td>
<td>CPXERR_ARC_INDEX_RANGE()</td>
</tr>
<tr>
<td>Summary</td>
<td>1231 Arc index %d out of range.</td>
</tr>
<tr>
<td>Description</td>
<td>The specified arc index is negative or greater than or equal to the number of arcs in the network.</td>
</tr>
</tbody>
</table>
CPXERR_ARRAY_BAD_SOS_TYPE

Category: Macro

Synopsis: CPXERR_ARRAY_BAD_SOS_TYPE

Summary: 3009 Illegal sostype entry %d.

Description: Only sostype values of 1 or 2 are legal.
### CPXERR_ARRAY_NOT_ASCENDING

<table>
<thead>
<tr>
<th>Category</th>
<th>Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synopsis</td>
<td>CPXERR_ARRAY_NOT_ASCENDING()</td>
</tr>
<tr>
<td>Summary</td>
<td>1226 Array entry %d not ascending.</td>
</tr>
<tr>
<td>Description</td>
<td>Entries in matbeg or sosbeg arrays must be ascending.</td>
</tr>
</tbody>
</table>
**CPXERR_ARRAY_TOO_LONG**

**Category**
Macro

**Synopsis**
CPXERR_ARRAY_TOO_LONG()

**Summary**
1208 Array length too long.

**Description**
The number of norm values passed to CPXcopypnorms exceeds the number of columns, or the number of norm values passed to CPXcopydnorms exceeds the number of rows.
**CPXERR_BAD_ARGUMENT**

<table>
<thead>
<tr>
<th>Category</th>
<th>Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synopsis</td>
<td><code>CPXERR_BAD_ARGUMENT()</code></td>
</tr>
<tr>
<td>Summary</td>
<td>1003 Bad argument to Callable Library routine.</td>
</tr>
<tr>
<td>Description</td>
<td>An invalid argument was passed.</td>
</tr>
</tbody>
</table>
CPXERR_BAD_BOUNDSENSE

Category       Macro
Synopsis        CPXERR_BAD_BOUNDSENSE
Summary         1622 Line %d: Invalid bound sense.
Description     An invalid bounds sense indicator appears in the LP file. Acceptable bound senses are <, >, =, or free.
CPXERR_BAD_BOUND_TYPE

Category: Macro

Synopsis: CPXERR_BAD_BOUND_TYPE()

Summary: 1457 Line %d: Unrecognized bound type '%s'.

Description: An unrecognized bounds sense indicator appears in the MPS file. Acceptable bound senses are BV, LI, UI, UP, LO, FX, FR, MI, PL, and SC.
**CPXERR_BAD_CHAR**

**Category**
Macro

**Synopsis**
CPXERR_BAD_CHAR()

**Summary**
1537 Illegal character.

**Description**
That character is not allowed. See specifications of the NET or MIN format.
CPXERR_BAD_CTYPE

Category: Macro

Synopsis: CPXERR_BAD_CTYPE()

Summary: 3021 Illegal ctype entry %d.

Description: An illegal ctype character has been passed to CPXchgctype. Use one of these: C, B, I, S, or N.
CPXERR_BAD_DIRECTION

Category: Macro

Synopsis: CPXERR_BAD_DIRECTION()

Summary: 3012 Line %d: Unrecognized direction '%c%c'.

Description: Only UP and DN are accepted as branching directions beginning in column 2 of an ORD file.
CPXERR_BAD_EXPONENT

Category: Macro

Synopsis: CPXERR_BAD_EXPONENT()

Summary: 1618 Line %d: Exponent '%s' not %s with number.

Description: The characters following an exponent on the indicated line are not numbers.
CPXERR_BAD_EXPO_RANGE

Category  Macro
Synopsis   CPXERR_BAD_EXPO_RANGE()
Summary   1435 Line %d: Exponent '%s' out of range.
Description An exponent on the indicated line is greater than the largest permitted for your computer system.
## CPXERR_BAD_FILETYPE

<table>
<thead>
<tr>
<th>Category</th>
<th>Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synopsis</td>
<td><code>CPXERR_BAD_FILETYPE()</code></td>
</tr>
<tr>
<td>Summary</td>
<td>1424 Invalid filetype.</td>
</tr>
<tr>
<td>Description</td>
<td>An invalid file type has been passed to a routine requiring a file type.</td>
</tr>
</tbody>
</table>
## CPXERR_BAD_ID

<table>
<thead>
<tr>
<th>Category</th>
<th>Macro</th>
</tr>
</thead>
</table>

### Synopsis

`CPXERR_BAD_ID()`

### Summary

1617 Line %d: '%s' not valid identifier.

### Description

An illegal variable or row name exists on the indicated line.
## CPXERR_BAD_INDICATOR

<table>
<thead>
<tr>
<th>Category</th>
<th>Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synopsis</td>
<td><code>CPXERR_BAD_INDICATOR()</code></td>
</tr>
<tr>
<td>Summary</td>
<td>1551 Line <code>%%d</code>: Unrecognized basis indicator ‘%%s’.</td>
</tr>
<tr>
<td>Description</td>
<td>An invalid basis indicator appears in the BAS file.</td>
</tr>
</tbody>
</table>
CPXERR_BAD_LUB

Category
Macro

Synopsis
CPXERR_BAD_LUB()

Summary
1229 Illegal bound change indicator entry %d.

Description
The bound change indicators must be L, U, or B.
CPXERR_BAD_METHOD

Category            Macro
Synopsis             CPXERR_BAD_METHOD()
Summary              1292 Invalid choice of optimization method.
Description          Unknown method selected for CPXhybnetopt or CPXhybbaropt. Select CPX_ALG_PRIMAL or CPX_ALG_DUAL.
## CPXERR_BAD_NUMBER

**Category**  
Macro

**Synopsis**  
CPXERR_BAD_NUMBER()

**Summary**  
1434 Line %d: Couldn’t convert '%s' to a number.

**Description**  
CPLEX was unable to interpret a string as a number on the indicated line.
**CPXERR_BAD_OBJ_SENSE**

**Category**  
Macro

**Synopsis**  
`CPXERR_BAD_OBJ_SENSE()`

**Summary**  
1487 Line %d: Unrecognized objective sense 's'.

**Description**  
There is an OBJSENSE line in an MPS problem file, but CPLEX can not locate the MIN or MAX objective sense statement. Check the MPS file for correct syntax. See the File Formats Manual for a description of MPS format.
CPXERR_BAD_PARAM_NAME

Category: Macro

Synopsis: CPXERR_BAD_PARAM_NAME()

Summary: 1028 Bad parameter name to CPLEX parameter routine.

Description: The parameter name does not exist.
**Category**  
Macro

**Synopsis**  
CPXERR_BAD_PARAM_NUM

**Summary**  
1013 Bad parameter number to CPLEX parameter routine.

**Description**  
The CPLEX parameter number does not exist.
CPXERR_BAD_PIVOT

Category: Macro

Synopsis: CPXERR_BAD_PIVOT()

Summary: 1267 Illegal pivot.

Description: This error occurs if illegal or bad simplex pivots are attempted. Examples are attempts to remove nonbasic variables from the basis or selection of a zero column to enter the basis. Also, this error code may be generated if a pivot would yield a numerically unstable or singular basis.
**CPXERR_BAD_PRIORITY**

**Category**       Macro

**Synopsis**       CPXERR_BAD_PRIORITY()

**Summary**        3006 Negative priority entry %d.

**Description**    Priority orders must be positive integer values.
CPXERR_BAD_PROB_TYPE

Category: Macro

Synopsis: CPXERR_BAD_PROB_TYPE()

Summary: 1022 Unknown problem type. Problem not changed.

Description: CPXchgpctype could not change the problem type since an unknown type was specified.
**CPXERR_BAD_ROW_ID**

**Category**  
Macro

**Synopsis**  
`CPXERR_BAD_ROW_ID`

**Summary**  
1532 Incorrect row identifier.

**Description**  
Selected row does not exist.
CPXERR_BAD_SECTION_BOUNDS

Category: Macro

Synopsis: CPXERR_BAD_SECTION_BOUNDS()

Summary: 1473 Line %d: Unrecognized section indicator. Expecting RANGES, BOUNDS, QMATRIX, or ENDATA.

Description: An unrecognized MPS file section indicator occurred after the COLUMNS section of the MPS file.
CPXERR_BAD_SECTION_ENDATA

<table>
<thead>
<tr>
<th>Category</th>
<th>Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synopsis</td>
<td><code>CPXERR_BAD_SECTION_ENDATA()</code></td>
</tr>
<tr>
<td>Summary</td>
<td>1462 Line %d: Unrecognized section indicator. Expecting ENDATA.</td>
</tr>
<tr>
<td>Description</td>
<td>An unrecognized MPS file section indicator occurred after the COLUMNS section of the MPS file.</td>
</tr>
</tbody>
</table>
CPXERR_BAD_SECTION_QMATRIX

Category     Macro
Synopsis     CPXERR_BAD_SECTION_QMATRIX()
Summary     1475 Line %d: Unrecognized section indicator. Expecting QMATRIX or ENDATA.
Description  An unrecognized MPS file section indicator occurred after the RHS or BOUNDS section of the MPS file
**Category**  Macro  

**Synopsis**  CPXERR_BAD_SENSE ()  

**Summary**  1215 Illegal sense entry %d.  

**Description**  Legal sense symbols are L, G, E, and R.
CPXERR_BAD_SOS_TYPE

Category          Macro

Synopsis          CPXERR_BAD_SOS_TYPE()

Summary           1442 Line %d: Unrecognized SOS type: %c%c.

Description       Only SOS Types S1 or S2 can be specified within an SOS or MPS file.
CPXERR_BAD_STATUS

Category: Macro

Synopsis: CPXERR_BAD_STATUS()

Summary: 1253 Invalid status entry %d for basis specification.

Description: The basis status values are out of range.
<table>
<thead>
<tr>
<th>Category</th>
<th>Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Synopsis</strong></td>
<td><strong>CPXERR_BAS_FILE_SHORT</strong>()</td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td>1550 Basis missing some basic variables.</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Number of basic variables is less than the number of rows.</td>
</tr>
</tbody>
</table>
**CPXERR_BAS_FILE_SIZE**

**Category**  
Macro

**Synopsis**  
CPXERR_BAS_FILE_SIZE()

**Summary**  
1555 \( \%d \) \( \%s \) basic variable(s).

**Description**  
Number of basic variables doesn't match the problem. Check the CPXcopybase call.
CPXERR_BOUNDS_INFEAS

Category: Macro

Synopsis: CPXERR_BOUNDS_INFEAS()

Summary: 1100 Bounds on '%s' contradictory.

Description: Check upper and lower bounds for the specified variable indicated by that string of characters.
CPXERR_BOUNDS_INT

Category	Macro

Synopsis	CPXERR_BOUNDS_INT()

Summary	3011 Non-integral %s bound value %g for integer column '%s'.

Description	Check to be sure that bounds for integer variables have integer values.
**CPXERR_CALLBACK**

**Category**  
Macro

**Synopsis**  
`CPXERR_CALLBACK()`

**Summary**  
1006 Error during callback.

**Description**  
An error condition occurred during the callback, as, for example, when solving a MIP problem, if a callback asks for information that is not available from CPLEX.
CPXERR_CANT_CLOSE_CHILD

Category  Macro
Synopsis   CPXERR_CANT_CLOSE_CHILD()
Summary   1021 Cannot close a child environment.
Description  It is not permitted to call CPXcloseCPLEX  for a child environment.
## CPXERR_CHILD_OF_CHILD

<table>
<thead>
<tr>
<th>Category</th>
<th>Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Synopsis</strong></td>
<td>CPXERR_CHILD_OF_CHILD()</td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td>1019 Cannot clone a cloned environment.</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>CPXparenv cannot be called from a child thread.</td>
</tr>
</tbody>
</table>
CPXERR_COL_INDEX_RANGE

Category: Macro

Synopsis: CPXERR_COL_INDEX_RANGE()

Summary: 1201 Column index %d out of range.

Description: The specified column index is negative or greater than or equal to the number of columns in the currently loaded problem.
CPXERR_COL_REPEATS

Category: Macro

Synopsis: CPXERR_COL_REPEATS()

Summary: 1446 Column '%%s' repeats.

Description: The MPS file contains duplicate column entries. Inspect and edit the file.
**CPXERR_COL_REPEAT_PRINT**

**Category**  
Macro

**Synopsis**  
CPXERR_COL_REPEAT_PRINT()

**Summary**  
1478 %d Column repeats messages not printed.

**Description**  
The MPS problem or REV file contains duplicate column entries. Inspect and edit the file.
CPXERR_COL_ROW_REPEATS

Category: Macro

Synopsis: CPXERR_COL_ROW_REPEATS()

Summary: 1443 Column '%s' has repeated row '%s'.

Description: The indicated column appears more than once in a row. Check the MPS file for duplicate entries.
CPXERR_COL_UNKNOWN

Category       Macro

Synopsis       CPXERR_COL_UNKNOWN()

Summary        1449 Line %d: '%s' is not a column name.

Description    The MPS file specifies a column name that does not exist.
CPXERR_COUNT_OVERLAP

Category: Macro

Synopsis: CPXERR_COUNT_OVERLAP

Summary: 1228 Count entry %d indicates overlapping entries.

Description: Entries in the matcnt array indicate that the specified items overlap.
CPXERR_COUNT_RANGE

Category          Macro
Synopsis          CPXERR_COUNT_RANGE()
Summary           1227 Count entry %d negative or larger than allowed.
Description       Entries in matcnt arrays must be nonnegative or less than the number of items possible (columns or rows, for example).
CPXERR_DBL_MAX

Category          Macro
Synopsis           CPXERR_DBL_MAX()
Summary            1233 Numeric entry %d is larger than allowed maximum of %g.
Description        Data checking detected a number too large.
### CPXERR_DECOMPRESSION

**Category**  
Macro

**Synopsis**  
CPXERR_DECOMPRESSION()

**Summary**  
1027 Decompression of unpresolved model failed.

**Description**  
CPLEX was unable to restore the original model, due, for example, to insufficient memory.
CPXERR_DUP_ENTRY

Category     Macro
Synopsis     CPXERR_DUP_ENTRY()
Summary     1222 Duplicate entry.
Description   In CPXchgcoeflist, a duplicate (row, column) pair was found.
CPXERR_EXTRA_BV_BOUND

Category: Macro

Synopsis: CPXERR_EXTRA_BV_BOUND()

Summary: 1456 Line %d: 'BV' bound type illegal when prior bound given.

Description: Check the MPS file for bound values which conflict with this type specification.
CPXERR_EXTRA_FR_BOUND

Category          Macro

Synopsis          CPXERR_EXTRA_FR_BOUND ()

Summary           1455 Line %d: 'FR' bound type illegal when prior bound given.

Description       A column with an upper or lower bound previously assigned has an illegal FR bound assignment. Since the FR bound type has neither an upper nor lower bound, no other bound type can be specified. Check the MPS file.
Category: Macro

Synopsis: \texttt{CPXERR\_EXTRA\_FX\_BOUND ()}

Summary: 1454 Line \texttt{%d}: 'FX' bound type illegal when prior bound given.

Description: A column with either an upper or lower bound previously assigned has an illegal FX bound assignment. Since the FX bound type fixes both upper and lower bounds, no additional bounds can be specified. Check the MPS file.
**CPXERR_EXTRA_INTEND**

**Category**  
Macro

**Synopsis**  
CPXERR_EXTRA_INTEND()

**Summary**  
1481 Line %d: 'INTEND' found while not reading integers.

**Description**  
Integer markers are incorrectly positioned in the MPS file.
<table>
<thead>
<tr>
<th>Category</th>
<th>Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synopsis</td>
<td>CPXERR_EXTRA_INTORG()</td>
</tr>
<tr>
<td>Summary</td>
<td>1480 Line %d: 'INTORG' found while reading integers.</td>
</tr>
<tr>
<td>Description</td>
<td>Integer markers are incorrectly positioned in the MPS file.</td>
</tr>
</tbody>
</table>
CPXERR_EXTRA_SOSEND

Category: Macro

Synopsis: CPXERR_EXTRA_SOSEND()

Summary: 1483 Line %d: 'SOSEND' found while not reading a SOS.

Description: SOS markers are incorrectly positioned in the MPS file.
CPXERR_EXTRA_SOSORG

Category: Macro

Synopsis: CPXERR_EXTRA_SOSORG()

Summary: 1482 Line %d: 'SOSORG' found while reading a SOS.

Description: SOS markers are incorrectly positioned in the MPS file.
**CPXERR_FAIL_OPEN_READ**

**Category**  
Macro

**Synopsis**  
CPXERR_FAIL_OPEN_READ()

**Summary**  
1423 Could not open file '%s' for reading.

**Description**  
CPLEX could not read the specified file. Check the file specification.
<table>
<thead>
<tr>
<th>Category</th>
<th>Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synopsis</td>
<td><code>CPXERR_FAIL_OPEN_WRITE()</code></td>
</tr>
<tr>
<td>Summary</td>
<td>1422 Could not open file '%s' for writing.</td>
</tr>
<tr>
<td>Description</td>
<td>CPLEX could not create the specified file. Check the file specification.</td>
</tr>
</tbody>
</table>
CPXERR_FILE_ENTRIES

Category    Macro
Synopsis     CPXERR_FILE_ENTRIES()
Summary     1553 Line %d: Wrong number of entries.
Description The BAS or VEC file contains a line with too many entries.
CPXERR_FILE_FORMAT

Category		Macro

Synopsis	CPXERR_FILE_FORMAT()

Summary	1563 File '%s' has an incompatible format. Try setting reverse flag.

Description	When reading a binary file has been produced on a different computer system, reversing the setting of the byte order may allow reading.
CPXERR_IIS_DEFAULT

Category          Macro

Synopsis          CPXERR_IIS_DEFAULT()

Summary           1706 Unable to compute an IIS.

Description       The problem solved with an indication of numeric uncertainty. It may be feasible.
# CPXERR_IIS_FEAS

<table>
<thead>
<tr>
<th>Category</th>
<th>Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synopsis</td>
<td><code>CPXERR_IIS_FEAS()</code></td>
</tr>
<tr>
<td>Summary</td>
<td>1703 Problem is feasible; no IIS available.</td>
</tr>
<tr>
<td>Description</td>
<td>The problem solved with an indication of numeric uncertainty. It may be feasible.</td>
</tr>
</tbody>
</table>
CPXERR_IIS_NOT_INFEAS

**Category**
Macro

**Synopsis**
CPXERR_IIS_NOT_INFEAS()

**Summary**
1704 Problem may be feasible; no IIS available.

**Description**
The problem solved with an indication of numeric uncertainty. It may be feasible.
CPXERR_IIS_NO_BASIC

Category  Macro

Synopsis  CPXERR_IIS_NO_BASIC()

Summary  1707 Infeasibility Finder requires a basic solution.

Description  Use primal or dual simplex or crossover from a barrier solution first prior to applying the IIS algorithm.
CPXERR_IIS_NO_INFO

Category: Macro

Synopsis: CPXERR_IIS_NO_INFO()

Summary: 1701 No IIS information exists.

Description: The IIS algorithm found no IIS sets, so the routine CPXgetiis fails. Users of the Callable Library see this message if CPXgetiis or CPXdisplayiis is called before computing the IIS.
**CPXERR_IIS_NO_LOAD**

**Category**
Macro

**Synopsis**
CPXERR_IIS_NO_LOAD()

**Summary**
1709 Failed to load IIS subproblem.

**Description**
Most likely there is insufficient memory to perform the IIS analysis.
CPXERR_IIS_NO_SOLN

Category  Macro
Synopsis    CPXERR_IIS_NO_SOLN()
Summary    1702 No solution exists; infeasibility analysis not available.
Description The requested command cannot be executed because no solution exists for the problem. Optimize the problem first
**CPXERR_IIS_NUM_BEST**

**Category**          Macro

**Synopsis**          CPXERR_IIS_NUM_BEST()

**Summary**           1713 Unable to optimize IIS subproblem.

**Description**       The status of the subproblem optimization is unknown due to numeric uncertainty. Check the scaling of the problem.
CPXERR_IIS_OPT_INFEAS

Category             Macro

Synopsis             CPXERR_IIS_OPT_INFEAS()

Summary             1705 Problem optimal with unscaled infeasibilities. No IIS available.

Description         The problem solved with only unscaled infeasibilities. The IIS algorithm cannot be applied.
CPXERR_IIS_SUB_IT_LIM

Category: Macro

Synopsis: CPXERR_IIS_SUB_IT_LIM()

Summary: Iteration limit reached on IIS subproblem.

Description: During the solution of a subproblem, CPLEX reached the indicated limit.
**CPXERR_IIS_SUB_OBJ_LIM**

**Category**  
Macro

**Synopsis**  
CPXERR_IIS_SUB_OBJ_LIM

**Summary**  
1710 Objective limit reached on IIS subproblem.

**Description**  
During the solution of a subproblem, CPLEX reached the indicated limit.
<table>
<thead>
<tr>
<th>Category</th>
<th>Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synopsis</td>
<td>CPXERR_IIS_SUB_TIME_LIM()</td>
</tr>
<tr>
<td>Summary</td>
<td>1712 Time limit reached on IIS subproblem.</td>
</tr>
<tr>
<td>Description</td>
<td>During the solution of a subproblem, CPLEX reached the indicated limit.</td>
</tr>
</tbody>
</table>
CPXERR_INDEX_NOT_BASIC

Category  Macro

Synopsis  CPXERR_INDEX_NOT_BASIC()

Summary  1251 Index must correspond to a basic variable.

Description  The requested variable is not basic.
CPXERR_INDEX_RANGE

Category          Macro
Synopsis          CPXERR_INDEX_RANGE()
Summary           1200 Index is outside range of valid values.
Description       Selected index is too large or small.
CPXERR_INDEX_RANGE_HIGH

Category: Macro

Synopsis: CPXERR_INDEX_RANGE_HIGH()

Summary: 1206 %s: 'end' value %d is greater than %d.

Description: The index in the query routine is too large. The symbol %s represents a string, %d a number.
CPXERR_INDEX_RANGE_LOW

Category: Macro

Synopsis: CPXERR_INDEX_RANGE_LOW()

Summary: 1205: 'begin' value %d is less than %d.

Description: The index in the query routine is too small. The symbol %s represents a string, %d a number.
**CPXERR_INT_TOO_BIG**

**Category**  
Macro

**Synopsis**  
CPXERR_INT_TOO_BIG()

**Summary**  
3018 Magnitude of variable %s: %g exceeds integer limit %d.

**Description**  
CPXmipopt tried to branch on the indicated integer variable at a value larger than representable in the branch & cut tree. Check the model formulation.
CPXERR_INT_TOO_BIG_INPUT

Category: Macro

Synopsis: CPXERR_INT_TOO_BIG_INPUT()

Summary: 1463 Line %d: Magnitude exceeds integer limit %d.

Description: A number has been read that is greater than the largest integer value that can be represented by the computer.
CPXERR_INVALID_NUMBER

Category       Macro

Synopsis       CPXERR_INVALID_NUMBER()

Summary        1650 Number not representable in exponential notation.

Description    The number to be printed is not representable.
CPXERR_LIMITS_TOO_BIG

Category: Macro

Synopsis: CPXERR_LIMITS_TOO_BIG()

Summary: 1012 Problem size limits too large.

Description: One of the problem dimensions or read limits requires an array length beyond the architectural maximum of the computer.
**Category**
Macro

**Synopsis**
CPXERR_LO_BOUND_REPEATS ()

**Summary**
1459 Line %d: Repeated lower bound.

**Description**
The lower bound for a column is repeated within the problem file on the indicated line. Two individual lower bounds could exist. Alternatively, an MI bound and individual lower bound could be in conflict. Check the MPS file.
CPXERR_MISS_SOS_TYPE

Category: Macro

Synopsis: CPXERR_MISS_SOS_TYPE()

Summary: 3301 Line %d: Missing SOS type.

Description: An SOS type has not been specified.
**CPXERR_MSG_NO_CHANNEL**

**Category**    Macro

**Synopsis**    

**Summary**  1051 No channel pointer supplied to message routine.

**Description**  The message routine needs a pointer to a channel.
CPXERR_MSG_NO_FILEPTR

Category       Macro
Synopsis        CPXERR_MSG_NO_FILEPTR()
Summary        1052 No file pointer found for message routine.
Description    The message routine needs a pointer to a file.
CPXERR_MSG_NO_FUNCTION

Category: Macro

Synopsis: CPXERR_MSG_NO_FUNCTION()

Summary: 1053 No function pointer found for message routine.

Description: The message routine needs a pointer to a function.
CPXERR_NAME_CREATION

Category: Macro

Synopsis: CPXERR_NAME_CREATION()

Summary: 1209 Unable to create default names.

Description: The current names of rows or columns don’t allow the creation of default names.
CPXERR_NAME_NOT_FOUND

Category        Macro
Synopsis        CPXERR_NAME_NOT_FOUND ()
Summary         1210 Name not found.
Description     Name does not exist. Check the arguments of CPXgetcolindex or CPXgetrowindex.
CPXERR_NAME_TOO_LONG

Category: Macro

Synopsis: CPXERR_NAME_TOO_LONG()

Summary: 1464 Line %d: Identifier/name too long to process.

Description: The length of the identifier or name was beyond the size CPLEX can process.
CPXERR_NAN

Category            Macro
Synopsis             CPXERR_NAN ()
Summary              1225 Numeric entry %d is not a double precision number (NAN).
Description          The value is not a number.
CPXERR_NEED_OPT_SOLN

Category: Macro

Synopsis: CPXERR_NEED_OPT_SOLN()

Summary: 1252 Optimal solution required.

Description: An optimal solution must exist before the requested operation can be performed.
CPXERR_NEGATIVE_SURPLUS

Category  Macro

Synopsis  CPXERR_NEGATIVE_SURPLUS()

Summary  1207 Insufficient array length.

Description  The array is too short to hold the requested data.
CPXERR_NET_DATA

Category       Macro

Synopsis       CPXERR_NET_DATA()

Summary        1530 Inconsistent network file.

Description    Check the NET format file for errors.
CPXERR_NET_FILE_SHORT

Category: Macro

Synopsis: CPXERR_NET_FILE_SHORT()

Summary: 1538 Unexpected end of network file.

Description: Check the NET format file for errors.
CPXERR_NODE_INDEX_RANGE

Category  
Macro

Synopsis  
CPXERR_NODE_INDEX_RANGE ()

Summary  
1230 Node index %d out of range.

Description  
The specified node index is negative or greater than or equal to the number of nodes in the network.
CPXERR_NODE_ON_DISK

Category: Macro

Synopsis: CPXERR_NODE_ON_DISK()

Summary: 3504 No callback info on disk/compressed nodes.

Description: Information about nodes stored in node files is not available through the advanced callback functions.
CPXERR_NOT_DUAL_UNBOUNDED

Category: Macro

Synopsis: CPXERR_NOT_DUAL_UNBOUNDED()

Summary: 1265 Dual unbounded solution required.

Description: The called function requires that the LP stored in the problem object has been determined to be primal infeasible by the dual simplex algorithm.
CPXERR_NOT_FIXED

Category: Macro

Synopsis: CPXERR_NOT_FIXED()

Summary: 1221 Only fixed variables are pivoted out.

Description: CPXpivotout can pivot out only fixed variables.
CPXERR_NOT_FOR_MIP

Category: Macro

Synopsis: CPXERR_NOT_FOR_MIP()

Summary: 1017 Not available for mixed-integer problems.

Description: The requested operation cannot be performed for mixed integer programs. Change the problem type.
CPXERR_NOT_FOR_QCP

Category: Macro

Synopsis: CPXERR_NOT_FOR_QCP()

Summary: 1031 Not available for QCP.

Description: Function is not available for quadratically constrained problems.
CPXERR_NOT_FOR_QP

Category  Macro
Synopsis    CPXERR_NOT_FOR_QP
Summary    1018 Not available for quadratic programs.
Description The requested operation can not be performed for quadratic programs. Change the problem type.
CPXERR_NOT_MILPCLASS

Category    Macro
Synopsis         CPXERR_NOT_MILPCLASS()
Summary  1024 Not a MILP, relaxed MILP, or fixed MILP.
Description   Function requires that problem type must be CPXPROB_MILP, CPXPROB_RELAXEDMILP, or CPXPROB_FIXEDMILP.
<table>
<thead>
<tr>
<th><strong>Category</strong></th>
<th>Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Synopsis</strong></td>
<td><code>CPXERR_NOT_MIN_COST_FLOW()</code></td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td>1531 Not a min-cost flow problem.</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Check the MIN format file for errors.</td>
</tr>
</tbody>
</table>
CPXERR_NOT_MIP

Category  Macro
Synopsis   CPXERR_NOT_MIP()
Summary   3003 Not a mixed-integer problem.
Description The requested operation can only be performed on a mixed integer problem.
CPXERR_NOT_MIQPCCLASS

CategoryMacro

SynopsisCPXERR_NOT_MIQPCCLASS()

Summary1029 Not a MIQP or fixed MIQP.

DescriptionFunction requires that problem type be CPXPROB_MIQP or CPXPROB_FIXEDMIQP (that is, it has a quadratic objective).
CPXERR_NOT_ONE_PROBLEM

Category  
Macro

Synopsis  
CPXERR_NOT_ONE_PROBLEM()

Summary  
1023 Not a single problem, relaxed or fixed.

Description  
No problem available, or problem is relaxed or fixed, and the operation is inappropriate for those types of problems.
CPXERR_NOT_QP

Category       Macro
Synopsis       CPXERR_NOT_QP()
Summary        5004 Not a quadratic program.
Description    The requested operation can be performed only on a quadratic problem.
CPXERR_NOT_QPCLASS

Category Macro

Synopsis CPXERR_NOT_QPCLASS()

Summary 1025 Not a QP or zeroed QP.

Description Function requires that QP data exist for the problem passed in.
CPXERR_NOT_SAV_FILE

Category: Macro

Synopsis: CPXERR_NOT_SAV_FILE()

Summary: 1560 File '%s' is not a SAV file.

Description: The selected file does not match the type specified.
CPXERR_NOT_UNBOUNDED

Category          Macro

Synopsis          CPXERR_NOT_UNBOUNDED()

Summary           1254 Unbounded solution required.

Description       The requested operation can be performed only on a problem determined to be unbounded.
CPXERR_NO_BARRIER_SOLN

Category  Macro
Synopsis    CPXERR_NO_BARRIER_SOLN()
Summary  1223 No barrier solution exists.
Description  The requested operation requires the existence of a barrier solution.
CPXERR_NO_BASIC_SOLN

Category: Macro

Synopsis: CPXERR_NO_BASIC_SOLN()

Summary: 1261 No basic solution exists.

Description: The requested operation requires the existence of a basic solution. Apply primal or dual simplex or crossover.
CPXERR_NO_BASIS

Category               Macro
Synopsis               CPXERR_NO_BASIS()
Summary                1262 No basis exists.
Description            The requested operation requires the existence of a basis.
CPXERR_NO_BOUND_SENSE

Category       Macro
Synopsis       CPXERR_NO_BOUND_SENSE ()
Summary        1621 Line %d: No bound sense.
Description    The sense indicator is missing from the indicated line.
CPXERR_NO_BOUND_TYPE

Category Macro

Synopsis CPXERR_NO_BOUND_TYPE()

Summary 1460 Line %d: Bound type missing.

Description No bound type could be found for the indicated column bound on the indicated line. Check the MPS file.
CPXERR_NO_COLUMNS_SECTION

Category: Macro

Synopsis: CPXERR_NO_COLUMNS_SECTION()

Summary: 1472 Line %d: No COLUMNS section.

Description: The required COLUMNS section is missing from the MPS file. Check the file.
CPXERR_NO_DUAL_SOLN

Category: Macro

Synopsis: CPXERR_NO_DUAL_SOLN()

Summary: 1232 No dual solution exists.

Description: There is no dual solution available, so there is no quality information about the dual either.
**CPXERR_NO_ENDATA**

**Category**  
Macro

**Synopsis**  
CPXERR_NO_ENDATA()

**Summary**  
1552 ENDATA missing.

**Description**  
BAS files must have an ENDATA record as the last line of the file.
CPXERR_NO_ENVIRONMENT

Category  Macro

Synopsis    CPXERR_NO_ENVIRONMENT()

Summary    1002 No environment.

Description    Be sure to pass a valid environment pointer to the routines.
CPXERR_NO_FILENAME

Category  Macro
Synopsis    CPXERR_NO_FILENAME
Summary    1421 File name not specified.
Description  A filename must be specified for the requested operation to succeed.
CPXERR_NO_ID

Category  Macro

Synopsis  CPXERR_NO_ID()

Summary  1616 Line %d: Expected identifier, found '%c'.

Description  Instead of the expected identifier CPLEX found the character shown in the error message.
CPXERR_NO_ID_FIRST

Category: Macro

Synopsis: CPXERR_NO_ID_FIRST()

Summary: 1609 Line %d: Expected identifier first.

