## Contents

1 Introduction ................................................. 1
   1.1 What is Mosel? ........................................ 1
   1.2 General organization .................................... 1
   1.3 Running Mosel ........................................... 2
      1.3.1 `mosel` command: invocation ...................... 2
      1.3.2 `mosel` command: interactive debugger .......... 6
      1.3.3 `mosel` command: restricted mode ............... 10
      1.3.4 `mosel` command: securing bim files .......... 11
   1.4 References ............................................. 11
   1.5 Structure of this manual ............................... 12

I Core System ................................................. 13

2 The Mosel Language ......................................... 14
   2.1 Introduction ........................................... 14
      2.1.1 Comments ......................................... 14
      2.1.2 Identifiers ........................................ 14
      2.1.3 Reserved words .................................... 15
      2.1.4 Separation of instructions, line breaking ..... 15
      2.1.5 Conventions in this document .................... 15
   2.2 Structure of the source file ........................... 16
   2.3 The compiler directives ................................ 16
      2.3.1 Directive `uses` .................................. 16
      2.3.2 Directive `imports` ................................ 17
      2.3.3 Directive `options` ................................ 18
      2.3.4 Directive `version` ................................ 18
   2.4 The parameters block .................................. 18
   2.5 Source file preprocessing ............................... 19
      2.5.1 Source file character encoding ................... 19
      2.5.2 Source file inclusion ................................ 19
      2.5.3 Line control directives ........................... 20
   2.6 The declaration block .................................. 20
      2.6.1 Elementary types .................................. 21
         2.6.1.1 Basic types .................................... 21
         2.6.1.2 MP types ....................................... 21
      2.6.2 Sets .................................................. 21
      2.6.3 Lists .................................................. 22
      2.6.4 Arrays ................................................ 22
         2.6.4.1 Special case of dynamic arrays of a type not supporting assignment . 23
      2.6.5 Records .............................................. 23
      2.6.6 Constants ............................................ 24
      2.6.7 User defined types .................................. 24
         2.6.7.1 Naming new types ................................ 24
         2.6.7.2 Combining types ................................ 25
   2.7 Expressions .............................................. 25
## Contents

2.10 Problems ........................................... 43
  2.10.1 The mpproblem type ......................... 44
2.11 The public qualifier ............................... 45
2.12 Packages .......................................... 45
  2.12.1 The requirements block ..................... 46
2.13 Annotations ........................................ 46
  2.13.1 Syntax ........................................ 46
  2.13.2 Symbol association ........................... 48
  2.13.3 Declaration ................................... 48
2.14 File names and input/output drivers ............... 50
2.15 Character encoding of text files .................... 51
2.16 Working directory and temporary directory ........ 52
2.17 Handling of input/output .......................... 52
2.18 Deploying models ................................ 53
2.19 Documenting models using annotations ............. 54
  2.19.1 doc annotation category .................... 54
    2.19.1.1 Global definitions ....................... 55
    2.19.1.2 Document structure ...................... 55
    2.19.1.3 Symbol definitions ...................... 56
    2.19.1.4 Annotation definitions ................ 57
2.7.1 Introduction .................................... 25
2.7.2 Aggregate operators ................................ 27
2.7.3 Arithmetic expressions .......................... 28
2.7.4 String expressions ............................... 28
2.7.5 Set expressions .................................. 29
2.7.6 List expressions ................................ 29
2.7.7 Boolean expressions .............................. 29
2.7.8 Linear constraint expressions .................... 30
2.7.9 Automatic arrays ................................ 31
2.8 Statements ......................................... 31
  2.8.1 Simple statements .............................. 31
    2.8.1.1 Assignment ................................. 31
    2.8.1.2 Assignment of structured types ........ 32
    2.8.1.3 About implicit declarations ............ 32
    2.8.1.4 Inline initialization ...................... 33
    2.8.1.5 Linear constraint expression ............ 33
    2.8.1.6 Procedure call ............................. 33
  2.8.2 Initialization block ............................ 34
    2.8.2.1 About automatic finalization .......... 36
2.8.3 Selections ....................................... 36
  2.8.3.1 If statement ................................ 36
  2.8.3.2 Case statement .............................. 37
2.8.4 Loops ............................................ 38
  2.8.4.1 Forall loop ................................ 38
  2.8.4.2 While loop ................................ 38
  2.8.4.3 Repeat loop ................................ 38
  2.8.4.4 break and next statements .............. 39
    2.8.4.5 with statement ............................. 39
2.9 Procedures and functions ........................... 40
  2.9.1 Definition .................................... 40
  2.9.2 Formal parameters: passing convention ....... 41
  2.9.3 Local declarations ............................ 41
  2.9.4 Overloading ................................... 42
  2.9.5 Forward declaration .......................... 42
  2.9.6 Suffix notation ............................... 42
2.10 Problems ........................................... 43
  2.10.1 The mpproblem type ......................... 44
<table>
<thead>
<tr>
<th>Function/Procedure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs</td>
<td>63</td>
</tr>
<tr>
<td>arctan</td>
<td>64</td>
</tr>
<tr>
<td>asproc</td>
<td>65</td>
</tr>
<tr>
<td>assert</td>
<td>66</td>
</tr>
<tr>
<td>bitflip</td>
<td>67</td>
</tr>
<tr>
<td>bitneg</td>
<td>68</td>
</tr>
<tr>
<td>bitset</td>
<td>69</td>
</tr>
<tr>
<td>bitshift</td>
<td>70</td>
</tr>
<tr>
<td>bittest</td>
<td>71</td>
</tr>
<tr>
<td>bitval</td>
<td>72</td>
</tr>
<tr>
<td>ceil</td>
<td>73</td>
</tr>
<tr>
<td>cos</td>
<td>74</td>
</tr>
<tr>
<td>create</td>
<td>75</td>
</tr>
<tr>
<td>currentdate</td>
<td>76</td>
</tr>
<tr>
<td>currenttime</td>
<td>77</td>
</tr>
<tr>
<td>cuthead</td>
<td>78</td>
</tr>
<tr>
<td>cuttail</td>
<td>79</td>
</tr>
<tr>
<td>delcell</td>
<td>80</td>
</tr>
<tr>
<td>exists</td>
<td>81</td>
</tr>
<tr>
<td>exit</td>
<td>82</td>
</tr>
<tr>
<td>exp</td>
<td>83</td>
</tr>
<tr>
<td>exportprob</td>
<td>84</td>
</tr>
<tr>
<td>fclose</td>
<td>85</td>
</tr>
<tr>
<td>fflush</td>
<td>86</td>
</tr>
<tr>
<td>finalize</td>
<td>87</td>
</tr>
<tr>
<td>findfirst</td>
<td>88</td>
</tr>
<tr>
<td>findlast</td>
<td>89</td>
</tr>
<tr>
<td>floor</td>
<td>90</td>
</tr>
<tr>
<td>fopen</td>
<td>91</td>
</tr>
<tr>
<td>fselect</td>
<td>92</td>
</tr>
<tr>
<td>fskipline</td>
<td>93</td>
</tr>
<tr>
<td>fwrite, fwriteln</td>
<td>94</td>
</tr>
<tr>
<td>getact</td>
<td>95</td>
</tr>
<tr>
<td>getcoeff</td>
<td>96</td>
</tr>
<tr>
<td>getcoeffs</td>
<td>97</td>
</tr>
<tr>
<td>getdual</td>
<td>98</td>
</tr>
<tr>
<td>getfid</td>
<td>99</td>
</tr>
<tr>
<td>getfirst</td>
<td>100</td>
</tr>
<tr>
<td>gethead</td>
<td>101</td>
</tr>
<tr>
<td>getname</td>
<td>102</td>
</tr>
<tr>
<td>getlast</td>
<td>103</td>
</tr>
<tr>
<td>getobjval</td>
<td>104</td>
</tr>
<tr>
<td>getparam</td>
<td>105</td>
</tr>
<tr>
<td>getcost</td>
<td>106</td>
</tr>
<tr>
<td>getreadcnt</td>
<td>108</td>
</tr>
<tr>
<td>getparam</td>
<td>109</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------</td>
<td>------</td>
</tr>
<tr>
<td>getreverse</td>
<td>110</td>
</tr>
<tr>
<td>getsize</td>
<td>111</td>
</tr>
<tr>
<td>getslack</td>
<td>112</td>
</tr>
<tr>
<td>getsol</td>
<td>113</td>
</tr>
<tr>
<td>gettail</td>
<td>114</td>
</tr>
<tr>
<td>gettype</td>
<td>115</td>
</tr>
<tr>
<td>getvars</td>
<td>116</td>
</tr>
<tr>
<td>isdynamic</td>
<td>117</td>
</tr>
<tr>
<td>iseof</td>
<td>118</td>
</tr>
<tr>
<td>isfinite</td>
<td>119</td>
</tr>
<tr>
<td>ishidden</td>
<td>120</td>
</tr>
<tr>
<td>isinf</td>
<td>121</td>
</tr>
<tr>
<td>isnan</td>
<td>122</td>
</tr>
<tr>
<td>isodd</td>
<td>123</td>
</tr>
<tr>
<td>ln</td>
<td>124</td>
</tr>
<tr>
<td>log</td>
<td>125</td>
</tr>
<tr>
<td>makesos1, makesos2</td>
<td>126</td>
</tr>
<tr>
<td>maxlist</td>
<td>127</td>
</tr>
<tr>
<td>minlist</td>
<td>128</td>
</tr>
<tr>
<td>newmuid</td>
<td>129</td>
</tr>
<tr>
<td>publish</td>
<td>130</td>
</tr>
<tr>
<td>random</td>
<td>131</td>
</tr>
<tr>
<td>read, readln</td>
<td>132</td>
</tr>
<tr>
<td>reset</td>
<td>133</td>
</tr>
<tr>
<td>reverse</td>
<td>134</td>
</tr>
<tr>
<td>round</td>
<td>135</td>
</tr>
<tr>
<td>setcoeff</td>
<td>136</td>
</tr>
<tr>
<td>sethidden</td>
<td>137</td>
</tr>
<tr>
<td>setioerr</td>
<td>138</td>
</tr>
<tr>
<td>setmatherr</td>
<td>139</td>
</tr>
<tr>
<td>setname</td>
<td>140</td>
</tr>
<tr>
<td>setparam</td>
<td>141</td>
</tr>
<tr>
<td>setrandseed</td>
<td>142</td>
</tr>
<tr>
<td>settype</td>
<td>143</td>
</tr>
<tr>
<td>sin</td>
<td>144</td>
</tr>
<tr>
<td>splithead</td>
<td>145</td>
</tr>
<tr>
<td>splitltail</td>
<td>146</td>
</tr>
<tr>
<td>sqrt</td>
<td>147</td>
</tr>
<tr>
<td>strftime</td>
<td>148</td>
</tr>
<tr>
<td>substr</td>
<td>149</td>
</tr>
<tr>
<td>timestamp</td>
<td>150</td>
</tr>
<tr>
<td>unpublish</td>
<td>151</td>
</tr>
<tr>
<td>write, writeln</td>
<td>152</td>
</tr>
</tbody>
</table>

## II Modules

### 4 mmec

#### 4.1 Procedures and functions

- disc
- diskdata

#### 4.2 I/O drivers

- Driver diskdata

### 5 mmhttp

- 158
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1 Example</td>
<td>194</td>
</tr>
<tr>
<td>7.2 Control parameters</td>
<td>195</td>
</tr>
<tr>
<td>conntmpl</td>
<td>195</td>
</tr>
<tr>
<td>nodenumber</td>
<td>195</td>
</tr>
<tr>
<td>defaultnode</td>
<td>196</td>
</tr>
<tr>
<td>jobid</td>
<td>196</td>
</tr>
<tr>
<td>parentnumber</td>
<td>196</td>
</tr>
<tr>
<td>keepalive</td>
<td>196</td>
</tr>
<tr>
<td>fsrvport</td>
<td>197</td>
</tr>
<tr>
<td>fsrvdelay</td>
<td>197</td>
</tr>
<tr>
<td>fsrvnbiter</td>
<td>197</td>
</tr>
<tr>
<td>sshcmd</td>
<td>198</td>
</tr>
<tr>
<td>7.3 Procedures and functions</td>
<td>198</td>
</tr>
<tr>
<td>7.3.1 Mosel instance management</td>
<td>198</td>
</tr>
<tr>
<td>connect</td>
<td>199</td>
</tr>
<tr>
<td>disconnect</td>
<td>200</td>
</tr>
<tr>
<td>clearaliases</td>
<td>201</td>
</tr>
<tr>
<td>getbanner</td>
<td>202</td>
</tr>
<tr>
<td>gethostalies</td>
<td>203</td>
</tr>
<tr>
<td>getalieses</td>
<td>204</td>
</tr>
<tr>
<td>sethostalies</td>
<td>205</td>
</tr>
<tr>
<td>findxsrvs</td>
<td>206</td>
</tr>
<tr>
<td>7.3.2 Model management</td>
<td>207</td>
</tr>
<tr>
<td>compile</td>
<td>208</td>
</tr>
<tr>
<td>detach</td>
<td>210</td>
</tr>
<tr>
<td>load</td>
<td>211</td>
</tr>
<tr>
<td>setdefstream</td>
<td>212</td>
</tr>
<tr>
<td>resetmodpar</td>
<td>213</td>
</tr>
<tr>
<td>setcontrol</td>
<td>214</td>
</tr>
<tr>
<td>setmodpar</td>
<td>215</td>
</tr>
<tr>
<td>setworkdir</td>
<td>216</td>
</tr>
<tr>
<td>run</td>
<td>217</td>
</tr>
<tr>
<td>getsdprop</td>
<td>218</td>
</tr>
<tr>
<td>getgid</td>
<td>219</td>
</tr>
<tr>
<td>getid</td>
<td>220</td>
</tr>
<tr>
<td>getmodprop</td>
<td>221</td>
</tr>
<tr>
<td>getnode</td>
<td>222</td>
</tr>
<tr>
<td>getrmtid</td>
<td>223</td>
</tr>
<tr>
<td>getstatus</td>
<td>224</td>
</tr>
<tr>
<td>getuid</td>
<td>225</td>
</tr>
<tr>
<td>getexitcode</td>
<td>226</td>
</tr>
<tr>
<td>stop</td>
<td>227</td>
</tr>
<tr>
<td>reset</td>
<td>228</td>
</tr>
<tr>
<td>unload</td>
<td>229</td>
</tr>
<tr>
<td>getannidents</td>
<td>230</td>
</tr>
<tr>
<td>getannotations</td>
<td>231</td>
</tr>
<tr>
<td>7.3.3 Synchronization</td>
<td>232</td>
</tr>
<tr>
<td>canceltimer</td>
<td>233</td>
</tr>
<tr>
<td>send</td>
<td>234</td>
</tr>
<tr>
<td>settimer</td>
<td>235</td>
</tr>
<tr>
<td>setuid</td>
<td>236</td>
</tr>
<tr>
<td>setgid</td>
<td>237</td>
</tr>
<tr>
<td>wait</td>
<td>238</td>
</tr>
<tr>
<td>waitexpired</td>
<td>239</td>
</tr>
<tr>
<td>waitfor</td>
<td>240</td>
</tr>
</tbody>
</table>
### Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.4</td>
<td>Procedures and functions</td>
<td>282</td>
</tr>
<tr>
<td>9.5</td>
<td>I/O drivers</td>
<td>291</td>
</tr>
<tr>
<td>10.2</td>
<td>Example</td>
<td>292</td>
</tr>
<tr>
<td>10.3</td>
<td>Data transfer between Mosel and the database</td>
<td>293</td>
</tr>
<tr>
<td>10.3.1</td>
<td>From the database to Mosel</td>
<td>293</td>
</tr>
<tr>
<td>10.3.2</td>
<td>From Mosel to the database</td>
<td>295</td>
</tr>
<tr>
<td>10.4</td>
<td>ODBC and MS Excel</td>
<td>296</td>
</tr>
<tr>
<td>10.5</td>
<td>Control parameters</td>
<td>297</td>
</tr>
<tr>
<td>10.6</td>
<td>Procedures and functions</td>
<td>301</td>
</tr>
<tr>
<td>10.7</td>
<td>I/O drivers</td>
<td>320</td>
</tr>
<tr>
<td>10.7.1</td>
<td>Driver odbc</td>
<td>320</td>
</tr>
</tbody>
</table>

**9.4 Procedures and functions**

- OCIlogon ................................................. 283
- OCIlogoff ............................................. 284
- OCIexecute ............................................ 285
- OCIreadinteger ........................................ 286
- OCIreadreal ........................................... 287
- OCIreadstring ........................................ 288
- OCIcommit ............................................. 289
- OClrollback ............................................ 290

**9.5 I/O drivers**

- OCIROWCOUNT ........................................... 291
- OCIROWXFR ............................................. 301

**10 mmodbc**

- 10.1 Prerequisite ...................................... 292
- 10.2 Example ........................................... 292
- 10.3 Data transfer between Mosel and the database | 293  |
| 10.3.1 From the database to Mosel | 293  |
| 10.3.2 From Mosel to the database | 295  |
| 10.4 ODBC and MS Excel | 296  |
| 10.5 Control parameters | 297  |
| SQLautocommit ........................................ 298
- SQLBUFSIZE ............................................. 298
- SQLCOLUMNSIZE .......................................... 298
- SQLCONNECTION .......................................... 299
- SQLDEBUG ............................................... 299
- SQLdm .................................................... 299
- SQLextn ............................................... 300
- SQLINDXCOL ............................................ 300
- SQLROWCOUNT .......................................... 300
- SQLROWXFR ............................................. 301
- SQLSUCCESS ............................................ 301
- SQLVERBOSE ............................................ 301

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.6</td>
<td>Procedures and functions</td>
<td>301</td>
</tr>
<tr>
<td>SQLcolumns</td>
<td></td>
<td>303</td>
</tr>
<tr>
<td>SQLcommit</td>
<td></td>
<td>304</td>
</tr>
<tr>
<td>SQLconnect</td>
<td></td>
<td>305</td>
</tr>
<tr>
<td>SQLdisconnect</td>
<td></td>
<td>307</td>
</tr>
<tr>
<td>SQLexecute</td>
<td></td>
<td>308</td>
</tr>
<tr>
<td>SQLparam</td>
<td></td>
<td>310</td>
</tr>
<tr>
<td>SQLgetparam</td>
<td></td>
<td>311</td>
</tr>
<tr>
<td>SQLindicies</td>
<td></td>
<td>312</td>
</tr>
<tr>
<td>SQLprimarykeys</td>
<td></td>
<td>313</td>
</tr>
<tr>
<td>SQLreadinteger</td>
<td></td>
<td>314</td>
</tr>
<tr>
<td>SQLreadreal</td>
<td></td>
<td>315</td>
</tr>
<tr>
<td>SQLreadstring</td>
<td></td>
<td>316</td>
</tr>
<tr>
<td>SQLrollback</td>
<td></td>
<td>317</td>
</tr>
<tr>
<td>SQLtables</td>
<td></td>
<td>318</td>
</tr>
<tr>
<td>SQLupdate</td>
<td></td>
<td>319</td>
</tr>
<tr>
<td>10.7</td>
<td>I/O drivers</td>
<td>320</td>
</tr>
<tr>
<td>10.7.1</td>
<td>Driver odbc</td>
<td>320</td>
</tr>
</tbody>
</table>
## 11 mmquad

11.1 New functionality for the Mosel language .................................................. 321
  11.1.1 The type qexp and its operators ......................................................... 321
    11.1.1.1 Example: using mmquad for Quadratic Programming ....................... 321
  11.1.2 Procedures and functions ....................................................................... 322
    exportprob .................................................................................................... 323
    getsol ........................................................................................................... 324

11.2 Published library functions ................................................................. 325
  11.2.1 Complete module example ................................................................... 325
  11.2.2 Description of the library functions ................................................... 327
    getqexpstat .................................................................................................. 328
    getqexpstat ................................................................................................. 329
    clearqexpstat ............................................................................................... 330
    getqexpnextterm ......................................................................................... 331

## 12 mmrobust

12.1 New functionality for the Mosel language ............................................... 332
  12.1.1 The problem type mpprobлем.xprs.robust ......................................... 332
  12.1.2 The type uncertain ................................................................................. 332
  12.1.3 The type robustctr and its operators .................................................. 333
  12.1.4 The type uncertainctr and its operators .............................................. 333
  12.1.5 Example: using mmrobust for solving a robust problem ..................... 333

12.2 Control parameters ....................................................................................... 334
  robust_uncertain_overlap ................................................................................ 334
  robust_check_feas_uncertainty_set ................................................................ 334
  robust_check_feas_original_problem ............................................................ 335

12.3 Procedures and functions ............................................................................. 335
  cardinality ....................................................................................................... 336
  getsol ............................................................................................................. 337
  getact ............................................................................................................. 338
  ishidden ......................................................................................................... 339
  scenario ......................................................................................................... 340
  sethidden ...................................................................................................... 341
  getnominal .................................................................................................... 342
  gettype ......................................................................................................... 343
  setnominal .................................................................................................... 344
  settype ......................................................................................................... 345

## 13 mmsheet

13.1 I/O drivers .................................................................................................... 346
  13.1.1 Driver excel ......................................................................................... 347
  13.1.2 Driver xls/xlsx ..................................................................................... 347
  13.1.3 Driver csv ............................................................................................ 348

## 14 mmssl

14.1 Overview ....................................................................................................... 349
  14.1.1 Document encryption in Mosel ............................................................. 349
  14.1.2 The mmssl command ......................................................................... 349