Description: A variable name is missing on the indicated line.
CPXERR_NO_INT_SOLN

Category: Macro

Synopsis: CPXERR_NO_INT_SOLN()

Summary: 3017 No integer feasible solution exists.

Description: An integer feasible solution must exist before the requested operation can be performed.
CPXERR_NO_INT_X

Category: Macro

Synopsis: CPXERR_NO_INT_X()

Summary: 3023 Integer feasible solution values are unavailable.

Description: When the incumbent for the problem has been provided by a MIP Start or by an advanced callback function working on the original problem, the incumbent solution values are not available for the reduced problem.
CPXERR_NO_LU_FACTOR

Category: Macro

Synopsis: CPXERR_NO_LU_FACTOR()

Summary: 1258 No LU factorization exists.

Description: The requested item requires the presence of factoring. You may need to optimize with a 0 iteration limit to factor.
CPXERR_NO_MEMORY

Category Macro

Synopsis CPXERR_NO_MEMORY()

Summary 1001 Out of memory.

Description The computer has insufficient memory available to complete the selected operation. Downsize problem or increase the amount of physical memory available. Depending on the command, several memory-conserving corrections can be made.
CPXERR_NO_MIPSTART

Category　　Macro

Synopsis　　CPXERR_NO_MIPSTART()

Summary　　3020 No MIP start exists.

Description　CPXgetmipstart failed because no MIP start data is available for the problem.
## CPXERR_NO_NAMES

<table>
<thead>
<tr>
<th>Category</th>
<th>Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synopsis</td>
<td><code>CPXERR_NO_NAMES()</code></td>
</tr>
<tr>
<td>Summary</td>
<td>1219 No names exist.</td>
</tr>
<tr>
<td>Description</td>
<td>The requested operation is successful only if names have been assigned. Typically, this failure occurs when a file is being read, such as an ORD file, when no names were assigned during the prior call to CPXreadcopyprob.</td>
</tr>
</tbody>
</table>
CPXERR_NO_NAME_SECTION

Category  Macro
Synopsis   CPXERR_NO_NAME_SECTION()
Summary   1441 Line %d: No NAME section.
Description The NAME section required in an MPS file is missing.
CPXERR_NO_NORMS

Category       Macro

Synopsis       CPXERR_NO_NORMS()

Summary        1264 No norms available.

Description    Norms are not present. Change pricing, and call the optimization routine.
CPXERR_NO_NUMBER

Category        Macro

Synopsis        CPXERR_NO_NUMBER()

Summary         1615 Line %d: Expected number, found '%c'.

Description     Some character other than a number, as required, appears on the indicated line.
CPXERR_NO_NUMBER_BOUND

Category          Macro

Synopsis          CPXERR_NO_NUMBER_BOUND(

Summary           1623 Line %d: Missing bound number.

Description       The bound data is missing from the LP file.  CPLEX expected a number where no number was found.
CPXERR_NO_NUMBER_FIRST

Category  Macro

Synopsis  CPXERR_NO_NUMBER_FIRST()

Summary  1611 Line %d: Expected number first.

Description  Some character other than a number, as required, appears on the indicated line.
**CPXERR_NO_OBJECTIVE**

**Category**  
Macro

**Synopsis**  
`CPXERR_NO_OBJECTIVE()`

**Summary**  
1476 Line %d: No objective row found.

**Description**  
No free row was found in the MPS file. Check the file. At least one free row must be present. Free rows have an N sense beginning in column 2.
Category: Macro

Synopsis: CPXERR_NO_OBJSENSE()

Summary: 1436 Max or Min missing.

Description: The indicator of the sense of the objective function is missing from the LP file. No problem has been read as a consequence.
CPXERR_NO_OPERATOR

Category          Macro
Synopsis          CPXERR_NO_OPERATOR()
Summary           1607 Line %d: Expected '+' or '-', found '%c'.
Description       Some character other than + or - appears between variable names on the indicated line.
CPXERR_NO_OP_OR_SENSE

Category: Macro

Synopsis: CPXERR_NO_OP_OR_SENSE()

Summary: 1608 Line %d: Expected '+', '-' or sense, found '%c'.

Description: Some character other than a + or - operator, as required, appears on the indicated line.
CPXERR_NO_ORDER

Category       Macro

Synopsis       CPXERR_NO_ORDER()

Summary        3016 No priority order exists.

Description    The requested command cannot be executed because no priority order has been loaded.
## CPXERR_NO_PROBLEM

<table>
<thead>
<tr>
<th>Category</th>
<th>Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Synopsis</strong></td>
<td><code>CPXERR_NO_PROBLEM()</code></td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td>1009 No problem exists.</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>The requested command cannot be executed because no problem has been loaded.</td>
</tr>
</tbody>
</table>
CPXERR_NO_QMATRIX_SECTION

Category: Macro

Synopsis: CPXERR_NO_QMATRIX_SECTION()

Summary: Line %d: No QMATRIX section.

Description: The required QMATRIX section for quadratic programs is missing from the QP file. Check the file.
**CPXERR_NO_QP_OPERATOR**

**Category**
Macro

**Synopsis**
CPXERR_NO_QP_OPERATOR()

**Summary**
1614 Line %d: Expected ^ or *.

**Description**
The ^ or * operator is missing from the QP term.
CPXERR_NO_QUAD_EXP

Category         Macro

Synopsis     CPXERR_NO_QUAD_EXP()

Summary         1612 Line %d: Expected quadratic exponent.

Description      An exponent of 2 is expected after the ^ operator.
CPXERR_NO_RHS_COEFF

Category: Macro

Synopsis: CPXERR_NO_RHS_COEFF()

Summary: 1610 Line %d: Expected RHS coefficient.

Description: No RHS coefficient is present after the sense indicator on the indicated line.
CPXERR_NO_RHS_IN_OBJ

Category  
Synopsis  
Summary  
Description  
You cannot make changes to the right-hand side of an objective row because no coefficients exist.
## CPXERR_NO_RNGVAL

**Category**  
Macro

**Synopsis**  
CPXERR_NO_RNGVAL()

**Summary**  
1216 No range values.

**Description**  
No ranges exist for this problem.
CPXERR_NO_ROWS_SECTION

Category: Macro

Synopsis: CPXERR_NO_ROWS_SECTION()

Summary: 1471 Line %d: No ROWS section.

Description: No ROW section was found in the MPS file.
CPXERR_NO_ROW_NAME

Category		Macro

Synopsis		CPXERR_NO_ROW_NAME()

Summary		1486 Line %d: No row name.

Description	A row name is missing within the ROWS section.
CPXERR_NO_ROWSENSE

Category: Macro

Synopsis: CPXERR_NO_ROWSENSE()

Summary: 1453 Line %d: No row sense.

Description: No sense indicator was found on the indicated line.
CPXERR_NO_SENSIT

Category: Macro

Synopsis: CPXERR_NO_SENSIT()

Summary: 1260 Sensitivity analysis not available for current status.

Description: Sensitivity information is not available because an optimal basic solution does not exist for the currently loaded problem. Optimize the problem and check to make sure that it is not infeasible or unbounded.
## CPXERR_NO_SOLN

**Category**  
Macro

**Synopsis**  
CPXERR_NO_SOLN()

**Summary**  
1217 No solution exists.

**Description**  
The requested command cannot be executed because no solution exists for the problem. Optimize the problem first.
CPXERR_NO_SOS

<table>
<thead>
<tr>
<th>Category</th>
<th>Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synopsis</td>
<td>CPXERR_NO_SOS()</td>
</tr>
<tr>
<td>Summary</td>
<td>3015 No user-defined SOSs exist.</td>
</tr>
<tr>
<td>Description</td>
<td>SOS information can be written to a file only if the SOS has already been defined. SOS Type 3 information (found by the SOSSCAN feature) cannot be written to an SOS file.</td>
</tr>
</tbody>
</table>
### CPXERR_NO_SOS_SEPARATOR

<table>
<thead>
<tr>
<th>Category</th>
<th>Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synopsis</td>
<td>CPXERR_NO_SOS_SEPARATOR ()</td>
</tr>
<tr>
<td>Summary</td>
<td>1627 Expected ':', found '%c'.</td>
</tr>
<tr>
<td>Description</td>
<td>The separator :: must follow the S1 or S2 declaration.</td>
</tr>
</tbody>
</table>
**CPXERR_NO_TREE**

**Category**: Macro

**Synopsis**: `CPXERR_NO_TREE()`

**Summary**: 3412 Current problem has no tree.

**Description**: No tree exists until after the mixed integer optimization has begun.
CPXERR_NO_VECTOR_SOLN

Category       Macro

Synopsis   CPXERR_NO_VECTOR_SOLN()

Summary   1556 Vector solution does not exist.

Description   CPLEX could not write VEC file because no vector solution is available.
<table>
<thead>
<tr>
<th>Category</th>
<th>Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synopsis</td>
<td><code>CPXERR_NULL_NAME</code>()</td>
</tr>
<tr>
<td>Summary</td>
<td>1224 Null pointer %d in name array.</td>
</tr>
<tr>
<td>Description</td>
<td>Null pointers are not allowed in name arrays.</td>
</tr>
</tbody>
</table>
CPXERR_NULL_POINTER

Category       Macro
Synopsis        CPXERR_NULL_POINTER()
Summary         1004 Null pointer for required data.
Description     A value of NULL was passed to a routine where NULL is not allowed.
CPXERR_ORDER_BAD_DIRECTION

Category: Macro

Synopsis: CPXERR_ORDER_BAD_DIRECTION()

Summary: 3007 Illegal direction entry %d.

Description: Legal direction entries are limited to the values CPX_BRANCH_GLOBAL, CPX_BRANCH_DOWN, and CPX_BRANCH_UP.
CPXERR_PARAM_TOO_BIG

Category: Macro

Synopsis: CPXERR_PARAM_TOO_BIG()

Summary: 1015 Parameter value too big.

Description: The value of the CPLEX parameter is outside the range of possible settings.
## CPXERR_PARAM_TOO_SMALL

<table>
<thead>
<tr>
<th>Category</th>
<th>Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synopsis</td>
<td>CPXERR_PARAM_TOO_SMALL()</td>
</tr>
<tr>
<td>Summary</td>
<td>1014 Parameter value too small.</td>
</tr>
<tr>
<td>Description</td>
<td>The value of the CPLEX parameter is outside the range of possible settings.</td>
</tr>
</tbody>
</table>
CPXERR_PRESLV_ABORT

Category: Macro

Synopsis: CPXERR_PRESLV_ABORT()

Summary: 1106 Aborted during presolve.

Description: The user halted preprocessing by means of a callback.
CPXERR_PRESLV_BAD_PARAM

Category     Macro
Synopsis     CPXERR_PRESLV_BAD_PARAM()
Summary     1122 Bad presolve parameter setting.
Description  Dual presolve reductions (CPX_PARAM_REDUCE) were specified in the presence of lazy constraints, or nonlinear reductions (CPX_PARAM_PRELINEAR) were specified in the presence of user cuts.
**CPXERR_PRESLV_BASIS_MEM**

**Category**  
Macro

**Synopsis**  
CPXERR_PRESLV_BASIS_MEM()

**Summary**  
1107 Not enough memory to build basis for original LP.

**Description**  
Insufficient memory exists to complete the uncrushing of the presolved problem.
**CPXERR_PRESLV_COPYORDER**

**Category**  
Macro

**Synopsis**  
CPXERR_PRESLV_COPYORDER()

**Summary**  
1109 Can't copy priority order info from original MIP.

**Description**  
The CPLEX call to CPXcopyorder failed.
## CPXERR_PRESLV_COPYSOSS

<table>
<thead>
<tr>
<th>Category</th>
<th>Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synopsis</td>
<td>CPXERR_PRESLV_COPYSOSS ()</td>
</tr>
<tr>
<td>Summary</td>
<td>1108 Can't copy SOS info from original MIP.</td>
</tr>
<tr>
<td>Description</td>
<td>The CPLEX call to CPXcopySOS failed.</td>
</tr>
</tbody>
</table>
CPXERR_PRESLV_CRUSHFORM

Category: Macro

Synopsis: CPXERR_PRESLV_CRUSHFORM()

Summary: 1121 Can't crush solution form.

Description: Presolve could not reduce the solution.
## CPXERR_PRESLV_DUAL

<table>
<thead>
<tr>
<th>Category</th>
<th>Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synopsis</td>
<td>CPXERR_PRESLV_DUAL()</td>
</tr>
<tr>
<td>Summary</td>
<td>1119 The feature is not available for solving dual formulation.</td>
</tr>
<tr>
<td>Description</td>
<td>Certain presolve features are not compatible with its creating an explicit dual formulation.</td>
</tr>
</tbody>
</table>
CPXERR_PRESLV_FAIL_BASIS

Category  Macro
Synopsis  CPXERR_PRESLV_FAIL_BASIS()
Summary  1114 Could not load unpresolved basis for original LP.
Description  Most likely insufficient memory exists to complete the uncrushing of the presolved problem.
**CPXERR_PRESLV_INF**

**Category**  Macro

**Synopsis**  \texttt{CPXERR_PRESLV_INF()}  

**Summary**  1117 Presolve determines problem is infeasible.

**Description**  The loaded problem contains blatant infeasibilities.
CPXERR_PRESLV_INFOrUNBD

Category: Macro

Synopsis: CPXERR_PRESLV_INFOrUNBD()

Summary: 1101 Presolve determines problem is infeasible or unbounded.

Description: The loaded problem contains blatant infeasibilities or unboundedness.
CPXERR_PRESLV_NO_BASIS

Category  Macro

Synopsis  CPXERR_PRESLV_NO_BASIS()

Summary  1115 Failed to find basis in presolved LP.

Description  A basis could not be recovered during uncrushing, most likely due to lack of memory.
CPXERR_PRESLV_NO_PROB

Category       Macro
Synopsis        CPXERR_PRESLV_NO_PROB ()
Summary        1103 No presolved problem created.
Description    Most likely insufficient memory exists to complete the loading of the presolved problem.
CPXERR_PRESLV_SOLN_MIP

Category  Macro

Synopsis  CPXERR_PRESLV_SOLN_MIP()

Summary  1110 Not enough memory to recover solution for original MIP.

Description  Most likely insufficient memory exists to complete the uncrushing of the presolved problem.
## CPXERR_PRESLV_SOLN_QP

**Category**  
Macro

**Synopsis**  
`CPXERR_PRESLV_SOLN_QP ()`

**Summary**  
1111 Not enough memory to compute solution to original QP.

**Description**  
Most likely insufficient memory exists to complete the uncrushing of the presolved problem.
**CPXERR_PRESLV_START_LP**

**Category**  
Macro

**Synopsis**  
`CPXERR_PRESLV_START_LP()`

**Summary**  
1112 Not enough memory to build start for original LP.

**Description**  
Most likely insufficient memory exists to complete the uncrushing of the presolved problem.
CPXERR_PRESLV_UNBD

Category       Macro
Synopsis        CPXERR_PRESLV_UNBD()
Summary        1118 Presolve determines problem is unbounded.
Description    The loaded problem contains blatant unboundedness.
CPXERR_PRESLV_UNCRUSHFORM

Category: Macro

Synopsis: CPXERR_PRESLV_UNCRUSHFORM()

Summary: 1120 Can't uncrush solution form.

Description: Presolve could not create a full solution.
CPXERR_PRIIND

Category       Macro
Synopsis       CPXERR_PRIIND()
Summary        1257 Incorrect usage of pricing indicator.
Description    The value of the pricing indicator is out of range.
**Category**  
Macro

**Synopsis**  
`CPXERR_PRM_DATA(1)`

**Summary**  
1660 Line %d: Not enough entries.

**Description**  
There were illegal or missing values in a parameter file (.prm).
<table>
<thead>
<tr>
<th>Category</th>
<th>Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synopsis</td>
<td>CPXERR_PRM_HEADER()</td>
</tr>
<tr>
<td>Summary</td>
<td>1661 Line %d: Missing or invalid header.</td>
</tr>
<tr>
<td>Description</td>
<td>Illegal or missing version number in the header of a parameter file (.prm).</td>
</tr>
</tbody>
</table>
CPXERR_PTHREAD_CREATE

Category      Macro
Synopsis       CPXERR_PTHREAD_CREATE ()
Summary        3603 Could not create thread.
Description    An error occurred during a system call needed to initialize parallel MIP.
**CPXERR_PTHREAD_MUTEX_INIT**

**Category**  
Macro

**Synopsis**  
CPXERR_PTHREAD_MUTEX_INIT()

**Summary**  
3601 Could not initialize mutex.

**Description**  
An error occurred during a system call needed to initialize parallel MIP.
CPXERR_QCP_SENSE_FILE

Category: Macro

Synopsis: CPXERR_QCP_SENSE_FILE()

Summary: 1437 Line %d: Equality sense is not allowed for quadratic constraints.

Description: LP reader does not allow equality in quadratic constraints; MPS file format does not allow equality in quadratic constraints.
**CPXERR_QUAD_EXP_NOT_2**

<table>
<thead>
<tr>
<th>Category</th>
<th>Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synopsis</td>
<td>CPXERR_QUAD_EXP_NOT_2 ()</td>
</tr>
<tr>
<td>Summary</td>
<td>1613 Line %d: Quadratic exponent must be 2.</td>
</tr>
<tr>
<td>Description</td>
<td>Only an exponent of 2 is allowed after the exponentiation operator ^.</td>
</tr>
</tbody>
</table>
null
Category: Macro

Synopsis: `CPXERR_Q_DIVISOR()`

Summary: 1619 Line %d: Missing or incorrect divisor for Q terms.

Description: Quadratic terms must be enclosed in square brackets and followed by a division sign with the divisor 2, that is, [ ]/2.
## CPXERR_Q_DUP_ENTRY

<table>
<thead>
<tr>
<th>Category</th>
<th>Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synopsis</td>
<td>CPXERR_Q_DUP_ENTRY()</td>
</tr>
<tr>
<td>Summary</td>
<td>5011 Duplicate entry for pair '%s' and '%s'.</td>
</tr>
<tr>
<td>Description</td>
<td>There are duplicate entries for the quadratic term.</td>
</tr>
</tbody>
</table>
CPXERR_Q_NOT_INDEF

Category            Macro
Synopsis             CPXERR_Q_NOT_INDEF ()
Summary              5014 Q is not indefinite.
Description          Function requires that the Q matrix be indefinite.
CPXERR_Q_NOT_POS_DEF

Category: Macro

Synopsis: CPXERR_Q_NOT_POS_DEF()

Summary: 5002 Q is not positive semi-definite.

Description: The Q matrix must be positive semi-definite (for minimizations) to apply the barrier optimizer. Check the quadratic term.
**CPXERR_Q_NOT_SYMMETRIC**

**Category**  
Macro

**Synopsis**  
`CPXERR_Q_NOT_SYMMETRIC()`

**Summary**  
5012 Q is not symmetric.

**Description**  
The Q matrix must be symmetric. Check off-diagonal elements. Look for either a missing or superfluous element.
<table>
<thead>
<tr>
<th>Category</th>
<th>Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synopsis</td>
<td>CPXERR_RANGE_SECTION_ORDER()</td>
</tr>
<tr>
<td>Summary</td>
<td>1474 Line %d: 'RANGES' section out of order.</td>
</tr>
<tr>
<td>Description</td>
<td>The RANGES section can appear only after the RHS section in an MPS file.</td>
</tr>
</tbody>
</table>
CPXERR_RESTRICTED_VERSION

Category       Macro

Synopsis       CPXERR_RESTRICTED_VERSION()

Summary        1016 Promotional version. Problem size limits exceeded.

Description    The current problem is too large for your version of CPLEX. Reduce the size of the problem.
**CPXERR_RHS_IN_OBJ**

<table>
<thead>
<tr>
<th>Category</th>
<th>Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synopsis</td>
<td>CPXERR_RHS_IN_OBJ()</td>
</tr>
<tr>
<td>Summary</td>
<td>1603 Line %d: RHS sense in objective.</td>
</tr>
<tr>
<td>Description</td>
<td>The objective row erroneously includes a sense indicator.</td>
</tr>
</tbody>
</table>
CPXERR_RIMNZ_REPEATS

Category  Macro
Synopsis  CPXERR_RIMNZ_REPEATS()
Summary  1479 Line %d: %s %s repeats.
Description  The MPS file contains duplicate entries in an extra rim vector.
CPXERR_RIM_REPEATS

Category: Macro

Synopsis: CPXERR_RIM_REPEATS()

Summary: 1447 Line %d: %s '%s' repeats.

Description: The MPS file contains duplicate names.
**CPXERR_RIM_ROW_REPEATS**

**Category**  
Macro

**Synopsis**  
`CPXERR_RIM_ROW_REPEATS()`

**Summary**  
1444 %s '%s' has repeated row '%s'.

**Description**  
The MPS file contains duplicate row names.
CPXERR_ROW_INDEX_RANGE

Category Macro

Synopsis CPXERR_ROW_INDEX_RANGE()

Summary 1203 Row index %d out of range.

Description The specified row index is negative or greater than or equal to the number of rows in the currently loaded problem.
CPXERR_ROW_REPEATS

Category  Macro
Synopsis  CPXERR_ROW_REPEATS()
Summary  1445 Row '%s' repeats.
Description  The MPS file contains duplicate row entries. Inspect and edit the file.
**CPXERR_ROW_REPEAT_PRINT**

<table>
<thead>
<tr>
<th>Category</th>
<th>Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synopsis</td>
<td><code>CPXERR_ROW_REPEAT_PRINT()</code></td>
</tr>
<tr>
<td>Summary</td>
<td>1477 %d Row repeats messages not printed.</td>
</tr>
<tr>
<td>Description</td>
<td>The MPS problem or REV file contains duplicate row entries. Inspect and edit the file.</td>
</tr>
</tbody>
</table>
CPXERR_ROW_UNKNOWN

Category: Macro

Synopsis: CPXERR_ROW_UNKNOWN

Summary: 1448 Line %d: '%s' is not a row name.

Description: The MPS file specifies a row name that does not exist.
CPXERR_SAV_FILE_DATA

Category: Macro

Synopsis: CPXERR_SAV_FILE_DATA()

Summary: 1561 Not enough data in SAV file.

Description: The file is corrupted or was generated by an incompatible version of the software.
CPXERR_SAV_FILE_WRITE

Category: Macro

Synopsis: CPXERR_SAV_FILE_WRITE()

Summary: 1562 Unable to write SAV file to disk.

Description: CPLEX could not open or write to the requested SAV file. Check the file designation and disk space.
CPXERR_SBASE_ILLEGAL

Category  Macro
Synopsis  CPXERR_SBASE_ILLEGAL()
Summary  1554 Superbases are not allowed.
Description  Basis or restart file contains superbasis that cannot be read.
CPXERR_SBASE_INCOMPAT

Category: Macro

Synopsis: CPXERR_SBASE_INCOMPAT()

Summary: 1255 Incompatible with superbasis.

Description: The requested operation is incompatible with an existing superbasis.
**CPXERR_SEMI_BDS**

**Category**          Macro

**Synopsis**          CPXERR_SEMI_BDS()

**Summary**           3022 Infinite bds on semi-continuous or semi-integer variable \%d.

**Description**       Semi-continuous or semi-integer variables must have finite bounds.
<table>
<thead>
<tr>
<th><strong>Category</strong></th>
<th>Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Synopsis</strong></td>
<td><code>CPXERR_SINGULAR()</code></td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td>1256 Basis singular.</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>CPLEX cannot factor a singular basis. See the discussion of numeric difficulties in the ILOG CPLEX User's Manual.</td>
</tr>
</tbody>
</table>
CPXERR_STR_PARAM_TOO_LONG

Category: Macro

Synopsis: CPXERR_STR_PARAM_TOO_LONG()

Summary: 1026 String parameter is too long.

Description: Length of the string was greater than 510.
CPXERR_SUBPROB_SOLVE

Category: Macro

Synopsis: CPXERR_SUBPROB_SOLVE()

Summary: 3019 Failure to solve MIP subproblem.

Description: CPXmipopt failed to solve one of the subproblems in the branch & cut tree. This failure can be due to a limit (for example, an iteration limit) or due to numeric trouble. Check the log, or add a call to CPXgetsubstat in the Callable Library) for information about the cause.
**CPXERR_TILIM_CONDITION_NO**

**Category**       Macro

**Synopsis**       CPXERR_TILIM_CONDITION_NO

**Summary**        1268 Time limit reached in computing condition number.

**Description**    Condition number computation was not completed due to a time limit.
CPXERR_TILIM_STRONGBRANCH

Category       Macro
Synopsis       CPXERR_TILIM_STRONGBRANCH
Summary        1266 Time limit reached in strong branching.
Description    Strong branching was not completed due to a time limit.
CPXERR_TOO_MANY_COEFFS

Category: Macro

Synopsis: CPXERR_TOO_MANY_COEFFS()

Summary: 1433 Too many coefficients.

Description: The problem contains more matrix coefficients than are allowed.
CPXERR_TOO_MANY_COLS

Category: Macro

Synopsis: CPXERR_TOO_MANY_COLS()

Summary: 1432 Too many columns.

Description: The problem contains more columns than are allowed.
CPXERR_TOO_MANY_RIMNZ

Category                  Macro
Synopsis                  CPXERR_TOO_MANY_RIMNZ ()
Summary                   1485 Too many rim nonzeros.
Description               Reset the rim vector nonzero read limit to a larger number.
## CPXERR_TOO_MANY_RIMS

**Category**
Macro

**Synopsis**
```
CPXERR_TOO_MANY_RIMS()
```

**Summary**
1484 Too many rim vectors.

**Description**
Reset the rim vector read limit to a larger number.
CPXERR_TOO_MANY_ROWS

Category
Macro

Synopsis
CPXERR_TOO_MANY_ROWS()

Summary
1431 Too many rows.

Description
The problem contains more rows than are allowed.
CPXERR_TOO_MANY_THREADS

Category: Macro

Synopsis: CPXERR_TOO_MANY_THREADS()

Summary: 1020 Thread limit exceeded.

Description: The maximum number of cloned threads has been exceeded.
<table>
<thead>
<tr>
<th>Category</th>
<th>Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synopsis</td>
<td><code>CPXERR_TREE_MEMORY_LIMIT()</code></td>
</tr>
<tr>
<td>Summary</td>
<td>3413 Tree memory limit exceeded.</td>
</tr>
<tr>
<td>Description</td>
<td>The reading of the tree file has stopped  because the tree memory limit has been reached.</td>
</tr>
</tbody>
</table>

**CPXERR_TREE_MEMORY_LIMIT**
<table>
<thead>
<tr>
<th>Category</th>
<th>Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synopsis</td>
<td>CPXERR_TRE_FILE_COLS ()</td>
</tr>
<tr>
<td>Summary</td>
<td>3405 Tree in file %s does not match the current problem. Number of variables in TRE file: %d. Number of variables in current problem: %d.</td>
</tr>
<tr>
<td>Description</td>
<td>The problem being solved must match that at the time the TRE file was created.</td>
</tr>
</tbody>
</table>
Category: Macro
Synopsis: \texttt{CPXERR\_TRE\_FILE\_DATA()}
Summary: 3401 Not enough data in TRE file.
Description: The TRE file became corrupted.
**CPXERR_TRE_FILE_FORMAT**

**Category**  
Macro

**Synopsis**  
`CPXERR_TRE_FILE_FORMAT()`

**Summary**  
3414 File '%s' has an incompatible format.

**Description**  
The TRE file was produced on a computer with a different byte order.
**CPXERR_TRE_FILE_INTS**

**Category**
Macro

**Synopsis**
`CPXERR_TRE_FILE_INTS()`

**Summary**
3407 Tree in file `%s` does not match the current problem. Number of integer variables in TRE file: `%d`. Number of integer variables in current problem: `%d`.

**Description**
The problem being solved must match that at the time the TRE file was created.
CPXERR_TRE_FILE_NONZ

Category: Macro

Synopsis: CPXERR_TRE_FILE_NONZ()

Summary: 3408 Tree in file %s does not match the current problem. Number of nonzeros in TRE file: %d. Number of nonzeros in current problem: %d.

Description: The problem being solved must match that at the time the TRE file was created.
CPXERR_TRE_FILE_OBJ

Category  Macro

Synopsis   CPXERR_TRE_FILE_OBJ()

Summary   3404 Objective names in TRE file %s do not match current problem.

Description  The problem being solved must match that at the time the TRE file was created.
CPXERR_TRE_FILE_OBJSEN

Category: Macro

Synopsis: CPXERR_TRE_FILE_OBJSEN()

Summary: 3415 Objective sense in TRE file %s does match current problem.

Description: The objective sense indicated by the TRE file is inconsistent with the active model.
CPXERR_TRE_FILE_PRESOLVE

Category: Macro

Synopsis: `CPXERR_TRE_FILE_PRESOLVE()`

Summary: 3410 Presolve setting for `%s` in file `%s` does not match the current setting.

Description: The presolve settings in effect at the time the TRE file was generated must match those at the time it is read.
CPXERR_TRE_FILE_ROWS

Category: Macro

Synopsis: CPXERR_TRE_FILE_ROWS()

Summary: 3406 Tree in file %s does not match the current problem. Number of constraints in TRE file: %d. Number of constraints in current problem: %d.

Description: The problem being solved must match that at the time the TRE file was created.
**CPXERR_TRE_FILE_TYPES**

**Category**  
Macro

**Synopsis**  
`CPXERR_TRE_FILE_TYPES()`

**Summary**  
3409 Different type in TRE file %s for variable '%s'.

**Description**  
The problem being solved must match that at the time the TRE file was created.
CPXERR_TRE_FILE_VERSION

Category         Macro

Synopsis         CPXERR_TRE_FILE_VERSION()

Summary          3403 Can't read TRE file written by CPLEX %s (Current version is %s).

Description      The TRE file could not be interpreted. It may have been corrupted or produced by an incompatible prior version of CPLEX.
CPXERR_TRE_FILE_WRITE

Category: Macro

Synopsis: CPXERR_TRE_FILE_WRITE(

Summary: 3402 Error on write to TRE file.

Description: The TRE file could not be written. Check available disk space.
CPXERR_UNIQUE_WEIGHTS

Category        Macro
Synopsis         CPXERR_UNIQUE_WEIGHTS()
Summary         3010 Set does not have unique weights.
Description     SOS weights must be unique.
CPXERR_UP_BOUND_REPEATS

Category: Macro

Synopsis: CPXERR_UP_BOUND_REPEATS()

Summary: 1458 Line %d: Repeated upper bound.

Description: The upper bound for a column is repeated within the problem file on the indicated line. Two individual upper bounds could exist. Alternatively, a PL bound and individual bound could be in conflict. Check the MPS file.
CPXERR_WORK_FILE_OPEN

Category Macro

Synopsis CPXERR_WORK_FILE_OPEN()

Summary 1801 Could not open temporary file.

Description CPLEX was unable to access a temporary file in the directory specified by CPX_PARAM_WORKDIR.
CPXERR_WORK_FILE_READ

Category: Macro

Synopsis: CPXERR_WORK_FILE_READ()

Summary: 1802 Failure on temporary file read.

Description: CPLEX was unable to read a temporary file in the directory specified by CPX_PARAM_WORKDIR.
Category: Macro

Synopsis: CPXERR_WORK_FILE_WRITE()

Summary: 1803 Failure on temporary file write.

Description: CPLEX was unable to write a temporary file in the directory specified by CPX_PARAM_WORKDIR.

Group: optim.cplex.solutionquality

The Callable Library macros that indicate the qualities of a solution, their symbolic constants, and their meaning. Methods for accessing solution quality are mentioned after the table.

<table>
<thead>
<tr>
<th>Macros Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPX_DUAL_OBJ</strong></td>
</tr>
<tr>
<td><strong>CPX_KAPPA</strong></td>
</tr>
<tr>
<td><strong>CPX_MAX_COMP_SLACK</strong></td>
</tr>
<tr>
<td><strong>CPX_MAX_DUAL_INFEAS</strong></td>
</tr>
<tr>
<td><strong>CPX_MAX_DUAL_RESIDUAL</strong></td>
</tr>
<tr>
<td><strong>CPX_MAX_INT_INFEAS</strong></td>
</tr>
<tr>
<td><strong>CPX_MAX_PI</strong></td>
</tr>
<tr>
<td><strong>CPX_MAX_PRIMAL_INFEAS</strong></td>
</tr>
<tr>
<td><strong>CPX_MAX_PRIMAL_RESIDUAL</strong></td>
</tr>
<tr>
<td><strong>CPX_MAX_QCPRIMAL_RESIDUAL</strong></td>
</tr>
<tr>
<td><strong>CPX_MAX_QCSLACK</strong></td>
</tr>
<tr>
<td><strong>CPX_MAX_QCSLACK_INFEAS</strong></td>
</tr>
<tr>
<td>Enum Name</td>
</tr>
<tr>
<td>-----------------------------------</td>
</tr>
<tr>
<td>CPX_MAX_RED_COST</td>
</tr>
<tr>
<td>CPX_MAX_SCALED_DUAL_INFEAS</td>
</tr>
<tr>
<td>CPX_MAX_SCALED_DUAL_RESIDUAL</td>
</tr>
<tr>
<td>CPX_MAX_SCALED_PI</td>
</tr>
<tr>
<td>CPX_MAX_SCALED_PRIMAL_INFEAS</td>
</tr>
<tr>
<td>CPX_MAX_SCALED_PRIMAL_RESIDUAL</td>
</tr>
<tr>
<td>CPX_MAX_SCALED_RED_COST</td>
</tr>
<tr>
<td>CPX_MAX_SCALED_SLACK</td>
</tr>
<tr>
<td>CPX_MAX_SCALED_X</td>
</tr>
<tr>
<td>CPX_MAX_SLACK</td>
</tr>
<tr>
<td>CPX_MAX_X</td>
</tr>
<tr>
<td>CPX_OBJ_GAP</td>
</tr>
<tr>
<td>CPX_PRIMAL_OBJ</td>
</tr>
<tr>
<td>CPX_SUM_COMP_SLACK</td>
</tr>
<tr>
<td>CPX_SUM_DUAL_INFEAS</td>
</tr>
<tr>
<td>CPX_SUM_DUAL_RESIDUAL</td>
</tr>
<tr>
<td>CPX_SUM_INT_INFEAS</td>
</tr>
<tr>
<td>CPX_SUM_PI</td>
</tr>
<tr>
<td>CPX_SUM_PRIMAL_INFEAS</td>
</tr>
<tr>
<td>CPX_SUM_PRIMAL_RESIDUAL</td>
</tr>
<tr>
<td>CPX_SUM_QCPRIMAL_RESIDUAL</td>
</tr>
<tr>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CPX_SUM_QCslack</td>
</tr>
<tr>
<td>CPX_SUM_QCslack_INFEAS</td>
</tr>
<tr>
<td>CPX_SUM_RedCost</td>
</tr>
<tr>
<td>CPX_SUM_SCALEd_DUAL_INFEAS</td>
</tr>
<tr>
<td>CPX_SUM_SCALEd_DUAL_RESIDUAL</td>
</tr>
<tr>
<td>CPX_SUM_SCALEd_PI</td>
</tr>
<tr>
<td>CPX_SUM_SCALEd_PRIMAL_INFEAS</td>
</tr>
<tr>
<td>CPX_SUM_SCALEd_PRIMAL_RESIDUAL</td>
</tr>
<tr>
<td>CPX_SUM_SCALEd_RED_COST</td>
</tr>
<tr>
<td>CPX_SUM_SCALEd_Slack</td>
</tr>
<tr>
<td>CPX_SUM_Slack</td>
</tr>
<tr>
<td>CPX_SUM_X</td>
</tr>
</tbody>
</table>

This table lists quality values.

Values that are stored in a numeric variable or double variable are accessed by the Concert Technology method `getQuality` of the class `IloCplex` or by the Callable Library routine `CPXgetdblquality`.

Values that are stored in an integer variable are accessed by the method `getQuality` of the class `IloCplex` or by the routine `CPXgetintquality`.
CPX_DUAL_OBJ

Category  Macro
Synopsis  CPX_DUAL_OBJ
Summary  Concert Technology enum: DualObj.
Description  
**Numeric meaning** (double): To access the objective value relative to the dual barrier solution. This feature is available only for a barrier solution.
**Integer meaning**: not applicable
CPX_KAPPA

Category: Macro

Synopsis: CPX_KAPPA()

Summary: Concert Technology enum: Kappa.

Description:

**Numeric meaning** (double): To access the condition number of the scaled basis matrix. This feature is available only for a **simplex** solution.

**Integer meaning**: not applicable
CPX_MAX_COMP_SLACK

Category: Macro

Synopsis: CPX_MAX_COMP_SLACK()

Summary: Concert Technology enum: MaxCompSlack.

Description:

**Numeric meaning** (`double`): To access the maximum violation of the complementary slackness conditions for the unscaled problem. This feature is available only for a barrier solution.

**Integer meaning**: To access the lowest index of a row or column with the largest violation of the complementary slackness conditions. An index (such as `*quality_p`) strictly less than zero denotes row (-i-1) or the slack variable for that row, in the case of columns. This feature is available only for a barrier solution.
CPX_MAX_DUAL_INFEAS

Category: Macro

Synopsis: CPX_MAX_DUAL_INFEAS()

Summary: Concert Technology enum: MaxDualInfeas.

Description:
- **Numeric meaning** *(double)*: To access the maximum of dual infeasibility or, equivalently, the maximum reduced-cost infeasibility for the unscaled problem.
- **Integer meaning**: To access the lowest index where the maximum dual infeasibility occurs for the unscaled problem.
CPX_MAX_DUAL_RESIDUAL

Category: Macro

Synopsis: CPX_MAX_DUAL_RESIDUAL()

Summary: Concert Technology enum: MaxDualResidual.

Description: Numeric meaning (double): To access maximum dual residual value. For a simplex solution, this is the maximum of the vector |c-B'pi|, and for a barrier solution, it is the maximum of the vector |A'pi+rc-c| for the unscaled problem.

Integer meaning: To access the lowest index where the maximum dual residual occurs for the unscaled problem.
**CPX_MAX_INT_INFEAS**

**Category**
Macro

**Synopsis**
`CPX_MAX_INT_INFEAS()`

**Summary**
Concert Technology enum: MaxIntInfeas.

**Description**

Numeric meaning (double): To access the maximum of integer infeasibility for the unscaled problem

Integer meaning: To access the lowest index where the maximum integer infeasibility occurs for the unscaled problem
**CPX_MAX_PI**

**Category**  
Macro

**Synopsis**  
`CPX_MAX_PI()`

**Summary**  
Concert Technology enum: MaxPi.

**Description**  
**Numeric meaning** (double): To access the maximum absolute value in the dual solution vector for the unscaled problem

**Integer meaning**: To access the lowest index where the maximum pi value occurs for the unscaled problem
**CPX_MAX_PRIMAL_INFEAS**

**Category**
Macro

**Synopsis**
CPX_MAX_PRIMAL_INFEAS()

**Summary**
Concert Technology enum: MaxPrimalInfeas.

**Description**

**Numeric meaning** (double): To access the maximum primal infeasibility or, equivalently, the maximum bound violation including slacks for the unscaled problem.

**Integer meaning**: To access the lowest index of a column or row where the maximum primal infeasibility occurs for the unscaled problem. An index (such as *quality_p*) strictly less than zero indicates that the maximum occurs at the slack variable for row (-i-1).
CPX_MAX_PRIMAL_RESIDUAL

Category: Macro

Synopsis: CPX_MAX_PRIMAL_RESIDUAL()

Summary: Concert Technology enum: MaxPrimalResidual.

Description:

**Numeric meaning** (double): To access the maximum of the vector $|Ax - b|$ for the unscaled problem.

**Integer meaning**: To access the lowest index where the maximum primal residual occurs for the unscaled problem.
CPX_MAX_QCPRIMAL_RESIDUAL

Category: Macro

Synopsis: CPX_MAX_QCPRIMAL_RESIDUAL()

Summary: Concert Technology enum:

Description:

Numeric meaning (double): To access the maximum residual \( |x'Qx + dx - f| \) over all the quadratic constraints in the unscaled problem.

Integer meaning: To access the lowest index over all the quadratic constraints where the maximum residual occurs in the unscaled problem.
CPX_MAX_QCSLACK

Category: Macro

Synopsis: CPX_MAX_QCSLACK()

Summary: Concert Technology enum: not applicable.

Description: Numeric meaning (double): To access the maximum absolute quadratic constraint slack value.

Integer meaning: To access the lowest index of the quadratic constraints where the maximum quadratic constraint slack values occurs.
CPX_MAX_QCSLACK_INFEAS

Category  
Macro

Synopsis  
CPX_MAX_QCSLACK_INFEAS()

Summary  
Concert Technology enum: not applicable.

Description

**Numeric meaning** (double): To access the maximum infeasibility of the quadratic constraints, or equivalently, the maximum bound violation of the quadratic constraint slacks.

**Integer meaning**: To access the lowest index of the quadratic constraints where the maximum quadratic slack infeasibility occurs.
CPX_MAX_RED_COST

Category: Macro

Synopsis: CPX_MAX_RED_COST()

Summary: Concert Technology enum: MaxRedCost.

Description:

**Numeric meaning** (double): To access the maximum absolute reduced cost value for the unscaled problem.

**Integer meaning**: To access the lowest index where the maximum reduced cost value occurs for the unscaled problem.
CPX_MAX_SCALED_DUAL_INFEAS

Category Macro

Synopsis CPX_MAX_SCALED_DUAL_INFEAS()

Summary Concert Technology enum: MaxScaledDualInfeas.

Description

**Numeric meaning** (double): To access the maximum of dual infeasibility or, equivalently, the maximum reduced-cost infeasibility for the scaled problem.

**Integer meaning**: To access the lowest index where the maximum dual infeasibility occurs for the scaled problem.
CPX_MAX_SCALED_DUAL_RESIDUAL

Category: Macro

Synopsis: CPX_MAX_SCALED_DUAL_RESIDUAL()

Summary: Concert Technology enum: MaxScaledDualResidual.

Description:
- **Numeric meaning** (double): To access maximum dual residual value for the scaled problem.
- **Integer meaning**: To access the lowest index where the maximum dual residual occurs for the scaled problem.
CPX_MAX_SCALED_PI

Category: Macro

Synopsis: CPX_MAX_SCALED_PI()

Summary: Concert Technology enum: MaxScaledPi.

Description:

**Numeric meaning** (double): To access the maximum absolute value in the dual solution vector for the scaled problem.

**Integer meaning**: To access the lowest index where the maximum pi value occurs for the scaled problem.
**CPX_MAX_SCALED_PRIMAL_INFEAS**

**Category**  
Macro

**Synopsis**  
`CPX_MAX_SCALED_PRIMAL_INFEAS()`

**Summary**  
Concert Technology enum: MaxScaledPrimalInfeas.

**Description**  
**Numeric meaning** (double): To access the maximum primal infeasibility or, equivalently, the maximum bound violation including slacks for the scaled problem

**Integer meaning**: To access the lowest index of a column or row where the maximum primal infeasibility occurs for the scaled problem
CPX_MAX_SCALED_PRIMAL_RESIDUAL

Category: Macro

Synopsis: CPX_MAX_SCALED_PRIMAL_RESIDUAL()

Summary: Concert Technology enum: MaxScaledPrimalResidual.

Description:

Numeric meaning (double): To access the maximum of the vector |Ax-b| for the scaled problem.

Integer meaning: To access the lowest index where the maximum primal residual occurs for the scaled problem.
**CPX_MAX_SCALED_RED_COST**

**Category**  
Macro

**Synopsis**  
`CPX_MAX_SCALED_RED_COST`()

**Summary**  
Concert Technology enum: MaxScaledRedCost.

**Description**  
**Numeric meaning** `(double)`: To access the maximum absolute reduced cost value for the scaled problem

**Integer meaning**: To access the lowest index where the maximum reduced cost value occurs for the scaled problem
**CPX_MAX_SCALED_SLACK**

**Category**  
Macro

**Synopsis**  
CPX_MAX_SCALED_SLACK()

**Summary**  
Concert Technology enum: MaxScaledSlack.

**Description**  
**Numeric meaning** (double): To access the maximum absolute slack value for the scaled problem  
**Integer meaning**: To access the lowest index where the maximum slack value occurs for the scaled problem
**CPX_MAX_SCALED_X**

**Category**  
Macro

**Synopsis**  
`CPX_MAX_SCALED_X()`

**Summary**  
Concert Technology enum: MaxScaledX.

**Description**  
**Numeric meaning** *(double)*: To access the maximum absolute value in the primal solution vector for the scaled problem

**Integer meaning**: To access the lowest index where the maximum x value occurs for the scaled problem
CPX_MAX_SLACK

Category          Macro

Synopsis          CPX_MAX_SLACK()

Summary           Concert Technology enum: MaxSlack.

Description

Numeric meaning  (double): To access the maximum absolute slack value for the unscaled problem

Integer meaning  To access the lowest index where the maximum slack value occurs for the unscaled problem
**CPX_MAX_X**

**Category**  
Macro

**Synopsis**  
`CPX_MAX_X()`

**Summary**  
Concert Technology enum: MaxX.

**Description**  

**Numeric meaning** (double): To access the maximum absolute value in the primal solution vector for the unscaled problem

**Integer meaning**: To access the lowest index where the maximum x value occurs for the unscaled problem
CPX_OBJ_GAP

Category     Macro
Synopsis     CPX_OBJ_GAP()
Summary     Concert Technology enum: ObjGap.
Description  Numeric meaning (double): To access the objective value gap between the primal and dual objective value solution. This feature is available only for a barrier solution.
            Integer meaning: not applicable
**CPX_PRIMAL_OBJ**

**Category**  
Macro

**Synopsis**  
`CPX_PRIMAL_OBJ()`

**Summary**  
Concert Technology enum: PrimalObj.

**Description**  
**Numeric meaning** (double): To access the objective value relative to the primal barrier solution. This feature is available only for a **barrier** solution.

**Integer meaning**: not applicable
CPX_SUM_COMP_SLACK

Category        Macro

Synopsis        CPX_SUM_COMP_SLACK()

Summary         Concert Technology enum: SumCompSlack.

Description     **Numeric meaning** (double): To access the sum of the violations of the complementary slackness conditions for the unscaled problem. This feature is available only for a barrier solution.

                  **Integer meaning**: not applicable
CPX_SUM_DUAL_INFEAS

Category                  Macro
Synopsis                   CPX_SUM_DUAL_INFEAS()
Summary                   Concert Technology enum: SumDualInfeas.
Description               **Numeric meaning** (double): To access the sum of dual infeasibilities or, equivalently, the sum of reduced-cost bound violations for the unscaled problem
                          **Integer meaning**: not applicable
CPX_SUM_DUAL_RESIDUAL

Category: Macro

Synopsis: CPX_SUM_DUAL_RESIDUAL()

Summary: Concert Technology enum: SumDualResidual.

Description:
- **Numeric meaning** (double): To access the sum of the absolute values of the dual residual vector for the unscaled problem.
- **Integer meaning**: not applicable
CPX_SUM_INT_INFEAS

Category   Macro

Synopsis   CPX_SUM_INT_INFEAS()

Summary   Concert Technology enum: SumIntInfeas.

Description   

   Numeric meaning (double): To access the sum of integer infeasibilities for the unscaled problem

   Integer meaning: not applicable
CPX_SUM_PI

Category: Macro

Synopsis: CPX_SUM_PI()

Summary: Concert Technology enum: SumPi.

Description:

**Numeric meaning (double):** To access the sum of the absolute values in the dual solution vector for the unscaled problem.

**Integer meaning:** Not applicable.
CPX_SUM_PRIMAL_INFEAS

Category            Macro
Synopsis             CPX_SUM_PRIMAL_INFEAS()
Summary             Concert Technology enum: SumPrimalInfeas.
Description          
                      **Numeric meaning** (double): To access the sum of primal infeasibilities or, equivalently, the sum of bound violations for the unscaled problem.
                      **Integer meaning**: not applicable
**CPX_SUM_PRIMAL_RESIDUAL**

**Category**  
Macro

**Synopsis**  
`CPX_SUM_PRIMAL_RESIDUAL()`

**Summary**  
Concert Technology enum: SumPrimalResidual.

**Description**  
**Numeric meaning** (double): To access the sum of the elements of vector $|Ax-b|$ for the unscaled problem  
**Integer meaning**: not applicable
**Category**
Macro

**Synopsis**
CPX\_SUM\_QCPRIMAL\_RESIDUAL()  

**Summary**
Concert Technology enum: MaxPrimalResidual.

**Description**

**Numeric meaning**\((\text{double})\): To access the sum of the residuals \(|x'^TQx + dx - f|\) for the unscaled quadratic constraints.

**Integer meaning**: not applicable
CPX_SUM_QCSLACK

Category: Macro

Synopsis: CPX_SUM_QCSLACK()

Summary: Concert Technology enum: MaxSlack.

Description:

**Numeric meaning** (double): To access the sum of the absolute quadratic constraint slack values.

**Integer meaning**: not applicable
CPX_SUM_QCSLACK_INFEAS

Category: Macro

Synopsis: CPX_SUM_QCSLACK_INFEAS()

Summary: Concert Technology enum: not applicable.

Description:

Numeric meaning (double): To access the sum of the infeasibilities of the quadratic constraints.

Integer meaning: not applicable
CPX_SUM_RED_COST

Category:  Macro

Synopsis:  CPX_SUM_RED_COST()

Summary:  Concert Technology enum: SumRedCost.

Description:  Numeric meaning (double): To access the sum of the absolute reduced cost values for
the unscaled problem

Integer meaning: not applicable
## CPX_SUM_SCALED_DUAL_INFEAS

**Category**  
Macro

**Synopsis**  
CPX_SUM_SCALED_DUAL_INFEAS()

**Summary**  
Concert Technology enum: SumScaledDualInfeas.

**Description**  
**Numeric meaning** (double): To access the sum of dual infeasibilities or, equivalently, the sum of reduced-cost bound violations for the scaled problem  
**Integer meaning**: not applicable
CPX_SUM_SCALED_DUAL_RESIDUAL

Category       Macro

Synopsis       CPX_SUM_SCALED_DUAL_RESIDUAL()

Summary        Concert Technology enum: SumScaledDualResidual.

Description    Numeric meaning (double): To access the sum of the absolute values of the dual residual vector for the scaled problem
                Integer meaning: not applicable
CPX_SUM_SCALED_PI

Category: Macro

Synopsis: CPX_SUM_SCALED_PI()

Summary: Concert Technology enum: SumScaledPi.

Description:

**Numeric meaning** (double): To access the sum of the absolute values in the dual solution vector for the scaled problem.

**Integer meaning**: not applicable.
CPX_SUM_SCALED_PRIMAL_INFEAS

Category     Macro
Synopsis       CPX_SUM_SCALED_PRIMAL_INFEAS()
Summary       Concert Technology enum: SumScaledPrimalInfeas.
Description   Numeric meaning (double): To access the sum of primal infeasibilities or, equivalently, the sum of bound violations for the scaled problem
               Integer meaning: not applicable
**CPX_SUM_SCALED_PRIMAL_RESIDUAL**

**Category**  
Macro

**Synopsis**  
`CPX_SUM_SCALED_PRIMAL_RESIDUAL()`

**Summary**  
Concert Technology enum: SumScaledPrimalResidual.

**Description**  
**Numeric meaning** (double): To access the sum of the elements of vector \( |Ax-b| \) for the unscaled problem

**Integer meaning**: not applicable
CPX_SUM_SCALED_RED_COST

Category: Macro

Synopsis: CPX_SUM_SCALED_RED_COST()

Summary: Concert Technology enum: SumScaledRedCost.

Description:

Numeric meaning (double): To access the sum of the absolute reduced cost values for the unscaled problem.

Integer meaning: not applicable
CPX_SUM_SCALED_SLACK

Category
Macro

Synopsis
CPX_SUM_SCALED_SLACK() 

Summary
Concert Technology enum: SumScaledSlack.

Description
Numeric meaning (double): To access the sum of the absolute slack values for the scaled problem

Integer meaning: not applicable
**CPX_SUM_SCALED_X**

**Category**  
Macro

**Synopsis**  
CPX_SUM_SCALED_X()

**Summary**  
Concert Technology enum: SumScaledX.

**Description**  
Numeric meaning (double): To access the sum of the absolute values in the primal solution vector for the scaled problem

Integer meaning: not applicable
**CPX_SUM_SLACK**

**Category**
Macro

**Synopsis**
CPX_SUM_SLACK()

**Summary**
Concert Technology enum: SumSlack.

**Description**

**Numeric meaning** (double): To access the sum of the absolute slack values for the unscaled problem

**Integer meaning**: not applicable
**CPX_SUM_X**

**Category**  
Macro

**Synopsis**  
CPX_SUM_X()

**Summary**  
Concert Technology enum: SumX.

**Description**  
**Numeric meaning** (double): To access the sum of the absolute values in the primal solution vector for the unscaled problem

**Integer meaning**: not applicable

**Group optim.cplex.solutionstatus**

The Callable Library macros that define solution status, their symbolic constants, their equivalent in Concert Technology enumerations, and their meaning. There is a note about unboundedness after the table.

<table>
<thead>
<tr>
<th>Macros Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_STAT_ABORT_DUAL_OBJ_LIM 22 (Barrier only) enum: AbortDualObjLim.</td>
</tr>
<tr>
<td>CPX_STAT_ABORT_IT_LIM 10 (Simplex or Barrier) enum: AbortItLim.</td>
</tr>
<tr>
<td>CPX_STAT_ABORT_OBJ_LIM 12 (Simplex or Barrier) enum: AbortObjLim.</td>
</tr>
<tr>
<td>CPX_STAT_ABORT_PRIM_OBJ_LIM 21 (Barrier only) enum: AbortPrimObjLim.</td>
</tr>
<tr>
<td>CPX_STAT_ABORT_TIME_LIM 11 (Simplex or Barrier) enum: AbortTimeLim.</td>
</tr>
<tr>
<td>CPX_STAT_ABORT_USER 13 (Simplex or Barrier) enum: AbortUser.</td>
</tr>
<tr>
<td>CPX_STAT_FEASIBLE_RELAXED 7 (Simplex or Barrier) enum: FeasibleRelaxed.</td>
</tr>
<tr>
<td>CPX_STAT_INFEASIBLE 3 (Simplex or Barrier) enum: Infeasible.</td>
</tr>
<tr>
<td>CPX_STAT_INFOrUNBD 4 (Simplex or Barrier) enum: InfOrUnbd.</td>
</tr>
<tr>
<td>ENUM NAME</td>
</tr>
<tr>
<td>-----------------------------------------</td>
</tr>
<tr>
<td>CPX_STAT_NUM_BEST</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>CPX_STAT_OPTIMAL</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>CPX_STAT_OPTIMAL_FACE_UNBOUNDED</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>CPX_STAT_OPTIMAL_INFEAS</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>CPX_STAT_OPTIMAL_RELAXED</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>CPX_STAT_UNBOUNDED</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>CPXMIP_ABORT_FEAS</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>CPXMIP_ABORT_INFEAS</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>CPXMIP_FAIL_FEAS</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>CPXMIP_FAIL_FEAS_NO_TREE</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>CPXMIP_FAIL_INFEAS</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>CPXMIP_FAIL_INFEAS_NO_TREE</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>CPXMIP_FEASIBLE_RELAXED</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>CPXMIP_INFEASIBLE</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>CPXMIP_INFOrUNBD</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>CPXMIP_MEM_LIM_FEAS</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>CPXMIP_MEM_LIM_INFEAS</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>CPXMIP_NODE_LIM_FEAS</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>CPXMIP_NODE_LIM_INFEAS</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>CPXMIP_OPTIMAL</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>CPXMIP_OPTIMAL_INFEAS</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>CPXMIP_OPTIMAL_RELAXED</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>CPXMIP_OPTIMAL_TOL</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>CPXMIP_SOL_LIM</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>CPXMIP_TIME_LIM_FEAS</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
This table lists the statuses for solutions to LP, QP, or MIP problems. These values are returned by the Callable Library routine `CPXgetstat` or by the Concert Technology methods `getCplexStatus` and `getCplexSubStatus` of the class `IloCplex`. If no solution exists, the return value is zero.

### About Unboundedness

The treatment of models that are unbounded involves a few subtleties. Specifically, a declaration of unboundedness means that ILOG CPLEX has determined that the model has an unbounded ray. Given any feasible solution $x$ with objective $z$, a multiple of the unbounded ray can be added to $x$ to give a feasible solution with objective $z-1$ (or $z+1$ for maximization models). Thus, if a feasible solution exists, then the optimal objective is unbounded. Note that ILOG CPLEX has not necessarily concluded that a feasible solution exists. Users can call the routine `CPXsolninfo` to determine whether ILOG CPLEX has also concluded that the model has a feasible solution.

<table>
<thead>
<tr>
<th>Enum</th>
<th>Description</th>
<th>Code</th>
<th>MIP only</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXMIP_TIME_LIM_INFEAS</td>
<td>Time Lim Infeas.</td>
<td>108</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPXMIP_UNBOUNDED</td>
<td>Unbounded.</td>
<td>118</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CPXMIP_ABORT_FEAS

Category: Macro

Synopsis: CPXMIP_ABORT_FEAS

Summary: 113 (MIP only) enum: AbortFeas.

Description: Stopped, but an integer solution exists
CPXMIP_ABORT_INFEAS

Category          Macro
Synopsis          CPXMIP_ABORT_INFEAS()
Summary           114 (MIP only) enum: AbortInfeas.
Description       Stopped; no integer solution
CPXMIP_FAIL_FEAS

Category: Macro

Synopsis: CPXMIP_FAIL_FEAS()

Summary: 109 (MIP only) enum: FailFeas.

Description: Terminated because of an error, but integer solution exists.
CPXMIP_FAIL_FEAS_NO_TREE

Category: Macro

Synopsis: CPXMIP_FAIL_FEAS_NO_TREE

Summary: 116 (MIP only) enum: FailFeasNoTree.

Description: Out of memory, no tree available, integer solution exists.
**Category** Macro

**Synopsis** `CPXMIP_FAIL_INFEAS()`

**Summary** 110 (MIP only) enum: FailInfeas.

**Description** Terminated because of an error; no integer solution
Category: Macro

Synopsis: CPXMIP_FAIL_INFEAS_NO_TREE()

Summary: 117 (MIP only) enum: FailInfeasNoTree.

Description: Out of memory, no tree available, no integer solution.
CPXMIP_FEASIBLE_RELAXED

Category       Macro

Synopsis       CPXMIP_FEASIBLE_RELAXED()

Summary        120 (MIP only) enum: FeasibleRelaxed.

Description    This status occurs only after a call to the Callable Library routine CPXfeasopt or the Concert Technology method feasOpt. A relaxation was successfully found and a feasible solution for the model (if relaxed according to that relaxation) was installed. The relaxation is minimal.
CPXMIP_INFEASIBLE

Category  
Macro

Synopsis  
CPXMIP_INFEASIBLE()

Summary  
103 (MIP only) enum: Infeasible.

Description  
Solution is integer infeasible
CPXMIP_INForUNBD

Category Macro

Synopsis CPXMIP_INForUNBD()

Summary 119 (MIP only) enum: InfOrUnbd.

Description Model has been proved either infeasible or unbounded
CPXMIP_MEM_LIM_FEAS

Category: Macro

Synopsis:
CPXMIP_MEM_LIM_FEAS()

Summary:
111 (MIP only) enum: MemLimFeas.

Description:
Limit on tree memory has been reached, but an integer solution exists.
CPXMIP_MEM_LIM_INFEAS

Category  Macro

Synopsis  CPXMIP_MEM_LIM_INFEAS()

Summary  112 (MIP only) enum: MemLimInfeas.

Description  Limit on tree memory has been reached; no integer solution
CPXMIP_NODE_LIM_FEAS

Category: Macro

Synopsis: CPXMIP_NODE_LIM_FEAS()

Summary: 105 (MIP only) enum: NodeLimFeas.

Description: Node limit has been exceeded but integer solution exists.
CPXMIP_NODE_LIM_INFEAS

Category  
Macro

Synopsis  
CPXMIP_NODE_LIM_INFEAS()

Summary  
106 (MIP only) enum: NodeLimInfeas.

Description  
Node limit has been reached; no integer solution
### CPXMIP_OPTIMAL

<table>
<thead>
<tr>
<th>Category</th>
<th>Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synopsis</td>
<td><code>CPXMIP_OPTIMAL()</code></td>
</tr>
<tr>
<td>Summary</td>
<td>101 (MIP only) enum: Optimal.</td>
</tr>
<tr>
<td>Description</td>
<td>Optimal integer solution has been found</td>
</tr>
</tbody>
</table>
CPXMIP_OPTIMAL_INFEAS

Category  Macro

Synopsis  CPXMIP_OPTIMAL_INFEAS ()

Summary  115 (MIP only) enum: OptimalInfeas.

Description  Problem is optimal with unscaled infeasibilities
CPXMIP_OPTIMAL_RELAXED

Category: Macro

Synopsis: CPXMIP_OPTIMAL_RELAXED()

Summary: 121 (MIP only) enum: OptimalRelaxed.

Description: This status occurs only after a call to the Callable Library routine CPXfeasopt or the Concert Technology method feasOpt. A relaxation was successfully found and an integer optimal solution for the model (if relaxed according to that relaxation) was installed. The solution is integer optimal in the sense that the relaxation is minimal and that there is no other minimal relaxation that would allow you to get a better value in the objective function for the relaxed model.
CPXMIP_OPTIMAL_TOL

Category  Macro
Synopsis    CPXMIP_OPTIMAL_TOL()
Summary    102 (MIP only) enum: OptimalTol.
Description Optimal solution with the tolerance defined by epgap or epagap has been found
CPXMIP_SOL_LIM

Category: Macro

Synopsis: CPXMIP_SOL_LIM()

Summary: 104 (MIP only) enum: SolLim.

Description: The limit on mixed integer solutions has been reached.
CPXMIP\_TIME\_LIM\_FEAS

**Category**
Macro

**Synopsis**
CPXMIP\_TIME\_LIM\_FEAS()

**Summary**
107 (MIP only) enum: AbortTimeLim.

**Description**
Time limit exceeded, but integer solution exists
<table>
<thead>
<tr>
<th>Category</th>
<th>Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synopsis</td>
<td><code>CPXMIP_TIME_LIM_INFEAS()</code></td>
</tr>
<tr>
<td>Summary</td>
<td>108 (MIP only) enum: TimeLimInfeas.</td>
</tr>
<tr>
<td>Description</td>
<td>Time limit exceeded; no integer solution</td>
</tr>
</tbody>
</table>
CPXMIP_UNBOUNDED

Category: Macro

Synopsis: CPXMIP_UNBOUNDED()

Summary: 118 (MIP only) enum: Unbounded.

Description: Model has an unbounded ray
**CPX_STAT_ABORT_DUAL_OBJ_LIM**

**Category**  
Macro

**Synopsis**  
CPX_STAT_ABORT_DUAL_OBJ_LIM

**Summary**  
22 (Barrier only) enum: AbortDualObjLim.

**Description**  
Stopped due to a limit on the dual objective
CPX_STAT_ABORT_IT_LIM

Category: Macro

Synopsis: CPX_STAT_ABORT_IT_LIM()

Summary: 10 (Simplex or Barrier) enum: AbortItLim.

Description: Stopped due to limit on number of iterations
CPX_STAT_ABORT_OBJ_LIM

**Category**  
Macro

**Synopsis**  
CPX_STAT_ABORT_OBJ_LIM

**Summary**  
12 (Simplex or Barrier) enum: AbortObjLim.

**Description**  
Stopped due to an objective limit
CPX_STAT_ABORT_PRIM_OBJ_LIM

Category: Macro

Synopsis: CPX_STAT_ABORT_PRIM_OBJ_LIM()

Summary: 21 (Barrier only) enum: AbortPrimObjLim.

Description: Stopped due to a limit on the primal objective.
CPX_STAT_ABORT_TIME_LIM

Category: Macro

Synopsis: CPX_STAT_ABORT_TIME_LIM()

Summary: 11 (Simplex or Barrier) enum: AbortTimeLim.

Description: Stopped due to a time limit
<table>
<thead>
<tr>
<th>Category</th>
<th>Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synopsis</td>
<td>CPX_STAT_ABORT_USER()</td>
</tr>
<tr>
<td>Summary</td>
<td>13 (Simplex or Barrier) enum: AbortUser.</td>
</tr>
<tr>
<td>Description</td>
<td>Stopped due to a request from the user</td>
</tr>
</tbody>
</table>
CPX_STAT_FEASIBLE_RELAXED

Category  Macro

Synopsis  CPX_STAT_FEASIBLE_RELAXED()

Summary  7 (Simplex or Barrier) enum: FeasibleRelaxed.

Description  This status occurs only after a call to the Callable Library routine CPXfeasopt or the Concert Technology method feasOpt. A relaxation was successfully found and a feasible solution for the model (if relaxed according to that relaxation) was installed. The relaxation is minimal.
### CPX_STAT_INFEASIBLE

**Category**  
Macro

**Synopsis**  
`CPX_STAT_INFEASIBLE()`

**Summary**  
3 (Simplex or Barrier) enum: Infeasible.

**Description**  
Model has been proved infeasible
CPX_STAT_INForUNBD

Category: Macro

Synopsis: CPX_STAT_INForUNBD()

Summary: 4 (Simplex or Barrier) enum: InfOrUnbd.