14.2 Control parameters ....................................................................................... 350
  https_cacerts .................................................................................................. 351
  https_ciphers .................................................................................................. 351
  https_cltcrt ..................................................................................................... 352
  https_cltkey .................................................................................................... 352
  https_srvcert .................................................................................................. 353
  https_srvkey .................................................................................................. 353
<table>
<thead>
<tr>
<th>Procedure/Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>https_trustsrv</td>
<td>353</td>
</tr>
<tr>
<td>ssl_cipher</td>
<td>353</td>
</tr>
<tr>
<td>ssl_digest</td>
<td>354</td>
</tr>
<tr>
<td>ssl_dir</td>
<td>354</td>
</tr>
<tr>
<td>ssl_privkey</td>
<td>355</td>
</tr>
<tr>
<td>14.3 Procedures and functions</td>
<td></td>
</tr>
<tr>
<td>RSAfingerprint</td>
<td>357</td>
</tr>
<tr>
<td>RSAGenkey</td>
<td>358</td>
</tr>
<tr>
<td>RSAGetkeysizes</td>
<td>359</td>
</tr>
<tr>
<td>RSAisprivate</td>
<td>360</td>
</tr>
<tr>
<td>RSAloadkey</td>
<td>361</td>
</tr>
<tr>
<td>RSAprivdecrypt</td>
<td>362</td>
</tr>
<tr>
<td>RSApubencrypt</td>
<td>363</td>
</tr>
<tr>
<td>RSAsavekey</td>
<td>364</td>
</tr>
<tr>
<td>msgdigest</td>
<td>365</td>
</tr>
<tr>
<td>msgsign</td>
<td>366</td>
</tr>
<tr>
<td>msgverify</td>
<td>367</td>
</tr>
<tr>
<td>sslivsize</td>
<td>368</td>
</tr>
<tr>
<td>ssikeysizes</td>
<td>369</td>
</tr>
<tr>
<td>sslmdsize</td>
<td>370</td>
</tr>
<tr>
<td>sslrandom</td>
<td>371</td>
</tr>
<tr>
<td>sslrandomdata</td>
<td>372</td>
</tr>
<tr>
<td>x509check</td>
<td>373</td>
</tr>
<tr>
<td>x509getinfo</td>
<td>374</td>
</tr>
<tr>
<td>x509newcrt</td>
<td>375</td>
</tr>
<tr>
<td>14.4 I/O drivers</td>
<td></td>
</tr>
<tr>
<td>14.4.1 Driver base64</td>
<td>376</td>
</tr>
<tr>
<td>14.4.2 Driver hex</td>
<td>376</td>
</tr>
<tr>
<td>14.4.3 Driver crypt</td>
<td>376</td>
</tr>
<tr>
<td>15 mmsvg</td>
<td>378</td>
</tr>
<tr>
<td>15.1 SVG graph structure</td>
<td></td>
</tr>
<tr>
<td>15.1.1 Object groups</td>
<td>378</td>
</tr>
<tr>
<td>15.1.2 SVG styling</td>
<td>379</td>
</tr>
<tr>
<td>15.1.3 Interaction with the graphical display</td>
<td>380</td>
</tr>
<tr>
<td>15.1.4 Example</td>
<td>380</td>
</tr>
<tr>
<td>15.2 Control parameters</td>
<td></td>
</tr>
<tr>
<td>MMSVGDISPPLAY</td>
<td>381</td>
</tr>
<tr>
<td>MMSVGTGZ</td>
<td>382</td>
</tr>
<tr>
<td>15.3 Procedures and Functions</td>
<td></td>
</tr>
<tr>
<td>svgaddgroup</td>
<td>384</td>
</tr>
<tr>
<td>svgaddarrow</td>
<td>385</td>
</tr>
<tr>
<td>svgaddcircle</td>
<td>386</td>
</tr>
<tr>
<td>svgaddellipse</td>
<td>387</td>
</tr>
<tr>
<td>svgaddfile</td>
<td>388</td>
</tr>
<tr>
<td>svgaddimage</td>
<td>389</td>
</tr>
<tr>
<td>svgaddline</td>
<td>390</td>
</tr>
<tr>
<td>svgaddpie</td>
<td>391</td>
</tr>
<tr>
<td>svgaddpoint</td>
<td>392</td>
</tr>
<tr>
<td>svgaddpolygon</td>
<td>393</td>
</tr>
<tr>
<td>svgaddrectangle</td>
<td>394</td>
</tr>
<tr>
<td>svgaddtext</td>
<td>395</td>
</tr>
<tr>
<td>svgaddxmltext</td>
<td>396</td>
</tr>
<tr>
<td>svgclosing</td>
<td>397</td>
</tr>
<tr>
<td>svgcolor</td>
<td>398</td>
</tr>
</tbody>
</table>
16 mmsystem 421

16.1 New functionality for the Mosel language 421
  16.1.1 The type text 421
  16.1.2 The type date 421
  16.1.3 The type time 421
  16.1.4 The type datetime 422
  16.1.5 The type parsectx 422
  16.1.6 The type textarea 422

16.2 Control parameters 422
datetimefmt 423
timefmt 423
datetimetimefmt 424
monthnames 424
sys_endparse 425
sys_fillchar 425
sys_pid 425
sys_qtype 425
sys_regcache 426
sys_sepchar 426
sys_trim 426
sys_txtmem 427

16.3 Procedures and functions 427
addmonths 431
copytext 432
cuttext 433
deltext 434
endswith 435
expandpath 436
fcopy 437
fdelete 438
findfiles 439
findtext 440

svgdelobj ........................................ 399
svgerase .......................................... 400
svggetgraphstyle .................................. 401
svggetgraphstylesheet ............................... 402
svggetgraphviewbbox ............................... 403
svggetlastobj ...................................... 404
svggettyle .......................... 405
svggetstylesheet .................................. 406
svgpause ........................................... 407
svgsave .......................... 408
svgsave ............................................ 409
svgsave ............................................ 410
svgsave ............................................ 411
svgsave ............................................ 412
svgsave ............................................ 413
svgsave ............................................ 414
svgsave ............................................ 415
svgsave ............................................ 416
svgsave ............................................ 417
svgsave ............................................ 418
svgsave ............................................ 419
svgsave ............................................ 420

findtext ............................................ 440
findfiles ........................................... 439
fdelete ............................................. 438
fdelete ............................................. 437
findfiles ........................................... 439
findfiles ........................................... 438
findfiles ........................................... 437
findfiles ........................................... 436
findfiles ........................................... 435
findfiles ........................................... 434
findfiles ........................................... 433
findfiles ........................................... 432
findfiles ........................................... 431
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>fmove</td>
<td>441</td>
</tr>
<tr>
<td>formatext</td>
<td>442</td>
</tr>
<tr>
<td>getasnumber</td>
<td>444</td>
</tr>
<tr>
<td>getchar</td>
<td>445</td>
</tr>
<tr>
<td>getcwd</td>
<td>446</td>
</tr>
<tr>
<td>getdate</td>
<td>447</td>
</tr>
<tr>
<td>getday</td>
<td>448</td>
</tr>
<tr>
<td>getdaynum</td>
<td>449</td>
</tr>
<tr>
<td>getdays</td>
<td>450</td>
</tr>
<tr>
<td>getdirsep</td>
<td>451</td>
</tr>
<tr>
<td>getendparse, setendparse</td>
<td>452</td>
</tr>
<tr>
<td>getenv</td>
<td>453</td>
</tr>
<tr>
<td>getfsize</td>
<td>454</td>
</tr>
<tr>
<td>getfstat</td>
<td>455</td>
</tr>
<tr>
<td>getftime</td>
<td>456</td>
</tr>
<tr>
<td>gethour</td>
<td>457</td>
</tr>
<tr>
<td>getminute</td>
<td>458</td>
</tr>
<tr>
<td>getmonth</td>
<td>459</td>
</tr>
<tr>
<td>getmsec</td>
<td>460</td>
</tr>
<tr>
<td>getpathsep</td>
<td>461</td>
</tr>
<tr>
<td>getsucc, setsucc</td>
<td>462</td>
</tr>
<tr>
<td>getqtype, setqtype</td>
<td>463</td>
</tr>
<tr>
<td>getsecond</td>
<td>464</td>
</tr>
<tr>
<td>getsepchar, setsepchar</td>
<td>465</td>
</tr>
<tr>
<td>getsize</td>
<td>466</td>
</tr>
<tr>
<td>getstart, setstart</td>
<td>467</td>
</tr>
<tr>
<td>getsysinfo</td>
<td>468</td>
</tr>
<tr>
<td>getsysstat</td>
<td>469</td>
</tr>
<tr>
<td>gettime</td>
<td>470</td>
</tr>
<tr>
<td>gettmpdir</td>
<td>471</td>
</tr>
<tr>
<td>gettrim, settrim</td>
<td>472</td>
</tr>
<tr>
<td>getweekday</td>
<td>473</td>
</tr>
<tr>
<td>getyear</td>
<td>474</td>
</tr>
<tr>
<td>inserttext</td>
<td>475</td>
</tr>
<tr>
<td>isvalid</td>
<td>476</td>
</tr>
<tr>
<td>makedir</td>
<td>477</td>
</tr>
<tr>
<td>makepath</td>
<td>478</td>
</tr>
<tr>
<td>newtar</td>
<td>479</td>
</tr>
<tr>
<td>newzip</td>
<td>480</td>
</tr>
<tr>
<td>nextfield</td>
<td>481</td>
</tr>
<tr>
<td>openpipe</td>
<td>482</td>
</tr>
<tr>
<td>parseextn</td>
<td>483</td>
</tr>
<tr>
<td>parseint</td>
<td>484</td>
</tr>
<tr>
<td>parsereal</td>
<td>485</td>
</tr>
<tr>
<td>parsetext</td>
<td>486</td>
</tr>
<tr>
<td>pastetext</td>
<td>487</td>
</tr>
<tr>
<td>pathmatch</td>
<td>488</td>
</tr>
<tr>
<td>pathsplit</td>
<td>489</td>
</tr>
<tr>
<td>qsort</td>
<td>490</td>
</tr>
<tr>
<td>quote</td>
<td>491</td>
</tr>
<tr>
<td>readtextline</td>
<td>492</td>
</tr>
<tr>
<td>regmatch</td>
<td>493</td>
</tr>
<tr>
<td>regreplace</td>
<td>494</td>
</tr>
<tr>
<td>removedir</td>
<td>495</td>
</tr>
<tr>
<td>removefiles</td>
<td>496</td>
</tr>
</tbody>
</table>
17 mmxml .......................................................... 535
  17.1 Document representation in mmxml .................. 535
  17.1.1 Data model ........................................ 535
  17.1.2 Paths in a document ............................ 536
    17.1.2.1 Axis specifier ................................ 537
    17.1.2.2 Node test .................................. 537
    17.1.2.3 Abbreviated notation ..................... 537
    17.1.2.4 Predicate ................................ 537
  17.1.3 JSON document as an XML tree ................... 538
  17.2 New functionality for the Mosel language ........... 540
    17.2.1 The type xmldoc .................................. 540
  17.3 Procedures and functions ............................ 540
    addnode .............................................. 542
    copynode ........................................... 544
    delattr ............................................. 545
    delnode ............................................. 546
    getattr ............................................. 547
    testattr ........................................... 548

setchar .................................................................. 500
setdate .................................................................. 501
setday .................................................................. 502
setenv .................................................................. 503
sethour .................................................................. 504
setminute ............................................................. 505
setmonth .............................................................. 506
setmsec ............................................................... 507
setsecond ............................................................. 508
settime .................................................................. 509
setyear .................................................................. 510
sleep ....................................................................... 511
startwith ................................................................ 512
system .................................................................... 513
tarlist .................................................................... 514
textfmt ................................................................... 515
tolower ................................................................... 517
toupper .................................................................... 518
trim ....................................................................... 519
untar ...................................................................... 520
unzip ...................................................................... 521
ziplist ..................................................................... 522

16.4 I/O drivers .................................................. 523
  16.4.1 Driver text .............................................. 523
  16.4.2 Driver pipe .............................................. 523

16.5 Published library functions .............................. 524
  16.5.1 Description of the library functions ................. 524
    gettext ....................................................... 526
    settime ...................................................... 527
    getdate ...................................................... 528
    setdate ...................................................... 529
    getdatetime ............................................... 530
    setdatetime ............................................... 531
    gettxtsize ................................................ 532
    gettxtbuf ................................................ 533
    txtresize .................................................. 534
### 18 mmxnlp

**18.1 New functionality for the Mosel language**
- The `userfunc` type  
- The `tolset` type  
- The `mpproblem.xprs.xnlp` problem type

**18.2 mmxnlp and the other Mosel modules**
- Overloaded functions  
- Module compatibility

**18.3 Control parameters**
- `XNLP_LOADASNL`  
- `XNLP_LOADNAMES`  
- `XNLP_NLPSTATUS`  
- `XNLP_SOLVER`
18.4 Procedures and functions .......................................................... 597
  addmultistart ................................................................. 599
  chgdeltatype .............................................................. 600
  F ................................................................. 601
  generateUFparallel .......................................................... 603
  printmodelmemory .......................................................... 604
  printmodelsclashing ......................................................... 605
  setcomplementary .......................................................... 606
  setdelayedctr .............................................................. 607
  setdetrow ................................................................. 608
  setenforcedctr ............................................................ 609
  setinitial ................................................................. 610
  settol ...................................................................... 611
  settolset ................................................................. 612
  userfuncDLL .............................................................. 613
  userfuncExcel ............................................................. 614
  userfuncExcelMacro .......................................................... 615
  userfuncinfo .............................................................. 616
  userfuncMosel ............................................................ 617
  validate ................................................................. 618
18.5 Error codes issued by mmxnlp ............................................... 619
19 mmxprs ................................................................. 621
  19.1 New functionality for the Mosel language .................................. 621
    19.1.1 The problem type mpproblem.xprs .................................. 621
    19.1.2 The type basis ......................................................... 621
    19.1.3 The type mpsol ....................................................... 621
    19.1.4 The type logctr ....................................................... 622
  19.2 Control parameters .......................................................... 622
    XPRS_colorder .......................................................... 623
    XPRS_enumsol ............................................................ 623
    XPRS_enummaxsol .......................................................... 623
    XPRS_enumdualpol .......................................................... 623
    XPRS_fullversion .......................................................... 624
    XPRS_loadnames .......................................................... 624
    XPRS_problem ............................................................ 624
    XPRS_probname ............................................................ 625
    XPRSVerbose ............................................................. 625
  19.3 Procedures and functions .................................................. 625
    addmipsol ............................................................... 628
    basisstability ............................................................ 629
    calcsolinfo .............................................................. 630
    clearmipdir .............................................................. 631
    clearmodcut .............................................................. 632
    command ................................................................. 633
    copyso1to1init ............................................................ 634
    defdelayedrows .......................................................... 635
    defsecurevecs ........................................................... 636
    estimatemarginals .......................................................... 637
    fixglobal ................................................................. 638
    getbstat ................................................................. 639
    getdualray ............................................................... 640
    getiis ................................................................. 641
    getiissense ............................................................. 642
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>getiistype</td>
<td>643</td>
</tr>
<tr>
<td>getinfcause</td>
<td>644</td>
</tr>
<tr>
<td>getinfeas</td>
<td>645</td>
</tr>
<tr>
<td>getlb</td>
<td>646</td>
</tr>
<tr>
<td>getloadedlinctrs</td>
<td>647</td>
</tr>
<tr>
<td>getloadedmipvars</td>
<td>648</td>
</tr>
<tr>
<td>getname</td>
<td>649</td>
</tr>
<tr>
<td>getprimalray</td>
<td>650</td>
</tr>
<tr>
<td>getprobstat</td>
<td>651</td>
</tr>
<tr>
<td>getrange</td>
<td>652</td>
</tr>
<tr>
<td>getsensrng</td>
<td>653</td>
</tr>
<tr>
<td>getsize</td>
<td>654</td>
</tr>
<tr>
<td>getsol</td>
<td>655</td>
</tr>
<tr>
<td>getub</td>
<td>656</td>
</tr>
<tr>
<td>getvars</td>
<td>657</td>
</tr>
<tr>
<td>hasfeature</td>
<td>658</td>
</tr>
<tr>
<td>implies</td>
<td>659</td>
</tr>
<tr>
<td>indicator</td>
<td>660</td>
</tr>
<tr>
<td>isisvalid</td>
<td>661</td>
</tr>
<tr>
<td>isintegral</td>
<td>662</td>
</tr>
<tr>
<td>loadbasis</td>
<td>663</td>
</tr>
<tr>
<td>loadmipsol</td>
<td>664</td>
</tr>
<tr>
<td>loadprob</td>
<td>666</td>
</tr>
<tr>
<td>maximize, minimize</td>
<td>667</td>
</tr>
<tr>
<td>postsolve</td>
<td>669</td>
</tr>
<tr>
<td>readbasis</td>
<td>670</td>
</tr>
<tr>
<td>readdirs</td>
<td>671</td>
</tr>
<tr>
<td>readsol</td>
<td>672</td>
</tr>
<tr>
<td>refinemipsol</td>
<td>673</td>
</tr>
<tr>
<td>rejectintsol</td>
<td>674</td>
</tr>
<tr>
<td>repairinfeas</td>
<td>675</td>
</tr>
<tr>
<td>resetbasis</td>
<td>677</td>
</tr>
<tr>
<td>resetiis</td>
<td>678</td>
</tr>
<tr>
<td>resetsol</td>
<td>679</td>
</tr>
<tr>
<td>savebasis</td>
<td>680</td>
</tr>
<tr>
<td>savemipsol</td>
<td>681</td>
</tr>
<tr>
<td>savesol</td>
<td>682</td>
</tr>
<tr>
<td>savestate</td>
<td>683</td>
</tr>
<tr>
<td>selectsol</td>
<td>684</td>
</tr>
<tr>
<td>setarchconsistency</td>
<td>685</td>
</tr>
<tr>
<td>setbstat</td>
<td>686</td>
</tr>
<tr>
<td>setcallback</td>
<td>687</td>
</tr>
<tr>
<td>setcbcutoff</td>
<td>690</td>
</tr>
<tr>
<td>setgndata</td>
<td>691</td>
</tr>
<tr>
<td>setlb</td>
<td>692</td>
</tr>
<tr>
<td>setmipdir</td>
<td>693</td>
</tr>
<tr>
<td>setmodcut</td>
<td>694</td>
</tr>
<tr>
<td>setsol</td>
<td>695</td>
</tr>
<tr>
<td>setub</td>
<td>696</td>
</tr>
<tr>
<td>setucbdata</td>
<td>697</td>
</tr>
<tr>
<td>stopoptimize</td>
<td>698</td>
</tr>
<tr>
<td>unloadprob</td>
<td>699</td>
</tr>
<tr>
<td>writebasis</td>
<td>700</td>
</tr>
<tr>
<td>writedirs</td>
<td>701</td>
</tr>
<tr>
<td>writeprob</td>
<td>702</td>
</tr>
</tbody>
</table>
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>writesol</td>
<td>703</td>
</tr>
<tr>
<td>xor</td>
<td>704</td>
</tr>
<tr>
<td>19.4 Cut Pool Manager</td>
<td></td>
</tr>
<tr>
<td>addcut</td>
<td>705</td>
</tr>
<tr>
<td>addcuts</td>
<td>706</td>
</tr>
<tr>
<td>delcuts</td>
<td>707</td>
</tr>
<tr>
<td>dropcuts</td>
<td>708</td>
</tr>
<tr>
<td>getcnlist</td>
<td>709</td>
</tr>
<tr>
<td>getcplist</td>
<td>710</td>
</tr>
<tr>
<td>loadcuts</td>
<td>711</td>
</tr>
<tr>
<td>storecut</td>
<td>712</td>
</tr>
<tr>
<td>storecuts</td>
<td>713</td>
</tr>
<tr>
<td>R</td>
<td></td>
</tr>
<tr>
<td>20.1 Introduction</td>
<td>715</td>
</tr>
<tr>
<td>20.1.1 Prerequisite</td>
<td>715</td>
</tr>
<tr>
<td>20.1.2 R initialization</td>
<td>716</td>
</tr>
<tr>
<td>20.1.3 Data types</td>
<td>716</td>
</tr>
<tr>
<td>20.2 Example</td>
<td>719</td>
</tr>
<tr>
<td>20.3 Control parameters</td>
<td>720</td>
</tr>
<tr>
<td>Rverbose</td>
<td>720</td>
</tr>
<tr>
<td>Rinteractive</td>
<td>720</td>
</tr>
<tr>
<td>Rusemosstreams</td>
<td>721</td>
</tr>
<tr>
<td>Rcleanscript</td>
<td>721</td>
</tr>
<tr>
<td>Runloadscript</td>
<td>721</td>
</tr>
<tr>
<td>Rsessionmode</td>
<td>721</td>
</tr>
<tr>
<td>20.4 Procedures and functions</td>
<td>722</td>
</tr>
<tr>
<td>Reval</td>
<td>723</td>
</tr>
<tr>
<td>Rfree</td>
<td>724</td>
</tr>
<tr>
<td>Rgetarr</td>
<td>725</td>
</tr>
<tr>
<td>Rgetbool</td>
<td>726</td>
</tr>
<tr>
<td>Rgetint</td>
<td>727</td>
</tr>
<tr>
<td>Rgetreal</td>
<td>728</td>
</tr>
<tr>
<td>Rgetstr</td>
<td>729</td>
</tr>
<tr>
<td>Rinit</td>
<td>730</td>
</tr>
<tr>
<td>Rprint</td>
<td>731</td>
</tr>
<tr>
<td>Rset</td>
<td>732</td>
</tr>
<tr>
<td>Rsource</td>
<td>733</td>
</tr>
<tr>
<td>Rerrcode</td>
<td>734</td>
</tr>
<tr>
<td>Rerrmsg</td>
<td>735</td>
</tr>
<tr>
<td>Rclearerr</td>
<td>736</td>
</tr>
<tr>
<td>20.5 I/O drivers</td>
<td></td>
</tr>
<tr>
<td>20.5.1 Driver rws</td>
<td>737</td>
</tr>
<tr>
<td>20.6 Troubleshooting</td>
<td>737</td>
</tr>
<tr>
<td>zlib</td>
<td></td>
</tr>
<tr>
<td>21.1 I/O drivers</td>
<td>738</td>
</tr>
<tr>
<td>21.1.1 Driver gzip</td>
<td>738</td>
</tr>
<tr>
<td>21.1.2 Driver deflate</td>
<td>738</td>
</tr>
<tr>
<td>21.1.3 Driver zip</td>
<td>738</td>
</tr>
<tr>
<td>Appendix</td>
<td></td>
</tr>
<tr>
<td>A Syntax diagrams for the Mosel language</td>
<td>741</td>
</tr>
<tr>
<td>A.1 Main structures and statements</td>
<td>741</td>
</tr>
</tbody>
</table>
## Contents

A.2 Expressions ................................................................. 745  
A.3 Initializations data file format ...................................... 747

B Remote Invocation Protocol ............................................. 748  
B.1 Instance control parameters ......................................... 748  
B.2 mcmd pseudo file ....................................................... 749  
B.3 Profiler interface ....................................................... 752  
B.4 Debugger interface ..................................................... 753

C Error messages ............................................................. 754  
C.1 General errors .......................................................... 754  
C.2 Parser/compiler errors ............................................... 756  
  C.2.1 Errors related to modules ...................................... 764  
  C.2.2 Errors related to packages .................................... 764  
C.3 Runtime errors .......................................................... 764  
  C.3.1 Initializations ....................................................... 765  
  C.3.2 General runtime errors ....................................... 765  
  C.3.3 BIM reader .......................................................... 767  
  C.3.4 Module manager errors ....................................... 768

D Contacting FICO ............................................................ 769  
  Product support .......................................................... 769  
  Product education ...................................................... 769  
  Product documentation ............................................... 769  
  Sales and maintenance ................................................... 770  
  Related services .......................................................... 770  
  About FICO ................................................................. 770

Index .................................................................. 771
Chapter 1
Introduction

1.1 What is Mosel?

Mosel is an environment for modeling and solving problems. To this aim, it provides a language that is both a modeling and a programming language. The originality of the Mosel language is that there is no separation between a modeling statement (e.g. declaring a decision variable or expressing a constraint) and a procedure that actually solves the problem (e.g. call to an optimizing command). Thanks to this synergy, one can program a complex solution algorithm by combining modeling and solving statements.