Description: Model has been proved either infeasible or unbounded.
CPX_STAT_NUM_BEST

Category  Macro
Synopsis   CPX_STAT_NUM_BEST()
Summary   6 (Simplex or Barrier) enum: NumBest.
Description  Solution is available, but not proved optimal, due to numeric difficulties during optimization
**Category**
Macro

**Synopsis**
CPX_STAT_OPTIMAL()

**Summary**
1 (Simplex or Barrier) enum: Optimal.

**Description**
Optimal solution is available
<table>
<thead>
<tr>
<th>Category</th>
<th>Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synopsis</td>
<td><code>CPX_STAT_OPTIMAL_FACE_UNBOUNDED()</code></td>
</tr>
<tr>
<td>Summary</td>
<td>20 (Barrier only) enum: OptimalFaceUnbounded.</td>
</tr>
<tr>
<td>Description</td>
<td>Model has an unbounded optimal face</td>
</tr>
<tr>
<td>Category</td>
<td>Macro</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Synopsis</td>
<td>CPX_STAT_OPTIMAL_INFEAS()</td>
</tr>
<tr>
<td>Summary</td>
<td>5 (Simplex or Barrier) enum: OptimalInfeas.</td>
</tr>
<tr>
<td>Description</td>
<td>Optimal solution is available, but with infeasibilities after unscaling</td>
</tr>
</tbody>
</table>
## CPX_STAT_OPTIMAL_RELAXED

**Category**  
Macro

**Synopsis**  
`CPX_STAT_OPTIMAL_RELAXED()`

**Summary**  
8 (Simplex or Barrier) enum: OptimalRelaxed.

**Description**  
This status occurs only after a call to the Callable Library routine `CPXfeasopt` or the Concert Technology method `feasOpt`. A relaxation was successfully found and an optimal solution for the model (if relaxed according to that relaxation) was installed. The solution is optimal in the sense that the relaxation is minimal and that there is no other minimal relaxation that would allow you to get a better value in the objective function for the relaxed model.
**CPX_STAT_UNBOUNDED**

**Category**  Macro

**Synopsis**  CPX_STAT_UNBOUNDED()

**Summary**  2 (Simplex or Barrier) enum: Unbounded.

**Description**  Model has an Unbounded ray

**Group optim.cplex.advanced.callable**

The API of the advanced C routines of the ILOG CPLEX Callable Library.

---

<table>
<thead>
<tr>
<th>Global Functions Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXbasicpresolve</td>
</tr>
<tr>
<td>CPXbinvacol</td>
</tr>
<tr>
<td>CPXbinvarow</td>
</tr>
<tr>
<td>CPXinvcol</td>
</tr>
<tr>
<td>CPXinvrow</td>
</tr>
<tr>
<td>CPXbranchcallbackbranchbds</td>
</tr>
<tr>
<td>CPXbranchcallbackbranchconstants</td>
</tr>
<tr>
<td>CPXbranchcallbackbranchgener al</td>
</tr>
<tr>
<td>CPXbtran</td>
</tr>
<tr>
<td>CPXcheckax</td>
</tr>
<tr>
<td>CPXcheckpib</td>
</tr>
<tr>
<td>CPXcopybasednorms</td>
</tr>
<tr>
<td>CPXcopydnorms</td>
</tr>
<tr>
<td>CPXcopylnorms</td>
</tr>
<tr>
<td>CPXcopyprotected</td>
</tr>
<tr>
<td>CPXcrushform</td>
</tr>
<tr>
<td>CPXcrushpi</td>
</tr>
<tr>
<td>CPXcrushx</td>
</tr>
<tr>
<td>CPXdjfrompi</td>
</tr>
<tr>
<td>CPXdualfarkas</td>
</tr>
<tr>
<td>CPXfreepresolve</td>
</tr>
<tr>
<td>CPXftran</td>
</tr>
<tr>
<td>CPXgetbasednorms</td>
</tr>
<tr>
<td>Function</td>
</tr>
<tr>
<td>---------------------------</td>
</tr>
<tr>
<td>CPXgetbhead</td>
</tr>
<tr>
<td>CPXgetbranchcallbackfunc</td>
</tr>
<tr>
<td>CPXgetcallbacktype</td>
</tr>
<tr>
<td>CPXgetcallbackgloballb</td>
</tr>
<tr>
<td>CPXgetcallbackglobalub</td>
</tr>
<tr>
<td>CPXgetcallbackincumbent</td>
</tr>
<tr>
<td>CPXgetcallbacklp</td>
</tr>
<tr>
<td>CPXgetcallbacknodeintfeas</td>
</tr>
<tr>
<td>CPXgetcallbacknodelb</td>
</tr>
<tr>
<td>CPXgetcallbacknodeulp</td>
</tr>
<tr>
<td>CPXgetcallbacknodeobjval</td>
</tr>
<tr>
<td>CPXgetcallbacknodestat</td>
</tr>
<tr>
<td>CPXgetcallbacknodeub</td>
</tr>
<tr>
<td>CPXgetcallbacknodevec</td>
</tr>
<tr>
<td>CPXgetcallbacknodevecex</td>
</tr>
<tr>
<td>CPXgetcallbacknodeorder</td>
</tr>
<tr>
<td>CPXgetcallbackpseudocosts</td>
</tr>
<tr>
<td>CPXgetcutcallbackfunc</td>
</tr>
<tr>
<td>CPXgetdeletenodecallbackfunc</td>
</tr>
<tr>
<td>CPXgetdnorms</td>
</tr>
<tr>
<td>CPXgetExactkappa</td>
</tr>
<tr>
<td>CPXgetheuristiccallbackfunc</td>
</tr>
<tr>
<td>CPXgetijdiv</td>
</tr>
<tr>
<td>CPXgetijrow</td>
</tr>
<tr>
<td>CPXgetincumbentcallbackfunc</td>
</tr>
<tr>
<td>CPXgetkappa</td>
</tr>
<tr>
<td>CPXgetnodecallbackfunc</td>
</tr>
<tr>
<td>CPXgetobjcallbackfunc</td>
</tr>
<tr>
<td>CPXgetobjoffset</td>
</tr>
<tr>
<td>CPXgetpnorms</td>
</tr>
<tr>
<td>CPXgetprestat</td>
</tr>
<tr>
<td>CPXgetprotected</td>
</tr>
<tr>
<td>CPXgetray</td>
</tr>
<tr>
<td>CPXgetredlp</td>
</tr>
<tr>
<td>CPXgetsolvecallbackfunc</td>
</tr>
<tr>
<td>CPXkilldnorms</td>
</tr>
<tr>
<td>CPXkillpnorms</td>
</tr>
<tr>
<td>CPXmdleave</td>
</tr>
<tr>
<td>CPXpivot</td>
</tr>
<tr>
<td>CPXpivotin</td>
</tr>
<tr>
<td>CPXpivotout</td>
</tr>
<tr>
<td>CPXpreaddrows</td>
</tr>
<tr>
<td>CPXprechgobj</td>
</tr>
<tr>
<td>CPXpresolve</td>
</tr>
</tbody>
</table>
### Description

These advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

### Warning

<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXqconstrslackfromx</td>
</tr>
<tr>
<td>CPXqpdjfrompi</td>
</tr>
<tr>
<td>CPXqpuncrushpi</td>
</tr>
<tr>
<td>CPXsetbranchcallbackfunc</td>
</tr>
<tr>
<td>CPXsetbranchnosolncallbackfunc</td>
</tr>
<tr>
<td>CPXsetcutcallbackfunc</td>
</tr>
<tr>
<td>CPXsetdeletenodecallbackfunc</td>
</tr>
<tr>
<td>CPXsetheuristiccallbackfunc</td>
</tr>
<tr>
<td>CPXsetincumbentcallbackfunc</td>
</tr>
<tr>
<td>CPXsetnodecallbackfunc</td>
</tr>
<tr>
<td>CPXsetsolvecallbackfunc</td>
</tr>
<tr>
<td>CPXslackfromx</td>
</tr>
<tr>
<td>CPXsolwrite</td>
</tr>
<tr>
<td>CPXstrongbranch</td>
</tr>
<tr>
<td>CPXtightenbds</td>
</tr>
<tr>
<td>CPXuncrushform</td>
</tr>
<tr>
<td>CPXuncrushpi</td>
</tr>
<tr>
<td>CPXuncrushx</td>
</tr>
<tr>
<td>CPXunscaleprob</td>
</tr>
</tbody>
</table>

**Description**

These advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.
CPXbasicpresolve

Category                      Global Function
Definition File              cplex.h
Include Files                cplex.h
Synopsis
int CPXPUBLIC CPXbasicpresolve (CPXCENVptr env,
                                CPXLPptr lp,
                                double * redlb,
                                double * reduub,
                                int * rstat)

Description

Note: This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXbasicpresolve performs bound strengthening and detects redundant rows. CPXbasicpresolve does not create a presolved problem. This routine cannot be used for quadratic programs.

Values for rstat[i]:
0 if row i is not redundant
-1 if row i is redundant

Example

status = CPXbasicpresolve (env, lp, redlb, reduub, rowstat);

Parameters

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.
lp
A pointer to a CPLEX LP problem object, as returned by CPXcreateprob.
redlb
An array to receive the strengthened lower bounds. The array must be of length at least the number of columns in the LP problem object. May be NULL.

redub

An array to receive the strengthened upper bounds. The array must be of length at least the number of columns in the LP problem object. May be NULL.

rstat

An array to receive the status of the row. The array must be of length at least the number of rows in the LP problem object. May be NULL.

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXbinvacol

Category  Global Function
Definition File  cplex.h
Include Files  cplex.h
Synopsis  

```c
int CPXPUBLIC CPXbinvacol(CPXCENVptr env, CPXCLPptr lp, int j, double * x)
```

Description

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXbinvacol computes the representation of the j-th column in terms of the basis. In other words, it solves Bx = Aj.

Parameters

- **env**
  The pointer to the ILOG CPLEX environment, as returned by CPXopenCPLEX.
- **lp**
  A pointer to a CPLEX LP problem object, as returned by CPXcreateprob.
- **j**
  An integer that indicates the index of the column to be computed.
- **x**
  An array containing the solution of Bx = Aj. The array must be of length at least equal to the number of rows in the problem.

Returns

The routine returns zero on success and nonzero if an error occurs.
**CPXbinvarow**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXbinvarow(CPXENVptr env, CPXCLPptr lp, int i, double * z)
```

**Description**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXbinvarow` computes the i-th row of $B^{-1}A$ where $B^{-1}$ represents the inverse of the matrix $B$ and juxtaposition indicates matrix multiplication. In other words, it computes the i-th row of the tableau.

**Parameters**

- **env**
  The pointer to the ILOG CPLEX environment, as returned by `CPXopenCPLEX`.
- **lp**
  A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.
- **i**
  An integer that indicates the index of the row to be computed.
- **z**
  An array containing the i-th row of $B^{-1}A$. The array must be of length at least equal to the number of columns in the problem.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXbinvcol

Category       Global Function
Definition File  cplex.h
Include Files  cplex.h
Synopsis

int CPXPUBLIC CPXbinvcol(CPXENVptr env,
                          CPXCLPptr lp,
                          int j,
                          double * x)

Description

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXbinvcol computes the j-th column of the basis inverse.

Parameters

- **env**
  The pointer to the ILOG CPLEX environment, as returned by CPXopenCPLEX.
- **lp**
  A pointer to a CPLEX LP problem object, as returned by CPXcreateprob.
- **j**
  An integer that indicates the index of the column of the basis inverse to be computed.
- **x**
  An array containing the j-th column of Binv (the inverse of the matrix B). The array must be of length at least equal to the number of rows in the problem.

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXbinvrow

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXbinvrow(CPXENVptr env,
                          CPXCLPptr lp,
                          int i,
                          double * y)
```

Description:

The routine CPXbinvrow computes the i-th row of the basis inverse.

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

Parameters:

- **env**
  The pointer to the ILOG CPLEX environment, as returned by CPXopenCPLEX.
- **lp**
  A pointer to the CPLEX LP problem object, as returned by CPXcreateprob.
- **i**
  An integer that indicates the index of the row to be computed.
- **y**
  An array containing the i-th row of Binv (the inverse of the matrix B). The array must be of length at least equal to the number of rows in the problem.

Returns:

The routine returns zero on success and nonzero if an error occurs.
CPXbranchcallbackbranchbds

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXbranchcallbackbranchbds(CPXENVptr env,
    void * cbdata,
    int wherefrom,
    double nodeest,
    int cnt,
    const int * indices,
    const char * lu,
    const int * bd,
    void * userhandle,
    int * seqnum_p)
```

Description:

Note: This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXbranchcallbackbranchbds specifies the branches to be taken from the current node. It may be called only from within a user-written branch callback function.

Branch variables are in terms of the original problem if the parameter CPX_PARAM_MIPCBREDLP is set to CPX_OFF before the call to CPXmipopt that calls the callback. Otherwise, branch variables are in terms of the presolved problem.

Parameters:

- **env**
  
  A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

- **cbdata**
  
  A pointer passed to the user-written callback. This argument must be the value of cbdata passed to the user-written callback.

- **wherefrom**
  
  An integer value that indicates where the user-written callback was called from. This argument must be the value of wherefrom passed to the user-written callback.
nodeest

A double that indicates the value of the node estimate for the node to be created with this branch. The node estimate is used to select nodes from the branch & cut tree with certain values of the NodeSel parameter.

cnt

An integer. The integer indicates the number of bound changes that are specified in the arrays indices, lu, and bd.

indices

An array. Together with lu and bd, this array defines the bound changes for the branch. The entry indices[i] is the index for the variable.

lu

An array. Together with indices and bd, this array defines the bound changes for each of the created nodes. The entry lu[i] is one of the three possible values indicating which bound to change: L for lower bound, U for upper bound, or B for both bounds.

bd

An array. Together with indices and lu, this array defines the bound changes for each of the created nodes. The entry bd[i] indicates the new value of the bound.

userhandle

A pointer to user private data that should be associated with the node created by this branch. May be NULL.

seqnum_p

A pointer to an integer. On return, that integer will contain the sequence number that CPLEX has assigned to the node created from this branch. The sequence number may be used to select this node in later calls to the node callback.

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXbranchcallbackbranchconstraints

Category Global Function

Definition File cplex.h

Include Files cplex.h

Synopsis

```c
int CPXPUBLIC CPXbranchcallbackbranchconstraints(CPXCENVptr env,
    void * cbdata,
    int wherefrom,
    double nodeest,
    int rcnt,
    int nzcnt,
    const double * rhs,
    const char * sense,
    const int * rmatbeg,
    const int * rmatind,
    const double * rmatval,
    void * userhandle,
    int * seqnum_p)
```

Description

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXbranchcallbackbranchconstraints` specifies the branches to be taken from the current node when the branch is specified by adding one or more constraints to the node problem. It may be called only from within a user-written branch callback function.

Constraints are in terms of the original problem if the parameter `CPX_PARAM_MIPCBREDLP` is set to `CPX_OFF` before the call to `CPXmipopt` that calls the callback. Otherwise, constraints are in terms of the presolved problem.

### Table 1: Values of `sense[i]`

<table>
<thead>
<tr>
<th>sense</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>less than or equal to constraint</td>
</tr>
<tr>
<td>E</td>
<td>equal to constraint</td>
</tr>
<tr>
<td>G</td>
<td>greater than or equal to constraint</td>
</tr>
</tbody>
</table>
Parameters

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbdata
A pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.

wherefrom
An integer value that indicates where the user-written callback was called from. This parameter must be the value of wherefrom passed to the user-written callback.

nodeest
A double that indicates the value of the node estimate for the node to be created with this branch. The node estimate is used to select nodes from the branch & cut tree with certain values of the NodeSel parameter.

rcnt
An integer that indicates the number of constraints for the branch.

ncnt
An integer that indicates the number of nonzero constraint coefficients for the branch. This specifies the length of the arrays rmatind and rmatval.

rhs
An array of length rcnt containing the right-hand side term for each constraint for the branch.

sense
An array of length rcnt containing the sense of each constraint to be added for the branch. Values of the sense appear in Table 1.

rmatbeg
An array that with rmatind and rmatval defines the constraints for the branch.

rmatind
An array that with rmatbeg and rmatval defines the constraints for the branch.

rmatval
An array that with rmatbeg and rmatind defines the constraints for the branch. The format is similar to the format used to describe the constraint matrix in the routine CPXaddrows. Every row must be stored in sequential locations in this array from position rmatbeg[i] to rmatbeg[i+1]-1 (or from rmatbeg[i] to nzcnt -1 if i=rcnt-1). Each entry, rmatind[i], indicates the column index of the corresponding coefficient, rmatval[i]. All rows must be contiguous, and rmatbeg[0] must be 0.
userhandle

A pointer to user private data that should be associated with the node created by this branch. May be NULL.

seqnum_p

A pointer to an integer that, on return, will contain the sequence number that CPLEX has assigned to the node created from this branch. The sequence number may be used to select this node in later calls to the node callback.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXbranchcallbackbranchgeneral

Category                Global Function
Definition File         cplex.h
Include Files           cplex.h

Synopsis
int CPXPUBLIC CPXbranchcallbackbranchgeneral(CPXENVptr env,
    void * cbdata,
    int wherefrom,
    double nodeest,
    int varcnt,
    const int * varind,
    const char * varlu,
    const int * varbd,
    int rcnt,
    int nzcnt,
    const double * rhs,
    const char * sense,
    const int * rmatbeg,
    const int * rmatind,
    const double * rmatval,
    void * userhandle,
    int * seqnum_p)

Description

Note: This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXbranchcallbackbranchgeneral specifies the branches to be taken from the current node when the branch includes variable bound changes and additional constraints. It may be called only from within a user-written branch callback function.

Branch variables are in terms of the original problem if the parameter CPX_PARAM_MIPCBREDLP is set to CPX_OFF before the call to CPXmipopt that calls the callback. Otherwise, branch variables are in terms of the presolved problem.
**Parameters**

- `env`  
  A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.

- `cbdata`  
  A pointer passed to the user-written callback. This parameter must be the value of `cbdata` passed to the user-written callback.

- `wherefrom`  
  An integer value that indicates where the user-written callback was called from. This parameter must be the value of `wherefrom` passed to the user-written callback.

- `nodeest`  
  A double that indicates the value of the node estimate for the node to be created with this branch. The node estimate is used to select nodes from the branch & cut tree with certain values of the `NodeSel` parameter.

- `varcnt`  
  An integer that indicates the number of bound changes that are specified in the arrays `varind`, `varlu`, and `varbd`.

- `varind`  
  Together with `varlu` and `varbd`, this array defines the bound changes for the branch. The entry `varind[i]` is the index for the variable.

- `varlu`  
  Together with `varind` and `varbd`, this array defines the bound changes for the branch. The entry `varlu[i]` is one of three possible values indicating which bound to change. Those values appear in Table 1.

- `varbd`  

**Table 1: Values of varlu[i]**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>change the lower bound</td>
</tr>
<tr>
<td>U</td>
<td>change the upper bound</td>
</tr>
<tr>
<td>R</td>
<td>change both upper and lower bounds</td>
</tr>
</tbody>
</table>

**Table 2: Values of sense[i]**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>less than or equal to constraint</td>
</tr>
<tr>
<td>E</td>
<td>equal to constraint</td>
</tr>
<tr>
<td>G</td>
<td>greater than or equal to constraint</td>
</tr>
</tbody>
</table>
Together with varind and varlu, this array defines the bound changes for the branch. The entry varbd[i] indicates the new value of the bound.

rcnt
An integer that indicates the number of constraints for the branch.

nzcnt
An integer that indicates the number of nonzero constraint coefficients for the branch. This specifies the length of the arrays rmatind and rmatval.

rhs
An array of length rcnt containing the right-hand side term for each constraint for the branch.

sense
An array of length rcnt containing the sense of each constraint to be added for the branch. Possible values appear in Table 2.

rmatbeg
An array that with rmatbeg and rmatind defines the constraints for the branch.

rmatind
An array that with rmatbeg and rmatind defines the constraints for the branch.

rmatval
An array that with rmatbeg and rmatind defines the constraints for the branch. The format is similar to the format used to describe the constraint matrix in the routine CPXaddrows. Every row must be stored in sequential locations in this array from position rmatbeg[i] to rmatbeg[i+1]-1 (or from rmatbeg[i] to nzcnt -1 if i=rcnt-1). Each entry, rmatind[i], indicates the column index of the corresponding coefficient, rmatval[i]. All rows must be contiguous, and rmatbeg[0] must be 0.

userhandle
A pointer to user private data that should be associated with the node created by this branch. May be NULL.

seqnum_p
A pointer to an integer that, on return, will contain the sequence number that CPLEX has assigned to the node created from this branch. The sequence number may be used to select this node in later calls to the node callback.

Returns
The routine returns zero on success and nonzero if an error occurs.
**CPXbtran**

**Category**    Global Function
**Definition File**    cplex.h
**Include Files**    cplex.h
**Synopsis**    int CPXPUBLIC CPXbtran(CPXENVptr env, CPXCLPptr lp, double * y)

**Description**

| Note: | This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead. |

The routine **CPXbtran** solves xTB = yT and puts the answer in y. B is the basis matrix.

**Parameters**

- **env**
  The pointer to the ILOG CPLEX environment, as returned by CPXopenCPLEX.
- **lp**
  A pointer to the CPLEX LP problem object, as returned by CPXcreateprob.
- **y**
  An array that holds the right-hand side vector on input and the solution vector on output. The array must be of length at least equal to the number of rows in the LP problem object.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
**CPXcheckax**

**Category**  Global Function

**Definition File**  cplex.h

**Include Files**  cplex.h

**Synopsis**

```c
double CPXPUBLIC CPXcheckax(CPXENVptr env,
                           CPXCLPptr lp,
                           int * imax_p,
                           int scalrimtype)
```

**Description**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXcheckax finds the $L_\infty$ norm of $Ax - b$. That is, this routine checks for numeric (roundoff) error in the computation of $x$ (the resident solution) by putting it into that formula and determining which row has the maximum error from zero. This routine also returns, in one of its arguments, the index of the row with the maximum error from zero.

To get the $L_\infty$ norm for the scaled problem, set the parameter `scalrimtype = 1`.

**Parameters**

- `env`
  The pointer to the ILOG CPLEX environment, as returned by `CPXopenCPLEX`.
- `lp`
  A pointer to the CPLEX LP problem object, as returned by `CPXcreateprob`.
- `imax_p`
  A pointer to the index of the row with the maximum absolute value in $Ax - b$. If no solution exists, *imax_p is set to $-1$.
- `scalrimtype`
  An integer that indicates the type of scaling to be applied to the returned $L_\infty$ norm. When this parameter is equal to 0 (zero), the returned $L_\infty$ norm will be unscaled. Otherwise, the $L_\infty$ norm has the same scaling as that applied to the problem currently in memory.
Returns

If successful, the routine returns the L_2 norm of Ax - b, where x is the resident solution. If no such solution exists, -1.0 is returned.
**CPXcheckpib**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
double CPXPUBLIC CPXcheckpib(CPXENVptr env,  
CPXCLPptr lp,  
int * ijmax_p,  
int scalrimtype)
```

**Description**

*Note:* This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXcheckpib finds the L? norm of cB' - ?B, where ? represents dual solution values and B represents the basis. That is, this routine checks for numeric (roundoff) error in the computation of ? by putting ? into the equation that defines it and then returning the value of the maximum deviation from zero of the elements of the resulting residual vector. This routine also returns, in one of its arguments, the index of the basic variable corresponding to this maximum.

To get the L? norm for the scaled problem, set the parameter scalrimtype = 1.

**Parameters**

- **env**
  The pointer to the ILOG CPLEX environment, as returned by CPXopenCPLEX.

- **lp**
  A pointer to the CPLEX LP problem object, as returned by CPXcreateprob.

- **ijmax_p**
  A pointer to the row or column with the maximum absolute value in cBT - ?B. If *ijmax_p corresponds to a row numbered rowindex (either a slack row or a ranged row), *ijmax_p is -1 - rowindex. If no solution exists, *ijmax_p is set to a large integer.

- **scalrimtype**
An integer that indicates the type of scaling to be applied to the returned $L^?$ norm. When this parameter is equal to 0 (zero), the returned $L^?$ norm is unscaled. Otherwise, the $L^?$ norm has the same scaling as that applied to the problem currently in memory.

Returns

If successful, this routine returns the $L^?$ norm of $cB^T - ?^T B$. If a basic solution does not exist, -1.0 is returned.
CPXcopybasednorms

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXcopybasednorms(CPXENVptr env,
                                 CPXLPptr lp,
                                 const int * cstat,
                                 const int * rstat,
                                 const double * dnorm)
```

Description:

Note: This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXcopybasednorms works in conjunction with the routine CPXgetbasednorms. CPXcopybasednorms copies the values in the arrays cstat, rstat, and dnorm, as returned by CPXgetbasednorms, into a specified problem object.

Each of the arrays cstat, rstat, and dnorm must be non NULL. Only data returned by CPXgetbasednorms should be copied by CPXcopybasednorms. (Other details of cstat, rstat, and dnorm are not documented.)

Note: The routine CPXcopybasednorms should be called only if the return values of CPXgetnumrows and CPXgetnumcols have not changed since the companion call to CPXgetbasednorms. If either of these values has increased since that companion call, a memory violation may occur. If one of those values has decreased, the call will be safe, but its meaning will be undefined.

See Also: CPXgetbasednorms

Parameters:

- env
The pointer to the ILOG CPLEX environment, as returned by CPXopenCPLEX.

lp

A pointer to the CPLEX LP problem object, as returned by CPXcreateprob.

cstat

An array containing the basis status of the columns in the constraint matrix returned by a call to CPXgetbasednorms. The length of the allocated array must be at least the value returned by CPXgetnumcols.

rstat

An array containing the basis status of the rows in the constraint matrix returned by a call to CPXgetbasednorms. The length of the allocated array must be at least the value returned by CPXgetnumrows.

dnorm

An array containing the dual steepest-edge norms returned by a call to CPXgetbasednorms. The length of the allocated array must be at least the value returned by CPXgetnumrows.

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXcopydnorms

Category          Global Function
Definition File   cplex.h
Include Files    cplex.h
Synopsis          int CPXPUBLIC CPXcopydnorms(CPXENVptr env,
                                 CPXLPptr lp,
                                 const double * norm,
                                 const int * head,
                                 int len)

Description       
Note: This is an advanced routine. Advanced routines typically demand a profound
understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher
risk of incorrect behavior in your application, behavior that can be difficult to
debug. Therefore, ILOG encourages you to consider carefully whether you can
accomplish the same task by means of other Callable Library routines instead.

The routine CPXcopydnorms copies the dual steepest-edge norms to the specified LP
problem object. The argument head is an array of column or row indices corresponding
to the array of norms. Column indices are indexed with nonnegative values. Row indices
are indexed with negative values offset by 1 (one). For example, if head[0] = -5,
then norm[0] is associated with row 4.

See Also          CPXcopypnorms, CPXgetdnorms

Parameters        
env
The pointer to the ILOG CPLEX environment, as returned by CPXopenCPLEX.
lp
A pointer to the CPLEX LP problem object, as returned by CPXcreateprob.
norm
An array containing values to be used in a subsequent call to CPXdualopt, with a setting
of CPX_PARAM_DGRADIENT different from 1 (one), as the initial values for the dual
steepest-edge norms of the corresponding basic variables specified in head[]. The
array must be of length at least equal to the value of the argument len. If any indices in
head[] are not basic, the corresponding values in norm[] are ignored.
head
An array containing the indices of the basic variables for which norms have been specified in norm[]. The array must be of length at least equal to the value of the argument len.

len

An integer that indicates the number of entries in norm[] and head[].

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXcopypnorms

Category               Global Function
Definition File        cplex.h
Include Files          cplex.h
Synopsis
int CPXPUBLIC CPXcopypnorms (CPXCENVptr env,
   CPXLPptr lp,
   const double * cnorm,
   const double * rnorm,
   int len)

Description

The routine CPXcopypnorms copies the primal steepest-edge norms to the specified LP problem object.

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXcopypnorms copies the primal steepest-edge norms to the specified LP problem object.

See Also

CPXcopydnorms, CPXgetpnorms

Parameters

env
The pointer to the ILOG CPLEX environment, as returned by CPXopenCPLEX.

lp
A pointer to a CPLEX LP problem object, as returned by CPXcreateprob.

cnorm
An array containing values to be used in a subsequent call to CPXprimopt, with a setting of CPX_PARAM_PGRADIENT of 2 or 3, as the initial values for the primal steepest-edge norms of the first len columns in the LP problem object. The array must be of length at least equal to the value of the argument len.

rnorm
An array containing values to be used in a subsequent call to CPXprimopt with a setting of CPX_PARAM_PGRADIENT of 2 or 3, as the initial values for the primal
steepest-edge norms of the slacks and ranged variables that are nonbasic. The array must be of length at least equal to the number of rows in the LP problem object.

\texttt{len}

An integer that indicates the number of entries in the array \texttt{cnorm}[].

\textbf{Returns}

The routine returns zero on success and nonzero if an error occurs.
**CPXcopyprotected**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXcopyprotected (CPXCENVptr env,  
   CPXLPptr lp,  
   int cnt,  
   const int * indices)
```

**Description**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXcopyprotected is used to specify a set of variables that should not be substituted out of the problem. If presolve can fix a variable to a value, it is removed, even if it is specified in the protected list.

**Example**

```c
status = CPXcopyprotected (env, lp, cnt, indices);
```

**Parameters**

**env**
A pointer to the CPLEX environment, as returned by CPXopenCplex.

**lp**
A pointer to a CPLEX LP problem object, as returned by CPXcreateprob.

**cnt**
The number of variables to be protected.

**indices**
An array of length cnt containing the column indices of variables to be protected from being substituted out.
Returns

The routine returns zero on success and nonzero if an error occurs.
CPXcrushform

Category       Global Function
Definition File  cplex.h
Include Files   cplex.h

Synopsis

    int CPXPUBLIC CPXcrushform(CPXENVptr env,
                                CPXCLPptr lp,
                                int len,
                                const int * ind,
                                const double * val,
                                int * plen_p,
                                double * poffset_p,
                                int * pind,
                                double * pval)

Description

Note: This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXcrushform crushes a linear formula of the original problem to a linear formula of the presolved problem.

Example

    status = CPXcrushform (env, lp, len, ind, val,
                           &plen, &poffset, pind, pval);

Parameters

    env
    A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

    lp
    A pointer to a CPLEX LP problem object, as returned by CPXcreateprob.

    len
    The number of entries in the arrays ind and val.

    ind
An array to hold the column indices of coefficients in the array \texttt{val}.

\texttt{val}

The linear formula in terms of the original problem. Each entry, \texttt{ind[i]}, indicates the column index of the corresponding coefficient, \texttt{val[i]}.

\texttt{plen_p}

A pointer to an integer to receive the number of nonzero coefficients, that is, the true length of the arrays \texttt{pind} and \texttt{pval}.

\texttt{poffset_p}

A pointer to a double to contain the value of the linear formula corresponding to variables that have been removed in the presolved problem.

\texttt{pind}

An array to hold the column indices of coefficients in the presolved problem in the array \texttt{pval}.

\texttt{pval}

The linear formula in terms of the presolved problem. Each entry, \texttt{pind[i]}, indicates the column index in the presolved problem of the corresponding coefficient, \texttt{pval[i]}. The arrays \texttt{pind} and \texttt{pval} must be of length at least the number of columns in the presolved LP problem object.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXcrushpi

Category       Global Function
Definition File cplex.h
Include Files  cplex.h

Synopsis       int CPXPUBLIC CPXcrushpi(CPXENVptr env,
                CPXCLPptr lp,
                const double * pi,
                double * prepi)

Description    The routine CPXcrushpi crushes a dual solution for the original problem to a dual solution for the presolved problem.

Note: This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

Example

status = CPXcrushpi (env, lp, origpi, reducepi);

Parameters

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

lp
A pointer to a CPLEX LP problem object, as returned by CPXcreateprob.

pi
An array that contains dual solution (pi) values for the original problem, as returned by routines such as CPXgetpi or CPXsolution. The array must be of length at least the number of rows in the LP problem object.

prepi
An array to receive dual values corresponding to the presolved problem. The array must be of length at least the number of rows in the presolved problem object.
Returns

The routine returns zero on success and nonzero if an error occurs.
CPXcrushx

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXcrushx(CPXCENVptr env,
                          CPXCLPptr lp,
                          const double * x,
                          double * prex)
```

Description:

```
Note: This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.
```

The routine CPXcrushx crushes a solution for the original problem to a solution for the presolved problem.

Example:

```c
status = CPXcrushx (env, lp, origx, reducex);
```

Parameters:

`env`

A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

`lp`

A pointer to a CPLEX LP problem object, as returned by CPXcreateprob.

`x`

An array that contains primal solution (x) values for the original problem, as returned by routines such as CPXgetx or CPXsolution. The array must be of length at least the number of columns in the problem object.

`prex`

An array to receive the primal values corresponding to the presolved problem. The array must be of length at least the number of columns in the presolved problem object.
Returns

The routine returns zero on success and nonzero if an error occurs.

CPXdjfrompi

Category       Global Function
Definition File cplex.h
Include Files  cplex.h

Synopsis       

\[
\text{int CPXPUBLIC CPXdjfrompi\{CPXCENVptr env,} \\
\quad \text{CPXCLPptr lp,} \\
\quad \text{const double * pi,} \\
\quad \text{double * dj}\}
\]

Description    

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXdjfrompi computes an array of reduced costs from an array of dual values. This routine is for linear programs. Use CPXqpdjfrompi for quadratic programs.

**Example**

\[
\text{status = CPXdjfrompi\{env, lp, pi, dj\};}
\]

**Parameters**

- **env**
  A pointer to the CPLEX environment as returned by CPXopenCPLEX.
- **lp**
  A pointer to a CPLEX LP problem object as returned by CPXcreateprob.
- **pi**
  An array that contains dual solution (pi) values for the problem, as returned by routines such as CPXuncrushpi and CPXcrushpi. The array must be of length at least the number of rows in the problem object.
- **dj**
  An array to receive the reduced cost values computed from the pi values for the problem object. The array must be of length at least the number of columns in the problem object.
Returns

The routine returns zero on success and nonzero if an error occurs.
CPXdualfarkas

Category: Global Function
Definition File: cplex.h
Include Files: cplex.h
Synopsis:
```c
int CPXPUBLIC CPXdualfarkas(CPXENVptr env,
    CPXCLPptr lp,
    double * y,
    double * proof_p)
```
Description:

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXdualfarkas assumes that there is a resident solution as produced by a call to CPXdualopt and that the status of this solution as returned by CPXgetstat is CPX_STAT_INFEASIBLE.

The values returned in the array `y[]` have the following interpretation. For the `ith` constraint, if that constraint is a less-than-or-equal-to constraint, \( y[i] \leq 0 \) holds; if that constraint is a greater-than-or-equal-to constraint, \( y[i] \geq 0 \) holds. Thus, where \( b \) is the right-hand-side vector for the given linear program, \( A \) is the constraint matrix, and \( x \) denotes the vector of variables, \( y \) may be used to derive the following valid inequality:

\[ y^T A x \geq y^T b \]

Here \( y \) is being interpreted as a column vector, and \( y^T \) denotes the transpose of \( y \).

The real point of computing \( y \) is the following. Suppose we define a vector \( z \) of dimension equal to the dimension of \( x \) and having the following value for entries

\[ z_j = u_j \] where \( y^T A_j > 0 \), and
\[ z_j = l_j \] where \( y^T A_j < 0 \),

where \( A_j \) denotes the column of \( A \) corresponding to \( x_j \), \( u_j \) the given upper bound on \( x_j \), and \( l_j \) is the specified lower bound. (\( z_j \) is arbitrary if \( y^T A_j = 0 \).) Then \( y \) and \( z \) will satisfy

\[ y^T b - y^T A z > 0. \]
This last inequality contradicts the validity of $y^T A x \geq y^T b$, and hence shows that the given linear program is infeasible. The quantity $\ast \text{proof}_p$ is set equal to $y^T b - y^T A z$. Thus, $\ast \text{proof}_p$ in some sense denotes the degree of infeasibility.

**Parameters**

- env
  A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
- lp
  A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.
- y
  An array of doubles of length at least equal to the number of rows in the problem.
- proof_p
  A pointer to a double. The parameter `proof_p` is allowed to have the value NULL.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXfreepresolve

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis: int CPXPUBLIC CPXfreepresolve (CPXCENVptr env, CPXLPptr lp)

Description:

Note: This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXfreepresolve frees the presolved problem from the LP problem object. Under the default setting of CPX_PARAM_REDUCE, the presolved problem is freed when an optimal solution is found. It is not freed when CPX_PARAM_REDUCE is set to CPX_PREREDUCE_PRIMALONLY (1) or CPX_PREREDUCE_DUALONLY (2), so the routine CPXfreepresolve can be used to free it manually.

Example:

status = CPXfreepresolve (env, lp);

Parameters:

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

lp
A pointer to a CPLEX LP problem object, as returned by CPXcreateprob.

Returns:
The routine returns zero on success and nonzero if an error occurs.
CPXftran

Category         Global Function
Definition File  cplex.h
Include Files    cplex.h
Synopsis         int CPXPUBLIC CPXftran(CPXENVptr env,
                          CPXCLPptr lp,
                          double * x)

Description

Note: This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXftran solves By = x and puts the answer in the vector x, where B is the basis matrix.

Parameters

env
The pointer to the ILOG CPLEX environment, as returned by CPXopenCPLEX.
lp
A pointer to a CPLEX LP problem object, as returned by CPXcreateprob.
x
An array that holds the right-hand side vector on input and the solution vector on output. The array must be of length at least equal to the number of rows in the LP problem object.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXgetExactkappa

Category          Global Function
Definition File   cplex.h
Include Files     cplex.h
Synopsis          double CPXPUBLIC CPXgetExactkappa(CPXENVptr env, CPXCLPptr lp)

Description

Note: This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus, they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetExactkappa computes and returns the condition number, kappa.

See Also            CPXgetkappa

Parameters

env
The pointer to the ILOG CPLEX environment, as returned by CPXopenCPLEX.
lp
A pointer to a CPLEX LP problem object, as returned by CPXcreateprob.

Returns            If successful, this routine returns the condition number, kappa.
CPXgetbasednorms

Category: Global Function
Definition File: cplex.h
Include Files: cplex.h
Synopsis:
```c
int CPXPUBLIC CPXgetbasednorms(CPXCENVptr env,
                           CPXCLPptr lp,
                           int * cstat,
                           int * rstat,
                           double * dnorm)
```

Description:

The routine `CPXgetbasednorms` works in conjunction with the routine `CPXcopybasednorms`. `CPXgetbasednorms` retrieves the resident basis and dual norms from a specified problem object.

Each of the arrays `cstat`, `rstat`, and `dnorm` must be non-NULL. That is, each of these arrays must be allocated. The allocated size of `cstat` is assumed by this routine to be at least the number returned by `CPXgetnumcols`. The allocated size of `rstat` and `dnorm` are assumed to be at least the number returned by `CPXgetnumrows`. (Other details of `cstat`, `rstat`, and `dnorm` are not documented.)

Success, Failure

If this routine succeeds, `cstat` and `rstat` contain information about the resident basis, and `dnorm` contains the dual steepest-edge norms. If there is no basis, or if there is no set of dual steepest-edge norms, this routine returns an error code. The returned data are intended solely for use by `CPXcopybasednorms`.

Example

For example, if a given LP has just been successfully solved by the ILOG CPLEX Callable Library optimizer `CPXdualopt` with the dual pricing option `CPX_PARAM_DPRIIND` set to `CPX_DPRIIND_STEEP`, `CPX_DPRIIND_FULLSTEEL`, or `CPX_DPRIIND_STEEPQSTART`, then a call to `CPXgetbasednorms` should succeed. (That optimizer and those pricing options are...

Motivation

When the ILOG CPLEX Callable Library optimizer CPXdualopt is called to solve a problem with the dual pricing option CPX_PARAM_DPRIIND set to CPX_DPRIIND_STEEP or CPX_DPRIIND_FULLSTEEP, there must be values of appropriate dual norms available before the optimizer can begin. If these norms are not already resident, they must be computed, and that computation may be expensive. The functions CPXgetbasednorms and CPXcopybasednorms can, in some cases, avoid that expense. Suppose, for example, that in some application an LP is solved by CPXdualopt with one of those pricing settings. After the solution of the LP, some intermediate optimizations are carried out on the same LP, and those subsequent optimizations are in turn followed by some changes to the LP, and a re-solve. In such a case, copying the basis and norms that were resident before the intermediate solves, back into ILOG CPLEX data structures can greatly increase the speed of the re-solve.

See Also

CPXcopybasednorms

Parameters

env
The pointer to the ILOG CPLEX environment, as returned by CPXopenCPLEX.

lp
A pointer to the CPLEX LP problem object, as returned by CPXcreateprob.

cstat
An array containing the basis status of the columns in the constraint matrix. The length of the allocated array is at least the value returned by CPXgetnumcols.

rstat
An array containing the basis status of the rows in the constraint matrix. The length of the allocated array is at least the value returned by CPXgetnumrows.

dnorm
An array containing the dual steepest-edge norms. The length of the allocated array is at least the value returned by CPXgetnumrows.

Returns
The routine returns zero on success and nonzero if an error occurs.
**CPXgetbhead**

**Category**          Global Function

**Definition File**   cplex.h

**Include Files**     cplex.h

**Synopsis**

```
int CPXPUBLIC CPXgetbhead(CPXENVptr env,
                          CPXCLPptr lp,
                          int * head,
                          double * x)
```

**Description**

The routine CPXgetbhead returns the basis header; it gives the negative value minus one of all row indices of slacks.

**Parameters**

- **env**
  
  The pointer to the ILOG CPLEX environment, as returned by CPXopenCPLEX.

- **lp**
  
  A pointer to a CPLEX LP problem object, as returned by CPXcreateprob.

- **head**
  
  An array. The array contains the indices of the variables in the resident basis, where basic slacks are specified by the negative of the corresponding row index minus 1 (one); that is, \(-\text{rowindex} - 1\). The array must be of length at least equal to the number of rows in the LP problem object.

- **x**
  
  An array. This array contains the values of the basic variables in the order specified by head[]. The array must be of length at least equal to number of rows in the LP problem object.

**Returns**

The routine returns zero on success and nonzero if an error occurs.

---

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.
**CPXgetbranchcallbackfunc**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**
```c
void CPXPUBLIC CPXgetbranchcallbackfunc(CPXENVptr env,
    int(CPXPUBLIC **branchcallback_p)(CALLBACK_BRANCH_ARGS),
    void ** cbhandle_p)
```

**Description**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine **CPXgetbranchcallbackfunc** accesses the user-written callback routine to be called during MIP optimization after a branch has been selected but before the branch is carried out. ILOG CPLEX uses the callback routine to change its branch selection.

**Example**

```c
CPXgetbranchcallbackfunc(env, &current_callback,
    &current_handle);
```

See also *Advanced MIP Control Interface* in the **ILOG CPLEX User's Manual**.

**Parameters**

- **env**
  A pointer to the CPLEX environment, as returned by **CPXopenCPLEX**.

- **branchcallback_p**
  The address of the pointer to the current user-written branch callback. If no callback has been set, the returned pointer evaluates to NULL.

- **cbhandle_p**
  The address of a variable to hold the user's private pointer.
The call to the branch callback occurs after a branch has been selected but before the branch is carried out. This function is written by the user. On entry to the callback, the ILOG CPLEX-selected branch is defined in the arguments. The arguments to the callback specify a list of changes to make to the bounds of variables when child nodes are created. One, two, or zero child nodes can be created, so one, two, or zero lists of changes are specified in the arguments. The first branch specified is considered first. The callback is called with zero lists of bound changes when the solution at the node is integer feasible.

Custom branching strategies can be implemented by calling the CPLEX function CPXbranchcallbackbranchbds and setting the useraction variable to CPX_CALLBACK_SET. Then CPLEX will carry out these branches instead of the CPLEX-selected branches.

Branch variables are in terms of the original problem if the parameter CPX_PARAM_MIPCBREDLP is set to CPX_OFF before the call to CPXmipopt that calls the callback. Otherwise, branch variables are in terms of the presolved problem.

Callback return value
The callback returns zero on success and nonzero if an error occurs.

Callback arguments
env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbdata
A pointer passed from the optimization routine to the user-written callback that identifies the problem being optimized. The only purpose of this pointer is to pass it to the callback information routines.

**wherefrom**

An integer value indicating where in the optimization this function was called. It will have the value CPX_CALLBACK_MIP_BRANCH.

**cbhandle**

A pointer to user-private data.

**int type**

An integer that indicates the type of branch. This table summarizes possible values.

### Branch Types Returned from a User-Written Branch Callback

<table>
<thead>
<tr>
<th>Symbolic Constant</th>
<th>Value</th>
<th>Branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_TYPE_VAR</td>
<td>0</td>
<td>variable branch</td>
</tr>
<tr>
<td>CPX_TYPE_SOS1</td>
<td>1</td>
<td>SOS1 branch</td>
</tr>
<tr>
<td>CPX_TYPE_SOS2</td>
<td>2</td>
<td>SOS2 branch</td>
</tr>
<tr>
<td>CPX_TYPE_USER</td>
<td>X</td>
<td>user-defined</td>
</tr>
</tbody>
</table>

**sos**

An integer that indicates the special ordered set (SOS) used for this branch. A value of 1 indicates that this branch is not an SOS-type branch.

**nodecnt**

An integer that indicates the number of nodes CPLEX will create from this branch. Possible values are:

- 0 (zero), or
- 1, or
- 2.

If the argument is 0, the node will be fathomed unless user-specified branches are made; that is, no child nodes are created and the node itself is discarded.

**bdcnt**

An integer that indicates the number of bound changes defined in the arrays `indices`, `lu`, and `bd` that define the CPLEX-selected branch.

**nodeest**
An array with \texttt{nodecnt} entries that contains estimates of the integer objective-function value that will be attained from the created node.

\texttt{nodebeg}

An array with \texttt{nodecnt} entries. The \texttt{i}-th entry is the index into the arrays \texttt{indices}, \texttt{lu}, and \texttt{bd} of the first bound changed for the \texttt{i}th node.

\texttt{indices}

Together with \texttt{lu} and \texttt{bd}, this array defines the bound changes for each of the created nodes. The entry \texttt{indices[i]} is the index for the variable.

\texttt{lu}

Together with \texttt{indices} and \texttt{bd}, this array defines the bound changes for each of the created nodes. The entry \texttt{lu[i]} is one of the three possible values indicating which bound to change:

- \texttt{L} for lower bound, or
- \texttt{U} for upper bound, or
- \texttt{B} for both bounds.

\texttt{bd}

Together with \texttt{indices} and \texttt{lu}, this array defines the bound changes for each of the created nodes. The entry \texttt{bd[i]} indicates the new value of the bound.

\texttt{useraction_p}

A pointer to an integer indicating the action for ILOG CPLEX to take at the completion of the user callback. The table summarizes the possible actions.

### Actions to be Taken After a User-Written Branch Callback

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CPX_CALLBACK_DEFAULT</td>
<td>Use CPLEX-selected branch</td>
</tr>
<tr>
<td>1</td>
<td>CPX_CALLBACK_FAIL</td>
<td>Exit optimization</td>
</tr>
<tr>
<td>2</td>
<td>CPX_CALLBACK_SET</td>
<td>Use user-selected branch, as defined by calls to CPXbranchcallbackbranchbds</td>
</tr>
<tr>
<td>3</td>
<td>CPX_CALLBACK_NO_SPACE</td>
<td>Allocate more space and call callback again</td>
</tr>
</tbody>
</table>

**See Also**

CPXsetbranchcallbackfunc

**Returns**

This routine does not return a result.
**CPXgetcallbackctype**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXgetcallbackctype(CPXENVptr env,
    void * cbdata,
    int wherefrom,
    char * xtype,
    int begin,
    int end)
```

**Description**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetcallbackctype` is used to get the cotypes for the MIP problem from within a user-written callback during MIP optimization. The values are from the original problem if `CPX_PARAM_MIPCBREDLP` is set to `CPX_OFF`. Otherwise, they are from the presolved problem.

This routine may be called only when the value of the `wherefrom` argument is one of the following:

- `CPX_CALLBACK_MIP`,
- `CPX_CALLBACK_MIP_BRANCH`,
- `CPX_CALLBACK_MIP_INCUMBENT`,
- `CPX_CALLBACK_MIP_NODE`,
- `CPX_CALLBACK_MIP_HEURISTIC`,
- `CPX_CALLBACK_MIP_SOLVE`, or
- `CPX_CALLBACK_MIP_CUT`.

**Example**
status = CPXgetcallbackctype (env, cbdata, wherefrom, 
                  prectype, 0, precols-1);

**Parameters**

- **env**
  A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

- **cbdata**
  The pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.

- **wherefrom**
  An integer value indicating from where the user-written callback was called. The parameter must be the value of wherefrom passed to the user-written callback.

- **xctype**
  An array where the ctype values for the MIP problem will be returned. The array must be of length at least (end - begin + 1). If successful, xctype[0] through xctype[end-begin] contain the variable types.

- **begin**
  An integer indicating the beginning of the range of ctype values to be returned.

- **end**
  An integer indicating the end of the range of ctype values to be returned.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXgetcallbackgloballb

Category                           Global Function
Definition File                    cplex.h
Include Files                      cplex.h
Synopsis                           int CPXPUBLIC CPXgetcallbackgloballb(CPXCENVptr env,
                                    void * cbdata,
                                    int wherefrom,
                                    double * lb,
                                    int begin,
                                    int end)

Description

Note: This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetcallbackgloballb is used to get the best known global lower bound values during MIP optimization from within a user-written callback. The values are from the original problem if CPX_PARAM_MIPCBREDLP is set to CPX_OFF, otherwise they are from the presolved problem.

This routine may be called only when the value of the wherefrom argument is one of the following:

- CPX CALLBACK MIP,
- CPX CALLBACK MIP BRANCH,
- CPX CALLBACK MIP INCUMBENT,
- CPX CALLBACK MIP NODE,
- CPX CALLBACK MIP HEURISTIC,
- CPX CALLBACK MIP SOLVE, or
- CPX CALLBACK MIP CUT.

Example
status = CPXgetcallbackgloballb (env, cbdata, wherefrom, glb, 0, cols-1);

Parameters

env

A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbdata

The pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.

wherefrom

An integer value indicating from where the user-written callback was called. The parameter must be the value of wherefrom passed to the user-written callback.

lb

An array to receive the values of the global lower bound values. This array must be of length at least (end - begin + 1). If successful, lb[0] through lb[end - begin] contain the global lower bound values.

begin

An integer indicating the beginning of the range of lower bound values to be returned.

end

An integer indicating the end of the range of lower bound values to be returned.

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXgetcallbackglobalub

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXgetcallbackglobalub(CPXENVptr env,
    void * cbdata,
    int wherefrom,
    double * ub,
    int begin,
    int end)
```

Description:

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetcallbackglobalub is used to get the best known global upper bound values during MIP optimization from within a user-written callback. The values are from the original problem if CPX_PARAM_MIPCBREDLP is set to CPX_OFF, otherwise they are from the presolved problem.

This routine may be called only when the value of the wherefrom argument is one of the following:

- CPX_CALLBACK_MIP,
- CPX_CALLBACK_MIP_BRANCH,
- CPX_CALLBACK_MIP_INCUMBENT,
- CPX_CALLBACK_MIP_NODE,
- CPX_CALLBACK_MIP_HEURISTIC,
- CPX_CALLBACK_MIP_SOLVE, or
- CPX_CALLBACK_MIP_CUT.

Example:
status = CPXgetcallbackglobalub (env, cbdata, wherefrom,  
    gub, 0, cols-1);

Parameters

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbdata
The pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.

wherefrom
An integer value indicating from where the user-written callback was called. The parameter must be the value of wherefrom passed to the user-written callback.

ub
An array to receive the values of the global upper bound values. This array must be of length at least (end - begin + 1). If successful, ub[0] through ub[end-begin] contain the global upper bound values.

begin
An integer indicating the beginning of the range of upper bound values to be returned.

d end
An integer indicating the end of the range of upper bound values to be returned.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXgetcallbackincumbent

Category       Global Function
Definition File cplex.h
Include Files  cplex.h
Synopsis       

```c
int CPXPUBLIC CPXgetcallbackincumbent(CPXENVptr env,
    void * cbdata,
    int wherefrom,
    double * x,
    int begin,
    int end)
```

Description

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetcallbackincumbent is used to get the incumbent values during MIP optimization from within a user-written callback. The values are from the original problem if CPX_PARAM_MIPCBREDLP is set to CPX_OFF. Otherwise, they are from the presolved problem.

This routine may be called only when the value of the wherefrom argument is one of the following:

- CPX_CALLBACK_MIP
- CPX_CALLBACK_MIP_BRANCH
- CPX_CALLBACK_MIP_INCUMBENT
- CPX_CALLBACK_MIP_NODE
- CPX_CALLBACK_MIP_HEURISTIC
- CPX_CALLBACK_MIP_SOLVE, or
- CPX_CALLBACK_MIP_CUT.

Example
status = CPXgetcallbackincumbent (env, cbdata, wherefrom,
    bestx, 0, cols-1);

Parameters

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbdata
The pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.

wherefrom
An integer value indicating from where the user-written callback was called. The parameter must be the value of wherefrom passed to the user-written callback.

x
An array to receive the values of the incumbent (best available) integer solution. This array must be of length at least (end - begin + 1). If successful, x[0] through x[end-begin] contain the incumbent values.

begin
An integer indicating the beginning of the range of incumbent values to be returned.

end
An integer indicating the end of the range of incumbent values to be returned.

Returns
The routine returns zero on success and nonzero if an error occurs.
**CPXgetcallbacklp**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXgetcallbacklp(CPXENVptr env,  
void * cbdata,  
int wherefrom,  
CPXCLPptr * lp_p)
```

**Description**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetcallbacklp is used to get the pointer to the MIP problem that is in use when the user-written callback function is called. It is the original MIP if CPX_PARAM_MIPCBREDLP is set to CPX_OFF; otherwise, it is the presolved MIP. In contrast, the function CPXgetcallbacknodelp returns a pointer to the node subproblem, which is an LP. Generally, this pointer may be used only in CPLEX Callable Library query routines, such as CPXsolution or CPXgetrows.

This routine may be called only when the value of the wherefrom argument is one of the following:

- CPX_CALLBACK_MIP,
- CPX_CALLBACK_MIP_BRANCH,
- CPX_CALLBACK_MIP_INCUMBENT,
- CPX_CALLBACK_MIP_NODE,
- CPX_CALLBACK_MIP_HEURISTIC,
- CPX_CALLBACK_MIP_SOLVE, or
- CPX_CALLBACK_MIP_CUT.

**Example**
status = CPXgetcallbacklp (env, cbdata, wherefrom, &origlp);

See also admipex1.c, admipex2.c, and admipex3.c in the standard distribution.

**Parameters**

- **env**
  A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

- **cbdata**
  The pointer passed to the user-written callback. This parameter must be the value of `cbdata` passed to the user-written callback.

- **wherefrom**
  An integer value indicating from where the user-written callback was called. The parameter must be the value of `wherefrom` passed to the user-written callback.

- **lp_p**
  A pointer to a variable of type CPXLPptr to receive the pointer to the LP problem object, which is a MIP.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
**CPXgetcallbacknodeintfeas**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXgetcallbacknodeintfeas(CPXENVptr env,
   void * cbdata,
   int wherefrom,
   int * feas,
   int begin,
   int end)
```

**Description**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetcallbacknodeintfeas is used to get indicators for each variable of whether or not the variable is integer feasible in the node subproblem. It can be used in a user-written callback during MIP optimization. The indicators are from the original problem if CPX_PARAM_MIPCBREDLP is set to CPX_OFF. Otherwise, they are from the presolved problem.

**Example**

```c
status = CPXgetcallbacknodeintfeas(env, cbdata, wherefrom,
   feas, 0, cols-1);
```

See admipex1.c and admipex2.c in the standard distribution.

This routine may be called only when the value of the wherefrom argument is one of the following:

- CPX_CALLBACK_MIP,
- CPX_CALLBACK_MIP_BRANCH,
- CPX_CALLBACK_MIP_INCUMBENT,
- CPX_CALLBACK_MIP_NODE,
- CPX_CALLBACK_MIP_HEURISTIC, or
- CPX_CALLBACK_MIP_CUT.

**Indicators of feasibility for a node of the subproblem**

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_INTEGER_FEASIBLE</td>
<td>0</td>
<td>variable (j+\text{begin}) is integer-valued</td>
</tr>
<tr>
<td>CPX_INTEGER_INFEASIBLE</td>
<td>1</td>
<td>variable (j+\text{begin}) is not integer-valued</td>
</tr>
<tr>
<td>CPX_IMPLIED_INTEGER_FEASIBLE</td>
<td>2</td>
<td>variable (j+\text{begin}) may have a fractional value in the current solution, but it will take on an integer value when all integer variables still in the problem have integer values. It should not be branched upon.</td>
</tr>
</tbody>
</table>

**Parameters**

- **env**
  A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

- **cbdata**
  The pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.

- **wherefrom**
  An integer value indicating from where the user-written callback was called. The parameter must be the value of wherefrom passed to the user-written callback.

- **feas**
  An array to receive an indicator of feasibility for the node subproblem. This array must be of length at least \((\text{end} - \text{begin} + 1)\). If successful, feas[0] through feas[\text{end}-\text{begin}] will contain the indicators. The indicators of feasibility for a node of the subproblem appear in the table.

- **begin**
  An integer indicating the beginning of the range of feasibility indicators to be returned.

- **end**
  An integer indicating the end of the range of feasibility indicators to be returned.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
**CPXgetcallbacknodelb**

**Category**  Global Function

**Definition File**  cplex.h

**Include Files**  cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXgetcallbacknodelb(CPXCENVptr env,
    void * cbdata,
    int wherefrom,
    double * lb,
    int begin,
    int end)
```

**Description**

**Note**: This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetcallbacknodelb is used to get the lower bound values for the subproblem at the current node during MIP optimization from within a user-written callback. The values are from the original problem if `CPX_PARAM_MIPCBREDLP` is set to `CPX_OFF`; otherwise, they are from the presolved problem.

This routine may be called only when the value of the `wherefrom` argument is one of the following:

- `CPX_CALLBACK_MIP`
- `CPX_CALLBACK_MIP_BRANCH`
- `CPX_CALLBACK_MIP_INCUMBENT`
- `CPX_CALLBACK_MIP_NODE`
- `CPX_CALLBACK_MIP_HEURISTIC`
- `CPX_CALLBACK_MIP_SOLVE`, or
- `CPX_CALLBACK_MIP_CUT`.

**Example**
status = CPXgetcallbacknodelb (env, cbdata, wherefrom, lb, 0, cols-1);

Parameters

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbdata
The pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.

wherefrom
An integer value indicating from where the user-written callback was called. The parameter must be the value of wherefrom passed to the user-written callback.

lb
An array to receive the values of the lower bound values. This array must be of length at least (end - begin + 1). If successful, lb[0] through lb[end-begin] contain the lower bound values for the current subproblem.

begin
An integer indicating the beginning of the range of lower bounds to be returned.

end
An integer indicating the end of the range of lower bounds to be returned.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXgetcallbacknodelp

Category: Global Function
Definition File: cplex.h
Include Files: cplex.h
Synopsis:
```
int CPXPUBLIC CPXgetcallbacknodelp(CPXENVptr env, 
    void * cbdata, 
    int wherefrom, 
    CPXLPptr * nodelp_p)
```

Description:

This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetcallbacknodelp accesses the lp pointer indicating the currently defined linear programming subproblem (LP) from within user-written callbacks. Generally, this pointer may be used only in ILOG CPLEX Callable Library query routines, such as CPXsolution or CPXgetrows.

Example:
```
status = CPXgetcallbacknodelp (env, cbdata, &nodelp);
```

See also the example admipex1.c and admipex6.c in the standard distribution.

CPXgetcallbacknodelp may be called only when its wherefrom argument has one of the following values:
- CPX_CALLBACK_MIP,
- CPX_CALLBACK_MIP_BRANCH,
- CPX_CALLBACK_MIP_CUT,
- CPX_CALLBACK_MIP_HEURISTIC,
- CPX_CALLBACK_MIP_INCUMBENT, or
- CPX_CALLBACK_MIP_SOLVE.
When the `wherefrom` argument has the value `CPX_CALLBACK_MIP_SOLVE`, the subproblem pointer may also be used in ILOG CPLEX optimization routines.

**Note:** *Any modification to the subproblem may result in corruption of the problem and of the ILOG CPLEX environment.*

### Parameters

- `env`
  A pointer to the CPLEX environment, as returned by `CPXopenCplex`.

- `cbdata`
  The `cbdata` pointer passed to the user-written callback. This parameter must be the value of `cbdata` passed to the user-written callback.

- `wherefrom`
  An integer value indicating where the user-written callback was called from. This parameter must be the value of the `wherefrom` passed to the user-written callback.

- `nodelp_p`
  The `lp` pointer indicating the current subproblem. If no subproblem is defined, the pointer is set to NULL.

### Returns

The routine returns zero on success and nonzero if an error occurs. A nonzero return value may mean that the requested value is not available.
### CPXgetcallbacknodeobjval

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXgetcallbacknodeobjval(CPXCNVptr env,
   void * cbdata,
   int wherefrom,
   double * objval_p)
```

**Description**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetcallbacknodeobjval` is used to get the objective value for the subproblem at the current node during MIP optimization from within a user-written callback.

This routine may be called only when the value of the `wherefrom` argument is one of the following:

- `CPX_CALLBACK_MIP`
- `CPX_CALLBACK_MIP_BRANCH`
- `CPX_CALLBACK_MIP_INCUMBENT`
- `CPX_CALLBACK_MIP_NODE`
- `CPX_CALLBACK_MIP_HEURISTIC`
- `CPX_CALLBACK_MIP_CUT`

**Example**

```c
status = CPXgetcallbacknodeobjval (env, cbdata, wherefrom, 
   &objval);
```

See also `admipex1.c` and `admipex3.c` in the standard distribution.
Parameters

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbddata
The pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.

wherefrom
An integer value indicating from where the user-written callback was called. The parameter must be the value of wherefrom passed to the user-written callback.

objval_p
A pointer to a variable of type double where the objective value of the node subproblem is to be stored.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXgetcallbacknodestat

Category  Global Function
Definition File  cplex.h
Include Files  cplex.h
Synopsis  

```c
int CPXPUBLIC CPXgetcallbacknodestat (CPXCENVptr env,
  void * cbdata,
  int wherefrom,
  int * nodestat_p)
```

Description

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetcallbacknodestat is used to get the optimization status of the subproblem at the current node from within a user-written callback during MIP optimization.

The optimization status will be either optimal or unbounded. An unbounded status can occur when some of the constraints are being treated as lazy constraints. When the node status is unbounded, then the function CPXgetcallbacknodex returns a ray that can be used to decide which lazy constraints need to be added to the subproblem.

This routine may be called only when the value of the wherefrom argument is CPX_CALLBACK_MIP_CUT.

Example

```c
status = CPXgetcallbacknodestat (env, cbdata, wherefrom,
   &nodestatus);
```

Parameters

**env**

A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

**cbdata**

The pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.
wherefrom

An integer value indicating from where the user-written callback was called. The parameter must be the value of `wherefrom` passed to the user-written callback.

`nodestat_p`

A pointer to an integer where the node subproblem optimization status is to be returned. The values of `*nodestat_p` may be `CPX_STAT_OPTIMAL` or `CPX_STAT_UNBOUNDED`.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
**CPXgetcallbacknodeub**

**Category**     Global Function

**Definition File**  cplex.h

**Include Files**  cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXgetcallbacknodeub(CPXENVptr env,
    void * cbdata,
    int wherefrom,
    double * ub,
    int begin,
    int end)
```

**Description**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetcallbacknodeub` is used to get the upper bound values for the subproblem at the current node during MIP optimization from within a user-written callback. The values are from the original problem if `CPX_PARAM_MIPCBREDLP` is set to `CPX_OFF`; otherwise, they are from the presolved problem.

This routine may be called only when the value of the `wherefrom` argument is one of the following:

- `CPX_CALLBACK_MIP`
- `CPX_CALLBACK_MIP_BRANCH`
- `CPX_CALLBACK_MIP_INCUMBENT`
- `CPX_CALLBACK_MIP_NODE`
- `CPX_CALLBACK_MIP_HEURISTIC`
- `CPX_CALLBACK_MIP_SOLVE`, or
- `CPX_CALLBACK_MIP_CUT`.

**Example**
status = CPXgetcallbacknodeub (env, cbdata, wherefrom, ub, 0, cols-1);

**Parameters**

- **env**
  A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

- **cbdata**
  The pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.

- **wherefrom**
  An integer value indicating from where the user-written callback was called. The parameter must be the value of wherefrom passed to the user-written callback.

- **ub**
  An array to receive the values of the upper bound values. This array must be of length at least (end - begin + 1). If successful, ub[0] through ub[end-begin] contain the upper bound values for the current subproblem.

- **begin**
  An integer indicating the beginning of the range of upper bound values to be returned.

- **end**
  An integer indicating the end of the range of upper bound values to be returned.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXgetcallbacknodex

Category       Global Function
Definition File cplex.h
Include Files  cplex.h
Synopsis

int CPXPUBLIC CPXgetcallbacknodex (CPXCENVptr env,
void * cbdata,
int wherefrom,
double * x,
int begin,
int end)

Description

The routine CPXgetcallbacknodex is used to get the primal variable (x) values for
the subproblem at the current node during MIP optimization from within a user-written
callback. The values are from the original problem if CPX_PARAM_MIPCBREDLP is
set to CPX_OFF; otherwise, they are from the presolved problem.

This routine may be called only when the value of the wherefrom argument is one of
the following:

◆ CPX_CALLBACK_MIP,
◆ CPX_CALLBACK_MIP_BRANCH,
◆ CPX_CALLBACK_MIP_INCUMBENT,
◆ CPX_CALLBACK_MIP_NODE,
◆ CPX_CALLBACK_MIP_HEURISTIC, or
◆ CPX_CALLBACK_MIP_CUT.