Each category of problem comes with its own particular types of variables and constraints and a single kind of solver cannot be efficient in all cases. To take this into account, the Mosel system does not integrate any solver by default but offers a dynamic interface to external solvers provided as modules. Each solver module comes with its own set of procedures and functions that directly extends the vocabulary and capabilities of the Mosel language. The link between Mosel and a solving module is achieved at the memory level and does not require any modification of the core system.

This open architecture can also be used as a means to connect Mosel to other software. For instance, a module could define the functionality required to communicate with a specific database.

The modeling and solving tasks are usually not the only operations performed by a software application. This is why the Mosel environment is provided either in the form of libraries or as a standalone program.

1.2 General organization

As input, Mosel expects a text file containing the source of the model/program to execute (henceforth we use just the term ‘model’ for ‘model/program’ except where there might be an ambiguity). This source file is first compiled by the Mosel compiler. During this operation, the syntax of the model is checked but no operation is executed. The result of the compilation is a Binary Model (BIM) that is saved in a second file. In this form, the model is ready to be executed and the source file is not required any more. To actually ‘run’ the model, the BIM file must be read in again by Mosel and then executed. These different phases are handled by different modules that comprise the Mosel environment:

The runtime library: This library contains the Virtual MAchine (VIMA) interpreter. It knows how to load a model in its binary format and how to execute it. It also implements a model manager (for handling several models at a time) and a Dynamic Shared Objects manager (for loading and unloading modules required by a given model). All the features of this library can be accessed from a user application.
The **compiler library**: The role of this module is to translate a source file into a binary format suitable for being executed by the VIMA Interpreter.

The **standalone application**: The ‘mosel’ application, also known as ‘Mosel Console’, is a command line interpreter linked to the two previous modules. It provides a single program to compile and execute models.

**Various modules**: These modules complete the Mosel set of functionalities by providing, for instance, optimization procedures. As an example, the `mmxprs` module extends the Mosel language with the procedure `maximize` that optimizes the current problem using the Xpress Optimizer.

This modularized structure offers various advantages:

- Once compiled, a model can be run several times, for instance with different data sets, without the need for recompiling it.
- The compiled form of the program is system and architecture independent: it can be run on any operating system equipped with the Mosel runtime library and any modules required.
- The BIM file can be generated in order to contain no symbols at all. It is then safe, in terms of intellectual property, to distribute a model in its binary form.
- As a library, Mosel can be easily integrated into a larger application. The model may be provided as a BIM file and the application only linked to the runtime library.
- The Mosel system does not integrate any kind of solver but is designed in a way that a module can provide solving facilities. The direct consequence of this is that Mosel can be linked to different solvers and communicate with them directly through memory.
- This open architecture of Mosel makes extensions of the functionality possible on a case by case basis, without the need to modify the Mosel internals.

### 1.3 Running Mosel

The Mosel environment may be accessed either through its libraries or by means of two applications, perhaps the simplest of which is Xpress Workbench, a development studio type environment for working with your Mosel models. Xpress Workbench is a complete modeling and optimization development environment that presents Mosel in an easy-to-use graphical interface with a built-in text editor.

In its standalone version, Mosel offers a simple interface to execute certain generic commands directly from the command prompt (or shell) of the operating system. The user may compile or execute source models or programs (.mos files), run binary models (.bim files) or retrieve information related to the Mosel environment itself (like properties of modules or version number of the system). An interactive debugger as well as a profiler are also included: the debugger allows to execute the model step by step, specify breakpoints from where status of the model can be examined. Running a model with the profiler provides detailed information on what part of the code is actually executed and how much time each statement requires. This information may be helpful for optimizing the model (by locating hot spots where the code is using a great deal of computer time) and also for building testsuites (by checking whether the data sets used in the test set exercise all statements of a given model).

#### 1.3.1 `mosel` command: invocation

The `mosel` executable is typically used with the following syntax from an operating system console:
Where the option \(-1\) selects the language for message translation (see Section 2.20); the option \(-d\) sets the working directory of the process; the option \(-sdm\) specifies the maximum size of stack dumps (displayed when a model terminates on a runtime error, this can also be set via the environment variable \MOSEL\_SDMAX, the default value of 0 disables the display of the stack trace); the option \(-sr\) defines the active restrictions (see Section 1.3.3) and \(-dp\) specify an initial DSO path (see Section 2.3.1). The \texttt{command} parameter is one of the following commands and \texttt{cmd\_args} are the associated arguments (square brackets indicate optional arguments):

\begin{verbatim}
\end{verbatim}

Compile the model \texttt{src} and generate the corresponding Binary Model (BIM) file if the compilation succeeds. The extension \texttt{.mos} is appended to \texttt{src} if no extension is provided. If option \texttt{\'-o outf\'} (filename to use for saving BIM file) is not given, the extension \texttt{.bim} is used to form the name of the binary file. If the flag \texttt{\'-s\'} is selected, the private object names (e.g. variables, constraints) are not saved into the BIM file. The flag \texttt{\'-g\'} adds debugging information: it is required to locate a runtime error. The flag \texttt{\'-G\'} adds both debugging and tracing information: it is required to run the model with the debugger. When the \texttt{\'-G\'} flag is used, the compiler adds instructions in the generated code that may slow down execution speed of the model. The flag \texttt{\'-I\'} may also be added to enable the \texttt{xbim} extension (see Section 2.3.3). The flag \texttt{\'-D\'} enables generation of documentation annotations in the resulting BIM file (by default documentation annotations are ignored). With the flag \texttt{\'-wi\'}, the compiler emits a warning message each time a symbol is implicitly declared and the flag \texttt{\'-ni\'} disables implicit declarations (see Section 2.8.1.3). When the flag \texttt{\'-we\'} is used warnings are handled like errors such that any warning will make the compilation fail. The option \texttt{\'-c usrcmd\'} may be used to add a commentary to the BIM file (see debugger command \texttt{LSMOD}). The options \texttt{\'-ix incpfx\'} and \texttt{\'-bx bimpfx\'} define respectively the file name prefix for file inclusion (see Section 2.5.2) and the file name prefix for package loading (see Section 2.3.1). If the flag \texttt{\'-p\'} is selected, only the syntax of the source file is checked, the compilation is not performed and no output file is generated. The flag \texttt{\'-x\'} will be used to generate a POT file (Portable Object Template) for message translation (see Section 2.20).

The other options are related to handling encrypted or signed BIM files (see Section 1.3.4): option \texttt{\'-S\'} will be used to produce a \texttt{signed} file. Unless the option \texttt{\'-pk\'} is specified, the default private key \texttt{personal.key} (see \texttt{ssl\_dir}) is used for the signature. The options \texttt{\'-V\'} and \texttt{\'-T\'} control how to handle signed packages: by default signature of packages is ignored but, if the first option is used, the signature is checked and the loading fails if it cannot be verified. With option \texttt{\'-T\'}, only signed packages with a valid signature can be used (i.e. packages without signature are not allowed). Public keys that are required for the verification are searched for in the default public keys directory \texttt{pubkeys} (see \texttt{ssl\_dir}). BIM file encryption is enabled by the \texttt{\'-E\'} option: the encryption key is either deduced from the password stated via option \texttt{\'-pwd\'} (if the flag \texttt{\'-F\'} is active, the value of \texttt{\'-pwd\'} is interpreted as a text file the first line of which is the password) or generated randomly. Optionally, the encryption key can be stored in the BIM file itself in encrypted form (this is required if it has been randomly generated): in this case the encryption requires public keys of the recipients of the BIM file (who will be able to decrypt the file using their own private keys). Public keys can be listed by using the \texttt{\'-k\'} or \texttt{\'-kf\'} options: in the first case, one public key is listed at a time (the \texttt{\'-k\'} parameter may be used several times) and in the second case a file containing a list of keys is specified. Each line of this file is interpreted as a key file name (except empty lines or lines starting with \texttt{\'#\'} or \texttt{"\#\'} that are ignored). Unless they include a path specification, key files are considered to be located in the default public keys directory (for instance the key file \texttt{"somekey"} is searched in the public keys directory but the file \texttt{"./somekey"} comes from the current working directory). An encrypted BIM file can always be decrypted by its creator thanks to his private key.
Several source files may be passed to the compiler command in a single step (this is not compatible with option ‘-o’): each file gets compiled individually.

```
        [-dbg|-prof|-cov] [-sdir dir] [-nl] bim [param=value [...]]
```

Load the provided BIM file `bim` and then run it. Options ‘-is in’, ‘-os out’ and ‘-es err’ can be specified to define alternative default input, output and error streams to be used by Mosel. With option ‘-prof’ or ‘-cov’ the model is run through the profiler (see commands profile and coverage below) and with option ‘-dbg’ it is passed to the interactive debugger (see command `debug`). The option ‘-sdir’ can be used in addition to the profiler or debugger to indicate alternative locations for source files (this option may be stated several times).

The options ‘-V’ and ‘-T’ control how to handle signed BIM files (see Section 1.3.4): by default signature of files is ignored but, if the first option is used, the signature is checked and the loading fails if it cannot be verified. With the second option, only signed BIM files with a valid signature can be used (i.e. files without signature are not allowed). For this verification task public keys are usually searched for in the default public keys directory `pubkeys` (see `ssl_dir`) but alternatively a list of expected keys may be specified with the ‘-k’ or ‘-kf’ options: in the first case, one public key is listed at a time (the ‘-k’ parameter may be used several times) and in the second case a list of keys is read from the given file. Each line of this file is interpreted as a key file name (except empty lines or lines starting with ‘!’ or ‘#’ that are ignored). Unless they include a path specification, key files are considered to be located in the default public keys directory (for instance the key file "somekey" is searched in the public keys directory but the file "./somekey" comes from the current working directory). Moreover, the special file name ‘.’ implies that keys stored in the default location can also be used.

The options ‘-pwd’ and ‘-pk’ may be required to load an encrypted BIM file: the former defines the password to use (if the flag ‘-F’ is active, the value of ‘-pwd’ is interpreted as a text file the first line of which is the password) and the option ‘-pk’ servers to specify a private key file (to be used in place of the default `personal.key` in `ssl_dir`).

Optionally, a list of parameter values may be provided in order to initialize the run-time parameters of the model and/or the control parameters of the modules used. The syntax of such an initialization is `param_name = value` for a model parameter and `dsoname.ctrpar_name = value` for a control parameter, where `dsoname` is the name of a module and `ctrpar_name` is the control parameter to set.

The option ‘-nl’ can be used when running the debugger on Unix/Linux systems to deactivate the command history if the terminal is not properly handled by the command history mechanism.

```
execute [compile_opt]s [run_opt]s src [param=value [...]]
```

Compile `src`, load, and then run the model. This command is equivalent to the consecutive execution of compile and run except that no BIM file is generated. All options documented for both, compile and run, can be used with this command. The use of option ‘-prof’ or ‘-cov’ implies the compiler flag ‘-g’ and the use of option ‘-dbg’ will also add compiler flag ‘-G’ if flag ‘-g’ is not explicitly specified.

```
debug [compile_opt]s [run_opt]s src [param=value [...]]
```

This command is equivalent to ‘execute -dbg’, the model is compiled and then run through the interactive debugger. If the model is compiled with flag ‘-G’ (the default with this command), the execution is immediately suspended before the first statement. Otherwise the execution starts as usual but can be suspended by pressing `ctrl-C`. Note that if a critical operation is being processed, the interruption is delayed until the operation completes (for instance, the Optimizer cannot be interrupted during an iteration of its algorithm). Execution is suspended once more just before the program terminates: this makes it possible to inspect model data before the end of execution. Refer to the Section 1.3.2 below for further information on the use of the debugger.
This command is equivalent to `execute -prof`, the model is compiled and then run through the profiler. After execution, the total execution time and some source coverage information is displayed. Moreover a file `sourcefile.prof` is generated based on the original source file. Each line of this file consists in:

- the number of times the corresponding statement has been executed;
- the total amount of time (in seconds) spent on this particular line (this measure is not valid if the statement is a recursive call);
- the elapsed time (in seconds) between the beginning of the execution and the last time the line was executed;
- the text of the model source

All lines of the original source file are transferred, lines that do not correspond to the beginning of a statement are directly copied without further information.

If the model runs additional submodels via `mmjobs`, a report for each model execution is also displayed and the associated annotated files are generated in a similar way as for the main model.

This command is equivalent to `execute -cov`, the model is compiled and then run through the profiler. The difference with the `profile` command described above is the type of reports generated: the files produced are taking the `.cov` extension and only collect the number of times each statement has been executed. Moreover existing files are updated instead of being replaced (i.e. iteration counts of each statement are added up).

Display the list of constants, procedures/functions, types, IO drivers, control parameters and annotations of modules, packages or the Mosel core library. Optional flags may be used to select which type of information is displayed: `'-h'` for general information, `'-c'` for constants, `'-s'` for subroutines, `'-v'` for variables, `'-r'` for requirements, `'-t'` for types, `'-i'` for IO drivers, `'-p'` for control parameters and `'-a'` for annotations. By default, listings are sorted in alphabetical order, option `'-u'` disables sorting. If both, a package and a module of the same name, are available only the information relating to the package is displayed. To select either the package or the module, extension `.bim` or `.dso` can be appended to the library name. If the flag `'-m'` is used and no package or module can be located then a binary model file is searched for in the current working directory. The displayed information is related to the Mosel core library if no name is specified with the command. The option `'-V'` can be added for checking the signature of signed BIM files (the result of the verification is reported in the header output). The options `'-pwd'` and `'-pk'` may be required to load an encrypted BIM file: the former defines the password to use (if the flag `'-F'` is active, the value of `'-pwd'` is interpreted as a text file the first line of which is the password) and the second option specifies a private key file (to be used in place of the default `personal.key` in `ssl_dir`).

Display a list of available modules and packages. Use the optional flag `'-p'` to list only packages and `'-m'` to get modules only.

If none of the above keywords is recognized, the first argument of the command is interpreted as a Mosel file. In the case of a BIM file, the command `run` is executed; otherwise the file name is passed to the command `execute`.

The `mosel` command may also be started using only flags. Besides options `'-V'` (Mosel version information) and `'-h'` (short help message), all other options relate to starting Mosel in server
mode when it is invoked from a remote instance: they should not be used directly (see the
documentation of module mmjobs in Chapter 7 for further explanations).

After the completion of a command the mosel executable returns a non-zero status to the
operating system in case of error and the execution status of the model if a model has been run
(e.g. with the command execute). This execution status is the value provided via the procedure
exit in the model (by default this is 0).

Some examples:
Execute model ‘mymodel.mos’ setting values for the model parameters A,B,C and D

    > mosel mymodel A=33 B="word" C=true D=5.3e-5

Compile model ‘m.mos’ located on a web service and store the bim file locally in compressed form

    > mosel comp -o zlib.gzip:m.bim.gz mmhttp.url:http://websrv/m.mos

Run ‘optmod.bim’ from the debugger enabling verbose mode of module ‘mmxprs’

    > mosel run -dbg optmod mmxprs.XPRS_verbose=true

List all available modules and packages

    > mosel lslib

Display the list of subroutines defined by ‘mmxprs’

    > mosel exam -s mmxprs

Display all constants defined in the Mosel language

    > mosel exam -c

Display version information of Mosel

    > model -V

1.3.2 mosel command: interactive debugger

When a model that is executed through the debugger is interrupted (for instance, because the
user has typed ctrl-C or an error has occurred), the execution is suspended, the text source of the
statement being processed is displayed and an interactive session starts. This mode is signaled by
the specific prompt ‘>dbg’ and the following commands may be entered (the arguments enclosed
in square brackets [] are optional). The command line interpreter is case-insensitive, although we
display commands in upper case for clarity:

**BCONDITION**  
Define or remove a condition on a breakpoint. This command may be used to put a
condition (Boolean expression) on the specified break point: the execution is suspended
at the breakpoint only if the given condition is verified. To remove a condition previously
set up, enter this command without specifying any condition.

**BREAK**  
Install a breakpoint. When a breakpoint has been set up, execution is interrupted
whenever the statement corresponding to the specified location is reached. A procedure
or function name may be used as the location: in this case a breakpoint is installed at the beginning of each procedure or function of the provided name. If this command is used without parameters, the breakpoint is defined at the current location.

BREAKPOINTS
List the defined breakpoints.

BREAKSUB [0|1]
Decide whether to suspend execution whenever a submodel is started.

CONTINUE
Resume execution. If the interruption was not due to an error, execution of the model continues, otherwise the execution of the model is aborted and Mosel exits.

DELETE [bk]
Delete a breakpoint.

DISPLAY [expression]
Record an expression to be displayed at every interruption. Used with no expression, this command gives a list of all recorded expressions.

DOWN [nblev]
Go down in the calling stack. If an argument is provided, it indicates how many levels down to go (default is 1).

EXPORTPROB [-pms] [filename [objective]]
Display or save to the given file (option filename) the matrix corresponding to the active problem. The matrix output uses the LP format (default) or the MPS format (flag ‘-m’). A problem is available after the execution of a model. The flags may be used to select the direction of the optimization (‘-p’: maximize), the file format (‘-m’: MPS format) and whether real object names should be used (‘-s’: scrambled names — this is the default if the object names are not available). The objective may also be selected by specifying a constraint name.

FINISH
Continue execution until the end of the current subroutine. The execution continues but will be interrupted again after the subroutine terminates.

INFO [*|symbol [symbol...]]
Without arguments, this command displays information about the program being executed (this may be useful for problem reporting). Any specified argument is interpreted as a symbol from the current model. If the requested symbol exists in the model, this command displays some information about its type and structure. Several symbols may be given in a single call and if ‘*’ is used in place of a symbol name then the information is displayed for every symbol of the model.

LIST [[start] nblines]
Display the source file that corresponds to the model being executed. When used with no extra argument, this command lists 10 lines of the source model starting at the current statement; used with a single positive parameter nblines, it displays nblines lines instead of the default 10 lines. If the parameter nblines is negative, it is interpreted as a starting point for the listing relative to the current statement. When 2 parameters are used, the first one is understood as the first line to display (a negative value is relative to the current line) and the second one as the number of lines to display. Examples (assuming current line is 5):

>list displays lines 5 to 14
>list 5 displays lines 5 to 9
>list -2 displays lines 3 to 14
>list -2 5 displays lines 3 to 7
LSATTR [typename]
Display the list of available attributes for all used native types or only those related to the specified type typename

LSLIBS
Display the list of all loaded dynamic shared objects (DSO) together with, for each module, its version number and its number of references (i.e. number of loaded models using it).

LSLOCAL
Display the list of symbols defined locally to the current context.

LSMODS
Display the list of all models currently loaded in core memory. The information displayed for each model is:

- name: the model name and version number (given by the model and version statements in the source file);
- number: the model number is automatically assigned when the model is loaded;
- size: the amount of memory used by the model (in bytes);
- system comment: a text string generated by the compiler indicating the source filename and if the model contains debugging information and/or symbols;
- user comment: the comment defined by the user at compile time (cf. command compile);
- modules: the name and version number of each module required by the model;
- pkg. req.: if the model is a package, the name and version number of each package required by a model using this package;
- pkg. imp.: the name and version number of each package included by this model.
The active model is marked by an asterisk ('*') in front of its name.

LSSYMB [-cspou]
Display the list of symbols published by the current model. The optional flags may be used to filter what kind of symbol to display: '-c' for constants, '-s' for subroutines, '-p' for parameters and '-o' for everything else. By default the list is sorted in alphabetical order, option '-u' disables sorting.

MODEL [modnum]
With no argument this command lists all models running concurrently. The active model (debugger commands are applied to this model) is identified by a star ("*"). If provided, the argument is interpreted as a model number that becomes the active model.

NEXT [line [file]]
Continue execution until the next statement. The execution continues but will be interrupted again after the current statement has been completed. If a location information is provided (by means of a line number and, if necessary, a file name), the next interruption will occur before the specified statement is executed.

OPTION name [=] value
View or change the value of a command line parameter. These parameters are used by the command line interpreter to display real values (especially in command PRINT):

- realfmt: C-style format for printing floating point numbers (default value: "%g")
- zerotol: zero tolerance to decide whether two values are equal (default value: 1e-13). It is also used when printing very small numbers: if a value is smaller than zerotol, "0" is displayed instead.
Although these parameters have the same name and function as those used by Mosel when running a model, they are not synchronised with their internal counterpart.

**PRINT expression [>>filename]**
Evaluate then display the value of the given arithmetic or Boolean expression. For building the expression, the following functions can be used: `getparam`, `ceil`, `floor`, `round`, `abs`, `getsize`, `getmodprop` as well as all attributes (see `LSATTR` command above). In addition to these Mosel functions, the interpreter implements `getnbdim` that returns the number of dimensions of an array and `getndx#` that gets the index set of dimension number `'#' of an array (`'#' being an integer between 1 and the number of dimensions of the array). `get-functions` may be called using the suffix notation (e.g. `getact(c)` is equivalent to `c.act`). Some functions can be applied to arrays: the result is the evaluation of the function for each cell of the array. Private symbols of packages may be accessed by prefixing the symbol name by the package name and the symbol `˜` (for instance the identifier `aa` declared in the package `mypkg` can be accessed using `mypkg˜aa`). It is possible to report only a part of a collection (array, set or list) by specifying range information. Ranges definitions take one of these two forms:

- `[ maxelt ]`: get at most `maxelt` elements
- `[ skip maxelt ]`: get at most `maxelt` entries after skipping `skip` elements

Several range definitions may be specified (separated by blanks): they are used when exploring complex structures (e.g. a list of list).