Example

status = CPXgetcallbacknodex(env, cbdata, wherefrom,
       nodex, 0, cols-1);
See also admipex1.c, admipex3.c, and admipex5.c in the standard distribution.

**Parameters**

- **env**
  A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

- **cbdata**
  The pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.

- **wherefrom**
  An integer value indicating from where the user-written callback was called. The parameter must be the value of wherefrom passed to the user-written callback.

- **x**
  An array to receive the values of the primal variables for the node subproblem. This array must be of length at least \((end - begin + 1)\). If successful, \(x[0] \) through \(x[end-begin]\) contain the primal values.

- **begin**
  An integer indicating the beginning of the range of primal variable values for the node subproblem to be returned.

- **end**
  An integer indicating the end of the range of primal variable values for the node subproblem to be returned.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXgetcallbackorder

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXgetcallbackorder(CPXENVptr env,
   void * cbdata,
   int wherefrom,
   int * priority,
   int * direction,
   int begin,
   int end)
```

Description:

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetcallbackorder is used to get MIP priority order information during MIP optimization from within a user-written callback. The values are from the original problem if CPX_PARAM_MIPCBREDLP is set to CPX_OFF. Otherwise, they are from the presolved problem.

This routine may be called only when the value of the `wherefrom` argument is one of the following values:

- `CPX_CALLBACK_MIP`,
- `CPX_CALLBACK_MIP_BRANCH`,
- `CPX_CALLBACK_MIP_INCUMBENT`,
- `CPX_CALLBACK_MIP_NODE`,
- `CPX_CALLBACK_MIP_HEURISTIC`,
- `CPX_CALLBACK_MIP_SOLVE`, or
- `CPX_CALLBACK_MIP_CUT`.

Example
status = CPXgetcallbackorder (env, cbdata, wherefrom,
priority, NULL, 0, cols-1);

Branching direction

<table>
<thead>
<tr>
<th>Branching direction</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_BRANCH_GLOBAL</td>
<td>0</td>
<td>use global branching direction setting</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CPX_PARAM_BRDIR</td>
</tr>
<tr>
<td>CPX_BRANCH_DOWN</td>
<td>-1</td>
<td>branch down first on variable j+begin</td>
</tr>
<tr>
<td>CPX_BRANCH_UP</td>
<td>1</td>
<td>branch up first on variable j+begin</td>
</tr>
</tbody>
</table>

Parameters

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbdata
The pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.

wherefrom
An integer value indicating from where the user-written callback was called. The parameter must be the value of wherefrom passed to the user-written callback.

priority
An array where the priority values are to be returned. This array must be of length at least (end - begin + 1). If successful, priority[0] through priority[end-begin] contain the priority order values. May be NULL. The value of direction[j] will be a value from the table of branching directions.

begin
An integer indicating the beginning of the range of priority order information to be returned.

dend
An integer indicating the end of the range of priority order information to be returned.

Returns
The routine returns zero on success and nonzero if an error occurs.
**CPXgetcallbackpseudocosts**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXgetcallbackpseudocosts(CPXCENVptr env,  
void * cbdata,  
int wherefrom,  
double * uppc,  
double * downpc,  
int begin,  
int end)
```

**Description**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine **CPXgetcallbackpseudocosts** is used to get the pseudo-cost values during MIP optimization from within a user-written callback. The values are from the original problem if **CPX_PARAM_MIPCBREDLP** is set to **CPX_OFF**. Otherwise, they are from the presolved problem.

**Note:** When pseudo-costs are retrieved for the original problem variables, pseudo-costs are zero for variables that have been removed from the problem, since they are never used for branching.

This routine may be called only when the value of the **wherefrom** argument is one of the following:

- **CPX_CALLBACK_MIP**,  
- **CPX_CALLBACK_MIP_BRANCH**,  
- **CPX_CALLBACK_MIP_INCUMBENT**,  
- **CPX_CALLBACK_MIP_NODE**.
Example

```c
status = CPXgetcallbackpseudocosts (env, cbdata, wherefrom,
                                      upcost, downcost,
                                      j, k);
```

Parameters

- **env**
  A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

- **cbdata**
  The pointer passed to the user-written callback. This parameter must be the value of cbdata passed to the user-written callback.

- **wherefrom**
  An integer value indicating from where the user-written callback was called. The parameter must be the value of wherefrom passed to the user-written callback.

- **uppc**
  An array to receive the values of up pseudo-costs. This array must be of length at least \((end - begin + 1)\). If successful, uppc[0] through uppc[end-begin] will contain the up pseudo-costs. May be NULL.

- **downpc**
  An array to receive the values of the down pseudo-costs. This array must be of length at least \((end - begin + 1)\). If successful, downpc[0] through downpc[end-begin] will contain the down pseudo-costs. May be NULL.

- **begin**
  An integer indicating the beginning of the range of pseudo-costs to be returned.

- **end**
  An integer indicating the end of the range of pseudo-costs to be returned.

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXgetcutcallbackfunc

Category               Global Function
Definition File            cplex.h
Include Files            cplex.h
Synopsis
void CPXPUBLIC CPXgetcutcallbackfunc(CPXCENVptr env,
    int (CPXPUBLIC **cutcallback_p)(CALLBACK_CUT_ARGS) ,
    void ** cbhandle_p)

Description

Note: This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetcutcallbackfunc accesses the user-written callback for adding cuts. The user-written callback is called by ILOG CPLEX during MIP branch & cut for every node that has an LP optimal solution with objective value below the cutoff and that is integer infeasible. The callback routine adds globally valid cuts to the LP subproblem.

Example

CPXgetcutcallbackfunc(env, &current_cutfunc,
    &current_data);

See also Advanced MIP Control Interface in the ILOG CPLEX User's Manual.

For documentation of callback arguments, see the routine CPXsetcutcallbackfunc.

Parameters

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cutcallback_p
The address of the pointer to the current user-written cut callback. If no callback has been set, the pointer evaluates to NULL.
cbhandle_p
The address of a variable to hold the user's private pointer.

See Also CPXcutcallbackadd, CPXsetcutcallbackfunc

Returns This routine does not return a result.
CPXgetdeletenodecallbackfunc

**Category**  Global Function

**Definition File**  cplex.h

**Include Files**  cplex.h

**Synopsis**

```c
void CPXPUBLIC CPXgetdeletenodecallbackfunc(CPXCENVptr env,
                                           void(CPXPUBLIC **deletecallback_p)(CALLBACK_DELETENODE_ARGS) ,
                                           void ** cbhandle_p)
```

**Description**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetdeletenodecallbackfunc accesses the user-written callback to be called during MIP optimization when a node is to be deleted. Nodes are deleted when a branch is carried out from that node, when the node relaxation is infeasible, or when the node relaxation objective value is worse than the cutoff. This callback can be used to delete user data associated with a node.

**Example**

```c
CPXgetdeletenodecallbackfunc(env,
                       &current_callback,
                       &current_cbdata);
```

See also Advanced MIP Control Interface in the *ILOG CPLEX User's Manual*.

For documentation of callback arguments, see the routine CPXsetdeletenodecallbackfunc.

**Parameters**

- **env**

  A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

- **deletenodecallback_p**
The address of the pointer to the current user-written delete-node callback. If no callback has been set, the pointer evaluates to NULL.

cbhandle_p

The address of a variable to hold the user’s private pointer.

**See Also**
- CPXsetdeletenodecallbackfunc
- CPXbranchcallbackbranchbds
- CPXbranchcallbackbranchconstraints
- CPXbranchcallbackbranchgeneral

**Returns**
This routine does not return a result.
**CPXgetdnorms**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**
```c
int CPXPUBLIC CPXgetdnorms(CPXENVptr env,
    CPXCLPptr lp,
    double * norm,
    int * head,
    int * len_p)
```

**Description**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetdnorms accesses the norms from the dual steepest edge. As in CPXcopydnorms, the argument head is an array of column or row indices corresponding to the array of norms. Column indices are indexed with nonnegative values. Row indices are indexed with negative values offset by 1 (one). For example, if head[0] = -5, norm[0] is associated with row 4.

**See Also**
CPXcopydnorms

**Parameters**

- **env**
The pointer to the ILOG CPLEX environment, as returned by CPXopenCPLEX.
- **lp**
A pointer to the CPLEX LP problem object, as returned by CPXcreateprob.
- **norm**
An array containing the dual steepest-edge norms in the ordered specified by head[]. The array must be of length at least equal to the number of rows in the LP problem object.
- **head**
An array containing column or row indices. The allocated length of the array must be at least equal to the number of rows in the LP problem object.

len_p

A pointer to an integer that indicates the number of entries in both norm[] and head[]. The value assigned to the pointer *len_p is needed by the routine CPXcopydnorms.

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXgetheuristiccallbackfunc

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
void CPXPUBLIC CPXgetheuristiccallbackfunc(CPXCENVptr env,
   int(CPXPUBLIC **heuristiccallback_p)(CALLBACK_HEURISTIC_ARGS) ,
   void ** cbhandle_p)
```

Description:

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetheuristiccallbackfunc accesses the user-written callback to be called by ILOG CPLEX during MIP optimization after the subproblem has been solved to optimality. That callback is not called when the subproblem is infeasible or cut off. The callback supplies ILOG CPLEX with heuristically-derived integer solutions.

**Example**

```c
CPXgetheuristiccallbackfunc(env, &current_callback, &current_handle);
```

See also **Advanced MIP Control Interface** in the *ILOG CPLEX User's Manual*.

For documentation of callback arguments, see the routine **CPXsetheuristiccallbackfunc**.

**Parameters**

- `env`
  A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

- `heuristiccallback_p`
  The address of the pointer to the current user-written heuristic callback. If no callback has been set, the pointer evaluates to NULL.

- `cbhandle_p`
The address of a variable to hold the user's private pointer.

**See Also**

CPXsetheuristiccallbackfunc

**Returns**

This routine does not return a result.
CPXgetijdiv

Category  Global Function
Definition File  cplex.h
Include Files  cplex.h
Synopsis

```c
int CPXPUBLIC CPXgetijdiv(CPXENVptr env,
CPXCLPptr lp,
int * idiv_p,
int * jdiv_p)
```

Description

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetijdiv returns the index of the diverging row (that is, constraint) or column (that is, variable) when one of the ILOG CPLEX simplex optimizers terminates due to a diverging vector. This function can be called after an unbounded solution status for a primal simplex call or after an infeasible solution status for a dual simplex call.

If one of the ILOG CPLEX simplex optimizers has concluded that the LP problem object is unbounded, and if the diverging variable is a slack or ranged variable, CPXgetijdiv returns the index of the corresponding row in *idiv_p. Otherwise, *idiv_p is set to -1.

If one of the ILOG CPLEX simplex optimizers has concluded that the LP problem object is unbounded, and if the diverging variable is a normal, structural variable, CPXgetijdiv sets *jdiv_p to the index of that variable. Otherwise, *jdiv_p is set to -1.

Parameters

- **env**
  The pointer to the ILOG CPLEX environment, as returned by CPXopenCPLEX.
- **lp**
  A pointer to the CPLEX LP problem object, as returned by CPXcreateprob.
- **idiv_p**
  The index of the diverging row (constraint) or column (variable).
A pointer to an integer indexing the row of a diverging variable.

If one of the ILOG CPLEX simplex optimizers has concluded that the LP problem object is unbounded, and if the diverging variable is a slack or ranged variable, CPXgetijdiv returns the index of the corresponding row in *idiv_p. Otherwise, *idiv_p is set to -1.

jdiv_p
A pointer to an integer indexing the row of a diverging variable.

If one of the ILOG CPLEX simplex optimizers has concluded that the LP problem object is unbounded, and if the diverging variable is a normal, structural variable, CPXgetijdiv sets *jdiv_p to the index of that variable. Otherwise, *jdiv_p is set to -1.

Returns
The routine returns zero on success and nonzero if an error occurs.
Cpxgetijrow

Category Global Function

Definition File cplex.h

Include Files cplex.h

Synopsis

```
int CPXPUBLIC CPXgetijrow(CPXENVptr env,
                          CPXCLPptr lp,
                          int i,
                          int j,
                          int * row_p)
```

Description

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetijrow returns the index of a specific basic variable as its position in the basis header. If the specified row indexes a constraint that is not basic, or if the specified column indexes a variable that is not basic, CPXgetijrow returns an error code and sets the value of its argument *row_p to -1. An error is also returned if both row and column indices are specified in the same call.

Parameters

**env**

The pointer to the ILOG CPLEX environment, as returned by CPXopenCPLEX.

**lp**

The pointer to the CPLEX LP problem object, as returned by CPXcreateprob.

**i**

An integer specifying the index of a basic row; CPXgetijrow must find the position of this basic row in the basis header. A negative value in this argument indicates to CPXgetijrow not to seek a basic row.

**j**

An integer specifying the index of a basic column; CPXgetijrow must find the position of this basic column in the basis header. A negative value in this argument indicates to CPXgetijrow not to seek a basic column.
row_p
A pointer to an integer indicating the position in the basis header of the row \textit{i} or column \textit{j}. If \texttt{CPXgetijrow} encounters an error, and if \texttt{row_p} is not NULL, \*\texttt{row_p} is set to -1.

\textbf{Returns} 
The routine returns zero on success and nonzero if an error occurs.
**CPXgetincumbentcallbackfunc**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
void CPXPUBLIC CPXgetincumbentcallbackfunc(CPXENVptr env, 
int(CPXPUBLIC **incumbentcallback_p)(CALLBACK_INCUMBENT_ARGS) , 
void ** cbhandle_p)
```

**Description**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetincumbentcallbackfunc` accesses the user-written callback to be called by CPLEX during MIP optimization after an integer solution has been found but before this solution replaces the incumbent. This callback can be used to discard solutions that do not meet criteria beyond that of the mixed integer programming formulation.

**Example**

```c
CPXgetincumbentcallbackfunc(env, &current_incumbentcallback, 
&current_handle);
```

See also Advanced MIP Control Interface in the ILOG CPLEX User's Manual.

For documentation of callback arguments, see the routine `CPXsetincumbentcallbackfunc`.

**Parameters**

- **env**
  
  A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.

- **incumbentcallback_p**
  
  The address of the pointer to the current user-written incumbent callback. If no callback has been set, the pointer evaluates to NULL.
cbhandle_p

The address of a variable to hold the user's private pointer.

See Also

CPXsetincumbentcallbackfunc

Returns

This routine does not return a result.
CPXgetkappa

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis: double CPXPUBLIC CPXgetkappa(CPXENVptr env, CPXCLPptr lp)

Description:

Note: This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetkappa computes and returns an estimate of the condition number, kappa.

Parameters:

env
The pointer to the ILOG CPLEX environment, as returned by CPXopenCPLEX.

lp
A pointer to the CPLEX LP problem object, as returned by CPXcreateprob.

Returns: If successful, this routine returns an estimate of the condition number, kappa.
CPXgetnodecallbackfunc

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
void CPXPUBLIC CPXgetnodecallbackfunc(CPXENVptr env,
                                  int(CPXPUBLIC **nodecallback_p)(CALLBACK_NODE_ARGS) ,
                                  void ** cbhandle_p)
```

Description:

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetnodecallbackfunc accesses the user-written callback to be called during MIP optimization after ILOG CPLEX has selected a node to explore, but before this exploration is carried out. The callback routine can change the node selected by ILOG CPLEX to a node selected by the user.

For documentation of callback arguments, see the routine CPXsetnodecallbackfunc.

**Example**

```c
CPXgetnodecallbackfunc(env, &current_callback, &current_handle);
```

See also the example admipex1.c in the standard distribution.

**Parameters**

- **env**
  
  A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

- **nodecallback_p**
  
  The address of the pointer to the current user-written node callback. If no callback has been set, the pointer will evaluate to NULL.

- **cbhandle_p**
  
  A pointer to the handle.
The address of a variable to hold the user's private pointer.

**Returns**

This routine does not return a result.
CPXgetobjoffset

Category       Global Function

Definition File cplex.h

Include Files  cplex.h

Synopsis

```c
int CPXPUBLIC CPXgetobjoffset(CPXENVptr env,
                               CPXCLPptr lp,
                               double * objoffset_p)
```

Description

Note: This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. They thus incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetobjoffset returns the objective offset between the original problem and the presolved problem.

Parameters

- `env`  
The pointer to the ILOG CPLEX environment, as returned by CPXopenCPLEX.
- `lp`  
A pointer to a reduced CPLEX LP problem object, as returned by CPXgetredlp.
- `objoffset_p`  
A pointer to a variable of type double to hold the objective offset value.

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXgetpnorms

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXgetpnorms(CPXENVptr env,
                               CPXCLPptr lp,
                               double * cnorm,
                               double * rnorm,
                               int * len_p)
```

Description:

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetpnorms returns the norms from the primal steepest-edge. There is no comparable argument in this routine for `rnorm[]`. If the rows of the problem have changed since the norms were computed, they are generally no longer valid. However, if columns have been deleted, or if columns have been added, the norms for all remaining columns present before the deletions or additions remain valid.

See Also:

- CPXcopypnorms

Parameters:

- `env`
  - The pointer to the ILOG CPLEX environment, as returned by CPXopenCPLEX.

- `lp`
  - A pointer to a CPLEX LP problem object, as returned by CPXcreateprob.

- `cnorm`
  - An array containing the primal steepest-edge norms for the normal, column variables. The array must be of length at least equal to the number of columns in the LP problem object.

- `rnorm`
  - An array containing the primal steepest-edge norms for the normal, column variables.

- `len_p`
  - The length of the array `cnorm` and `rnorm`. The length must be equal to the number of columns in the LP problem object.
An array containing the primal steepest-edge norms for ranged variables and slacks. The array must be of length at least equal to the number of rows in the LP problem object.

`len_p`

A pointer to the number of entries in the array `cnorm[]`. When this routine is called, `*len_p` is equal to the number of columns in the LP problem object when optimization occurred. The routine `CPXcopynorms` needs the value `*len_p`.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
**Synopsis**

```c
int CPXPUBLIC CPXgetprestat(CPXENVptr env, CPXCLPptr lp, int * prestat_p, int * pcstat, int * prstat, int * ocstat, int * orstat);
```

**Description**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetprestat` is used to get presolve status information for the columns and rows of the presolved problem in the original problem and of the original problem in the presolved problem.

**Table 1: Value of prestat_p**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td><code>lp</code> is not presolved or there were no reductions</td>
</tr>
<tr>
<td>1</td>
<td><code>lp</code> has a presolved problem</td>
</tr>
<tr>
<td>2</td>
<td><code>lp</code> was reduced to an empty problem</td>
</tr>
</tbody>
</table>

For variable `i` in the original problem, values for `pcstat[i]` appear in Table 2.

**Table 2: Values for pcstat[i]**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 0</td>
<td><code>variable i</code> corresponds to <code>variable pcstat[i]</code> in the presolved problem</td>
</tr>
</tbody>
</table>
For row $i$ in the original problem, values for $\text{prstat}[i]$ appear in Table 3.

### Table 2: Values for $\text{pcstat}[i]$  

<table>
<thead>
<tr>
<th>$\text{CPX_PRECOL_LOW}$</th>
<th>-1</th>
<th>$i$ is fixed to its lower bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{CPX_PRECOL_UP}$</td>
<td>-2</td>
<td>$i$ is fixed to its upper bound</td>
</tr>
<tr>
<td>$\text{CPX_PRECOL_FIX}$</td>
<td>-3</td>
<td>$i$ is fixed to some other value</td>
</tr>
<tr>
<td>$\text{CPX_PRECOL_AGG}$</td>
<td>-4</td>
<td>$i$ is aggregated out</td>
</tr>
<tr>
<td>$\text{CPX_PRECOL_OTHER}$</td>
<td>-5</td>
<td>$i$ is deleted or merged for some other reason</td>
</tr>
</tbody>
</table>

For variable $i$ in the presolved problem, values for $\text{ocstat}[i]$ appear in Table 4.

### Table 3: Values for $\text{prstat}[i]$  

<table>
<thead>
<tr>
<th>$\text{CPX_PREROW_RED}$</th>
<th>-1</th>
<th>row $i$ is redundant</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{CPX_PREROW_AGG}$</td>
<td>-2</td>
<td>row $i$ is used for aggregation</td>
</tr>
<tr>
<td>$\text{CPX_PREROW_OTHER}$</td>
<td>-3</td>
<td>row $i$ is deleted for some other reason</td>
</tr>
</tbody>
</table>

For row $i$ in the original problem, values for $\text{orstat}[i]$ appear in Table 5.

### Table 4: Values for $\text{ocstat}[i]$  

<table>
<thead>
<tr>
<th>$\geq 0$</th>
<th>$i$ corresponds to a linear combination of some variables in the original problem.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\leq 0$</td>
<td>$i$ corresponds to $\text{ocstat}[i]$ in the presolved problem.</td>
</tr>
</tbody>
</table>

For row $i$ in the original problem, values for $\text{orstat}[i]$ appear in Table 5.

### Table 5: Values for $\text{orstat}$  

<table>
<thead>
<tr>
<th>$\geq 0$</th>
<th>row $i$ corresponds to row $\text{orstat}[i]$ in the original problem.</th>
</tr>
</thead>
</table>
Example

```c
status = CPXgetprestat (env, lp, &presolvestat, precstat, prerstat, origcstat, origrstat);
```

See also admipex6.c in the *ILOG CPLEX User's Manual*.

**Parameters**

- **env**
  A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

- **lp**
  A pointer to the original CPLEX LP problem object, as returned by CPXcreateprob.

- **prestat_p**
  A pointer to an integer that will receive the status of the presolved problem associated with LP problem object lp. May be NULL.

- **pcstat**
  The array where the presolve statuses of the columns are to be returned. The array must be of length at least the number of columns in the original problem object. May be NULL.

- **prstat**
  The array where the presolve statuses of the rows are to be returned. The array must be of length at least the number of rows in the original problem object. May be NULL.

- **ocstat**
  The array where the presolve statuses of the columns of the presolved problem are to be returned. The array must be of length at least the number of columns in the presolved problem object. May be NULL.

- **orstat**
  The array where the presolve statuses of the rows of the presolved problem are to be returned. The array must be of length at least the number of rows in the presolved problem object. May be NULL.

**Returns**

The routine returns zero on success and nonzero if an error occurs.

---

**Table 5: Values for orstat**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>If row i is created by, for example, merging two rows in the original problem.</td>
</tr>
</tbody>
</table>
**CPXgetprotected**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**
```c
int CPXPUBLIC CPXgetprotected(CPXENVptr env, CPXCLPptr lp, int * cnt_p, int * indices, int pspace, int * surplus_p)
```

**Description**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetprotected is used to get the set of variables that cannot be aggregated out.

**Note:** If the value of pspace is 0, the negative of the value of *surplus_p returned indicates the length needed for array indices.

**Example**
```c
status = CPXgetprotected (env, lp, &protectcnt, protectind, 10, &surplus);
```

**Parameters**

- `env`
  A pointer to the CPLEX environment, as returned by CPXopenCPLEX.
- `lp`
  A pointer to a CPLEX LP problem object, as returned by CPXcreateprob.
- `cnt_p`
  A pointer to an integer that is set to the number of protected variables.
- `indices`
  A pointer to an array of integers that contains the indices of the protected variables.
- `pspace`
  A nonnegative integer that specifies the size of the space to allocate.
- `surplus_p`
  A pointer to an integer that is set to the negative of the space needed for the array `indices`.
A pointer to an integer to contain the number of protected variables returned, that is, the true length of the array `indices`.

`indices`  
The array to contain the indices of the protected variables.

`pspace`  
An integer indicating the length of the array `indices`.

`surplus_p`  
A pointer to an integer to contain the difference between `pspace` and the number of entries in `indices`. A nonnegative value of `*surplus_p` indicates that the length of the arrays was sufficient. A negative value indicates that the length was insufficient and that the routine could not complete its task. In that case, the routine `CPXgetprotected` returns the value `CPXERR_NEGATIVE_SURPLUS`, and the value of `*surplus_p` indicates the amount of insufficient space in the arrays.

**Returns**

The routine returns zero on success and nonzero if an error occurs. The value `CPXERR_NEGATIVE_SURPLUS` indicates that insufficient space was available in the array `indices` to hold the protected variable indices.
CPXgetray

Category   Global Function
Definition File   cplex.h
Include Files    cplex.h
Synopsis

```
int CPXPUBLIC CPXgetray(CPXCENVptr env, 
CPXCLPtr lp, 
double * z)
```

Description

The routine `CPXgetray` is used to find an unbounded direction (also known as a ray) for a linear program where the CPLEX simplex algorithm concludes that the LP is unbounded (solution status `CPX_STAT_UNBOUNDED`). An error is returned, `CPXERR_NOT_UNBOUNDED`, if this case does not hold.

As an illustration, consider a linear program of the form:

```
Minimize       c'x
Subject to     Ax = b
                x \geq 0
```

where ' indicates the transpose.

If the CPLEX simplex algorithm completes optimization with a solution status of `CPX_STAT_UNBOUNDED`, the vector `z` returned by `CPXgetray` would satisfy the following:

```
c'z < 0
Az = 0
z \geq 0
```

if computations could be carried out in exact arithmetic.
Example

\[
\text{status} = \text{CPXgetray} (\text{env}, \text{lp}, z);
\]

Parameters

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

lp
A pointer to the CPLEX LP problem object, as returned by CPXcreateprob.

z
The array where the unbounded direction is returned. This array must be at least as large as the number of columns in the problem object.

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXgetredlp

Category          Global Function
Definition File   cplex.h
Include Files     cplex.h
Synopsis           int CPXPUBLIC CPXgetredlp (CPXCENVptr env,
                                                                  CPXCLPptr lp,
                                                                  CPXCLPptr * redlp_p)

Description

Note: This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXgetredlp returns a pointer for the presolved problem. It returns NULL if the problem is not presolved or if all the columns and rows are removed by presolve. Generally, the returned pointer may be used only in CPLEX Callable Library query routines, such as CPXsolution or CPXgetrows.

The presolved problem must not be modified. Any modifications must be done on the original problem. If CPX_PARAM_REDUCE is set appropriately, the modifications are automatically carried out on the presolved problem at the same time. Optimization and query routines can be used on the presolved problem.

Example

status = CPXgetredlp (env, lp, &reducelp);

Parameters

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.
lp
A pointer to a CPLEX LP problem object, as returned by CPXcreateprob.
redlp_p
A pointer to receive the problem object pointer that results when presolve has been applied to the LP problem object.
Returns

The routine returns zero on success and nonzero if an error occurs.
CPXgetsolvecallbackfunc

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
void CPXPUBLIC CPXgetsolvecallbackfunc(CPXCENVptr env,
                                      int (CPXPUBLIC **solvecallback_p)(CALLBACK_SOLVE_ARGS) ,
                                      void ** cbhandle_p)
```

Description:

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetsolvecallbackfunc` accesses the user-written callback to be called during MIP optimization to optimize the subproblem.

**Example**

```c
CPXgetsolvecallbackfunc(env, &current_callback,
&current_cbdata);
```

See also *Advanced MIP Control Interface* in the *ILOG CPLEX User's Manual*.

For documentation of callback arguments, see the routine `CPXsetsolvecallbackfunc`.

**Parameters**

- **env**
  A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.

- **solvecallback_p**
  The address of the pointer to the current user-written solve callback. If no callback has been set, the pointer evaluates to NULL.

- **cbhandle_p**
  The address of a variable to hold the user's private pointer.
See Also

CPXgetcallbacknodelp, CPXsetsolvecallbackfunc

Returns

This routine does not return a result.
**CPXkilldnorms**

**Category**   
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
void CPXPUBLIC CPXkilldnorms(CPXLPtr lp)

**Description**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXkilldnorms deletes any dual steepest-edge norms that have been retained relative to an active basis. If the user believes that the values of these norms may be significantly in error, and the setting of the CPX_PARAM_DPRIIND parameter is CPX_DPRIIND_STEEP or CPX_DPRIIND_FULLSTEEP, calling CPXkilldnorms means that fresh dual steepest-edge norms will be computed on the next call to CPXdualopt.

**Parameters**

lp  
The pointer to a CPLEX LP problem object, as returned by CPXcreateprob.
CPXkillpnorms

Category                        Global Function
Definition File                cplex.h
Include Files                  cplex.h
Synopsis                        void CPXPUBLIC CPXkillpnorms(CPXLPptr lp)
Description

Note: This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXkillpnorms deletes any primal steepest-edge norms that have been retained relative to an active basis. If the user believes that the values of these norms may be significantly in error, and the setting of the CPX_PARAM_PPRIIND parameter is CPX_PPRIIND_STEEP, calling CPXkillpnorms means that fresh primal steepest-edge norms will be computed on the next call to CPXprimopt.

Parameters

lp

A pointer to a CPLEX LP problem object, as returned by CPXcreateprob.
CPXmdleave

Category          Global Function
Definition File   cplex.h
Include Files     cplex.h
Synopsis
int CPXPUBLIC CPXmdleave(CPXENVptr env, 
                         CPXLPptr lp, 
                         const int * goodlist, 
                         int goodlen, 
                         double * downratio, 
                         double * upratio)

Description

Note: This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXmdleave assumes that there is a resident optimal simplex basis, and a resident LU-factorization associated with this basis. It takes as input a list of basic variables as specified by goodlist[] and goodlen, and returns values commonly known as Driebeck penalties in the two arrays downratio[] and upratio[].

For a given j = goodlist[i], downratio[i] has the following meaning. Let $x_j$ be the name of the basic variable with index j, and suppose that $x_j$ is fixed to some value $t' < t$. In a subsequent call to CPXdualopt, the leaving variable in the first iteration of this call is uniquely determined: It must be $x_j$.

There are then two possibilities. Either an entering variable is determined, or it is concluded (in the first iteration) that the changed model is dual unbounded (primal infeasible). In the latter case, downratio[i] is set equal to a large positive value (this number is system dependent, but is usually $1.0E+75$). In the former case, where $r$ is the value of the objective function after this one iteration, downratio[i] is determined by $|r| = (t - t') * \text{downratio[i]}$.

Parameters

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.
lp
A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.

**goodlist**

An array of integers that must be of length at least `goodlen`. The entries in `goodlist[]` must all be indices of current basic variables. Moreover, these indices must all be indices of original model variables; that is, they must all take values smaller than the number of columns in the model as returned by `CPXgetnumcols`. Negative indices and indices bigger than or equal to `CPXgetnumcols` result in an error.

**goodlen**

An integer indicating the number of entries in `goodlist[]`. If `goodlen < 0`, an error is returned.

**downratio**

An array of type `double` that must be of length at least `goodlen`.

**upratio**

An array of type `double` that must be of length at least `goodlen`.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXpivot

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis: int CPXPUBLIC CPXpivot(CPXENVptr env, CPXLPptr lp, int jenter, int jleave, int leavestat)

Description:

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXpivot performs a basis change where variable *jenter* replaces variable *jleave* in the basis.

Use the constant CPX_NO_VARIABLE for *jenter* or for *jleave* if you want ILOG CPLEX to determine one of the two variables involved in the basis change.

It is invalid to pass a basic variable for *jenter*. Also, no nonbasic variable may be specified for *jleave*, except for *jenter == jleave* when the variable has both finite upper and lower bounds. In that case the variable is moved from the current to the other bound. No shifting or perturbation is performed.

**Example**

```c
status = CPXpivot (env, lp, jenter, jleave, CPX_AT_LOWER);
```

Parameters:

- **env**
  A pointer to the CPLEX environment, as returned by the CPXopenCPLEX.

- **lp**
  A pointer to a CPLEX LP problem object, as returned by CPXcreateprob.

- **jenter**
An index indicating the variable to enter the basis. The slack or artificial variable for row i is denoted by \( j_{\text{enter}} = -i-1 \). The argument \( j_{\text{enter}} \) must either identify a nonbasic variable or take the value \( \text{CPX\_NO\_VARIABLE} \). When \( j_{\text{enter}} \) is set to \( \text{CPX\_NO\_VARIABLE} \), ILOG CPLEX will use the leaving variable \( j_{\text{leave}} \) to perform a dual simplex method ratio test that determines the entering variable.

\( j_{\text{leave}} \)

An index indicating the variable to leave the basis. The slack or artificial variable for row i is denoted by \( j_{\text{enter}} = -i-1 \). The argument \( j_{\text{leave}} \) typically identifies a basic variable. However, if \( j_{\text{enter}} \) denotes a variable with finite upper and lower bounds, \( j_{\text{leave}} \) may be set to \( j_{\text{enter}} \) to indicate that the variable moves from its current bound to the other. The argument \( j_{\text{leave}} \) may also be set to \( \text{CPX\_NO\_VARIABLE} \). In that case, ILOG CPLEX will use the incoming variable \( j_{\text{enter}} \) to perform a primal simplex method ratio test that determines the leaving variable.

\( \text{leavestat} \)

An integer indicating the nonbasic status to be assigned to the leaving variable after the basis change. This is important for the case where \( j_{\text{leave}} \) indicates a variable with finite upper and lower bounds, as it may become nonbasic at its lower or upper bound.

**Example**

```c
status = CPXpivot (env, lp, jenter, jleave, CPX_AT_LOWER);
```

**Returns**

This routine returns zero on success and nonzero if an error occurs.
**CPXpivotin**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**
```c
int CPXPUBLIC CPXpivotin(CPXENVptr env, 
CPXLPptr lp, 
const int * rlist, 
int rlen)
```

**Description**

The routine `CPXpivotin` forcibly pivots slacks that appear on a list of inequality rows into the basis. If equality rows appear among those specified on the list, they are ignored.

**Motivation**

In the implementation of cutting-plane algorithms for integer programming, it is occasionally desirable to delete some of the added constraints (that is, cutting planes) when they no longer appear to be useful. If the slack on some such constraint (that is, row) is not in the resident basis, the deletion of that row may destroy the quality of the basis. Pivoting the slack in before the deletion avoids that difficulty.

**Dual Steepest-Edge Norms**

If one of the dual steepest-edge algorithms is in use when this routine is called, the corresponding norms are automatically updated as part of the pivot. (Primal steepest-edge norms are not automatically updated in this way because, in general, the deletion of rows invalidates those norms.)

**Parameters**

- **env**
  The pointer to the ILOG CPLEX environment, as returned by `CPXopenCPLEX`.

- **ip**
  A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.

- **rlist**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.
An array of length \( rlen \), containing distinct row indices of slack variables that are not basic in the current solution. If \( rlist[] \) contains negative entries or entries exceeding the number of rows, CPXpivotin returns an error code. Entries of nonslack rows are ignored.

\( rlen \)

An integer that indicates the number of entries in the array \( rlist[] \). If \( rlen \) is negative or greater than the number of rows, CPXpivotin returns an error code.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXpivotout

Category                  Global Function
Definition File           cplex.h
Include Files            cplex.h
Synopsis                  int CPXPUBLIC CPXpivotout(CPXENVptr env,
                           CPXLPptr lp,
                           const int * clist,
                           int clen)

Description

Note: This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXpivotout pivots a list of fixed variables out of the resident basis. Variables are fixed when the absolute difference between the lower and upper bounds is at most 1.0e-10.

Parameters

env
The pointer to the ILOG CPLEX environment, as returned by CPXopenCPLEX.

ip
A pointer to a CPLEX LP problem object, as returned by CPXcreateprob.

clist
An array of length clen, containing the column indices of the variables to be pivoted out of the basis. If any of these variables is not fixed, CPXpivotout returns an error code.

clen
An integer that indicates the number of entries in the array clist[].

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXpreaddrows

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXpreaddrows (CPXCENVptr env,
                         CPXLPptr lp,
                         int rcnt,
                         int nzcnt,
                         const double * rhs,
                         const char * sense,
                         const int * rmatbeg,
                         const int * rmatind,
                         const double * rmatval,
                         char ** rowname)
```

Description:

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXpreaddrows is used to add rows to an LP problem object and its associated presolved LP problem object. Note that the CPLEX parameter CPX_PARAM_REDUCE must be set to CPX_PREREDEUCE_PRIMALONLY (1) or CPX_PREREDEUCE_NOPRIMALORDUAL (0) at the time of the presolve in order to add rows and preserve the presolved problem. This routine should be used in place of CPXaddrows when you want to preserve the presolved problem.

The arguments of CPXpreaddrows are the same as those of CPXaddrows, with the exception that new columns may not be added, so there are no ccnt and colname arguments. The new rows are added to both the original LP problem object and the associated presolved LP problem object.

**Examples:**

```c
status = CPXpreaddrows (env, lp, rcnt, nzcnt, rhs, sense, rmatbeg, rmatind, rmatval, newrowname);
```
See also the example adpreex1.c in the standard distribution.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
**CPXprechgobj**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXprechgobj(CPXENVptr env,  
                           CPXLPptr lp,  
                           int cnt,  
                           const int * indices,  
                           const double * values)
```

**Description**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXprechgobj is used to change the objective function coefficients of an LP problem object and its associated presolved LP problem object. Note that the CPLEX parameter CPX_PARAM_REDUCE must be set to CPX_PREREDECE_PRIMALONLY (1) or CPX_PREREDECE_NOPRIMALORDUAL (0) at the time of the presolve in order to change objective coefficients and preserve the presolved problem. This routine should be used in place of CPXchgobj when it is desired to preserve the presolved problem.

The arguments and operation of CPXprechgobj are the same as those of CPXchgobj. The objective coefficient changes are applied to both the original LP problem object and the associated presolved LP problem object.

**Example**

```c
status = CPXprechgobj (env, lp, objcnt, objind, objval);
```

See also the example adpreex1.c in the standard distribution.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
**CPXpresolve**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXpresolve(CPXENVptr env,
            CPXLPptr lp,
            int method)
```

**Description**

The routine CPXpresolve performs LP or MIP presolve depending whether a problem object is an LP or a MIP. If the problem is already presolved, the existing presolved problem is freed, and a new presolved problem is created.

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

**Example**

```c
status = CPXpresolve (env, lp, CPX_ALG_DUAL);
```

**Parameters**

- **env**
  A pointer to the CPLEX environment, as returned by CPXopenCPLEX.
- **lp**
  A pointer to a CPLEX LP problem object, as returned by CPXcreateprob.
- **method**
  An integer specifying the optimization algorithm to be used to solve the problem after the presolve is completed. Some presolve reductions are specific to an optimization algorithm, so specifying the algorithm makes sure that the problem is presolved for that algorithm, and that presolve does not have to be repeated when that optimization routine is called. Possible values are CPX_ALG_NONE, CPX_ALG_PRIMAL, CPX_ALG_DUAL, and CPX_ALG_BARRIER for LP; CPX_ALG_NONE should be used for MIP.
Returns

The routine returns zero on success and nonzero if an error occurs.
CPXqconstrslackfromx

Category          Global Function
Definition File   cplex.h
Include Files     cplex.h
Synopsis
int CPXPUBLIC CPXqconstrslackfromx(CPXCENVptr env,
                                   CPXCLPptr lp,
                                   const double * x,
                                   double * qcslack)

Description

Note: This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXqconstrslackfromx computes an array of slack values for quadratic constraints from primal solution values.

Example

status = CPXqconstrslackfromx (env, lp, x, qcslack);

Parameters

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

lp
A pointer to a CPLEX LP problem object, as returned by CPXcreateprob.

x
An array that contains primal solution (x) values for the problem, as returned by routines such as CPXcrushx and CPXuncrushx. The array must be of length at least the number of columns in the LP problem object.

qcslack
An array to receive the quadratic constraint slack values computed from the x values for the problem object. The array must be of length at least the number of quadratic constraints in the LP problem object.
Returns

The routine returns zero on success and nonzero if an error occurs.
**CXPqpdjfrompi**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**
```c
int CPXPUBLIC CXPqpdjfrompi(CPXENVptr env,
                           CPXCLPptr lp,
                           const double * pi,
                           const double * x,
                           double * dj)
```

**Description**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CXPqpdjfrompi computes an array of reduced costs from an array of dual values for a QP.

**Example**
```c
status = CXPqpdjfrompi (env, lp, origpi, reducepi);
```

**Parameters**

**env**
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

**lp**
A pointer to a CPLEX LP problem object, as returned by CPXcreateprob.

**pi**
An array that contains dual solution (pi) values for a problem, as returned by such routines as CXPqpuncrushpi and CXPcrushpi. The length of the array must at least equal the number of rows in the LP problem object.

**x**
An array that contains primal solution \((x)\) values for a problem, as returned by such routines as \texttt{CPXuncrushx} and \texttt{CPXcrushx}. The length of the array must at least equal the number of columns in the LP problem object.

\(dj\)

An array to receive the reduced cost values computed from the \(pi\) values for the problem object. The length of the array must at least equal the number of columns in the problem object.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXqpuncrushpi

Category: Global Function

Definition File: cplex.h

Include Files: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXqpuncrushpi(CPXENVptr env,
        CPXCLPptr lp,
        double * pi,
        const double * prepi,
        const double * x)
```

Description:

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXqpuncrushpi uncrushes a dual solution for the presolved problem to a dual solution for the original problem if the original problem is a QP.

**Example**

```c
status = CPXqpuncrushpi (env, lp, pi, prepi, x);
```

Parameters:

- **env**
  A pointer to the CPLEX environment, as returned by CPXopenCPLEX.
- **lp**
  A pointer to a CPLEX LP problem object, as returned by CPXcreateprob.
- **pi**
  An array to receive dual solution (pi) values for the original problem as computed from the dual values of the presolved problem object. The length of the array must at least equal the number of rows in the LP problem object.
- **prepi**
  An array that contains dual solution (pi) values for the presolved problem, as returned by such routines as CPXgetpi and CPXsolution when applied to the presolved
problem object. The length of the array must at least equal the number of rows in the presolved problem object.

\( \mathbf{x} \)

An array that contains primal solution (\( \mathbf{x} \)) values for a problem, as returned by such routines as CPXuncrushx and CPXcrushx. The length of the array must at least equal the number of columns in the LP problem object.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
**CPXsetbranchcallbackfunc**

**Category**
Global Function

**Definition File**
cplex.h

**Include Files**
cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXsetbranchcallbackfunc (CPXENVptr env,  
    int (CPXPUBLIC *branchcallback)(CALLBACK_BRANCH_ARGS) ,  
    void * cbhandle)
```

**Description**

The routine `CPXsetbranchcallbackfunc` sets and modifies the user-written callback routine to be called after a branch has been selected but before the branch is carried out during MIP optimization. In the callback routine, the CPLEX-selected branch can be changed to a user-selected branch.

**Example**

```c
status = CPXsetbranchcallbackfunc (env, mybranchfunc, mydata);
```

See also the example `admipex1.c` in the standard distribution.

**Parameters**

- **env**
  A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.

- **branchcallback**
  A pointer to a user-written branch callback. If the callback is set to NULL, no callback can be called during optimization.

- **cbhandle**
  A pointer to user private data. This pointer is passed to the callback.

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.
Callback description

```c
int callback (CPXENVptr env,
             void       *cbdata,
             int        wherefrom,
             void       *cbhandle,
             int        type,
             int        sos,
             int        nodecnt,
             int        bdcnt,
             double     *nodeest,
             int        *nodebeg,
             int        *indices,
             char       *lu,
             int        *bd,
             int        *useraction_p);
```

The call to the branch callback occurs after a branch has been selected but before the branch is carried out. This function is written by the user. On entry to the callback, the ILOG CPLEX-selected branch is defined in the arguments. The arguments to the callback specify a list of changes to make to the bounds of variables when child nodes are created. One, two, or zero child nodes can be created, so one, two, or zero lists of changes are specified in the arguments. The first branch specified is considered first. The callback is called with zero lists of bound changes when the solution at the node is integer feasible. ILOG CPLEX occasionally elects to branch by changing a number of variables bounds or by adding constraints to the node subproblem; the branch type is then CPX_TYPE_ANY. The details of the constraints added for a CPX_TYPE_ANY branch are not available to the user.

Custom branching strategies can be implemented by calling the CPLEX routine CPXbranchcallbackbranchbds, CPXbranchcallbackbranchconstraints, or CPXbranchcallbackbranchgeneral and setting the useraction variable to CPX_CALLBACK_SET. Then CPLEX will carry out these branches instead of the CPLEX-selected branches.

Branch variables are in terms of the original problem if the parameter CPX_PARAM_MIPCBREDLP is set to CPX_OFF before the call to CPXmipopt that calls the callback. Otherwise, branch variables are in terms of the presolved problem.

Callback return value

The callback returns zero on success and nonzero if an error occurs.

Callback arguments

- `env`
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbdata

A pointer passed from the optimization routine to the user-written callback that identifies the problem being optimized. The only purpose of this pointer is to pass it to the callback information routines.

wherefrom

An integer value indicating where in the optimization this function was called. It will have the value CPX_CALLBACK_MIP_BRANCH.

cbhandle

A pointer to user-private data.

int type

An integer that indicates the type of branch. This table summarizes possible values.

<table>
<thead>
<tr>
<th>Symbolic Constant</th>
<th>Value</th>
<th>Branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_TYPE_VAR</td>
<td>'0'</td>
<td>variable branch</td>
</tr>
<tr>
<td>CPX_TYPE_SOS1</td>
<td>'1'</td>
<td>SOS1 branch</td>
</tr>
<tr>
<td>CPX_TYPE_SOS2</td>
<td>'2'</td>
<td>SOS2 branch</td>
</tr>
<tr>
<td>CPX_TYPE_ANY</td>
<td>'A'</td>
<td>multiple bound changes and/or constraints will be used for branching</td>
</tr>
</tbody>
</table>

sos

An integer that indicates the special ordered set (SOS) used for this branch. A value of -1 indicates that this branch is not an SOS-type branch.

nodecnt

An integer that indicates the number of nodes CPLEX will create from this branch. Possible values are:

◆ 0 (zero), or
◆ 1, or
◆ 2.

If the argument is 0, the node will be fathomed unless user-specified branches are made; that is, no child nodes are created and the node itself is discarded.

dcnt
An integer that indicates the number of bound changes defined in the arrays `indices`, `lu`, and `bd` that define the CPLEX-selected branch.

`nodeest`

An array with `nodecnt` entries that contains estimates of the integer objective-function value that will be attained from the created node.

`nodebeg`

An array with `nodecnt` entries. The i-th entry is the index into the arrays `indices`, `lu`, and `bd` of the first bound changed for the i-th node.

`indices`

Together with `lu` and `bd`, this array defines the bound changes for each of the created nodes. The entry `indices[i]` is the index for the variable.

`lu`

Together with `indices` and `bd`, this array defines the bound changes for each of the created nodes. The entry `lu[i]` is one of the three possible values indicating which bound to change:

- 'L' for lower bound, or
- 'U' for upper bound, or
- 'B' for both bounds.

`bd`

Together with `indices` and `lu`, this array defines the bound changes for each of the created nodes. The entry `bd[i]` indicates the new value of the bound.

`useraction_p`

A pointer to an integer indicating the action for ILOG CPLEX to take at the completion of the user callback. The table summarizes the possible actions.

### Actions to be Taken After a User-Written Branch Callback

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CPX_CALLBACK_DEFAULT</td>
<td>Use CPLEX-selected branch</td>
</tr>
<tr>
<td>1</td>
<td>CPX_CALLBACK_FAIL</td>
<td>Exit optimization</td>
</tr>
<tr>
<td>2</td>
<td>CPX_CALLBACK_SET</td>
<td>Use user-selected branch, as defined by calls to CPXbranchcallbackbranch bds</td>
</tr>
</tbody>
</table>

### Returns

The routine returns zero on success and nonzero if an error occurs.
**CPXsetbranchnosolncallbackfunc**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXsetbranchnosolncallbackfunc(CPXENVptr env,
int(CPXPUBLIC *branchnosolncallback)(CALLBACK_BRANCH_ARGS) ,
void * cbhandle)
```

**Description**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXsetbranchnosolncallbackfunc sets the callback function that will be called instead of the branch callback when there is a failure due to such situations as an iteration limit being reached, unboundedness being detected, numeric difficulties being encountered, while the node LP is being solved. In consequence of the failure, whether the node is feasible or infeasible cannot be known and thus CPLEX routines such as CPXsolution may fail. In this situation, CPLEX will attempt to fix some variables and continue.

These conditions are rare (except when the user has set a very low iteration limit), so it is acceptable to let CPLEX follow its default action in these cases.
**CPXsetcutcallbackfunc**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXsetcutcallbackfunc(CPXENVptr env,
    int(CPXPUBLIC *cutcallback)(CALLBACK_CUT_ARGS) ,
    void * cbhandle)
```

**Description**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine **CPXsetcutcallbackfunc** sets and modifies the user-written callback for adding cuts. The user-written callback is called by ILOG CPLEX during MIP branch & cut for every node that has an LP optimal solution with objective value below the cutoff and is integer infeasible. The callback routine adds globally valid cuts to the LP subproblem. The cut may be for the original problem if the parameter CPX_PARAM_MIPCBREDLP was set to CPX_OFF before the call to CPXmipopt that calls the callback. Otherwise, the cut is for the presolved problem.

Within the user-written cut callback, the routine **CPXgetcallbacknodelp** and other query routines from the Callable Library access information about the subproblem. The routines **CPXgetcallbacknodeintfeas** and **CPXgetcallbackssosinfo** examines the status of integer entities.

The routine **CPXcutcallbackadd** adds cuts to the problem. Cuts added to the problem are first put into a cut pool, so they are not present in the subproblem LP until after the user-written cut callback is finished.

Any cuts that are duplicates of cuts already in the subproblem are not added to the subproblem. Cuts that are added remain part of all subsequent subproblems; there is no cut deletion.

If cuts have been added, the subproblem is re-solved and evaluated, and, if the LP solution is still integer infeasible and not cut off, the cut callback is called again.
If the problem has names, user-added cuts have names of the form \texttt{X\text{number}} where \texttt{number} is a sequence number among all cuts generated.

The parameter \texttt{CPX\_PARAM\_REDUCE} must be set to \texttt{CPX\_PREREDUCE\_PRIMALONLY} (1) or \texttt{CPX\_PREREDUCE\_NOPRIMALORDUAL} (0) if the constraints to be added in the callback are lazy constraints, that is, not implied by the constraints in the constraint matrix. The parameter \texttt{CPX\_PARAM\_PRELINEAR} must be set to 0 if the constraints to be added are in terms of the original problem and the constraints are valid cutting planes.

**Example**

```c
status = CPXsetcutcallbackfunc(env, mycutfunc, mydata);
```

See also the example \texttt{admipex5.c} in the standard distribution.

**Parameters**

\texttt{env}

A pointer to the CPLEX environment, as returned by \texttt{CPXopenCPLEX}.

\texttt{cutcallback}

The pointer to the current user-written cut callback. If no callback has been set, the pointer evaluates to \texttt{NULL}.

\texttt{cbhandle}

A pointer to user private data. This pointer is passed to the user-written cut callback.

**Callback description**

```c
int callback (CPXENVptr env,
              void      *cbdata,
              int       wherefrom,
              void      *cbhandle,
              int       *useraction_p);
```

ILOG CPLEX calls the cut callback when the LP subproblem for a node has an optimal solution with objective value below the cutoff and is integer infeasible.

**Callback return value**

The callback returns zero on success and nonzero if an error occurs.

**Callback arguments**

\texttt{env}
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbdata
A pointer passed from the optimization routine to the user-written callback that identifies the problem being optimized. The only purpose of this pointer is to pass it to the callback information routines.

wherefrom
An integer value indicating where in the optimization this function was called. It has the value CPX_CALLBACK_MIP_CUT.

cbhandle
A pointer to user private data.

useraction_p
A pointer to an integer indicating the action for ILOG CPLEX to take at the completion of the user callback. The table summarizes possible actions.

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CPX_CALLBACK_DEFAULT</td>
<td>Use cuts as added</td>
</tr>
<tr>
<td>1</td>
<td>CPX_CALLBACK_FAIL</td>
<td>Exit optimization</td>
</tr>
<tr>
<td>2</td>
<td>CPX_CALLBACK_SET</td>
<td>Use cuts as added</td>
</tr>
</tbody>
</table>

Returns
The routine returns zero on success and nonzero if an error occurs.
CPXsetdeletenodecallbackfunc

Category: Global Function
Definition File: cplex.h

Synopsis:

```c
int CPXPUBLIC CPXsetdeletenodecallbackfunc(CPXENVptr env,
                                           void(CPXPUBLIC *deletecallback)(CALLBACK_DELETENODE_ARGS) ,
                                           void * cbhandle)
```

Description:

```
Note: This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.
```

The routine CPXsetdeletenodecallbackfunc sets and modifies the user-written callback to be called during MIP optimization when a node is to be deleted. Nodes are deleted in these circumstances:

- when a branch is carried out from that node, or
- when the node relaxation is infeasible, or
- when the node relaxation objective value is worse than the cutoff.

Example:

```c
status = CPXsetdeletenodecallbackfunc (env,
                                         mybranchfunc,
                                         mydata);
```

See also the example admipex1.c in the standard distribution.

Parameters:

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

deletecallback
A pointer to a user-written branch callback. If the callback is set to NULL, no callback is called during optimization.
cbhandle
A pointer to user private data. This pointer is passed to the callback.

Callback description

```c
int callback (CPXENVptr env,
    void      *cbdata,
    int       wherefrom,
    void      *cbhandle,
    int       seqnum,
    void      *handle);
```

The call to the delete node callback routine occurs during MIP optimization when a node is to be deleted.

The main purpose of the callback is to provide an opportunity to free any user data associated with the node, thus preventing memory leaks.

Callback return value
The callback returns zero on success and nonzero if an error occurs.

Callback arguments

env
A pointer to the CPLEX environment, as returned by one of the CPXopenCPLEX routines.

cbdata
A pointer passed from the optimization routine to the user-written callback that identifies the problem being optimized. The only purpose of this pointer is to pass it to the callback information routines.

wherefrom
An integer value indicating where in the optimization this function was called. It will have the value CPX_CALLBACK_MIP_DELETENODE.

cbhandle
A pointer to user private data.

seqnum
The sequence number of the node that is being deleted.

handle
A pointer to the user private data that was assigned to the node when it was created with one of the callback branching routines:

- CPXbranchcallbackbranchbds, or
- CPXbranchcallbackbranchconstraints, or
- CPXbranchcallbackbranchgeneral.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXsetheuristiccallbackfunc

Category                       Global Function
Definition File                cplex.h
Include Files                  cplex.h

Synopsis

\[
\text{int CPXPUBLIC CPXsetheuristiccallbackfunc}(\text{CPXENVptr env, int CPAUBLIC *heuristiccallback)(CALLBACK_HEURISTIC_ARGS) , void * cbhandle})
\]

Description

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXsetheuristiccallbackfunc` sets or modifies the user-written callback to be called by ILOG CPLEX during MIP optimization after the subproblem has been solved to optimality. That callback is not called when the subproblem is infeasible or cut off. The callback supplies ILOG CPLEX with heuristically-derived integer solutions.

If a linear program must be solved as part of a heuristic callback, make a copy of the node LP and solve the copy, not the CPLEX node LP.

**Example**

\[
\text{status} = \text{CPXsetheuristiccallbackfunc}(\text{env, myheuristicfunc, mydata});
\]

See also the example `admipex2.c` in the standard distribution.

**Parameters**

- `env`
  A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
- `heuristiccallback`
  A pointer to a user-written heuristic callback. If this callback is set to NULL, no callback is called during optimization.
- `cbhandle`
A pointer to the user's private data. This pointer is passed to the callback.

**Callback description**

```c
int callback (CPXENVptr env,
              void      *cbdata,
              int       wherefrom,
              void      *cbhandle,
              double    *objval_p,
              double    *x,
              int       *checkfeas_p,
              int       *useraction_p);
```

The call to the heuristic callback occurs after an optimal solution to the subproblem has been obtained. The user can provide that solution to start a heuristic for finding an integer solution. The integer solution provided to ILOG CPLEX replaces the incumbent if it has a better objective value. The basis that is saved as part of the incumbent is the optimal basis from the subproblem; it may not be a good basis for starting optimization of the fixed problem.

The integer solution returned to CPLEX is for the original problem if the parameter `CPX_PARAM_MIPCBREDLP` was set to `CPX_OFF` before the call to `CPXmipopt` that calls the callback. Otherwise, it is for the presolved problem.

**Callback return value**

The callback returns zero on success and nonzero if an error occurs.

**Callback arguments**

- `env`: A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
- `cbdata`: A pointer passed from the optimization routine to the user-written callback to identify the problem being optimized. The only purpose of the `cbdata` pointer is to pass it to the callback information routines.
- `wherefrom`: An integer value indicating at which point in the optimization this function was called. It has the value `CPX_CALLBACK_MIP_HEURISTIC` for the heuristic callback.
- `cbhandle`: A pointer to user private data.
- `objval_p`:
A pointer to a variable that on entry contains the optimal objective value of the subproblem and on return contains the objective value of the integer solution found, if any.

x

An array that on entry contains primal solution values for the subproblem and on return contains solution values for the integer solution found, if any.

checkfeas_p

A pointer to an integer that indicates whether or not ILOG CPLEX should check the returned integer solution for integer feasibility. The solution is checked if checkfeas_p is nonzero. When the solution is checked and found to be integer infeasible, it is discarded, and optimization continues.

useraction_p

A pointer to an integer to contain the indicator for the action to be taken on completion of the user callback. The table summarizes the possible values.

### Actions to be Taken after a User-Written Heuristic Callback

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CPX_CALLBACK_DEFAULT</td>
<td>No solution found</td>
</tr>
<tr>
<td>1</td>
<td>CPX_CALLBACK_FAIL</td>
<td>Exit optimization</td>
</tr>
<tr>
<td>2</td>
<td>CPX_CALLBACK_SET</td>
<td>Use user solution as indicated in return values</td>
</tr>
</tbody>
</table>

### Returns

The routine returns zero on success and nonzero if an error occurs.
**CPXsetincumbentcallbackfunc**

**Category**  Global Function

**Definition File**  cplex.h

**Include Files**  cplex.h

**Synopsis**

```c
int CPXPUBLIC CPXsetincumbentcallbackfunc(CPXENVptr env,
    int(CPXPUBLIC *incumbentallback)(CALLBACK_INCUMBENT_ARGS) ,
    void * cbhandle)
```

**Description**

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXsetincumbentcallbackfunc sets and modifies the user-written incumbent callback routine to be called when an integer solution has been found but before this solution replaces the incumbent. This callback can be used to discard solutions that do not meet criteria beyond that of the mixed integer programming formulation.

Variables are in terms of the original problem if the parameter CPX_PARAM_MIPCBREDLP is set to CPX_OFF before the call to CPXmipopt that calls the callback. Otherwise, variables are in terms of the presolved problem.

**Example**

```c
status = CPXsetincumbentcallbackfunc (env, myincumbentcheck, mydata);
```

See also *Advanced MIP Control Interface* in the *ILOG CPLEX User's Manual*.

**Parameters**

**env**

A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

**incumbentcallback**

A pointer to a user-written incumbent callback. If the callback is set to NULL, no callback can be called during optimization.
cbhandle

A pointer to user private data. This pointer is passed to the callback.

**Callback description**

```c
int callback (CPXENVptr env,
void      *cbdata,
int       wherefrom,
void      *cbhandle,
double    objval,
double    *x,
int       *isfeas_p,
int       *useraction_p);
```

The incumbent callback is called when CPLEX has found an integer solution, but before this solution replaces the incumbent integer solution.

Variables are in terms of the original problem if the parameter CPX_PARAM_MIPCBREDLP is set to CPX_OFF before the call to CPXmipopt that calls the callback. Otherwise, variables are in terms of the presolved problem.

**Callback return value**

The callback returns zero on success and nonzero if an error occurs.

**Callback arguments**

`env`

A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

`cbdata`

A pointer passed from the optimization routine to the user-written callback that identifies the problem being optimized. The only purpose of this pointer is to pass it to the callback information routines.

`wherefrom`

An integer value indicating where in the optimization this function was called. It will have the value CPX_CALLBACK_MIP_BRANCH.

`cbhandle`

A pointer to user private data.

`objval`

A variable that contains the objective value of the integer solution.

`x`
An array that contains primal solution values for the integer solution.

isfeas_p

A pointer to an integer variable that indicates whether or not CPLEX should use the integer solution specified in x to replace the current incumbent. A nonzero value indicates that the incumbent should be replaced by x; a zero value indicates that it should not.

useraction_p

A pointer to an integer to contain the indicator for the action to be taken on completion of the user callback. The table summarizes the possible values.

### Actions to be Taken after a User-Written Incumbent Callback

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CPX_CALLBACK_DEFAULT</td>
<td>Proceed with optimization</td>
</tr>
<tr>
<td>1</td>
<td>CPX_CALLBACK_FAIL</td>
<td>Exit optimization</td>
</tr>
<tr>
<td>2</td>
<td>CPX_CALLBACK_SET</td>
<td>Proceed with optimization</td>
</tr>
</tbody>
</table>

**See Also**  
CPXgetincumbentcallbackfunc

**Returns**  
The routine returns zero on success and nonzero if an error occurs.
CPXsetnodecallbackfunc

Category Global Function

Definition File cplex.h

Include Files cplex.h

Synopsis

```c
int CPXPUBLIC CPXsetnodecallbackfunc(CPXENVptr env,
    int(CPXPUBLIC *nodecallback)(CALLBACK_NODE_ARGS) ,
    void * cbhandle)
```

Description

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXsetnodecallbackfunc sets and modifies the user-written callback to be called during MIP optimization after ILOG CPLEX has selected a node to explore, but before this exploration is carried out. The callback routine can change the node selected by ILOG CPLEX to a node selected by the user.

**Example**

```c
status = CPXgetnodecallbackfunc(env, mynodefunc, mydata);
```

See also the example admipex1.c in the standard distribution.

**Parameters**

- **env**
  A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

- **nodecallback**
  A pointer to the current user-written node callback. If no callback has been set, the pointer evaluates to NULL.

- **cbhandle**
  A pointer to user private data. This pointer is passed to the user-written node callback.

**Callback description**
int callback (CPXENVptr env,
       void      *cbdata,
       int       wherefrom,
       void      *cbhandle,
       int       *nodeindex_p,
       int       *useraction_p);

ILOG CPLEX calls the node callback after selecting the next node to explore. The user can choose another node by setting the argument values of the callback.

**Callback return value**

The callback returns zero on success and nonzero if an error occurs.

**Callback arguments**

*env*

A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

*cbdata*

A pointer passed from the optimization routine to the user-written callback that identifies the problem being optimized. The only purpose of this pointer is to pass it to the callback information routines.

*wherefrom*

An integer value indicating where in the optimization this function was called. It has the value CPX_CALLBACK_MIP_NODE.

*cbhandle*

A pointer to user private data.

*nodeindex_p*

A pointer to an integer that indicates the node number of the user-selected node. The node selected by ILOG CPLEX is node number 0 (zero). Other nodes are numbered relative to their position in the tree, and this number changes with each tree operation. The unchanging identifier for a node is its sequence number. To access the sequence number of a node, use the ILOG CPLEX Callable Library routine CPXgetcallbacknodeinfo. An error results if a user attempts to select a node that has been moved to a node file. (See the ILOG CPLEX User's Manual for more information about node files.)

*useraction_p*

A pointer to an integer indicating the action to be taken on completion of the user callback. The table summarizes the possible actions.
Actions to be Taken after a User-Written Node Callback

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CPX_CALLBACK_DEFAULT</td>
<td>Use ILOG CPLEX-selected node</td>
</tr>
<tr>
<td>1</td>
<td>CPX_CALLBACK_FAIL</td>
<td>Exit optimization</td>
</tr>
<tr>
<td>2</td>
<td>CPX_CALLBACK_SET</td>
<td>Use user-selected node as defined in returned values</td>
</tr>
</tbody>
</table>

Returns

The routine returns zero on success and nonzero if an error occurs.
CPXsetsolvecallbackfunc

Category                  Global Function
Definition File           cplex.h
Include Files             cplex.h

Synopsis

int CPXPUBLIC CPXsetsolvecallbackfunc(CPXENVptr env,
                                       intCPXPUBLIC *solvecallback)(CALLBACK_SOLVE_ARGS) ,
                                       void * cbhandle)

Description

Note: This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXsetsolvecallbackfunc sets and modifies the user-written callback to be called during MIP optimization to optimize the subproblem.

Example

status = CPXsetsolvecallbackfunc(env, mysolvefunc, mydata);

See also the example admipex1.c in the standard distribution.

Parameters

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.
solvecallback
A pointer to a user-written solve callback. If the callback is set to NULL, no callback is called during optimization.

cbhandle
A pointer to user private data. This pointer is passed to the callback.

Callback description

int callback (CPXENVptr env,
              void *cbdata,
ILOG CPLEX calls the solve callback before ILOG CPLEX solves the subproblem defined by the current node. The user can choose to solve the subproblem in the solve callback instead by setting the user action parameter of the callback. The optimization that the user provides to solve the subproblem must provide a CPLEX solution. That is, the Callable Library routine CPXgetstat must return a nonzero value. The user may access the lp pointer of the subproblem with the Callable Library routine CPXgetcallbacknodelp.

Callback return value

The callback returns zero on success and nonzero if an error occurs.

Callback arguments

env

A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbdata

A pointer passed from the optimization routine to the user-written callback that identifies the problem being optimized. The only purpose of this pointer is to pass it to the callback information routines.

wherefrom

An integer value indicating where in the optimization this function was called. It will have the value CPX_CALLBACK_MIP_SOLVE.

cbhandle

A pointer to user private data.

useraction_p

A pointer to an integer indicating the to be taken on completion of the user callback. Table 11 summarizes the possible actions.