The display format of this command is compatible with the data file format of Mosel. Use the operator `>>filename` to append output of the command to the file `filename`.

Examples:

```
>print getsol(x) >> solfile.txt
>print getact(C(1,"tut")+c.size
>print toto˜a
>print abs(mytol)>1
>print myarray.ndx2 [3]
```

**QUIT**
Terminate the debug session. Model execution is aborted and Mosel exits.

**STEP**
Continue execution until the next statement stepping into procedures and functions. The execution continues but will be interrupted again after the current statement has been completed. If the current statement contains function or procedure calls, interruption will happen in these procedures or functions.

**UNDISPLAY [disp]**
Remove an expression recorded with `DISPLAY`. If no parameter is provided, all recorded expressions are removed, otherwise the parameter is understood as a record number.

**UP [nblev]**
Go up in the calling stack. If an argument is provided, it indicates how many levels up to go (default is 1). Note that expressions are evaluated according to the current stack frame. For example, if variable `i` is defined in procedure `B` and execution is suspended in procedure `A` called by `B`; it is necessary to go up in the stack in order to view the value of `i` because it does not exist in the current frame.

**WHERE [nblev]**
Display the calling stack. The calling stack corresponds to the sequence of procedure and function calls being processed. For instance assume the model calls procedure `A` which calls procedure `B` and the execution is suspended in procedure `B`: the calling stack will contain 3 records (location where `A` is called, location where `B` is called and current statement).
If a command is not recognized, a list of possible keywords is displayed together with a short explanation. The command names can be shortened as long as there is no ambiguity (e.g. \texttt{un} can be used in place of \texttt{UNDISPLAY} but \texttt{u} is not sufficient because it could equally denote the \texttt{UP} command). String arguments (the parameter \texttt{10} is a number, but "\texttt{10}" or '10' are text strings) may be quoted with either single or double quotes. Quoting is required if the text string starts with a digit or contains spaces and/or quotes.

Execution step by step and breakpoints can be used only if the model has been compiled using option \texttt{-G}. In this case, before the execution starts, a breakpoint is automatically put at the first statement of the model. Otherwise (model has been compiled with option \texttt{-g}), the model will be interrupted only if an error occurs or keys ctrl-C are pressed.

When debugging a model that runs submodels via \texttt{mmjobs} a message is displayed each time a submodel starts or terminates. Moreover, interrupting the execution of the model also suspends the execution of all submodels: the entered commands are applied to the selected active model, the choice of which can be changed with the command \texttt{MODEL}.

1.3.3 mosel command: restricted mode

Mosel may be run in restricted mode: by selecting which restrictions are to be applied, it is possible to control what operations models can perform (in particular regarding disk access). Upon startup, if the option \texttt{-sr} is not stated, the command line interpreter uses the value of the environment variable \texttt{MOSEL\_RESTR} for setting the execution restrictions. These restrictions are bit-encoded as an integer (each bit corresponding to a specific restriction) but restrictions can also be expressed by a list of one or more of the following keywords (symbols are not case-sensitive and can be optionally separated by spaces):

\begin{itemize}
\item \texttt{NoWrite} (bit 0, value 1) \
\quad Disable write access on the local system. This restriction concerns all file access except databases. Access to the temporary directory is not affected.
\item \texttt{NoRead} (bit 1, value 2) \
\quad Disable read access on the local system (this also implies \texttt{NoWrite}). This restriction concerns all file access except databases. When this option is selected, the current working directory is automatically set to the temporary directory (which can still be accessed).
\item \texttt{NoExec} (bit 2, value 4) \
\quad Disable external command execution. This restriction deactivates some procedures/functions allowing execution of commands external to Mosel (for instance \texttt{system} or \texttt{command}). Also, Mosel can only load modules from read-only locations when this restriction is active.
\item \texttt{WDOnly} (bit 3, value 8) \
\quad File access is limited to the current working directory and its subdirectories as well as the paths specified by the environment variables \texttt{MOSEL\_RWPATH} (for reading and writing) and \texttt{MOSEL\_ROPATH} (for reading only). The temporary directory can still be accessed.
\item \texttt{NoTmp} (bit 4, value 16) \
\quad Access to the temporary directory is disabled.
\item \texttt{NoDB} (bit 5, value 32) \
\quad Disable access to databases by blocking connection routines (e.g. \texttt{SQLconnect} or \texttt{OCIlogon}).
\end{itemize}

For example, to disable write access and execution of external commands the environment variable \texttt{MOSEL\_RESTR} will have to be either the integer value 5 (1+4) or the string "\texttt{NoWrite NoExec}".
Restricted mode is observed by the Mosel core libraries (when accessing files and managing directories) and the system requires that modules also satisfy the stated restrictions (although implementation of restrictions may vary depending on the type of functionality provided by a given module): a module that does not support the restricted mode of execution will fail to load when Mosel is running in this mode.

1.3.4 mosel command: securing bim files

The bim file format is secure with respect to the intellectual property of the author of the model (i.e. it is not possible to recover the original model from the bim file). However, further security mechanisms may be required when a bim file is to be transferred over an insecure media (like the internet): in particular it might be necessary to (1) make sure the file has not been modified during the transfer and (2) guarantee that only the addressee can access the file.

A digital signature ensures the first requirement: it is computed using a private key (exclusively owned by the sender of the document) such that any addressee having the corresponding public key (provided by the sender) can, at the same time, verify that the document has been prepared by the sender and that it has not been altered during the transfer. From the Mosel command line tool, creating a signed bim file can be done by using the ‘-S’ compiler option. When loading a signed bim file with the run command, it is required to enable the signature verification with options ‘-V’ or ‘-T’ as verification is not performed by default.

The second requirement can be satisfied by encrypting the bim file such that it appears as random data during the transfer. Mosel supports two kinds of encryption processes: it can use a usual password based key. In this case the same password is used for both encrypting and decrypting the bim file (the sender and the recipient have to share this key). The alternative is to rely on private/public key pairs like for the signature procedure outlined above: encryption is achieved with the public key of the addressee. Only the recipient will be able to decrypt the bim file using his private key. From the Mosel command line tool, creating an encrypted bim file can be done by using the ‘-E’ compiler option. A password is specified with the ‘-pwd’ option otherwise the public keys of the recipients have to be stated with the ‘-k’ or ‘-kf’ options (a bim file can be encrypted for up to 128 public keys).

Both signature and encryption require the management of private and public keys. These keys are expected to be stored in a predefined location specified by the module parameter ssl_dir.

Mosel relies on the RSA cryptographic system for the management of private/public key pairs (keys must be of at least 1024bits). The signature procedure uses the SHA256 message digest algorithm. Bim files are encrypted using the AES block cipher with keys of 128 bits.

1.4 References

Mosel could be described as an original combination of a couple of well known technologies. Here is a non-exhaustive list of the most important ‘originators’ of Mosel:

- The overall architecture of the system (compiler, virtual machine, native interface) is directly inspired by the Java language. Similar implementations are also commonly used in the languages for artificial intelligence (e.g. Prolog, Lisp).
- The syntax and the major building blocks of the Mosel language are in some aspects a simplification and for other aspects extensions of the Pascal language.
- The aggregate operators (like ‘sum’) are inherited from the ‘tradition of model builders’ and can be found in most of today’s modeling languages.
- The dynamic arrays and their particular link with sets are probably unique to Mosel but are at their origin a generalization of the sparse tables of the mp-model model builder.
1.5 Structure of this manual

The main body of this manual is essentially organized into two parts. In Chapter 2, the basic building blocks of Mosel's modeling and programming language are discussed.

Chapter 3 begins the reference section of this manual, providing a full description of all the functions and procedures defined as part of the core Mosel language. The functionality of the Mosel language may be expanded by loading modules: the following chapters describe the modules currently provided with the standard Mosel distribution.
I. Core System
CHAPTER 2

The Mosel Language

The Mosel language can be thought of as both a modeling language and a programming language. Like other modeling languages it offers the required facilities to declare and manipulate problems, decision variables, constraints and various data types and structures like sets and arrays. On the other hand, it also provides a complete set of functionalities proper to programming languages: it is compiled and optimized, all usual control flow constructs are supported (selection, loops) and can be extended by means of modules. Among these extensions, optimizers can be loaded just like any other type of modules and the functionality they offer may be used in the same way as any Mosel procedures or functions. These properties make of Mosel a powerful modeling, programming and solving language with which it is possible to write complex solution algorithms.

The syntax has been designed to be easy to learn and maintain. As a consequence, the set of reserved words and syntax constructs has deliberately been kept small avoiding shortcuts and ‘tricks’ often provided by modeling languages. These facilities are sometimes useful to reduce the size of a model source (not its readability) but also are likely to introduce inconsistencies and ambiguities in the language itself, making it harder to understand and maintain.

2.1 Introduction

2.1.1 Comments

A comment is a part of the source file that is ignored by the compiler. It is usually used to explain what the program is supposed to do. Either single line comments or multi lines comments can be used in a source file. For the first case, the comment starts with the ‘!’ character and terminates with the end of the line. A multi-line commentary must be inclosed in ‘(!’ and ‘!)’. Note that it is possible to nest several multi-line commentaries.

! In a comment
This text will be analyzed
(! Start of a multi line
  (! Another comment
    blabla
    end of the second level comment !)
 end of the first level !) Analysis continues here

Comments may appear anywhere in the source file.

2.1.2 Identifiers

Identifiers are used to name objects (variables, for instance). An identifier is an alphanumeric (plus ‘_’) character string starting with an alphabetic character or ‘_’. All characters of an identifier are significant and the case is important (the identifier ‘word’ is not equivalent to ‘Word’).
2.1.3 Reserved words

The reserved words are identifiers with a particular meaning that determine a specific behaviour within the language. Because of their special role, these *keywords* cannot be used to name user-defined objects (i.e., they cannot be redefined). The list of reserved words is:

- `and`, `array`, `as`, `boolean`, `break`, `case`, `count`, `counter`, `declarations`, `div`, `do`, `dynamic`, `elif`, `else`, `end`, `evaluation`, `false`, `forall`, `forward`, `from`, `function`, `if`, `imports`, `in`, `include`, `initialisations`, `initializations`, `integer`, `inter`, `is_binary`, `is_continuous`, `is_free`, `is_integer`, `is_partint`, `is_semcont`, `is_seminit`, `is_sos1`, `is_sos2`, `linctr`, `list`, `max`, `min`, `mod`, `model`, `mpvar`, `next`, `not`, `of`, `options`, `or`, `package`, `parameters`, `procedure`, `public`, `prod`, `range`, `real`, `record`, `repeat`, `requirements`, `return`, `set`, `string`, `sum`, `then`, `to`, `true`, `union`, `until`, `uses`, `version`, `while`, `with`.

Note that, although the lexical analyzer of Mosel is case-sensitive, the reserved words are defined both as lower and upper case (i.e. `AND` and `and` are keywords but not `And`).

2.1.4 Separation of instructions, line breaking

In order to improve the readability of the source code, each statement may be split across several lines and indented using as many spaces or tabulations as required. However, as the line breaking is the expression terminator, if an expression is to be split, it must be cut after a symbol that implies a continuation like an operator (`+', '-', ...) or a comma (`,'`) in order to warn the analyzer that the expression continues in the following line(s).

```
A+B ! Expression 1
-A+C+D ! Expression 2
A+B- ! Expression 3...
C+D ! ...end of expression 3
```

Moreover, the character `';'` can be used as an expression terminator.

```
A+B ; -C+D ! 2 expressions on the same line
```

Some users prefer to explicitly mark the end of each expression with a particular symbol. This is possible using the option `explterm` (see Section 2.3) which disables the default behaviour of the compiler. In that case, the line breaking is not considered any more as an expression separator and each statement finishing with an expression must be terminated by the symbol `';'`.

```
A+B; ! Expression 1
-A+C+D; ! Expression 2
A+B; ! Expression 3...
-C+D; ! ...end of expression 3
```

2.1.5 Conventions in this document

In the following sections, the language syntax is explained. In all code templates, the following conventions are employed:

- **word**: ‘word’ is a keyword and should be typed as is;
- **todo**: ‘todo’ is to be replaced by something else that is explained later;
- `[ something ]`: ‘something’ is optional and the entire block of instructions may be omitted;
- `[ something ...]`: ‘something’ is optional but if used, it can be repeated several times.
2.2 Structure of the source file

The Mosel compiler may compile both *models* and *packages* source files. Once compiled, a model is ready for execution but a package is intended to be used by a model or another package (see Section 2.3).

The general structure of a model source file is as follows:

```plaintext
model model_name
  [ Directives ]
  [ Parameters ]
  [ Body ]
end-model
```

The `model` statement marks the beginning the program and the statement `end-model` its end. Any text following this instruction is ignored (this can be used for adding plain text comments after the end of the program). The model name may be any quoted string or identifier, this name will be used as the model name in the Mosel model manager. An optional set of `directives` and a `parameters` block may follow. The actual program/model is described in the `body` of the source file which consists of a succession of declaration blocks, subroutine definitions and statements.

The structure of a *package* (see Section 2.12) source file is similar to the one of a model:

```plaintext
package package_name
  [ Directives ]
  [ Parameters ]
  [ Body ]
end-package
```

The `package` statement marks the beginning the library and the statement `end-package` its end. The package name must be a valid identifier.

It is important to understand that the language is *procedural* and not *declarative*: the declarations and statements are compiled and executed in the order of their appearance. As a consequence, it is not possible to refer to an identifier that is declared later in the source file or consider that a statement located later in the source file has already been executed. Moreover, the language is *compiled* and not *interpreted*: the entire source file is first translated — as a whole — into a binary form (the *BIM file*), then this binary form of the program is read again to be executed. During the compilation, except for some simple constant expressions, no action is actually performed. This is why only some errors can be detected during the compilation time, any others being detected when running the program.

2.3 The compiler directives

The compiler accepts four different types of directives: the `uses` statement, the `imports` statement, the `options` statement and the `version` statement.

2.3.1 Directive uses

The general form of a `uses` statement is:

```plaintext
uses libname1 [, libname2 ...][;]
```

This clause asks the compiler to load the listed modules or packages and import the symbols they
define. Modules must still be available for running the model but packages are incorporated into the generated bim file when compiling a model. If the source file being processed is a package, the bim files associated to the listed packages must be available for compiling another file using this package. It is also possible to merge bim files of several packages by using imports instead of uses when building packages.

By default the compiler tries first to find a package (the corresponding file is libname.bim) then, if this fails, it searches for a module (which file name is libname.dso). It is possible to indicate the type of library to look for by appending either ".bim" or ".dso" to the name (then the compiler does not try the alternative in case of failure). A package may also be specified by an extended file name (see Section 2.14) including the IO driver in order to disable the automatic search (i.e. "a.bim" searches the file a.bim in the library path but ":a.bim" takes the file a.bim from the current directory).

For example,

```
uses 'mmsystem','mmxprs.dso','mypkg.bim'
uses '://tmp/otherpkg.bim'
```

Both packages and modules are searched in a list of possible locations. Upon startup, Mosel uses as the default for this list the value of the environment variable MOSEL_DSO completed by a path deduced from the location (rtdir) of the Mosel runtime library (in the following # can be "32" on a 32bit system, "64" on a 64bit system or an empty string):

```
"rtdir\.dso#" Under Windows if rtdir terminates by ":\bin#" and "rtdir\.dso#" exists or
"rtdir/../dso#" On Posix systems if rtdir terminates by ":/lib#" and "rtdir/../dso#" exists or
"rtdir/dso#" if this directory exists or
"rtdir" if none of the above rules apply
```

The variable MOSEL_DSO is expected to be a list of paths conforming to the operating system conventions: for a Posix system the path separator is ':' (e.g. "/opt/Mosel/dso:/tmp") and it is ';' under Win32 (e.g. "E:\Mosel\Dso;C:\Temp"). The search path for modules and packages may also be set from the mosel command (using the -dp option, see Section 1.3) as well as inspected and modified from the Mosel Libraries (see functions XPRMgetdsopath and XPRMsetdsopath in the Mosel Libraries Reference Manual). Note however that Mosel will ignore modules not located in read-only locations when the restriction NoExec is active (see Section 1.3.3).

If the compiler option -bx is used (Section 1.3) a package with the specified prefix will be tried before proceeding to the search as decribed above. For instance if the option -bx "bimdir/" is used with the directive uses 'mypkg', the compiler will try to load the package "bimdir/mypkg.bim" before looking for "mypkg.bim" and "mypkg.dso" in the usual locations.

### 2.3.2 Directive imports

The general form of an imports statement is:

```
imports pkgname1 [, pkgname2 ...][;]
```

This clause is a special version of the uses directive that can only be used in packages: it asks the compiler to load the listed packages, import the symbols they define and incorporate the corresponding bim file. As a consequence, the generated package provides the functionality of the packages it imports.

For example,

```
imports 'mypkg'
```
2.3.3 Directive options

The compiler options may be used to modify the default behaviour of the compiler. The general form of an options statement is:

```
options optname1 [, optname2 ...]
```

The supported options are:

- **explterm**: asks the compiler to expect explicit expression termination (see Section 2.1.4).
- **noimplicit**: disables the implicit declarations (see Section 2.8.1.3). This option can also be activated by using the ‘-ni’ compiler flag (see Section 1.3).
- **noautofinal**: by default initialization blocks finalize sets they populate (section 2.8.2.1). This option disables this behaviour that may be activated afterwards using the autofinal control parameter (cf. setparam).
- **keepassert**: assertions (cf. assert) are compiled only in debug mode. With this option assertions are preserved regardless of the compilation mode.
- **xbim**: store additional symbol information in the generated bim file (in particular array index names). This option can also be enabled by using the ‘-I’ compiler flag (see Section 1.3).
- **fctasproc**: by default return values of functions must be used such that a function call is not a valid statement. With this option functions can be used as procedures: when a statement consists in a function call its return value is silently ignored (see also asproc).

For example,

```
options noimplicit,explterm
```

2.3.4 Directive version

In addition to the model/package name, a file version number may be specified using this directive: a version number consists in 1, 2 or 3 integers between 0 and 999 separated by the character ‘.’.

```
version major [. minor [. release ]]
```

For example,

```
version 1.2
```

The file version is stored in the BIM file and can be displayed from the Mosel console (command list) or retrieved using the Mosel Libraries (see function XPRMgetmodprop in the Mosel Libraries Reference Manual). From the model itself, the version number is recorded as a string in the control parameter model_version (see function getparam).

2.4 The parameters block

A model parameter is a symbol, the value of which can be set just before running the model (optional parameter of the ‘run’ command of the command line interpreter). The general form of the parameters block is:
The Mosel Language

parameters
ident1 = Expression1
[ ident2 = Expression2 ...]
end-parameters

where each identifier identi is the name of a parameter and the corresponding expression Expressioni its default value. This value is assigned to the parameter if no explicit value is provided at the start of the execution of the program (e.g. as a parameter of the ‘run’ command). Note that the type (integer, real, text string or Boolean) of a parameter is implied by its default value. Model parameters are manipulated as constants in the rest of the source file (it is not possible to alter their original value).

parameters
size=12 ! Integer parameter
R=12.67 ! Real parameter
F=“myfile” ! Text string parameter
B=true ! Boolean parameter
end-parameters

In addition to model parameters, Mosel and some modules provide control parameters: they can be used to give information on the system (e.g. success of an I/O operation) or control its behaviour (e.g. select output format of real numbers). These parameters can be accessed and modified using the routines getparam and setparam. Refer to the documentation of these functions for a complete listing of available Mosel parameters. The documentation of the modules include the description of the parameters they publish.

2.5 Source file preprocessing

2.5.1 Source file character encoding

The Mosel compiler expects source files to be encoded in UTF-8 and will handle properly UTF-16 and UTF-32 encodings when the file begins with a BOM (Byte Order Mark). It is also possible to select an alternative encoding using the encoding annotation (see section 2.13).

For instance to notify the compiler that the source file is encoded using ISO-8859-1, the following comment has to be copied at the beginning of the file:

!@encoding:iso-8859-1

2.5.2 Source file inclusion

A Mosel program may be split into several source files by means of file inclusion. The ‘include’ instruction performs this task:

include filename

where filename is the name of the file to be included. This file name may contain environment variable references using the notation ${varname} (e.g. ‘${MOSEL}/examples/mymodel’) that are expanded to generate the actual name. The ‘include’ instruction is replaced at compile time by the contents of the file filename.

Assuming the file a.mos contains:

model "Example for file inclusion"
writeln(‘From the main file’)


include "b.mos"
end-model

And the file b.mos:

writeln('From an included file')

Due to the inclusion of b.mos, the file a.mos is equivalent to:

model "Example for file inclusion"
writeln('From the main file')
writeln('From an included file')
end-model

If the compiler option -ix is used (Section 1.3) all file names used in the 'include' instruction will be prefixed as requested. For instance, if the option -ix "incdir/" is used with the compiler, the statement include "myfile.mos" will be replaced by the content of "incdir/myfile.mos".

Note that file inclusion cannot be used inside of blocks of instructions or before the body of the program (as a consequence, a file included cannot contain any of the following statements: uses, options or parameters).

2.5.3 Line control directives

In some cases it may be useful to process a Mosel source through an external preprocessor before compilation. For instance this may enable the use of facilities not supported by the Mosel compiler like macros, unrestricted file inclusion or conditional compilation. In order to generate meaningful error messages, the Mosel compiler supports line control directives: these directives are inserted by preprocessors (e.g. cpp or m4) to indicate the original location (file name and line number) of generated text.

```
#(line) linenum [filename]
```

To be properly interpreted, a line control directive must be the only statement of the line. Malformed directives and text following valid directives are silently ignored.

2.6 The declaration block

The role of the declaration block is to give a name, a type, and a structure to the entities that the processing part of the program/model will use. The type of a value defines its domain (for instance integer or real) and its structure, how it is organized, stored (for instance a reference to a single value or an ordered collection in the form of an array). The declaration block is composed of a list of declaration statements enclosed between the instructions declarations and end-declarations.

```
declarations
    Declare_stat
    [ Declare_stat ...]
end-declarations
```

Several declaration blocks may appear in a single source file but a symbol introduced in a given block cannot be used before that block. Once a name has been assigned to an entity, it cannot be reused for anything else.
2.6.1 Elementary types

Elementary objects are used to build up more complex data structures like sets or arrays. It is, of course, possible to declare an entity as a reference to a value of one of these elementary types. Such a declaration looks as follows:

\[ \text{ident1 [, ident2 ...]: type\_name} \]

where type\_name is the type of the objects to create. Each of the identifiers identi is then declared as a reference to a value of the given type. The type name may be either a basic type (integer, real, string, boolean), an MP type (mpvar, linctr), an external type or a user defined type (see section 2.6.7). MP types are related to Mathematical Programming and allow declaration of decision variables and linear constraints. Note that the linear constraint objects can also be used to store linear expressions. External types are defined by modules (the documentation of each module describes how to use the type(s) it implements).

```
declarations
  i,j: integer
  str: string
  x,y,z: mpvar
end-declarations
```

2.6.1.1 Basic types

The basic types are:

- **integer**: an integer value between \(-2147483648\) and \(2147483647\)
- **real**: a real value between \(-1.7e+308\) and \(1.7e+308\).
- **string**: some text.
- **boolean**: the result of a Boolean (logical) expression. The value of a Boolean entity is either the symbol `true` or the symbol `false`.

After its declaration, each entity receives an initial value of `0`, an empty string, or `false` depending on its type.

2.6.1.2 MP types

Two special types are provided for mathematical programming.

- **mpvar**: a decision variable
- **linctr**: a linear constraint

2.6.2 Sets

Sets are used to group an unordered collection of elements of a given type. Set elements are unique: if an element is added several times it is only contained once in the set. Declaring a set consists of defining the type of elements to be collected.

The general form of a set declaration is:

\[ \text{ident1 [, ident2 ...]: set of type\_name} \]
where \textit{type\_name} is one of the elementary types. Each of the identifiers \textit{identi} is then declared as a set of the given type.

A particular set type is also available that should be preferred to the general form wherever possible because of its better efficiency: the range set is an ordered collection of consecutive integers in a given interval. The declaration of a range set is achieved by:

\begin{verbatim}
ident1 [ident2 ...]: range [set of integer]
\end{verbatim}

Each of the identifiers \textit{identi} is then declared as a range set of integers. Every newly created set is empty.

```
declarations
    s1: set of string
    r1: range
end-declarations
```

### 2.6.3 Lists

Lists are used to group a collection of elements of a given type. An element can be stored several times in a list and order of the elements is specified by construction. Declaring a list consists of defining the type of elements to be collected.

The general form of a list declaration is:

\begin{verbatim}
ident1 [ident2 ...]: list of type\_name
\end{verbatim}

where \textit{type\_name} is one of the elementary types. Each of the identifiers \textit{identi} is then declared as a list of the given type.

Every newly created list is empty.

```
declarations
    l1: list of string
    l2: list of real
end-declarations
```

### 2.6.4 Arrays

An array is a collection of labelled objects of a given type. A label is defined by a list of indices taking their values in domains characterized by sets: the indexing sets. An array may be either of fixed size or dynamic. For fixed size arrays, the size (i.e. the total number of objects it contains, or cells) is known when it is declared. All the required cells (one for each object) are created and initialized immediately. Dynamic arrays are created empty. The cells are created explicitly (\textit{cf.} procedure \texttt{create}) or when they are assigned a value (\textit{cf.} Section 2.8.1.1) and the array may then grow 'on demand'. It is also possible to delete some or all cells of a dynamic array using the procedure \texttt{delcell}. A cell that has not been created can be identified using the \texttt{exists} function and its value is the default initial value of the type of the array. The general form of an array declaration is:

\begin{verbatim}
ident1 [ident2 ...]: [dynamic] array(list\_of\_sets) of type\_name
\end{verbatim}

where \textit{list\_of\_sets} is a list of set declarations/expressions separated by commas and \textit{type\_name} is one of the elementary types. Each of the identifiers \textit{identi} is then declared as an array of the given type and indexed by the given sets. In the list of indexing sets, a set declaration can be anonymous (i.e. \texttt{rs: set of real} can be replaced by \texttt{set of real} if no reference to \texttt{rs} is required) or shortened to the type of the set (i.e. \texttt{set of real} can be replaced by \texttt{real} in that context).
An array is of fixed size if all of its indexing sets are of fixed size (i.e. they are either constant or finalized (cf. procedure finalize)). If the qualifier dynamic is used, the array is dynamic and created empty. Otherwise (at least one indexing set is not constant), the array is created with as many cells as possible (i.e., the array is empty if one of the indexing sets is not initialized) and may grow if necessary. Such an array is not the same as a dynamic array even if it is created empty: Mosel may use a dedicated internal representation through which the creation of a single cell (via an assignment for instance) may induce the creation of a row of adjacent cells. Also, if no cell can be created at declaration time, the array is effectively allocated when it is first accessed. As a consequence, if all its indexing sets are finalized at that time, the array is created as a fixed size array. The following example shows the different behaviour of an array that is simply declared with unknown index set (a and c) and an explicit dynamic array (b).

Note that once a set is employed as an indexing set, Mosel makes sure that its size is never reduced in order to guarantee that no entry of any array becomes inaccessible. Such a set is called fixed.

2.6.4.1 Special case of dynamic arrays of a type not supporting assignment

Certain types do not have assignment operators: for instance, writing \texttt{x:=1} is a syntax error if \texttt{x} is of type \texttt{mpvar}. If an array of such a type is defined as dynamic or the size of at least one of its indexing sets is unknown at declaration time (i.e. empty set), the corresponding cells are not created. In that case, it is required to create each of the relevant entries of the array by using the procedure \texttt{create} since entries cannot be defined by assignment.

2.6.5 Records

A record is a finite collection of objects of any type. Each component of a record is called a field and is characterized by its name (an identifier) and its type. The general form of a record declaration is:

\begin{verbatim}
declarations
ident1[,ident2...]:record
  field1[,field2...]:type_name
  [...]
end-record
\end{verbatim}

where \texttt{fieldi} are the identifiers of the fields of the record and \texttt{type_name} one of the elementary types. Each of the identifiers \texttt{identi} is then declared as a record including the listed fields.

Example:
declarations
  r1: record
    i,j:integer
    r:real
  end-record
end-declarations

Each record declaration is considered unique by the compiler. In the following example, although r1 and r2 have the same definitions, they are not of the same type (but r3 is of course of the type of r2):

declarations
  r1: record
    i,j:integer
  end-record
  r2,r3: record
    i,j:integer
  end-record
end-declarations

2.6.6 Constants

A constant is an identifier for which the value is known at declaration time and that will never be modified. The general form of a constant declaration is:

\[ \text{identifier} = \text{Expression} \]

where identifier is the name of the constant and Expression its initial and only value. The expression must be of one of the basic types, a set or a list of one of these types.

Example:

declarations
  STR='my const string'
  I1=12
  R=1..10 ! constant range
  S={2.3,5.6,7.01} ! constant set
  L=[2,4,6] ! constant list
end-declarations

The compiler supports two kinds of constants: a compile time constant is a constant which value can be computed by the compiler. A run time constant will be known only when the model is run.

Example:

parameters
  P=0
end-parameters
declarations
  I=1/3 ! compile time constant
  J=P*2 ! run time constant
end-declarations

2.6.7 User defined types

2.6.7.1 Naming new types

A new type may be defined by associating an identifier to a type declaration. The general form of a type definition is:

\[ \text{identifier} = \text{Type_def} \]
where \textit{Type\_def} is a type (elementary, set, list, array or record) to be associated to the symbol \textit{identifier}. After such a definition, the new type may be used wherever a type name is required.

Example:

\begin{verbatim}
definitions
  entier=integer
  setint=set of entier
  i:entier ! <= i:integer
  s:setint ! <= s:set of integer
end-declarations
\end{verbatim}

Note that only compile time constant or globally defined sets are allowed as indices to array types:

\begin{verbatim}
definitions
  ar1=array(1..10) of integer ! OK
  ar2=array(range) of integer ! incorrect
  R:range
  ar3=array(R) of integer ! OK
end-declarations
\end{verbatim}

2.6.7.2 Combining types

Thanks to user defined types one can create complex data structures by combining structures offered by the language. For instance an array of sets may be defined as follows:

\begin{verbatim}
definitions
typset=set of integer
  a1:array(1..10) of typset
end-declarations
\end{verbatim}

In order to simplify the description of complex data structures, the Mosel compiler can generate automatically the intermediate user types. Using this property, the example above can be written as follows (both arrays \textit{a1} and \textit{a2} are of the same type):

\begin{verbatim}
definitions
  a2:array(1..10) of set of integer
end-declarations
\end{verbatim}

2.7 Expressions

Expressions are, together with the keywords, the major building blocks of a language. This section summarizes the different basic operators and connectors used to build expressions.

2.7.1 Introduction

Expressions are constructed using constants, operators and identifiers (of objects or functions). If an identifier appears in an expression its value is the value referenced by this identifier. In the case of a set, a list, an array or a record, it is the whole structure. To access a single cell of an array, it is required to 'dereference' this array. The dereferencing of an array is denoted as follows:

\begin{verbatim}
array\_ident(Exp1 [, Exp2 ...])
\end{verbatim}

where \textit{array\_ident} is the name of the array and \textit{Exp\textsubscript{i}} an expression of the type of the \textsubscript{i\textsuperscript{th}} indexing set of the array. The type of such an expression is the type of the array and its value the value
stored in the array with the label 'Exp1 [, Exp2 ...]'. In order to access the cell of an array of arrays, the list of indices for the second array has to be appended to the list of indices of the first array. For instance, the array a:array(1..10) of array(1..10) of integer can be dereferenced with a(1,2).

Similarly, to access the field of a record, it is required to ‘dereference’ this record. The dereferencing of a record is denoted as follows:

\[
\text{record_ident.field_ident}
\]

where \text{record_ident} is the name of the record and \text{field_ident} the name of the required field.

Dereferencing arrays of records is achieved by combining the syntax for the two structures. For instance \(a(1).b\)

A function call is denoted as follows:

\[
\text{function_ident}
\]
\[
\text{function_ident}(\text{Exp1 [, Exp2 ...]})
\]

where \text{function_ident} is the name of the function and \text{Expi} the \(i^{th}\) parameter required by this function. The first form is for a function requiring no parameter.

The special function \(\text{if}\) allows one to make a selection among expressions. Its syntax is the following:

\[
\text{if (Bool_expr, Exp1, Exp2)}
\]

which evaluates to \(\text{Exp1}\) if \(\text{Bool_expr}\) is \text{true} or \(\text{Exp2}\) otherwise. The type of this expression is the type of \(\text{Exp1}\) and \(\text{Exp2}\) which must be of the same type.

The Mosel compiler operates automatic conversions to the type required by a given operator in the following cases:

- in the dereference list of an array:
  - \text{integer} \rightarrow \text{real};

- in a function or procedure parameter list:
  - \text{integer} \rightarrow \text{real, lincr};
  - \text{real} \rightarrow \text{lincr};
  - \text{mpvar} \rightarrow \text{lincr};

- anywhere else:
  - \text{integer} \rightarrow \text{real, string, lincr};
  - \text{real} \rightarrow \text{string, lincr};
  - \text{mpvar} \rightarrow \text{lincr};
  - \text{boolean} \rightarrow \text{string}.

It is possible to force a basic type conversion using the type name as a function (i.e. \text{integer, real, string, boolean}). In the case of \text{string}, the result is the textual representation of the converted expression. In the case of \text{boolean}, for numerical values, the result is \text{true} if the value is nonzero and for strings the result is \text{true} if the string is the word ‘true’. Note that explicit conversions are not defined for MP types, and structured types (\text{e.g. lincr(x) is a syntax error}).

\[
! \text{Assuming } A=3.5, B=2 \\
\text{integer}(A+B) \quad ! = 5
\]
The Mosel Language

```plaintext
string(A-B) ! = "1.5"
real(integer(A+B)) ! = 5.5 (because the compiler simplifies the expression)
```

Parentheses may be used to modify the predefined evaluation order of the operators or simply to group subexpressions.

### 2.7.2 Aggregate operators

An operator is said to be aggregate when it is associated to a list of indices for each of which a set or list of values is defined. This operator is then applied to its operands for each possible tuple of values (e.g. the summation operator `sum` is an aggregate operator). The general form of an aggregate operator is:

```
Aggregate_ident (Iterator1 [, Iterator2 ...]) Expression
```

or

```
count (Iterator1 [, Iterator2 ...])
```

where the `Aggregate_ident` is the name of the operator and `Expression` an expression compatible with this operator (see below for the different available operators). The type of the result of such an aggregate expression is the type of `Expression`. The `count` operator does not require an additional expression: its value, an integer, corresponds to the number of times the expression of another aggregate operator used with the same iterator list would be evaluated (i.e. it is equivalent to `sum(iteratorlist) 1`).

An iterator is one of the following constructs:

```
SetList_expr
```

or

```
ident1 [, ident2 ...] in SetList_expr [ | Bool_expr]
```

or

```
ident = Expression [ | Bool_expr]
```

or

```
ident as counter
```

The first form gives the list of the values to be taken without specifying an index name. With the second form, the indices named `identi` take successively all values of the set or list defined by `SetList_expr`. With the third form, the index `ident` is assigned a single value (which must be a scalar). For the second and third cases, the scope of the created identifier is limited to the scope of the operator (i.e. it exists only for the following iterators and for the operand of the aggregate operator). Moreover, an optional condition can be stated by means of `Bool_expr` which can be used as a filter to select the relevant elements of the domain of the index. It is important to note that this condition is evaluated as early as possible. As a consequence, a Boolean expression that does not depend on any of the defined indices in the considered iterator list is evaluated only once, namely before the aggregate operator itself and not for each possible tuple of indices. The last form of an iterator declares a `counter` for the operator: the value of the corresponding symbol is incremented each time the operator’s expression is evaluated. For this case, if `ident` has been declared before, it must be integer or real and its value is not reset. Otherwise, as for indices, the scope of the created integer identifier is limited to the scope of the operator and its initial value is 0. There can be only one counter for a given aggregate operator.

The Mosel compiler performs loop optimization when function `exists` is used as the first factors of the condition in order to enumerate only those tuples of indices that correspond to actual cells in the array instead of all possible tuples. To be effective, this optimization requires that sets used to declare the array on which the exist condition applies must be named and the same sets must be used to define the index domains. Moreover, the maximum speedup is obtained when order
of indices is respected and all indices are defined in the same aggregate operator.

An index is considered to be a constant: it is not possible to change explicitly the value of a
named index (using an assignment for instance).

2.7.3 Arithmetic expressions

Numerical constants can be written using the common scientific notation. Arithmetic expressions
are naturally expressed by means of the usual operators (+, −, *, / division, unary −, unary +,’ raise
to the power). For integer values, the operators mod (remainder of division) and div (integral
division) are also defined. Note that mpvar objects are handled like real values in expression.

The sum (summation) aggregate operators is defined on integers, real and mpvar. The aggregate
operators prod (product), min (minimum) and max (maximum) can be used on integer and real
values.

\[
x \times 5.5 + (2+z)^4 + \cos(12.4) \\
\sum (i \text{ in } 1..10) \ (\min (j \text{ in } s) \ t(i) \times (a(j) \mod 2))
\]

2.7.4 String expressions

Constant strings of characters must be quoted with single (‘) or double quote ("), and may extend
over several lines. Strings enclosed in double quotes may contain C-like escape sequences
introduced by the ‘backslash’ character (\a \b \f \n \r \t \v \xxx \uhhhh with xxx being the
character code as an octal number and hhhh a Unicode code as a four hexadecimal digits number).

Each sequence is replaced by the corresponding control character (e.g. \n is the ‘new line’
command) or, if no control character exists, by the second character of the sequence itself (e.g. \ is
replaced by ‘\’).

The escape sequences are not interpreted if they are contained in strings that are enclosed in
single quotes.

Example:

'c:\ddd1\ddd2\ddd3' is understood as c:\ddd1\ddd2\ddd3
"c:\ddd1\ddd2\ddd3" is understood as c:ddd1ddd2ddd3

There are two basic operators for strings: the concatenation, written ‘+’ and the difference,
written ‘-’.

"a1b2c3d5"+"e6" != "a1b2c3d5e6"
'a1b2c3d5'-'3d5' != "a1b2c"

A constant string may also take 2 additional forms: initialised from the content of an external file
or as a portion of the current input file. For the first case, a text string enclosed in backquotes will
be replaced by the content of the file identified by this enclosed text. For the second case, a line
ending by the backquote character optionally followed by some label (consisting in any sequence
of characters not including backquote) will be interpreted as the beginning of a text block. The
end of this text block is marked by a line starting with the previously used label (if any) followed
by the backquote character.

Example:

'afile.txt' ! This string is the content of "afile.txt"
'MyMarker
line1
line2
MyMarker' ! This string is equivalent to "line1\nline2\n"
2.7.5 Set expressions

Constant sets are described using one of the following constructs:

\( \{ \text{Exp}_1, \text{Exp}_2, \ldots \} \)

or

\( \text{Integer}_{\text{exp}1} \ldots \text{Integer}_{\text{exp}2} \)

The first form enumerates all the values contained in the set and the second form, restricted to sets of integers, gives an interval of integer values. This form implicitly defines a range set.

The basic operators on sets are the union written +, the difference written − and the intersection written *.

The aggregate operators union and inter can also be used to build up set expressions.

\[
\{1,2,3\} + \{4,5,6\} - (6..8) * \{6,10\} \quad ! = \{1,2,3,4,5\}
\]

\[
\{a',b',c'\} * \{b',c',d'\} \quad ! = \{b',c'\}
\]

\[
\text{union}(i \text{ in } 1..4 | i < 2) \{i*3\} \quad ! = \{3,9,12\}
\]

If several range sets are combined in the same expression, the result is either a range or a set of integers depending on the continuity of the produced domain. If range sets and sets of integers of more than one element are combined in an expression, the result is a set of integers. It is however possible to convert a set of integers to a range by using the notation \text{range(setexpr)} where setexpr is a set expression which result is either a set of integers or a range.

2.7.6 List expressions

A constant list consist in a list of expressions enclosed in square brackets:

\( \{ \text{Exp}_1, \text{Exp}_2, \ldots \} \)

There are two basic operators for lists: the concatenation, written ‘+’ and the difference, written ‘−’. The aggregate operator sum can also be used to build up list expressions.

\[
[1,2,3] + [1,2,3] \quad ! = [1,2,3,1,2,3]
\]

\[
[1,2,3,4] - [3,4] \quad ! = [1,2]
\]

\[
\text{sum}(i \text{ in } 1..3) \{i*3\} \quad ! = [3,6,9]
\]

2.7.7 Boolean expressions

A Boolean expression is an expression whose result is either true or false. The traditional comparators are defined on integer and real values: <, <=, =, <> (not equal), >, >.

These operators are also defined for string expressions. In that case, the order is defined by the ISO-8859-1 character set (i.e. roughly: punctuation <digits <capitals <lower case letters <accented letters).

With sets, the comparators <= (‘is subset of’), >= (‘is superset of’), = (‘equality of contents’) and <> (‘difference of contents’) are defined. These comparators must be used with two sets of the same type. Moreover, the operator ‘expr in Set_expr’ is true if the expression expr is contained in the set Set_expr. The opposite, the operator not in is also defined.

With lists, the comparators = (‘equality of contents’) and <> (‘difference of contents’) are defined. These comparators must be used with two lists of the same type.

With arrays, the comparators = (‘equality of contents’) and <> (‘difference of contents’) are defined. These comparators must be used with two arrays of the same type and this type must
support the requested operator (for instance arrays of mpvar cannot be compared).

With records, the comparators « ('equality of contents') and <> ('difference of contents') are defined. These comparators must be used with two records of the same type and all fields of this record type must support the requested operator (for instance records including mpvar entries cannot be compared).

To combine Boolean expressions, the operators and (logical and) and or (logical or) as well as the unary operator not (logical negation) can be used. The evaluation of an arithmetic expression stops as soon as its value is known.

The aggregate operators and and or are the natural extension of their binary counterparts.

3<=x and y>=45 or t<>r and not r in {1..10}
and(i in 1..10) 3<=x(i)

2.7.8 Linear constraint expressions

Linear constraints are built up using linear expressions on the decision variables (type mpvar).

The different forms of constraints are:

\[
\begin{align*}
\text{Linear}_\text{expr} & \\
\text{or} & \\
\text{Linear}_\text{expr1} & \text{Ctr}_\text{cmp} \text{ Linear}_\text{expr2} \\
\text{or} & \\
\text{Linear}_\text{expr} & \text{SOS}_\text{type} \\
\text{or} & \\
\text{mpvar}_\text{ref} & \text{mpvar}_\text{type1} \\
\text{or} & \\
\text{mpvar}_\text{ref} & \text{mpvar}_\text{type2} \text{ Arith}_\text{expr}
\end{align*}
\]

In the case of the first form, the constraint is unconstrained and is just a linear expression. For the second form, the valid comparators are <=, >=, =. The third form is used to declare special ordered sets. The types are then is_sos1 and is_sos2. The coefficients of the variables in the linear expression are used as weights for the SOS (as a consequence, a 0-weighted variable cannot be represented this way, procedure makesos1 or makesos2 has to be used instead).

The last two types are used to set up special types for decision variables. The first series does not require any extra information: is_continuous (default), is_integer, is_binary, is_free. Continuous and integer variables have the default lower bound 0, binary variables only take the values 0 or 1, and 'free' means that the variable is unbounded (i.e., ranging from -\(\infty\) to +\(\infty\)). The second series of types is associated with a threshold value stated by an arithmetic expressions: is_partint for partial integer, the value indicates the limit up to which the variable must be integer, above which it is continuous. For is_semcont (semi-continuous) and is_semint (semi-continuous integer) the value gives the semi-continuous limit of the variable (that is, the lower bound on the part of its domain that is continuous or consecutive integers respectively).

Note that these constraints on single variables are also considered as common linear constraints.

\begin{verbatim}
3*y+\sum(i in 1..10) x(i)*i >= z-t
x is_free
x <= -2
x is Integer
t >= -7
t is_integer
\sum(i in 1..10) i*x(i) is_sos1
y is_partint 5
y <= 20
\end{verbatim}
 Internally all linear constraints are stored in the same form: a linear expression (including a constant term) and a constraint type (the right hand side is always 0). This means, the constraint expression $3x \geq 5y - 10$ is internally represented by: $3x - 5y + 10$ and the type ‘greater than or equal to’. When a reference to a linear constraint appears in an expression, its value is the linear expression it contains. For example, if the identifier $\text{ctl}$ refers to the linear constraint $3x \geq 5y - 10$, the expression $z - x + \text{ctl}$ is equal to: $z - 2x - 5y + 10$.