### Actions to be Taken after a User-Written Solve Callback

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Constant</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CPX_CALLBACK_DEFAULT</td>
<td>Use ILOG CPLEX subproblem optimizer</td>
</tr>
<tr>
<td>1</td>
<td>CPX_CALLBACK_FAIL</td>
<td>Exit optimization</td>
</tr>
</tbody>
</table>
Actions to be Taken after a User-Written Solve Callback

| 2 | CPX_CALLBACK_SET | The subproblem has been solved in the callback |

Returns

The routine returns zero on success and nonzero if an error occurs.
**CPXslackfromx**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXslackfromx(CPXENVptr env,  
   CPXCLPptr lp,  
   const double * x,  
   double * slack)
```

**Description**

The routine `CPXslackfromx` computes an array of slack values from primal solution values.

**Example**

```c
status = CPXslackfromx (env, lp, x, slack);
```

**Parameters**

- **env**
  A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
- **lp**
  A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.
- **x**
  An array that contains primal solution (x) values for the problem, as returned by routines such as `CPXcrushx` and `CPXuncrushx`. The array must be of length at least the number of columns in the LP problem object.
- **slack**
  An array to receive the slack values computed from the x values for the problem object. The array must be of length at least the number of rows in the LP problem object.

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.
Returns

The routine returns zero on success and nonzero if an error occurs.
The routine CPXsolwrite is a generic routine for writing solutions. It performs all the calculations needed to produce a solution file, but it writes only through functions that the user provides to it, so that the user may choose the data representation, data selection, and file format.

The user must open the file before calling CPXsolwrite and close the file after calling CPXsolwrite.

The arguments to CPXsolwrite are functions it calls to write the file. CPXsolwrite does not “know” anything about the file or the type of output being written. The argument info, the last parameter, communicates information to the routines hsection, rsectionbeg, csectionbeg, write_entry, and sectionend.

The function indicated by hsection writes a header section in a formatted file. Its only arguments are the ILOG CPLEX environment pointer env, the problem pointer lp, and the info parameter. CPXsolwrite calls this function first. It uses the lp problem pointer to retrieve any information needed by the header-section function.

The function indicated by rsectionbeg writes information at the beginning of the row section of a formatted file. It is called after hsection and sectionend. Its only argument is the info parameter.
The function indicated by `csection` writes information at the beginning of the column section of a formatted file. It is called after all row entries have been completed. Its only argument is the `info` parameter.

The function indicated by `write_entry` is called once for each row and column in the problem. The following table summarizes its arguments.

The function indicated by `sectionend` is used at the end of each header, row, and column section. The only argument to this function is the `info` parameter.

<table>
<thead>
<tr>
<th>Type?</th>
<th>Name?</th>
<th>Meaning?</th>
</tr>
</thead>
<tbody>
<tr>
<td>void?</td>
<td>*info?</td>
<td>the <code>info</code> parameter of <code>CPXsolwrite</code></td>
</tr>
<tr>
<td>int?</td>
<td>aflag?</td>
<td>a for alternate optimum?</td>
</tr>
<tr>
<td>int?</td>
<td>num?</td>
<td>sequence number; cumulative over rows and columns?</td>
</tr>
<tr>
<td>char?</td>
<td>*name?</td>
<td>name of row or column?</td>
</tr>
<tr>
<td>char?</td>
<td>*state?</td>
<td>state of row or column; one of: UL, LL, BS, EQ, FR, **?</td>
</tr>
<tr>
<td>double?</td>
<td>val1?</td>
<td>for rows, row activity; for columns, column solution value?</td>
</tr>
<tr>
<td>double?</td>
<td>val2?</td>
<td>for rows, slack activity; for columns, objective coefficient?</td>
</tr>
<tr>
<td>double?</td>
<td>ll?</td>
<td>for rows, lower limit; for columns, lower bound?</td>
</tr>
<tr>
<td>double?</td>
<td>ul?</td>
<td>for rows, upper limit; for columns, upper bound?</td>
</tr>
<tr>
<td>double?</td>
<td>val3?</td>
<td>for rows, dual value; for columns, reduced cost?</td>
</tr>
</tbody>
</table>

**Parameters**

- **env**
  - The pointer to the ILOG CPLEX environment, as returned by one of the `CPXopenCPLEX` routines.
- **lp**
  - A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.
- **info**
  - A generic pointer that passes information to each of the functions called by `CPXsolwrite`.
Returns

The routine returns zero on success and nonzero if an error occurs.
CPXstrongbranch

Category  Global Function
Definition File  cplex.h
Include Files  cplex.h
Synopsis  

```
int CPXPUBLIC CPXstrongbranch(CPXCENVptr env,
   CPXLptr lp,
   const int * goodlist,
   int goodlen,
   double * downpen,
   double * uppen,
   int itlim)
```

Description

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXstrongbranch computes information for selecting a branching variable in an integer-programming branch & cut search.

To describe this routine, let’s assume that an LP has been solved and that the optimal solution is resident. Let goodlist[] be the list of variable indices for this problem and goodlen be the length of that list. Then goodlist[] gives rise to 2*goodlen different LPs in which each of the listed variables in turn is fixed to the greatest integer value less than or equal to its value in the current optimal solution, and then each variable is fixed to the least integer value greater than or equal to its value in the current optimal solution. CPXstrongbranch performs at most itlim dual steepest-edge iterations on each of these 2*goodlen LPs, starting from the current optimal solution of the base LP. The values that these iterations yield are placed in the arrays downpen[] for the downward fix and uppen[] for the upward fix. Setting CPX_PARAM_DGRADIENT to 2 may give more informative values for the arguments downpen[] and uppen[] for a given number of iterations itlim.

For a given \( j = \text{goodlist}[i] \), upratio[i] has the following meaning. Let \( x_j \) be the name of the basic variable with index \( j \), and suppose that \( x_j \) is fixed to some value \( t' > t \). Then in a subsequent call to CPXdualopt, the leaving variable in the first iteration of this call is uniquely determined. It must be \( x_j \).
There are then two possibilities. Either an entering variable is determined, or it is concluded (in the first iteration) that the changed model is dual unbounded (primal infeasible). In the latter case, \( \text{upratio}[i] \) is set equal to a large positive value (this number is system dependent, but is usually \( 1.0E+75 \)). In the former case, where \( r \) is the value of the objective function after this one iteration, \( \text{upratio}[i] \) is determined by

\[
|r| = (t' - t) \times \text{upratio}[i]
\]

A user might use other routines of the ILOG CPLEX Callable Library directly to build a function that computes the same values as \text{CPXstrongbranch}. However, \text{CPXstrongbranch} should be faster because it takes advantage of direct access to internal ILOG CPLEX data structures.

**Parameters**

- **env**
  The pointer to the ILOG CPLEX environment, as returned by \text{CPXopenCplex}.

- **lp**
  A pointer to a CPLEX LP problem object, as returned by \text{CPXcreateprob}.

- **goodlist**
  An array of integers. The length of the array must be at least \text{goodlen}. As in other ILOG CPLEX Callable Library routines, row variables in \text{goodlist[0]} are specified by the negative of row index shifted down by one; that is, \(-\text{rowindex} - 1\).

- **goodlen**
  An integer indicating the number of entries in \text{goodlist[0]}.

- **downpen**
  An array containing values that are the result of the downward fix of branching variables in dual steepest-edge iterations carried out by \text{CPXstrongbranch}. The length of the array must be at least \text{goodlen}.

- **uppen**
  An array containing values that are the result of the upward fix of branching variables in dual steepest-edge iterations carried out by \text{CPXstrongbranch}. The length of the array must be at least \text{goodlen}.

- **itlim**
  An integer indicating the limit on the number of dual steepest-edge iterations carried out by \text{CPXstrongbranch} on each LP.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXtightenbds

Category Global Function

Definition File cplex.h

Include Files cplex.h

Synopsis

```c
int CPXPUBLIC CPXtightenbds(CPXENVptr env,
    CPXLPptr lp,
    int cnt,
    const int * indices,
    const char * lu,
    const double * bd)
```

Description

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXtightenbds changes the upper or lower bounds on a set of variables in a problem. Several bounds can be changed at once. Each bound is specified by the index of the variable associated with it. The value of a variable can be fixed at one value by setting both the upper and lower bounds to the same value.

In contrast to the ILOG CPLEX Callable Library routine CPXchgbd, also used to change bounds, CPXtightenbds preserves more of the internal ILOG CPLEX data structures so it is more efficient for re-optimization, particularly when changes are made to bounds on basic variables.

**Bound Indicators in the Argument lu of CPXtightenbds**

<table>
<thead>
<tr>
<th>Value of lu[j]</th>
<th>Meaning for bd[j]</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>bd[j] is an upper bound</td>
</tr>
<tr>
<td>L</td>
<td>bd[j] is a lower bound</td>
</tr>
<tr>
<td>B</td>
<td>bd[j] is the lower and upper bound</td>
</tr>
</tbody>
</table>

Example
status = CPXtightenbds (env, lp, cnt, indices, lu, bd);

**Parameters**

- **env**
  The pointer to the ILOG CPLEX environment, as returned by CPXopenCPLEX.

- **lp**
  A pointer to a CPLEX LP problem object, as returned by CPXcreateprob.

- **cnt**
  An integer indicating the total number of bounds to change. That is, `cnt` specifies the length of the arrays `indices`, `lu`, and `bd`.

- **indices**
  An array containing the numeric indices of the columns corresponding to the variables for which bounds will be changed. The allocated length of the array is `cnt`. Column `j` of the constraint matrix has the internal index `j - 1`.

- **lu**
  An array. This array contains characters indicating whether the corresponding entry in the array `bd` specifies the lower or upper bound on column `indices[j]`. The allocated length of the array is `cnt`. The table summarizes the values that entries in this array may assume.

- **bd**
  An array. This array contains the new values of the upper or lower bounds of the variables present in the array `indices`. The allocated length of the array is `cnt`.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXuncrushform

Category          Global Function
Definition File   cplex.h
Include Files    cplex.h

Synopsis

```
int CPXPUBLIC CPXuncrushform(CPXENVptr env, 
    CPXCLPptr lp, 
    int plen, 
    const int * pind, 
    const double * pval, 
    int * len_p, 
    double * offset_p, 
    int * ind, 
    double * val)
```

Description

This is an advanced routine. Advanced routines typically demand a profound
understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher
risk of incorrect behavior in your application, behavior that can be difficult to
detect. Therefore, ILOG encourages you to consider carefully whether you can
accomplish the same task by means of other Callable Library routines instead.

The routine CPXuncrushform uncrushes a linear formula of the presolved problem to
a linear formula of the original problem.

Let \( \text{cols} = \text{CPXgetnumcols(env, lp)} \). If \( \text{ind}[i] < \text{cols} \) then the \( i \)th variable in the formula is variable with index \( \text{ind}[i] \) in the original problem. If \( \text{ind}[i] \geq \text{cols} \), then the \( i \)th variable in the formula is the slack for the
\((\text{ind}[i] - \text{cols})\)th ranged row. The arrays \text{ind} and \text{val} must be of length at least the
number of columns plus the number of ranged rows in the original LP problem
object.

Example

```
status = CPXuncrushform(env, lp, plen, pind, pval, 
                        &len, &offset, ind, val);
```

Parameters

- env

A pointer to the CPLEX environment, as returned by CPXopenCPLEX.
A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.

The number of entries in the arrays `pind` and `pval`.

The linear formula in terms of the presolved problem. Each entry, `pind[i]`, indicates the column index of the corresponding coefficient, `pval[i]`.

A pointer to an integer to receive the number of nonzero coefficients, that is, the true length of the arrays `ind` and `val`.

A pointer to a double to contain the value of the linear formula corresponding to variables that have been removed in the presolved problem.

The linear formula in terms of the original problem.

The routine returns zero on success and nonzero if an error occurs.
CPXuncrushpi

Category Global Function

Definition File cplex.h

Include Files cplex.h

Synopsis

```c
int CPXPUBLIC CPXuncrushpi (CPXCENVptr env,
    CPXCLPptr lp,
    double * pi,
    const double * prepi)
```

Description

**Note:** This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXuncrushpi uncrushes a dual solution for the presolved problem to a dual solution for the original problem. This routine is for linear programs. Use CPXqpuncrushpi for quadratic programs.

**Example**

```c
status = CPXuncrushpi (env, lp, pi, prepi);
```

**Parameters**

- **env**
  A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

- **lp**
  A pointer to a CPLEX LP problem object, as returned by CPXcreateprob.

- **pi**
  An array to receive dual solution (pi) values for the original problem as computed from the dual values of the presolved problem object. The array must be of length at least the number of rows in the LP problem object.

- **prepi**
  An array that contains dual solution (pi) values for the presolved problem, as returned by routines such as CPXgetpi and CPXsolution when applied to the presolved problem.
problem object. The array must be of length at least the number of rows in the presolved problem object.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
CPXuncrushx

Category            Global Function
Definition File     cplex.h
Include Files       cplex.h
Synopsis             int CPXPUBLIC CPXuncrushx(CPXENVptr env,
                                         CPXCLPptr lp,
                                         double * x,
                                         double * prex)

Description

Note: This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXuncrushx uncrushes a solution for the presolved problem to the solution for the original problem.

Example

status = CPXuncrushx (env, lp, x, prex);

Parameters

env
A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

lp
A pointer to a CPLEX LP problem object, as returned by CPXcreateprob.

x
An array to receive the primal solution ($x$) values for the original problem as computed from primal values of the presolved problem object. The array must be of length at least the number of columns in the LP problem object.

prex
An array that contains primal solution ($x$) values for the presolved problem, as returned by routines such as CPXgetx and CPXsolution when applied to the presolved
problem object. The array must be of length at least the number of columns in the presolved problem object.

**Returns**

The routine returns zero on success and nonzero if an error occurs.
**CPXunscaleprob**

**Category**  
Global Function

**Definition File**  
cplex.h

**Include Files**  
cplex.h

**Synopsis**  
```c
int CPXPUBLIC CPXunscaleprob(CPXENVptr env, CPXLPtr lp)
```

**Description**

This is an advanced routine. Advanced routines typically demand a profound understanding of the algorithms used by ILOG CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, ILOG encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine CPXunscaleprob removes any scaling that ILOG CPLEX has applied to the resident problem and its associated data. A side effect is that if there is a resident solution, any associated factorization is discarded and the solution itself is deactivated, meaning that it can no longer be accessed with a call to CPXsolution, nor by any other query routine. However, any starting point information for the current solution (such as an associated basis) is retained.

**Parameters**

- **env**  
The pointer to the ILOG CPLEX environment, as returned by CPXopenCPLEX.
- **lp**  
A pointer to a CPLEX LP problem object, as returned by CPXcreateprob.

**Returns**  
The routine returns zero on success and nonzero if an error occurs.
## Index

<table>
<thead>
<tr>
<th>C</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX_DUAL_OBJ 1293</td>
<td>CPX_STAT_ABORT_PRIM_OBJ_LIM 1365</td>
</tr>
<tr>
<td>CPX_KAPPA 1294</td>
<td>CPX_STAT_ABORT_TIME_LIM 1366</td>
</tr>
<tr>
<td>CPX_MAX_COMP_SLACK 1295</td>
<td>CPX_STAT_ABORT_USER 1367</td>
</tr>
<tr>
<td>CPX_MAX_DUAL_INFEAS 1296</td>
<td>CPX_STAT_FEASIBLE_RELAXED 1368</td>
</tr>
<tr>
<td>CPX_MAX_DUAL_RESIDUAL 1297</td>
<td>CPX_STAT_INFEASIBLE 1369</td>
</tr>
<tr>
<td>CPX_MAX_INT_INFEAS 1298</td>
<td>CPX_STAT_INFOrUNBD 1370</td>
</tr>
<tr>
<td>CPX_MAX_PI 1299</td>
<td>CPX_STAT_NUM_BEST 1371</td>
</tr>
<tr>
<td>CPX_MAX_PRIMAL_INFEAS 1300</td>
<td>CPX_STAT_OPTIMAL 1372</td>
</tr>
<tr>
<td>CPX_MAX_PRIMAL_RESIDUAL 1301</td>
<td>CPX_STAT_OPTIMAL_FACE_UNBOUNDED 1373</td>
</tr>
<tr>
<td>CPX_MAX_QCPRIMAL_RESIDUAL 1302</td>
<td>CPX_STAT_OPTIMAL_INFEAS 1374</td>
</tr>
<tr>
<td>CPX_MAX_QCSLACK 1303</td>
<td>CPX_STAT_OPTIMAL_RELAXED 1375</td>
</tr>
<tr>
<td>CPX_MAX_QCSLACK_INFEAS 1304</td>
<td>CPX_STAT_UNBOUNDED 1376</td>
</tr>
<tr>
<td>CPX_MAX_RED_COST 1305</td>
<td>CPX_SUM_COMP_SLACK 1311</td>
</tr>
<tr>
<td>CPX_MAX_SCALED_DUAL_INFEAS 1306</td>
<td>CPX_SUM_ICONST_LIM 1322</td>
</tr>
<tr>
<td>CPX_MAX_SCALED_DUAL_RESIDUAL 1307</td>
<td></td>
</tr>
<tr>
<td>CPX_MAX_SCALED_PI 1308</td>
<td></td>
</tr>
<tr>
<td>CPX_MAX_SCALED_PRIMAL_INFEAS 1309</td>
<td></td>
</tr>
<tr>
<td>CPX_MAX_SCALED_PRIMAL_RESIDUAL 1310</td>
<td></td>
</tr>
<tr>
<td>CPX_MAX_SCALED_RED_COST 1311</td>
<td></td>
</tr>
<tr>
<td>CPX_MAX_SCALED_SLACK 1312</td>
<td></td>
</tr>
<tr>
<td>CPX_MAX_SCALED_X 1313</td>
<td></td>
</tr>
<tr>
<td>CPX_MAX_SLACK 1314</td>
<td></td>
</tr>
<tr>
<td>CPX_MAX_X 1315</td>
<td></td>
</tr>
<tr>
<td>CPX_OBJ_GAP 1316</td>
<td></td>
</tr>
<tr>
<td>CPX_PRIMAL_OBJ 1317</td>
<td></td>
</tr>
<tr>
<td>CPX_STAT_ABORT_DUAL_OBJ_LIM 1362</td>
<td></td>
</tr>
<tr>
<td>CPX_STAT_ABORT_IT_LIM 1363</td>
<td></td>
</tr>
<tr>
<td>CPX_STAT_ABORT_OBJ_LIM 1364</td>
<td></td>
</tr>
</tbody>
</table>
INDEX

CPXERR_MSG_NO_CHANNEL 1141
CPXERR_MSG_NO_FILEPTR 1142
CPXERR_MSG_NO_FUNCTION 1143
CPXERR_NAME_CREATION 1144
CPXERR_NAME_NOT_FOUND 1145
CPXERR_NAME_TOO_LONG 1146
CPXERR_NAN 1147
CPXERR_NEED_OPT_SOLN 1148
CPXERR_NEGATIVE_SURPLUS 1149
CPXERR_NET_DATA 1150
CPXERR_NET_FILE_SHORT 1151
CPXERR_NO_BARRIER_SOLN 1168
CPXERR_NO_BASIC_SOLN 1169
CPXERR_NO_BASIS 1170
CPXERR_NO_BOUND_SENSE 1171
CPXERR_NO_BOUND_TYPE 1172
CPXERR_NO_COLUMNS_SECTION 1173
CPXERR_NO_DUAL_SOLN 1174
CPXERR_NO_ENDATA 1175
CPXERR_NO_ENVIRONMENT 1176
CPXERR_NO_FILENAME 1177
CPXERR_NO_ID 1178
CPXERR_NO_ID_FIRST 1179
CPXERR_NO_INT_SOLN 1180
CPXERR_NO_INT_X 1181
CPXERR_NO_LU_FACTOR 1182
CPXERR_NO_MEMORY 1183
CPXERR_NO_MIPSTART 1184
CPXERR_NO_NAME_SECTION 1186
CPXERR_NO_NAMES 1185
CPXERR_NO_NORMS 1187
CPXERR_NO_NUMBER 1188
CPXERR_NO_NUMBER_BOUND 1189
CPXERR_NO_NUMBER_FIRST 1190
CPXERR_NO_OBJ_SENSE 1192
CPXERR_NO_OBJECTIVE 1191
CPXERR_NO_OP_or_SENSE 1194
CPXERR_NO_OPERATOR 1193
CPXERR_NO_ORDER 1195
CPXERR_NO_PROBLEM 1196
CPXERR_NO_QMATRIX_SECTION 1197
CPXERR_NO_QP_OPERATOR 1198
CPXERR_NO_QUAD_EXP 1199
CPXERR_NO_RHS_COEFF 1200
CPXERR_NO_RHS_IN_OBJ 1201
CPXERR_NO_RNGVAL 1202
CPXERR_NO_ROW_NAME 1204
CPXERR_NO_ROW_SENSE 1205
CPXERR_NO_ROWS_SECTION 1203
CPXERR_NO_SENSIT 1206
CPXERR_NO_SOLN 1207
CPXERR_NO_SOS 1208
CPXERR_NO_SOS_SEPARATOR 1209
CPXERR_NO_TREE 1210
CPXERR_NO_VECTOR_SOLN 1211
CPXERR_NODE_INDEX_RANGE 1152
CPXERR_NODE_ON_DISK 1153
CPXERR_NOT_DUAL_UNBOUNDED 1154
CPXERR_NOT_FIXED 1155
CPXERR_NOT_FOR_MIP 1156
CPXERR_NOT_FOR_QCP 1157
CPXERR_NOT_FOR_QP 1158
CPXERR_NOT_MILPCLASS 1159
CPXERR_NOT_MIN_COST_FLOW 1160
CPXERR_NOT_MIP 1161
CPXERR_NOT_MIQPCLASS 1162
CPXERR_NOT_ONE_PROBLEM 1163
CPXERR_NOT_QP 1164
CPXERR_NOT_QPCCLASS 1165
CPXERR_NOT_SAV_FILE 1166
CPXERR_NOT_UNBOUNDED 1167
CPXERR_NULL_NAME 1212
CPXERR_NULL_POINTER 1213
CPXERR_ORDER_BAD_DIRECTION 1214
CPXERR_PARAM_TOO_BIG 1215
CPXERR_PARAM_TOO_SMALL 1216
CPXERR_PRESLV_ABORT 1217
CPXERR_PRESLV_BAD_PARAM 1218
CPXERR_PRESLV_BASIS_MEM 1219
CPXERR_PRESLV_COPYORDER 1220
CPXERR_PRESLV_COPY_SOS 1221
CPXERR_PRESLV_CRUSHFORM 1222
CPXERR_PRESLV_DUAL 1223
CPXERR_PRESLV_FAIL_BASE 1224
CPXERR_PRESLV_INF 1225
CPXERR_PRESLV_INFForUNBD 1226
CPXERR_PRESLV_NO_BASIS 1227
CPXERR_PRESLV_NO_PROB 1228
CPXERR_PRESLV_SOLN_MIP 1229
CPXERR_PRESLV_SOLN_QP 1230
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXERR_PRESLV_START_LP</td>
<td>1231</td>
</tr>
<tr>
<td>CPXERR_PRESLV_UNBD</td>
<td>1232</td>
</tr>
<tr>
<td>CPXERR_PRESLV_UNCRUSHFORM</td>
<td>1233</td>
</tr>
<tr>
<td>CPXERR_PRIND</td>
<td>1234</td>
</tr>
<tr>
<td>CPXERR_PRM_DATA</td>
<td>1235</td>
</tr>
<tr>
<td>CPXERR_PRM_HEADER</td>
<td>1236</td>
</tr>
<tr>
<td>CPXERR_PTHREAD_CREATE</td>
<td>1237</td>
</tr>
<tr>
<td>CPXERR_PTHREAD_MUTEX_INIT</td>
<td>1238</td>
</tr>
<tr>
<td>CPXERR_Q_DIVISOR</td>
<td>1242</td>
</tr>
<tr>
<td>CPXERR_Q_DUP_ENTRY</td>
<td>1243</td>
</tr>
<tr>
<td>CPXERR_Q_NOT_INDEF</td>
<td>1244</td>
</tr>
<tr>
<td>CPXERR_Q_NOT_POS_DEF</td>
<td>1245</td>
</tr>
<tr>
<td>CPXERR_Q_NOT_SYMMETRIC</td>
<td>1246</td>
</tr>
<tr>
<td>CPXERR_QCP_SENSE_FILE</td>
<td>1239</td>
</tr>
<tr>
<td>CPXERR_QUAD_EXP_NOT_2</td>
<td>1240</td>
</tr>
<tr>
<td>CPXERR_QUAD_IN_ROW</td>
<td>1241</td>
</tr>
<tr>
<td>CPXERR_RANGE_SECTION_ORDER</td>
<td>1247</td>
</tr>
<tr>
<td>CPXERR_RESTRICTED_VERSION</td>
<td>1248</td>
</tr>
<tr>
<td>CPXERR_RHS_IN_OBJ</td>
<td>1249</td>
</tr>
<tr>
<td>CPXERR_RIM_REPEATS</td>
<td>1251</td>
</tr>
<tr>
<td>CPXERR_RIM_ROW_REPEATS</td>
<td>1252</td>
</tr>
<tr>
<td>CPXERR_RIMNZ_REPEATS</td>
<td>1250</td>
</tr>
<tr>
<td>CPXERR_ROW_INDEX_RANGE</td>
<td>1253</td>
</tr>
<tr>
<td>CPXERR_ROW_REPEAT_PRINT</td>
<td>1255</td>
</tr>
<tr>
<td>CPXERR_ROW_REPEATS</td>
<td>1254</td>
</tr>
<tr>
<td>CPXERR_ROW_UNKNOWN</td>
<td>1256</td>
</tr>
<tr>
<td>CPXERR_SAV_FILE_DATA</td>
<td>1257</td>
</tr>
<tr>
<td>CPXERR_SAV_FILE_WRITE</td>
<td>1258</td>
</tr>
<tr>
<td>CPXERR_SBASE_ILLEGAL</td>
<td>1259</td>
</tr>
<tr>
<td>CPXERR_SBASE_INCOMPACT</td>
<td>1260</td>
</tr>
<tr>
<td>CPXERR_SEMI_BDS</td>
<td>1261</td>
</tr>
<tr>
<td>CPXERR_SINGULAR</td>
<td>1262</td>
</tr>
<tr>
<td>CPXERR_STR_PARAM_TOO_LONG</td>
<td>1263</td>
</tr>
<tr>
<td>CPXERR_SUBPROB_SOLVE</td>
<td>1264</td>
</tr>
<tr>
<td>CPXERR_TILIM_CONDITION_NO</td>
<td>1265</td>
</tr>
<tr>
<td>CPXERR_TILIM_STRONGBRANCH</td>
<td>1266</td>
</tr>
<tr>
<td>CPXERR_TOO_MANY_COEFS</td>
<td>1267</td>
</tr>
<tr>
<td>CPXERR_TOO_MANY_COLS</td>
<td>1268</td>
</tr>
<tr>
<td>CPXERR_TOO_MANY_RIMNZ</td>
<td>1269</td>
</tr>
<tr>
<td>CPXERR_TOO_MANY_RIMS</td>
<td>1270</td>
</tr>
<tr>
<td>CPXERR_TOO_MANY_ROWS</td>
<td>1271</td>
</tr>
<tr>
<td>CPXERR_TOO_MANY_THREADS</td>
<td>1272</td>
</tr>
<tr>
<td>CPXERR_TRE_FILE_COLS</td>
<td>1274</td>
</tr>
<tr>
<td>CPXERR_TRE_FILE_DATA</td>
<td>1275</td>
</tr>
<tr>
<td>CPXERR_TRE_FILE_FORMAT</td>
<td>1276</td>
</tr>
<tr>
<td>CPXERR_TRE_FILE_INTS</td>
<td>1277</td>
</tr>
<tr>
<td>CPXERR_TRE_FILE_NONZ</td>
<td>1278</td>
</tr>
<tr>
<td>CPXERR_TRE_FILE_OBJ</td>
<td>1279</td>
</tr>
<tr>
<td>CPXERR_TRE_FILE_OBSEN</td>
<td>1280</td>
</tr>
<tr>
<td>CPXERR_TRE_FILE_PRESOLVE</td>
<td>1281</td>
</tr>
<tr>
<td>CPXERR_TRE_FILE_ROWS</td>
<td>1282</td>
</tr>
<tr>
<td>CPXERR_TRE_FILE_TYPES</td>
<td>1283</td>
</tr>
<tr>
<td>CPXERR_TRE_FILE_VERSION</td>
<td>1284</td>
</tr>
<tr>
<td>CPXERR_TRE_FILE_WRITE</td>
<td>1285</td>
</tr>
<tr>
<td>CPXERR_TREE_MEMORY_LIMIT</td>
<td>1273</td>
</tr>
<tr>
<td>CPXERR_UNIQUE_WEIGHTS</td>
<td>1286</td>
</tr>
<tr>
<td>CPXERR_UP_BOUND_REPEATS</td>
<td>1287</td>
</tr>
<tr>
<td>CPXERR_WORK_FILE_OPEN</td>
<td>1288</td>
</tr>
<tr>
<td>CPXERR_WORK_FILE_READ</td>
<td>1289</td>
</tr>
<tr>
<td>CPXERR_WORK_FILE_WRITE</td>
<td>1290</td>
</tr>
<tr>
<td>CPXfclose</td>
<td>241, 912</td>
</tr>
<tr>
<td>CPXfeasopt</td>
<td>242, 649</td>
</tr>
<tr>
<td>CPXfindiis</td>
<td>244, 651</td>
</tr>
<tr>
<td>CPXflushchannel</td>
<td>245, 942</td>
</tr>
<tr>
<td>CPXflushstdchannels</td>
<td>246, 943</td>
</tr>
<tr>
<td>CPXfopen</td>
<td>247, 913</td>
</tr>
<tr>
<td>CPXfputs</td>
<td>248, 914</td>
</tr>
<tr>
<td>CPXfreelaazyconstraints</td>
<td>249</td>
</tr>
<tr>
<td>CPXfreepresolve</td>
<td>1416</td>
</tr>
<tr>
<td>CPXfreeprob</td>
<td>250, 565</td>
</tr>
<tr>
<td>CPXfreesercuts</td>
<td>251</td>
</tr>
<tr>
<td>CPXftran</td>
<td>1417</td>
</tr>
<tr>
<td>CPXgetax</td>
<td>252, 580</td>
</tr>
<tr>
<td>CPXgetbaritcnt</td>
<td>253, 581</td>
</tr>
<tr>
<td>CPXgetbase</td>
<td>254, 582</td>
</tr>
<tr>
<td>CPXgetbasednorms</td>
<td>1419</td>
</tr>
<tr>
<td>CPXgetbestobjval</td>
<td>256, 608</td>
</tr>
<tr>
<td>CPXgetbhead</td>
<td>1421</td>
</tr>
<tr>
<td>CPXgetbranchcallbackfunc</td>
<td>257, 960, 1422</td>
</tr>
<tr>
<td>CPXgetcallbacktype</td>
<td>261, 964, 1426</td>
</tr>
<tr>
<td>CPXgetcallbackglobalhb</td>
<td>263, 966, 1428</td>
</tr>
<tr>
<td>CPXgetcallbackglobalh</td>
<td>265, 968, 1430</td>
</tr>
<tr>
<td>CPXgetcallbackincumbent</td>
<td>267, 970, 1432</td>
</tr>
<tr>
<td>CPXgetcallbackinfo</td>
<td>269, 972</td>
</tr>
<tr>
<td>CPXgetcallbacklp</td>
<td>275, 978, 1434</td>
</tr>
<tr>
<td>CPXgetcallbacknodeinfo</td>
<td>277</td>
</tr>
<tr>
<td>CPXgetcallbacknodeinfofile</td>
<td>281, 980, 1436</td>
</tr>
<tr>
<td>CPXgetcallbacknodeinfofile</td>
<td>283, 982, 1438</td>
</tr>
<tr>
<td>CPXgetcallbacknodelp</td>
<td>285, 984, 1440</td>
</tr>
<tr>
<td>CPXgetcallbacknodeobjval</td>
<td>287, 986, 1442</td>
</tr>
</tbody>
</table>
INDEX

optim.cplex.callable.callbacks 949
optim.cplex.callable.createdeletecopy 523
optim.cplex.callable.debug 918
optim.cplex.callable.manageparameters 732
optim.cplex.callable.message 932
optim.cplex.callable.modifyparameters 788
optim.cplex.callable.modifyparameters 747
optim.cplex.callable.network 842
optim.cplex.callable.optimizers 566
optim.cplex.callable.portability 911
optim.cplex.callable.querygeneralproblem 667
optim.cplex.callable.querymip 695
optim.cplex.callable.querynetwork 712
optim.cplex.callable.queryup 707
optim.cplex.callable.readfiles 803
optim.cplex.callable.readnetworkfiles 813
optim.cplex.callable.util 908
optim.cplex.callable.writefiles 816
optim.cplex.callable.writenetworkfiles 839
optim.cplex.errorcodes 1044
optim.cplex.solutionquality 1290
optim.cplex.solutionstatus 1338