Note that the value of a unary constraint of the type $x \ is\_type\ threshold\ is\ x\ -\ threshold$.

2.7.9 Automatic arrays

The `array` keyword can be used as an aggregate operator in order to create an array that will exist only for the duration of the expression.

```
array (Iterator1 [, Iterator2 ...]) Expression
```

here, the iterators define the indices of the array and the expression, the associated values.

This automatic array may be used wherever a reference to an array is expected: for instance to save the solution values of an array of decision variables in an initialization block (see Section 2.8.2).

```
initializations to "mydata.txt"
evaluation of array(i in 1..10) x(i).sol as "mylabel"
end-initializations
```

2.8 Statements

Four types of statements are supported by the Mosel language. The simple statements can be seen as elementary operations. The initialization block is used to load data from a file or save data to a file. Selection statements allow one to choose between different sets of statements depending on conditions. Finally, the loop statements are used to repeat operations.

Each of these constructs is considered as a single statement. A list of statements is a succession of statements. No particular statement separator is required between statements except if a statement terminates by an expression. In that case, the expression must be finished by either a line break or the symbol ‘;’.

2.8.1 Simple statements

2.8.1.1 Assignment

An assignment consists in changing the value associated to an identifier. The general form of an assignment is:

```
ident_ref := Expression
```

or

```
ident_ref += Expression
```

or

```
ident_ref -= Expression
```

where `ident_ref` is a reference to a value (i.e. an identifier or an array/record dereference) and `Expression` is an expression of a compatible type with `ident_ref`. The direct assignment, denoted `:=` replaces the value associated with `ident_ref` by the value of the expression. The additive
**assignment**, denoted `+=`, and the **subtractive assignment**, denoted `-=`, are basically combinations of a direct assignment with an addition or a subtraction. They require an expression of a type that supports these operators (for instance it is not possible to use additive assignment with Boolean objects).

The additive and subtractive assignments have a special meaning with linear constraints in the sense that they preserve the constraint type of the assigned identifier: normally a constraint used in an expression has the value of the linear expression it contains, the constraint type is ignored.

```plaintext
\[\begin{align*}
c &:= 3x+y \geq 5 \\
c &+= y \quad \text{! Implies c is } 3x+2y-5 \geq 0 \\
c &:= 3x+y \geq 5 \\
c &:= c + y \quad \text{! Implies c is } 3x+2y-5 (c \text{ becomes unconstrained})
\end{align*}\]
```

### 2.8.1.2 Assignment of structured types

The direct assignment `:=` can also be used with sets, lists, arrays and records under certain conditions. For sets and lists, reference and value must be of the same type, the system performing no conversion on structures. For instance it is not possible to assign a set of integers to a set of reals although assigning an integer value to a real object is valid.

When assigning records, reference and value must be of the same type and this type must be **assignment compatible**: two records having identical definitions are not considered to be the same type by the compiler. In most cases it will be necessary to employ a user type to declare the objects. A record is assignment compatible if all the fields it includes can be assigned a value. For instance a record including a decision variable (type `mpvar`) cannot be used in an assignment: copying a value of such a type has to be performed one field at a time skipping those fields that cannot be assigned.

Two arrays can be used in an assignment if they have strictly the same definition and are assignment compatible (i.e. their type supports assignment). Note that in a few cases arrays sharing the same definition cannot be assigned because their internal representations differ like in the following example:

```plaintext
declarations
  a: array(R:range) of integer \quad \text{! 'a' is dynamic}
end-declarations
R := 1..10
finalise(R)
declarations
  b: array(R) of integer \quad \text{! 'b' is static}
end-declarations
a := b \quad \text{! fails at run time}
```

### 2.8.1.3 About implicit declarations

Each symbol should be declared before being used. However, an **implicit declaration** is issued when a new symbol is assigned a value the type of which is unambiguous.

```plaintext
! Assuming A,S,SE are unknown symbols
A := 1 \quad \text{! A is automatically defined as an integer reference}
S := \{1,2,3\} \quad \text{! S is automatically defined as a set of integers}
SE := \{} \quad \text{! This produces a parser error as the type of SE is unknown}
```

In the case of arrays, the implicit declaration should be avoided or used with particular care as Mosel tries to deduce the indexing sets from the context and decides automatically whether the
created array must be dynamic. The result is not necessarily what is expected.

\[
A(1):=1 ! \text{ Implies: } A: \text{array}(1..1) \text{ of integer}
\]
\[
A(t):=2.5 ! \text{ Assuming } "t \text{ in } 1..10|f(t) > 0" \text{ implies: } A: \text{dynamic array}(\text{range}) \text{ of real}
\]

The option noimplicit disables implicit declarations (see Section 2.3.3).

### 2.8.1.4 Inline initialization

Using *inline initialization* it is possible to assign several cells of an array in a single statement. The general form of an inline initialization is:

\[
\text{ident_ref} ::= \{ \text{Exp1 \[ \text{Exp2 \[ ... \]} \}}
\]

or

\[
\text{ident_ref} ::= (\text{Ind1 \[ \text{Ind2 \[ ... \]} \}) \{ \text{Exp1 \[ \text{Exp2 \[ ... \]} \}}
\]

where *ident_ref* is the object to initialize (array, set or list) and *Exp* are expressions of a compatible type with *ident_ref*. The first form of this statement may be used with lists, sets and arrays indexed by ranges: the list of expressions is used to initialize the object. In the case of lists and sets this operation is similar to a direct assignment, with an array, the first index of each dimension is the lower bound of the indexing range or 1 if the range is empty.

The second form is used to initialize regions of arrays or arrays indexed by general sets: each *Ind* expression indicates the index or list of indices for the corresponding dimension. An index list can be a constant, a list of constants (e.g. ["a", "b", "c"]) or a constant range (e.g. 1..10) but all values must be known at compile time.

\[
\text{declarations}
\]

\[
T: \text{array}(1..10) \text{ of integer}
\]

\[
U: \text{array}(1..9,\{"a","b","c"\}) \text{ of integer}
\]

\text{end-declarations}

\[
T::[2,4,6,8] ! \iff T(1):=2; T(2):=4;...;
\]

\[
T::(2..5)[7,8,9,19] ! \iff T(2):=7; T(3):=8;...;
\]

\[
U::([1,3,6],"b")[1,2,3] ! \iff U(1,"b"):=1; U(3,"b"):=2;...
\]

### 2.8.1.5 Linear constraint expression

A linear constraint expression can be assigned to an identifier but can also be stated on its own. In that case, the constraint is said to be anonymous and is added to the set of already defined constraints. The difference from a *named constraint* is that it is not possible to refer to an anonymous constraint again, for instance to modify it.

\[
10<x; x<20
\]

\[
x \text{ is integer}
\]

### 2.8.1.6 Procedure call

Not all required actions are coded in a given source file. The language comes with a set of predefined procedures that perform specific actions (like displaying a message). It is also possible to import procedures from external locations by using modules or packages (cf. Section 2.3).

The general form of a procedure call is:

\[
\text{procedure_ident}
\]

\[
\text{procedure_ident (Exp1 [, Exp2 ...])}
\]
where \texttt{procedure\_ident} is the name of the procedure and, if required, \texttt{Expi} is the \texttt{i}\textsuperscript{th} parameter for the call. Refer to Chapter 3 of this manual for a comprehensive listing of the predefined procedures. The modules documentation should also be consulted for explanations about the procedures provided by each module.

\begin{verbatim}
writeln("hello!") ! Displays the message: hello!
\end{verbatim}

\subsection{Initialization block}

The initialization block may be used to initialize objects (scalars, arrays, lists or sets) of basic type from files or to save the values of such objects to files. Scalars and arrays of external/user types supporting this feature may also be initialized using this facility.

The first form of an initialization block is used to \textit{initialize} data from a file:

\begin{verbatim}
initializations from Filename
    item1 [ as Label1] or
    [itemT11,itemT12 [ ,IdentT13 ...]] asLabelT1
    [    item2 [ as Label2] or
    [itemT21,itemT22 [ ,IdentT23 ...]] asLabelT2
...
end-initializations
\end{verbatim}

where \texttt{Filename}, a string expression, is the name of the file to read, \texttt{itemi} any object identifier and \texttt{itemTij} an array identifier. Each identifier is automatically associated to a label: by default this label is the identifier itself but a different name may be specified explicitly using a string expression \texttt{Labeli}. If a given item is of a record type, the operation is permitted only if all fields it contains can be initialized. For instance, if one of the fields is a decision variable (type \texttt{mpvar}), the compilation will fail. Alternatively, the fields to be initialized can be listed using the following syntax as an item:

\begin{verbatim}
Identifier(field1 [ ,filedi ...])
\end{verbatim}

When an initialization block is executed, the given file is opened and the requested labels are searched for in this file to initialize the corresponding objects. Several arrays may be initialized with a single record. In this case they must be all indexed by the same sets and the label is obligatory. After the execution of an \texttt{initializations from} block, the control parameter \texttt{nbread} reports the number of items actually read in. Moreover, if control parameter \texttt{readcnt} is set to \texttt{true} before the execution of the block, counting is also achieved at the label level: the number of items actually read in for each label may be obtained using function \texttt{getreadcnt}.

An initialization file must contain one or several records of the following form:

\begin{verbatim}
Label: value
\end{verbatim}

where \texttt{Label} is a text string and \texttt{value} either a constant of a basic type (\texttt{integer, real, string} or \texttt{boolean}) or a collection of \texttt{values} separated by spaces and enclosed in square brackets. Collections of \texttt{values} are used to initialize lists, sets records or arrays — if such a record is requested for a scalar, then the first value of the collection is selected. When used for arrays, indices enclosed in round brackets may be inserted in the list of values to specify a location in the corresponding array.

Note also that:
no particular formatting is required: spaces, tabulations, and line breaks are just normal separators

- the special value ‘*’ implies a no-operation (i.e. the corresponding entity is not initialized)
- single line comments are supported (i.e. starting with ‘!’ and terminated by the end of the line)
- Boolean constants are either the identifiers false (FALSE) and true (TRUE) or the numerical constants 0 and 1
- all text strings (including the labels) may be quoted using either single or double quotes. In the latter case, escape sequences are interpreted (i.e. use of ‘\’).

By default Mosel expects that initialization files are encoded in UTF-8 and it can handle UTF-16 and UTF-32 when a BOM (Byte Order Mark) is used. To process files in another encoding, a special encoding comment line must be put at the beginning of the file (see section 2.5.1). For instance a data file encoded with CP1252 should start with the following comment line:

```
!@encoding:CP1252
```

The second form of an initialization block is used to save data to a file:

```
initializations to Filename

   item1 [as Label1]
   or
   [itemT11 ,itemT12 [ ,IdentT13 ...]] as LabelT1

   / [ item2 [ as Label2]
   or [itemT21 ,itemT22 [ ,IdentT23 ...]] as LabelT2

   ...

end-initializations
```

In this form, any `itemi` can be replaced by the value of an expression using the following construct (`Labeli` is mandatory in this case):

```
evaluation of expression
```

When this second form is executed, the value of all provided labels is updated with the current value of the corresponding identifier in the given file. If a label cannot be found, a new record is appended to the end of the file and the file is created if it does not yet exist.

For example, assuming the file `a.dat` contains:

```
! Example of the use of initialization blocks

t: [ (1 un) [10 11] (2 deux) [* 22] (3 trois) [30 33]]
t2: [ [10 (4) 30 40 ]

'nb used': 0
```

consider the following program:

```
model "Example initblk"
declarations
   nb_used: integer
   s: set of string
   ta, tb: dynamic array(1..3,s) of real
```

A copy of the original file is saved prior to the update (i.e. the original version of `fname` can be found in `fname~`).
t2: array(1..5) of integer
end-declarations

initializations from 'a.dat'
[ta,tb] as 't'
  ! ta=[(1,'un',10),(3,'trois',30)]
  ! tb=[(1,'un',11),(2,'deux',22),(3,'trois',33)]
  t2 ! t2=[10,0,0,30,40]
  nb_used as "nb used" ! nb_used=0
end-initializations

nb_used+=1
  ta(2,"quatre"):=1000

initializations to 'a.dat'
[ta,tb] as 't'
  nb_used as "nb used"
  s
end-initializations
end-model

After the execution of this model, the data file contains:

! Example of the use of initialization blocks
  t:=[(1 'un') [10 11] (2 'deux') [* 22] (2 'quatre') [1000 *]
  (3 'trois') [30 33]]
  t2:[ 10 (4) 30 40 ]
  'nb used': 1
  's': [un 'deux' 'trois' 'quatre']

In case of error (e.g. file not found, corrupted data format) during the processing of an initialization block, the execution of the model is interrupted. However if the value of control parameter ioctrl is true, executions continues. It is up to the user to verify whether data has been properly transferred by checking the value of control parameter iostatus.

2.8.2.1 About automatic finalization

During the execution of an initializations from block all sets are automatically finalized just after having been initialized. This also applies to sets indirectly initialized through the non-dynamic arrays for which they are index sets. In addition, such an array is created as a static array if it has not been used before the initialization block.

This behaviour is controled by the autofinal control parameter which value may be changed using the setparam procedure (i.e. it is therefore possible to have automatic finalization active for only some initializations blocks). The compiler option noautofinal (see section 2.3.3) allows to disable this feature from the beginning of the model (although it can be re-enabled as required using the control parameter).

2.8.3 Selections

2.8.3.1 If statement

The general form of the if statement is:
The selection is executed as follows: if \( \text{Bool} \_\text{exp} \_1 \) is true then \( \text{Statement} \_\text{list} \_1 \) is executed and the process continues after the end-if instruction. Otherwise, if there are elif statements, they are executed in the same manner as the if instruction itself. If, all boolean expressions evaluated are false and there is an else instruction, then \( \text{Statement} \_\text{list} \_\text{E} \) are executed; otherwise no statement is executed and the process continues after the end-if keyword.

```plaintext
if c=1 then writeln('c=1')
  elif c=2 then writeln('c=2')
  else writeln('c<>1 and c<>2')
end-if
```

2.8.3.2 Case statement

The general form of the case statement is:

```plaintext
case Expression\_0 of
  Expression\_1: Statement\_1
  or
  Expression\_1: do Statement\_list\_1 end-do
  [  
    Expression\_2: Statement\_2
    or
    Expression\_2: do Statement\_list\_2 end-do
  ]
  [...]
  [ else Statement\_list\_E ]
end-case
```

The selection is executed as follows: \( \text{Expression} \_0 \) is evaluated and compared sequentially with each expression of the list \( \text{Expression} \_i \) until a match is found. Then the statement \( \text{Statement} \_i \) (resp. list of statements \( \text{Statement} \_\text{list} \_i \)) corresponding to the matching expression is executed and the execution continues after the end-case instruction. If no matching is found and an else statement is present, the list of statements \( \text{Statement} \_\text{list} \_\text{E} \) is executed, otherwise the execution continues after the end-case instruction. Note that, each of the expression lists \( \text{Expression} \_i \) can be either a scalar, a set or a list of expressions separated by commas. In the last two cases, the matching succeeds if the expression \( \text{Expression} \_0 \) corresponds to an element of the set or an entry of the list.

```plaintext
case c of
  1 : writeln('c=1')
  2..5 : writeln('c in 2..5')
  6,8,10: writeln('c in {6,8,10}')
  else writeln('c in {7,9} or c>10 or c<1')
end-case
```
2.8.4 Loops

2.8.4.1 Forall loop

The general form of the forall statement is:

\[
\text{forall (Iterator_list) Statement} \\
\text{or} \\
\text{forall (Iterator_list) do Statement_list end-do}
\]

The statement Statement (resp. list of statements Statement_list) is repeated for each possible index tuple generated by the iterator list (cf. Section 2.7.2).

\[
\text{forall (i in 1..10, j in 1..10 | i<>j) do} \\
\text{write(' (' , i, ',' , j, ')')} \\
\text{if isodd(i*j) then s+=i*j} \\
\text{end-if} \\
\text{end-do}
\]

2.8.4.2 While loop

The general form of the while statement is:

\[
\text{while (Bool_expr) Statement} \\
\text{or} \\
\text{while (Bool_expr) do Statement_list end-do}
\]

The statement Statement (resp. list of statements Statement_list) is repeated as long as the condition Bool_expr is true. If the condition is false at the first evaluation, the while statement is entirely skipped.

\[
i:=1 \\
\text{while(i<=10) do} \\
\text{write(' ',i)} \\
\text{if isodd(i) then s+=i} \\
i+=1 \\
\text{end-do}
\]

2.8.4.3 Repeat loop

The general form of the repeat statement is:

\[
\text{repeat} \\
\text{Statement1} \\
[ \text{Statement2 ...} ] \\
\text{until Bool_expr}
\]

The list of statements enclosed in the instructions repeat and until is repeated until the condition Bool_expr is true. As opposed to the while loop, the statement(s) is (are) executed at least once.

\[
i:=1 \\
\text{repeat} \\
\text{write(' ',i)} \\
\text{if isodd(i) then s+=i} \\
\text{end-if}
\]
2.8.4.4 break and next statements

The statements break and next are respectively used to interrupt and jump to the next iteration of a loop. The general form of the break and next statements is:

```
break [n|label]

or

text [n|label]
```

where \( n \) is an optional integer constant: \( n-1 \) nested loops are stopped before applying the operation. This optional argument may also be a label (in the form an identifier or a string constant): in this case the loop to consider is identified by a label that must be defined just before the corresponding loop using the following syntax:

```
label:
```

The label can be either an identifier (that is not associated to any entity) or a constant string. The scope of each label is limited to the loop it identifies.

```
! in this example only the loop controls are shown
L1: ! 1: Define label "L1"
repeat ! 2: Loop L1
forall (i in S) do ! 3: Loop L2
while (C3) do ! 4: Loop L3
  break 3 ! 5: Stop the 3 loops and continue after line 12
  next ! 6: Go to next iteration of L3 (line 4)
  next 2 ! 7: Stop L3 and go to next 'i' (line 3)
end-do ! 8: End of L3
next "L1" ! 9: Stop L2, go to next iteration of L1 (line 12)
break !10: Stop L2 and continue after line 11
end-do !11: End of L2
until C1 !12: End of L1
```

2.8.4.5 with statement

The general syntax of this statement is:

```
with ident_1=exp_1 [, ident_2=exp_2...] do
  Statement
  [ Statement ...]
end-do
```

Although the with statement is not a loop it is handled like a single iteration forall loop such that it is possible to use the break statement within the block of instructions. The identifiers \( ident_i \) are defined as local symbols to the block.

```
! in this example LR is an array of records
with r=LR(10) do
  r.x:=10 ! update LR(10).x
  r.y:=20 ! update LR(10).y
end-do
```
2.9 Procedures and functions

It is possible to group sets of statements and declarations in the form of subroutines that, once defined, can be called several times during the execution of the model. There are two kinds of subroutines in Mosel, procedures and functions. Procedures are used in the place of statements (e.g. writeln("Hi!")) and functions as part of expressions (because a value is returned, e.g. round(12.3)). Procedures and functions may both receive arguments, define local data and call themselves recursively.

2.9.1 Definition

Defining a subroutine consists of describing its external properties (i.e. its name and arguments) and the actions to be performed when it is executed (i.e. the statements to perform). The general form of a procedure definition is:

```mosel
procedure name_proc ([list_of_parms])
    Proc_body
end-procedure
```

where `name_proc` is the name of the procedure and `list_ofParms` its list of formal parameters (if any). This list is composed of symbol declarations (cf. Section 2.6) separated by commas. The only difference from usual declarations is that no constants or expressions are allowed, including in the indexing list of an array (for instance A=12 or t1:array(1..4) of real are not valid parameter declarations). The body of the procedure is the usual list of statements and declaration blocks except that no procedure or function definition can be included.

```mosel
procedure myproc
    writeln("In myproc")
end-procedure

procedure withparams(a:array(r:range) of real, i,j:integer)
    writeln("I received: i=",i," j=",j)
    forall(n in r) writeln("a(",n,")=",a(n))
end-procedure

declarations
    mytab:array(1..10) of real
end-declarations

myproc        ! Call myproc
withparams(mytab,23,67) ! Call withparams
```

The definition of a function is very similar to the one of a procedure:

```mosel
function name_func ([List_of_params]): Type
    Func_body
end-function
```

The only difference with a procedure is that the function type must be specified: it can be any type name except mpvar. Inside the body of a function, a special variable of the type of the function is automatically defined: returned. This variable is used as the return value of the function, it must therefore be assigned a value during the execution of the function.

```mosel
function multiply_by_3(i:integer):integer
    returned:=i*3
end-function
```
writeln("3*12=", multiply_by_3(12)) ! Call the function

Normally all statements of a subroutine are executed in sequence. It is however possible to interrupt the execution and return to the caller by using the special statement return.

2.9.2 Formal parameters: passing convention

Formal Parameters of basic types are passed by value and all other types are passed by reference. In practice, when a parameter is passed by value, the subroutine receives a copy of the information so, if the subroutine modifies this parameter, the effective parameter remains unchanged. But if a parameter is passed by reference, the subroutine receives the parameter itself. As a consequence, if the parameter is modified during the process of the subroutine, the effective parameter is also affected.

```mosel
procedure alter(s:set of integer,i:integer) i+=1 s+={i} end-procedure
gs:={1} gi:=5 alter(gs,gi) writeln(gs," ",gi) ! Displays: {1,6} 5
```

2.9.3 Local declarations

Several declaration blocks may be used in a subroutine and all identifiers declared are local to this subroutine. This means that all of these symbols exist only in the scope of the subroutine (i.e. between the declaration and the end-procedure or end-function statement) and all of the resource they use is released once the subroutine terminates its execution unless they are referenced outside of the routine (e.g. member of a set defined globally). As a consequence, active constraints (lincr that are not just linear expressions) declared inside a subroutine and the variables they employ are still effective after the termination of the subroutine (because they are part of the current problem) even if the symbols used to name the related objects are not defined any more. Note also that a local declaration may hide a global symbol.

```mosel
declarations ! Global definition
i,j:integer end-declarations

procedure myproc
declarations
i:string ! This declaration hides the global symbol
end-declarations
i="a string" ! Local 'i'
j:=4 writeln("Inside of myproc, i="i," j=",j)
end-procedure

i:=45 ! Global 'i'
j:=10 myproc writeln("Outside of myproc, i="i," j=",j)
```

This code extract displays:

```
Inside of myproc, i=a string j=4
Outside of myproc, i=45 j=4
```
2.9.4 Overloading

Mosel supports overloading of procedures and functions. One can define the same function several times with different sets of parameters and the compiler decides which subroutine to use depending on the parameter list. This also applies to predefined procedures and functions.

! Returns a random number between 1 and a given upper limit
function random(limit:integer):integer
    returned:=round(.5+random*limit) ! Use the predefined 'random' function
end-function

It is important to note that:

■ a procedure cannot overload a function and vice versa;
■ it is not possible to redefine any identifier; this rule also applies to procedures and functions. A subroutine definition can be used to overload another subroutine only if it differs for at least one parameter. This means, a difference in the type of the return value of a function is not sufficient.

2.9.5 Forward declaration

During the compilation phase of a source file, only symbols that have been previously declared can be used at any given point. If two procedures call themselves recursively (cross recursion), it is therefore necessary to be able to declare one of the two procedures in advance. Moreover, for the sake of clarity it is sometimes useful to group all procedure and function definitions at the end of the source file. A forward declaration is provided for these uses: it consists of stating only the header of a subroutine that will be defined later. The general form of a forward declaration is:

forward procedure Proc_name [(List_of_params)]
or
forward function Func_name [(List_of_params)]: Basic_type

where the procedure or function Func_name will be defined later in the source file. Note that a forward definition for which no actual definition can be found is considered as an error by Mosel.

forward function f2(x:integer):integer
function f1(x:integer):integer
    returned:=x+if(x>0,f2(x-1),0) ! f1 needs to know f2
end-function
function f2(x:integer):integer
    returned:=x+if(x>0,f1(x-1),0) ! f2 needs to know f1
end-function

2.9.6 Suffix notation

Functions which name begins with get and taking a single argument may be called using a suffix notation. This alternative syntax is constructed by appending to the variable name (the intended function parameter) a dot followed by the function name without its prefix get. For instance the call getsol(x) is the same as x.sol. The compiler performing internally the translation from the suffix notation to the usual function call notation, the two syntaxes are equivalent.

Similarly, calls to procedures which name begins with set and taking two arguments may be written as an assignment combined with a suffix notation. In this case the statement can be
replaced by the variable name (the intended first procedure parameter) followed by a dot and the procedure name without its prefix \texttt{set} then the assignment sign \texttt{:=} and the value corresponding to the second parameter. For instance the statement \texttt{sethidden(ctl,true)} can also be written \texttt{ctl.hidden:=true}. As for the other alternative notation, the compiler performs the rewriting internally and the two syntaxes are equivalent.

2.10 Problems

In Mosel terms, a \textit{problem} is a container holding various attributes and entities. The nature of the information stored is characterised by a \textit{problem type}. The core system of Mosel provides the \texttt{mpproblem} problem type for the representation of mathematical programming problems with linear constraints. Other types may be published by modules either as entirely new problem types \textit{or as problem type extensions}. An extension adds extra functionality or properties to an existing type; for instance, \texttt{mpproblem.xprs} provided by the module \texttt{mmxprs} adds support for \textit{solving} \texttt{mpproblem} problems while the type \texttt{mpproblem.nl} of \texttt{mmnl} makes it possible to include non-linear constraints in an \texttt{mpproblem}.

When the execution of the model starts, an instance of each of the available problem types is created: this \textit{main problem} constitutes the default \textit{problem context}. As a consequence, all problem related operations (\textit{e.g.}, add constraints, solve...) refer to this context. Further problem instances may be declared just like any other symbol using a declarations section. The specification of a problem type (that is used as an elementary type in a declaration) has two forms:

\begin{verbatim}
problem_type
or
problem_type1 and problem_type2 [and problem_typen ...]
\end{verbatim}

where \textit{problem type*} are problem type names. The second syntax allows to define a problem instance that refers to several problem types: this can be useful if a particular problem consists in the combination of several problem types. Note also that the main problem can be seen as an instance of the combination of all available problem types.

The \texttt{with} construct can be used to switch to a different problem context for the duration of a block of instructions. The general form of this construct is:

\begin{verbatim}
with prob do
  Statement
  [ Statement ...]
end-do
\end{verbatim}

where \textit{prob} is a problem reference or a problem type specification. In the first case the referenced problem is selected, in the second case, a new problem instance is created for the duration of the block (\textit{i.e.}, it is released after the block has been processed). Both statements and declaration blocks as well as other \texttt{with} constructs may be included in this section: they are all executed in the context of the selected problem.

\begin{verbatim}
declarations
p1,p2:mpproblem
p3:mpproblem and mypb ! assuming 'mypb' is a problem type
PT=mpproblem and mypb ! user defined problem type
a:array(1..10) of PT
x,y:mpvar
end-declarations
with p1 do
  x+y>=0
end-with
\end{verbatim}
Some problem types support assignment (operator :=) and additive assignment (operator +=). These operators can be used between objects of same type but also when the right parameter of the operator is a component of the assigned object. For instance, assuming the declarations of the previous example we could state p3 := p2 meaning that the mpproblem part of p3 must be replaced by a copy of p2, the mypb part of p3 remaining unchanged. From the same context, the assignment p2 := p3 produces a compilation error.

2.10.1 The mpproblem type

An mpproblem instance basically consists in a set of linear constraints (the decision variables defined anywhere in a model are shared by all problems). A constraint is incorporated into a problem when it is expressed, so having the declaration of a linctr identifier in the context of a problem is not sufficient to attach it to this problem. The association will occur when the symbol is assigned its first value. Afterwards, the constraint will remain part of the same problem even if it is altered from within the context of another problem (a constraint cannot belong to several problems at the same time).

In the example above, the constraint C1 is part of problem p1. From the context of a second problem p2 the constraint C1 is modified using solution information of p2: this change affects only the first problem since the constraint does not belong to the current context. Note that since is_integer is a (unary) constraint, the decision variable x is integer for problem p1 but it is a continuous variable in p2.

When a problem is released or reset (see reset), all its constraints are detached. Constraints which are not referenced (anonymous constraints) are released at the same time, named constraints however are not freed, they become available to be associated to some other problem.

In this example, at the end of the first with block, the local problem is released. As a consequence the constraint C1 is detached from this problem (but remains unchanged) and the 2 other constraints are freed. The following statements add C1 to the problem p1.

The type mpproblem supports both assignment (operator :=) and additive assignment (operator +=).
2.11 The public qualifier

Once a source file has been compiled, the identifiers used to designate the objects of the model become useless for Mosel. In order to access information after a model has been executed (for instance using the `print` command of the command line interpreter), a table of symbols is saved in the BIM file. If the source is compiled with the `strip` option (`-s`), all private symbols are removed from the symbol table — by default all symbols (except parameters) are considered to be private.

The qualifier `public` can be used in declaration and definition of objects to mark those identifiers (including subroutines) that must be published in the table of symbols even when the strip option is in use.

```mosel
public declarations
  e:integer ! e is published
  f:integer ! f is published
end-declarations

declarations
  public a,b,c:integer ! a,b and c are published
  d:real ! d is private
end-declarations

forward public procedure myproc(i:integer) ! 'myproc' is published
```

This qualifier can also be used when declaring record types in order to select the fields of the record that can be accessed from outside of the file making the definitions: this allows to make available only a few fields of a record, hidding what is considered to be internal data.

```mosel
declarations
  public t1=record
    i:integer ! t1.i is private
    public j:real ! t1.j is public
  end-record
  public t2=public record
    i:integer ! t2.i is public
    j:real ! t2.j is public
  end-record
end-declarations
```

2.12 Packages

Declarations may be stored in a package: once compiled, the package can be used by any model by means of the `uses` statement. Except for its beginning and termination (keyword `model` is replaced by `package`) a package source is similar to a normal model source. The following points should be noticed:

- all statements and declarations outside procedure or function definitions are used as an initialization routine: they are automatically executed before statements of the model using the package;
- symbols that should be published by the package must be made explicitly public using the `public` qualifier (see Section 2.11);
- parameters of a package are automatically added to the list of parameters of the model using the package;
- as opposed to modules that are dynamically linked, bim files of packages are used only at compilation time — they are not required for execution;
a package cannot be imported several times by a model and packages publish symbols of packages they use. For instance, assuming package $P1$ imports package $P2$, a model using $P1$ cannot import explicitly $P2$ (with a \texttt{uses} statement) but has access to the functionality of $P2$ via $P1$.

### 2.12.1 The requirements block

Requirements are symbols a package requires for its processing but does not define. These required symbols are declared in requirement blocks which are a special kind of declaration blocks in which constants are not allowed but procedure/functions can be declared. The symbols of such a block have to be defined when the model using the package is compiled: the definitions may appear either in the model or in another package but cannot come from a module. Several packages used by a given model may have the same requirements (i.e. same identifier and same declaration). It is also worth noting that a package inherits the requirements of the packages it uses.

```mosel
requirements
  an_int: integer
  s0: set of string
  bigar: array(s0) of real
  procedure doit(i: integer)
end-requirements
```

### 2.13 Annotations

Annotations are meta data expressed in the Mosel source file that are stored in the resulting bim file after compilation. Thanks to a dedicated API it is possible to retrieve the information both from the model itself during its execution (see \texttt{getannotations}) or before/after execution from a host application (see function \texttt{XPRMgetannotations} in the Mosel Libraries Reference Manual).

#### 2.13.1 Syntax

Annotations are organised in categories. A category groups a set of annotations and other categories (or sub-categories). When expressing a full annotation name, categories are separated by the `.` symbol. For instance:

```mosel
doc.name
```

will be used to select the annotation \texttt{name} that is a member of the \texttt{doc} category. Similarly:

```mosel
mycat1.cat2.info
```

will reference the annotation \texttt{info} recorded in the category \texttt{cat2} that is itself part of category \texttt{mycat1}. Annotations and annotation categories must be valid Mosel identifiers: their names can only use alpha-numeric symbols plus `_'.

Some predefined categories are available at the beginning of the compilation:

- the default category (its name is empty) collects annotations that are not explicitly member of any particular category. For instance the annotation \texttt{myannot} will be recorded in the default category. This annotation may also be referenced by its full name \texttt{.myannot}

- \texttt{mc} (for Mosel Compiler) is used to pass information to the compiler during the compilation. For example, the \texttt{mc.def} annotation makes it possible to declare an annotation type (see section 2.13.3)
The Mosel Language

- doc can be used to document a model or package file (see section 2.19)

In the Mosel source file annotations are included in special comments. A single-line annotation is of the form:

```mosel
!@ name value
```

Here `name` is the name of the annotation (spaces between `@` and the name are ignored) and the following text (up to the end of line) its corresponding value. The separation character between the name and the value can be a space, `;` or `=` (there must be no space between the name and the symbol). There is no restriction on the content of the value: it can be any kind of text (unless the annotation is typed—see section 2.13.3).

A multi-line annotation is of the form:

```mosel
(!@name value
  ...
  @name2 value2
  ...
  !)
```

where `name` is an annotation name while the text following this name is its associated value. With this syntax the value may spread over several lines, its termination is marked either by the end of the comment block or by a new annotation specification. In this context, a new annotation must start with the `@` symbol at the beginning of a new line (leading spaces are ignored). As for a one-line annotation, symbols `;` and `=` can be used instead of a space to separate the name and its value.

If several annotations of the same category have to be defined in the same block, a current category may be defined such that following annotation names can be shortened. This mechanism is activated by specifying the category name terminated by a dot (the remainder of this line is ignored) before the first annotation statement. The category selection is effective for the current comment block only and remains active until the next selection. Using a dot in place of a category name restores the default behaviour (i.e. the full path must be used for annotation reference). For instance:

```mosel
(!@doc.       Switch to 'doc' category (this text is ignored)
 @name:my_function
 @type:integer
 @mycat.cat1. Switch to 'mycat.cat1'
 @memb1 10
 @memb2 20
 @. Unselect current category
 @glb=useless
 !)
```

Is equivalent to:

```mosel
(!@doc.name:my_function
 @doc.type:integer
 @mycat.cat1.memb1 10
 @mycat.cat1.memb2 20
 @glb=useless
 !)
```

By default any new annotation name is added to the internal dictionary and no checking is applied to the provided value. If a given annotation is defined several times only the last assignment is preserved. The compiler will however emit a warning if an attempt is made to assign a value to a category or to use an annotation as a category. For instance:
The second definition will fail to use mycat.memb1 as a category because the first one has already implicitly declared it as an annotation.

### 2.13.2 Symbol association

An annotation is either global or associated with a specific public symbol (see section 2.11). The association depends on the location of the definition in the source code:

- annotations preceding a subroutine declaration (*forward statement*) or definition are associated with the subroutine name
- annotations preceding a declarations block are distributed to all the symbols declared in the block
- inside of a declarations block: annotations preceding or terminating the line of a declaration are associated with the corresponding symbols

In all other cases the annotations are global (*i.e.* not associated with any particular symbol) — in particular trying to associate annotations to private symbols will result in global annotations.

Annotations that precede a subroutine declaration, a declarations block or an entity in a declarations block can be turned into global annotations by inserting the compiler annotation `mc.flush` between the annotation and the following code.

### 2.13.3 Declaration

Declaration of annotations is achieved via the `mc.def` compiler annotation. Once an annotation is declared, the compiler checks the validity of definitions and rejects those that are not compliant, issuing a warning message (invalid annotations will not make the compilation fail unless the flag `strict` is used).

The general syntax of the annotation declaration statement is:

```mc.def aname [prop1[,prop2...]]
```

Where `aname` is an annotation name and `prop?` a property keyword. The possible keywords are:

- `alias name1 name2...` aname Defining an alias to name1, name2...
- `text|integer|real|boolean` Type of the annotation value (*default:* `text`)
- `last|first|merge|multi` Handling of multiple definitions of an annotation (*default:* `last`)
  - `last`: the last definition is kept
  - `first`: keep the first definition (the following ones are ignored)
  - `merge`: definitions are concatenated (separated by new lines)
  - `multi`: all definitions are kept
- `global|specific` By default, the association of annotations depends on the location of the definition. If `global` is stated, the annotation is always global; with option `specific`, the annotation will be kept only if it can be associated with a symbol (otherwise it is ignored instead of being stored as a global one).
values=v1 v2 v3... If used, this option must be the last one of the definition and it cannot be combined with range. It defines a list of possible values for the annotation.

range=lb ub If used, this option must be the last one of the definition, it requires the type to be specified (integer or real) and it cannot be combined with values. It defines a range of possible values.

strict When this option has been stated any error detected on this annotation (or path when applied to a category) will make the compilation fail

Example:

```!
@mc.def person.name text,first,specific
@mc.def person.age integer,first,specific,range=0 150
@mc.def person.gender values=male female
```

Categories are implicitly declared by the annotations they include (for instance declaring `@mycat.myann` implies the creation of `mycat` as a category). It is also possible to explicitly declare an empty category (i.e. containing no annotation) using the `mc.def` construct by appending a dot to the category name (the only supported property is `strict`). For instance:

```!
@mc.def mycat.
```

For a given annotation the declaration may be stated several times but the properties of an annotation cannot be changed. For instance, the following declarations can be used in the same source:

```!
@mc.def myann
@mc.def myann text,last
```

But the following declaration cannot be combined with any of the two preceding ones as they both result in the annotation type `text`:

```!
@mc.def myann integer
```

Declarations included in models are not exported to the bim file (i.e. they are only used during the compilation procedure) but declarations stated in packages are published if they are relative to a user defined category: any model using the package inherits the annotation declarations of the package.

Additional properties can be set using the `mc.set` compiler annotation. The general syntax of this special statement is:

```!
@mc.set name flag
```

Where `name` is an annotation or category name and `flag` one of the following keywords:

- **complete**: Applied to a category this flag indicates that no other annotation can be added to this category (ignored for an annotation). It is however still possible to declare aliases. Note that sub-categories are not concerned by this flag: if required each sub-category has also to be tagged.

- **disable**: Disable the named category or annotation. From the point where this flag has been set onwards, all definitions deriving from the provided name are silently ignored.

- **enable**: Revert the effect of `disable`. 
unpublish  Disable the automatic publication of the specified declaration.
publish   Publish the specified declaration.

Note that mc.set expects a full explicit name: for this command ann refers to category ann and not to annotation .ann as in other places.

2.14 File names and input/output drivers

Mosel handles data streams using IO drivers: a driver is an interface between Mosel and a physical data source. Its role is to expose the data source in a standard way such that from the user perspective, all data sources can be accessed using the same methods (i.e. initializations blocks, file handling functions). Drivers are specified in file names: all Mosel functions supporting IO operations though drivers can be given an extended file name. This type of name is composed of the pair driver_name:file_name. When Mosel needs to access a file, it looks for the specified driver in the table of available drivers. This table contains all predefined drivers as well as drivers published by modules currently loaded in memory. If the driver is provided by a module, the module name may also be indicated in the extended file name: module_name:driver_name:file_name. Using this notation, Mosel loads the required module if necessary (otherwise the file operation fails if the module is not already loaded). For instance it is better to use mmodbc.odbc:database than odbc:database.

The file name part of the extended file name is specific to the driver and its structure and meaning depends on the driver. For instance, the sysfd driver expects a numerical file descriptor so file sysfd:1 is a valid name but sysfd:myfile cannot work. A driver may act as a filter and expects as file name another extended file name (e.g. zlib deflate:mem:myblk).

When no driver name is specified, Mosel uses the default driver which name is an empty string (myfile is equivalent to :myfile). This driver relies on OS functions to access files from the file system.

The tmp driver is an extension to the default driver: it locates the specified file in the temporary directory used by Mosel (i.e. tmp:toto is equivalent to getparam("tmpdir")/toto).

The null driver can be used to disable a stream: whatever written to file "null:" is ignored and reading from it is like reading from an empty file.

The mem driver uses a memory block instead of a file handled by the operating system. A file name for this driver is of the form mem:label[/minsize[/incstep]] where label is a letter and minsize an optional initial amount of memory to be reserved (size is expressed in bytes, in kilobytes with suffix "k" or in megabytes with suffix "m"). The label being recorded in the dictionary of the model symbols it cannot be identical to any of the identifiers of the model (the function newmuid might be used to generate a unique identifier). The memory block is allocated dynamically and resized as necessary. By default the size of the memory block is increased by pages of 4 kilobytes: the optional parameter incstep may be used to change this page size (i.e. the default setting is "label/0/4k"). The special value 0 modifies the allocation policy: instead of being increased of a fixed amount, the block size is doubled. In all cases unused memory is released when the file is closed.

The mem driver may also be used to exchange data with an application using the Mosel libraries (refer to the Mosel Libraries Reference Manual for further explanation).

The tee driver can only be open for writing and expects as file name a list of up to 6 extended file names separated with ‘&’: it opens all the specified files and duplicates what it receives to each of them. If only one file is given or if the string terminates with ‘&’, output is also sent to the default output stream (or error stream if the file is used for errors). For instance, writing to the file "tee:log1&log2k" has the effect of writing at the same time to files "log1" and "log2" as well as sending a copy to the console.
The `bin` driver can only be used for initializations blocks as a replacement of the default driver: it allows to write (and read) data files in a platform independent binary format. This file format is generally smaller than its ASCII equivalent and preserves accuracy of floating point numbers. This driver can be used in 2 different ways: a single file including all records of the initialisations block is produced if a file name is provided. For instance, in the following example the file "mydata" will contain both A and B:

```
initialisations to "bin:mydata"
  A
  B
end-initialisations
```

With the second form (without file name) one file is generated for each record of the block. The following example produces 2 files: "mydata_A" to contain the values of record A and "mydata_B" for values of B:

```
initialisations to "bin:"
  A as "mydata_A"
  B as "mydata_B"
end-initialisations
```

When using this form in an initialisations to block, the option append may be specified such that files are open in append mode.

The other predefined drivers (`sysfd`, `cb` and `raw`) are useful when interfacing Mosel with a host application. They are described in detail in the Mosel Libraries Reference Manual.

### 2.15 Character encoding of text files

Mosel uses UTF-8 for its internal representation of text strings and this is also the default character encoding for text files. It is however possible to read and write text files in different encodings: for model source and initialization block files the selection can be achieved by means of a special comment (see sections 2.5.1 and 2.8.2) but the encoding may also be specified at the time of opening a file by prefixing its name with the "enc: " prefix:

```
enc:encoding [+unix]+dos]+sys]/+[bom]+nobom],filename
```

Mosel supports natively the encodings UTF-8, UTF-16, UTF-32, ISO-8859-1, ISO-8859-15, CP1252 and US-ASCII. For UTF-16 and UTF-32 the byte ordering depends on the architecture of the running system (e.g. this is Little Endian on an x86 processor) but it can also be specified by appending LE (Little Endian) or BE (Big Endian) to the encoding name (e.g. UTF-16LE). The availability and names of other encodings depends on the operating system.

The following aliases may also be used in place of an encoding name: RAW (no encoding), SYS (default system encoding), WCHAR (wide character for the C library), FNAME (encoding used for file names), TTY (encoding of the output stream of the console), TTYIN (encoding of the input stream of the console), STDIN, STDOUT, STDERR (encoding of the default input/output/error stream).

In addition to the encoding name a couple of options might be applied: +unix and +dos select the line termination (note that +dos is automatically used when writing to a physical file on Windows). Options +bom and +nobom decides whether a Byte Order Mark is to be inserted at the beginning of the file (this option only applies to UTF encodings when the file is not open in appending mode). By default a BOM is inserted when the encoding is UTF-16 or UTF-32, the option +nobom disables this insertion. The option +bom implies the insertion of a BOM on UTF-8 encoded files (this is usually not required for this encoding but often used on Windows systems). The option +sys selects the line termination and BOM convention of the running system (i.e. it is
equivalent to +unix on a Posix system and +dos+bom on a Windows machine).

2.16 Working directory and temporary directory

Except for absolute path names, file or path name expansion are relative to the current working directory. By default this reference location corresponds to the operating system current working directory which usually is the directory from which Mosel has been started. Since the working directory is an execution parameter, a model may be running with a current working directory which might be different from the one used by the operating system. It is therefore recommended to use absolute file names when a Mosel model communicates with an external component (for instance when a file name is part of the DSN to be used for an ODBC connection).

In addition to the current working directory, Mosel creates a temporary directory that is shared by all models for storing temporary data handled as physical files. This directory is located in the system temporary directory as specified by one of the environment variables TMP, TEMP or USERPROFILE under Windows and TMPDIR on Posix systems. If none of these environment variables is defined, the default base directory will be “C: \” on Windows and “/tmp” on Posix systems. The Mosel temporary directory is automatically created when needed and deleted at program termination.

The path names of the working directory and the temporary directory are identified respectively by the "workdir" and "tmpdir" control parameters and can be retrieved using the getparam function. It is possible to change the current working directory of a running model by updating the "workdir" parameter using setparam.

2.17 Handling of input/output

At the start of the execution of a program/model, three text streams are created automatically: the standard input, output and error streams. The standard output stream is used by the procedures writing text (write, writeln, fflush). The standard input stream is used by the procedures reading text (read, readln, fskipline). The standard error stream is the destination of error messages reported by Mosel during its execution. These streams are inherited from the environment in which Mosel is being run: usually using an output procedure implies printing something to the console and using an input procedure implies expecting something to be typed by the user.

The procedures fopen and fclose make it possible to associate text files to the input, output and error streams: in this case the IO functions can be used to read from or write to files. Note that when a file is opened, it is automatically made the active input, output or error stream (according to its opening status) but the file that was previously assigned to the corresponding stream remains open. It is however possible to switch between different open files using the procedure fselect in combination with the function getfid.

model "test IO"
def_out:=getfid(F_OUTPUT) ! Save file ID of default output fopen("mylog.txt",F_OUTPUT) ! Switch output to ‘mylog.txt’my_out:=getfid(F_OUTPUT) ! Save ID of current output streamrepeatfselect(def_out) ! Select default ouput...write("Text? ") ! ...to print a message
text:=''readln(text) ! Read a string from the default inputfselect(my_out) ! Select the file ‘mylog.txt’writeln(text) ! Write the string into the file
2.18 Deploying models

Once a model has been compiled to a BIM file it may be deployed in a variety of ways. It may be

- run from some remote code using the remote invocation library XPRD (see the XPRD
  reference manual),
- be integrated in an application through the Mosel libraries (see Mosel libraries reference
  manual),
- form part of an Optimization Modeler application (see the Optimization Modeler Developer
  Guide), or
- simply be invoked from a command window or shell.

For the last option the usual approach consists in using the mosel command line tool (see section
1.3) with the run command. For instance, the following command may be used to run the model
mycmd.bim:

```bash
> mosel run mycmd.bim
```

The aim of the deploy module is to ease the use of a model published this way. This module
makes it possible to generate an executable program from the BIM file. Moreover, it gives the
model access to the command line arguments and exposes a method for embedding
configuration files into the resulting program. The deploy module is usually used through one of
its two IO drivers: the first driver, csrc, generates a C program (based on the Mosel libraries) from
a BIM file and the second one, exe, produces directly the executable by running a C compiler on
the generated C source (this requires the availability of a C compiler on the system). For example
the following command will create the program runmycmd (or runmycmd.exe on Windows) from
the model mycmd.mos:

```bash
> mosel comp mycmd.mos -o deploy.exe:runmycmd
```

In addition to its IO drivers, the deploy module publishes two functions for accessing the program
arguments: argc returns the number of parameters passed to the command (counting the
command itself as the first) and argv(i) returns the i\textsuperscript{th} argument (as a string). As an example, the
following model displays the arguments it receives:

```mosel
model mycmd
uses 'deploy'
writeln("My arguments:")
forall(i in 1..argc) writeln(argv(i))
end-model
```

After compiling this example into an executable with the command shown above, an execution
of the command runmycmd a b c will display:

```
My arguments:
runmycmd
a
b
c
```
The Mosel Language

In addition to giving access to command line arguments, deploy makes it possible to embed files
into the resulting executable. File locations are passed via model parameters. The following
example outputs its source when the program is called with the argument ’src’ — otherwise it
reports an error message:
model mycmd2
uses ’deploy’,’mmsystem’
parameters
SRC="null:"
end-parameters
if argc<>2 or argv(2)<>"src" then
writeln("Usage: ", argv(1), " src")
exit(1)
else
writeln("Source:")
fcopy(SRC,"")
end-if
end-model

In this example, the source file is identified by the model parameter SRC. To generate the
program, the following command has to be issued:
> mosel comp mycmd2.mos -o deploy.exe:runmycmd2,SRC=mycmd2.mos

With the command above, the file mycmd2.mos is included in the executable and the SRC
parameter is redefined such that the model can access the file through memory. Note that the
model file can also be included in the executable in compressed form. To enable this feature, the
parameter name has to be suffixed with -z in the compilation command:
> mosel comp mycmd2.mos -o deploy.exe:runmycmd2,SRC-z=mycmd2.mos

2.19

Documenting models using annotations
The predefined doc annotation category can be used to document a Mosel file. Using a dedicated
set of annotations the model author can add descriptions to the various entities defined in the
source, the user-defined descriptions are completed by definitions automatically generated by
the Mosel compiler.
From a bim file that includes such definitions a documentation processor may produce a complete
document: as an example, the Xpress distribution comes with the moseldoc processor that
generates an HTML documentation from an annotated bim file.

2.19.1 doc annotation category
Unlike other annotation categories, the doc annotation category is disabled by default such that
the corresponding annotations are silently ignored. To generate a documentation-enabled bim
file the compiler has to be run with the option -D. In addition to enabling the doc category, this
flag also activates the automatic generation of certain documentation annotations by the
compiler. Alternatively to using this flag, a model may define the following annotations:
!@mc.set doc enable
!@doc.autogen=true

Note that these special annotations can also be used in the source file as a means to exclude some
definitions from the documentation, setting doc.autogen to false right before the definitions to

Fair Isaac Corporation Confidential and Proprietary Information

54


be excluded and back to true immediately after.

2.19.1.1 Global definitions

The following global annotations are automatically generated by the compiler:

@doc.name Name of the package or model
@doc.version Version number as stated by the 'version' statement
@doc.date Current date
@doc.ispkg Set to ‘true’ if the file is a package

All automatic annotations can also be defined explicitly in the Mosel source to overwrite their default values.

The following annotations may be added to complete the general appearance of the document to be produced (they are used by the mosel/doc documentation processor):

@doc.title Title of the document
@doc.subtitleSubtitle of the document
@doc.xmlheader Header of the XML document
@doc.xmlroot Name of the XML element containing the documentation
@doc.id Prefix used to generate IDs of chapters, sections, and subsections. If the documentation for several packages is generated from a single master model then a unique ID must be explicitly defined in each of the packages in order to avoid ID collision

The @doc category is complete (i.e. it is not possible to create new doc.X annotations), however, the category @doc.ext can be used to define further information assuming a particular documentation processor can exploit it.

2.19.1.2 Document structure

Optionally, the resulting document may be organised in chapters, sections and subsections. Each of these constructs can contain both text paragraphs and entity descriptions (declarations and subroutines). To enter a new documentation component, one of the following annotations has to be defined:

@doc.chapter Start a chapter
@doc.section Start a section inside of a chapter
@doc.subsection Start a subsection inside of a section

In addition to the provided title a short title might also be defined (using @doc.shorttitle) that will be used in place of the (long) title in the table of contents. Whenever a new division starts, a unique ID is automatically generated based on the section number and any defined prefix specified in the header of the document with @doc.id. It is also possible to explicitly define an ID using @doc.id just after entering the section (this is required when the section has to be referenced using a <ref> tag).
From inside of any of these divisions a new paragraph is added with the `@doc.p` annotation. By default any new addition (paragraph or entity description) is appended to the current component but it is possible to select an alternative location. A target location has first to be defined using the annotation `@doc.location`: this creates a label associated with the current section. Defining the annotation `@doc.relocate` with this target elsewhere in the source file will move all subsequent additions to the target location; this relocation will continue up to the next division marker or relocation definition. Note that defining an empty relocation reverts to the effective current location. Example:

```
(!@doc.
  @chapter My first chapter
  @p some text related to the first chapter
  @location first_chap
  @section first section of first chapter
  @p something about the section
  @relocate first_chap
  @p this paragraph will be inserted directly under first chapter
  @relocate
  @p but this one will remain in the section
!)
```

### 2.19.1.3 Symbol definitions

The following sections list the various documentation annotations that can be defined depending on the kind of the entity (parameter, variable, type or subroutine) to be documented. Some of these annotations are automatically defined by the compiler: in the case of values (like the value of a constant) the automatic definition may not be performed if the value is the result of a calculation that cannot be evaluated at compile time (“runtime constant”). In this case it is required to explicitly specify the text that should be retained in the documentation.

#### Parameters

- `@doc.descr` Description (1-2 text lines)
- `@doc.default` Default value (automatically generated)
- `@doc.type` Type (automatically generated)
- `@doc.value` Possible value and explanation of its meaning (may be defined several times)
- `@doc.info` Some more detailed explanations (may be defined several times)
- `@doc.ignore` The symbol will be ignored by the documentation processor

#### Types, constants and variables

This set of annotations apply to symbols declared in *declarations blocks*. Record fields (both for a type declaration and for a variable) can be described using `@doc.recflddescr`: the value of this annotation consists in the name of the field followed by its description (a space should separate these two components)

- `@doc.descr` Description (1-2 text lines)
- `@doc.const` For a constant: value (automatically generated)
- `@doc.type` Type (automatically generated)
- `@doc.typedef` For a type definition: type (automatically generated)
- `@doc.value` Possible value and explanation of its meaning (may be defined several times)
Procedures and functions

Information from different overloaded versions of a given subroutine is merged automatically. The @doc.group annotation may be used to merge information of routines with different names but used for a similar task (up to 3 different subroutine names can be grouped). The @doc.param annotation is used to describe the parameters of the routine: the value of this annotation consists in the name of the parameter followed by its description (a space should separate these two components).

@doc.group     Name of another subroutine that this one should be grouped with
@doc.descr     Description (1-2 text lines)
@doc.shortdescr Shortened description for table of contents and list display
@doc.syntax    Routine signature (automatically generated)
@doc.param     Name and meaning of a subroutine argument (may be defined several times)
@doc.paramval  Possible value and meaning of a subroutine argument (may be defined several times). The value of this annotation is the name of the parameter (as specified with a preceding @doc.param) followed by the value and the explanation
@doc.return    For functions only: what is returned
@doc.err       Possible error code (may be defined several times)
@doc.example   Example of use (may be defined several times)
@doc.info      Some more detailed explanations (may be defined several times)
@doc.related   List of related symbols
@doc.ignore    The subroutine will be ignored by the documentation processor

2.19.1.4 Annotation definitions

A special set of annotations (category @doc.annot) is available for documenting annotation definitions in Mosel packages (not supported for Mosel models). The annotations for documenting annotation definitions are global annotations, their value must start with an annotation name in order to associate them with the corresponding annotation definition.

@doc.annot.descr Annotation name followed by a short description (1-2 text lines)
@doc.annot.default Annotation name and default value
@doc.annot.value  Annotation name, possible value and explanation of its meaning (may be defined several times)
@doc.annot.type   Annotation type
2.19.2 moseldoc documentation processor

2.19.2.1 Running moseldoc

The moseldoc program takes as input either a bim file produced from a Mosel model compiled with the -D compiler option or directly a Mosel source file (in which case a compilation step is automatically executed). Typically the generation of the documentation from a source file will be obtained with the following command:

```
>moseldoc mymodel.mos
```

The result of this process is an XML file ("mymodel_doc.xml") and a directory containing an HTML version of the documentation ("mymodel_html"). The program will produce only the XML file (from a bim or source file) if option -xml is used and only the HTML output (from an XML file) if -html is selected. The option -f is required to force the replacement of existing files.

As a Mosel program available in source form, moseldoc can be adapted to fit specific requirements. To re-generate the executable use this compilation command:

```
>mosel comp -s moseldoc.mos -o deploy.exe:moseldoc,css-z=moseldoc.css
```

2.19.2.2 Structure of the generated document

The resulting document respects the structure defined by the dedicated annotations (chapter, section, subsection). In each of these divisions, the paragraphs are exposed first, then the parameters and variables and finally the list of subroutines. If no structural elements have been defined, a chapter per entity type is automatically created to group similar objects (Parameters, Constants, Types, Variables and Subroutines).

2.19.2.3 Processing of annotation values

Values associated with descriptive text annotations (like section titles or descriptions) are interpreted as XML. Paragraphs (@doc.p) and examples (@doc.example) are handled in a specific way: by default the value is inserted as XML but, if the value starts with [TXT], the content is treated as plain text; if it starts with [SRC], the value is considered to be some example code and it is reproduced preserving spacing. If it starts with [NOD], it is interpreted as a self-contained XML node (i.e. it is not inserted in a paragraph block). In an XML block of text, the markers ref (chapter/section/subsection reference), fctRef (subroutine reference) and entRef (entity reference) are processed such that in the HTML document they are turned into hyperlinks to the corresponding objects. Similarly, the tt element type is replaced by an appropriate style for displaying code samples.

2.20 Message translation

Mosel supports a message translation mechanism that makes it possible to display messages in the current language of the operating environment. This system requires that all messages are
originally written in English and identified as messages to be translated (it is usually not desirable
to translate all text strings of a model). The Mosel compiler can then collect all messages to be
translated for building message catalogs. Each message catalog file contains the translations of
the messages for a given language: Mosel will select the appropriate file for the current
language during its execution to use the right set of translations. The system is designed such
that it will not fail if a translation or an entire language is missing: in such a case the original
English text is used.

2.20.1 Preparing the model source

Most often, not all text strings occurring in a program are to be translated to native language.
This is why it is necessary to tag each message to be translated such that the automatic message
translation system can process only the relevant texts. The tagging is achieved by using the
operators \_c(), \_() or the modified procedures write(), writeln(), fwrite() and
fwriteln().

The operator \_c() is used to identify constant strings that should be collected for translation but
the string will not be translated at the place where it is used. This operator can be applied to a
list of string constants. A similar effect can be obtained with the annotation mc.msgid.

The function \_() applies to both constant strings and variables: it replaces its argument by the
translated string. As with the operator \_c() constant strings are collected for the message
catalogs, but they will also be replaced by their translation at the place where the operator is
applied to the string.

The write_ and writeln_ procedures are equivalent to their normal versions except that they
process the constant strings they have to display for translation.

All translations of a model (or package) are grouped under a domain: this identifier is used to
name the message catalog files. The default domain name is the model (or package) name after
having replaced spaces and non-ascii characters by underscores (for instance the domain name of
the model "my mod" is "my_mod"). The domain name can also be specified using the mc.msgdom
annotation.

The following model example shows the use of the various markers:

```
model translate
! The message domain is 'trs' (default name would be the model name 'translate')
!@mc.msgdom trs

declarations
! The elements of 'nums' are kept in English, but collected for translation
nums=[\_c("one","two","three")]
! Add 'four' to the message catalogs (although it is not used here)
!@mc.msgid:four
end-declarations

! Translate the message text, without translating 'nums'
writeln_("all numbers (in English): ", nums)
n:=getfirst(nums)
! Translate the message text and the first occurrence of 'n', but not
! its second occurrence
writeln_("the first number is: ", \_\_(n), " (in English:\", n, ")")
end-model
```

2.20.2 Building the message catalogs

Once the model source has been prepared, the list of messages to be translated can be extracted.
This operation is performed by the Mosel compiler when executed with the option \-x:

```
>mosel comp \-x mymod.mos \-o trs.pot
```
The output of this command is a Portable Object Template (POT): this is a text file consisting of a list of pairs msgid (message to translate), msgstr (translation) for which only the first entry is populated.

With the example model from the previous section the generated POT file results in the following:

```plaintext
# Created by Mosel v4.0.0 from 'translate.mos'
# Domain name: trs

msgid "all numbers (in English): %L"
msgstr ""

msgid "four"
msgstr ""

msgid "one"
msgstr ""

msgid "the first number is: %s (in English:%s)"
msgstr ""

msgid "three"
msgstr ""

msgid "two"
msgstr ""
```

For each of the supported languages a separate PO (Portable Object) file that will contain the corresponding translations has to be created from this template. The command xprnls is used for this task (for further details please refer to the XPRNLS Reference Manual). For instance the following command will create the file for the Italian translations of the messages:

```bash
>xprnls init -o trs.it.po trs.pot
```

Here we name the file domain.language.po in order to ease the management of these translation files (where language stands for the ISO639 language code).

The generated file is a copy of the template with an additional header that should be completed by the translator (it is pre-populated with information obtained from the system), in particular the language (property "Language") and the encoding (property "Content-Type"). For each of the msgid records the translation in the language associated to the file has to be provided in the msgstr record. Note that some messages include escape sequences (like "\n") and format markers (e.g. "%s"): the corresponding translation must include the same format markers as the original text and they must appear in the same order (otherwise the translation will be ignored).

The beginning of the translation file of our example for French (named "trs.fr.po") should be similar to the following (the extract below shows only the header and the translation of the first message):

```plaintext
msgid ""
msgstr ""
"Project-Id-Version: My translation example"
"POT-Creation-Date: 2015-12-04 18:16+0100"
"PO-Revision-Date: 2015-12-04 18:16+0100"
"Last-Translator: Jules Verne"
"Language: fr"
"Content-Type: text/plain; charset=ISO8859-15"

msgid "all numbers (in English): %L"
msgstr "tous les nombres (en anglais): %L"
```

The message catalogs for the PO files are obtained by running once more the xprnls command,
this time using the option \texttt{mogen}:

\begin{verbatim}
  >xprnls mogen -d locale trs.*.po
\end{verbatim}

This command will compile each of the PO files into a Machine Object (MO) file named \texttt{trs.mo} that will be saved under the directory \texttt{locale/lang/LC_MESSAGES}. This directory tree must be distributed along with the model file for the automatic translation to work.

### 2.20.3 Model execution

During the execution of the model the message catalogs for the current language (as indicated by the operating system) are loaded automatically from the `locale` directory. This location is defined by the "localedir" control parameter (by default this is "./locale"). If no message catalog can be found for the requested language then the original English text is used. This will also be the case if a translation is missing (e.g. if the message catalog has not been updated after some model source change).

When run on a computer configured for French our example displays:

\begin{verbatim}
  tous les nombres (en anglais): ['one','two','three']
  le premier nombre est: un (en anglais: one)
\end{verbatim}
This chapter lists in alphabetical order all predefined functions and procedures included in the Mosel language. Certain functions or procedures take predefined constants as input values or return values that correspond to predefined constants. In every case, these constants are documented with the function or procedure. In addition, Mosel defines a few other useful numerical constants:

- MAX_INT: maximum integer number
- MAX_REAL: maximum real number
- M_E: base of natural logarithms $e$
- M_PI: value of $\pi$
- INFINITY: Infinity
- NAN: Not A Number
Predefined functions and procedures

abs

**Purpose**
Get the absolute value of an integer or real.

**Synopsis**

```pascal
function abs(i:integer):integer
function abs(r:real):real
```

**Arguments**
- `i` Integer number for which to calculate the absolute value
- `r` Real number for which to calculate the absolute value

**Return value**
Absolute value of an integer or real number.

**Further information**
This function returns the absolute value of an integer or real number. The returned type corresponds to the type of the input.

**Related topics**
- `exp`, `ln`, `log`, `sqrt`. 
**arctan**

**Purpose**
Get the arctangent of a value.

**Synopsis**

```plaintext
function arctan(r:real):real
```

**Argument**

- `r` Real number to which to apply the trigonometric function

**Return value**
Arctangent of the argument.

**Example**
The following functions compute the arcsine and arccosine of a value:

```plaintext
function arcsin(s:real):real
    returned:=arctan(s/sqrt(1-s^2))
end-function

function arccos(c:real):real
    returned:=arctan(sqrt(1-c^2)/c)
end-function
```

**Related topics**
`cos, sin`
asproc

Purpose
Ignore the return value of a function call.

Synopsis
procedure asproc(fctcall)

Argument
fctcall A function call

Example
asproc(splithead(L,2))

Further information
This procedure makes it possible to call a function and ignore its return value (see also option fctasproc in section 2.3.3).
assert

Purpose
Abort execution if a condition is not satisfied.

Synopsis
procedure assert(c:boolean)
procedure assert(c:boolean,m:string)
procedure assert(c:boolean,m:string,e:integer)

Arguments
- c  Condition to verify
- m  Error message to display in case of failure
- e  Error code to return in case of failure (default: 8)

Example
assert(and(i in I) mydata(i)>0)
assert(isodd(a),"a is not odd!!")

Further information
1. If the condition \(c\) is satisfied, this procedure has no effect, otherwise it displays an error message and aborts execution by calling \texttt{exit}. The versions of the procedure with 2 and 3 parameters can be used to replace the default message (location of the statement in the source) and default exit value (8).

2. Assertions are usually used as a debugging tool and are ignored when the model is compiled without debugging information (i.e. none of options \(-g\) or \(-G\) is used). It is however possible to keep assert statements even when no debugging information is included by specifying the compiler directive \texttt{keepassert} (see Section 2.3).

Related topics
exit
Predefined functions and procedures

bitflip

Purpose
Flip bits (bitwise XOR).

Synopsis
function bitflip(i:integer, j:integer):integer

Arguments
i  Integer to be set
j  Value to flip

Return value
Bitwise XOR of the operands.

Example
In the following, i takes the value 9, j takes the value 141, and k takes the value 7:

i:= bitflip(12, 5)
j:= bitflip(13, 128)
k:= bitflip(13, 10)

Further information
This function computes the bitwise exclusive OR of its operands.

Related topics
bittest, bitshift, bitset, bitneg, bitval
Predefined functions and procedures

bitneg

Purpose
Bitwise negation (bitwise NOT).

Synopsis
function bitneg(i:integer):integer

Argument
i Integer to negate

Return value
Negated value of argument.

Example
In the following, i takes the value -6, j takes the value 2147483647, and k takes the value -4:

i:= bitneg(5)
j:= bitneg(-2147483647-1)
k:= bitneg(3)

Further information
The bitwise NOT (or complement) consists in computing the logical negation of each bit: 1 is replaced by 0 and 0 is replaced by 1.

Related topics
bitset, bittest, bitflip, bitshift, bitval
**bitset**

**Purpose**
Set bits (bitwise OR).

**Synopsis**
```
function bitset(i:integer, j:integer):integer
```

**Arguments**
- **i** Integer to be set
- **j** Value to set

**Return value**
Bitwise OR of the operands.

**Example**
In the following, i takes the value 13, j takes the value 141, and k takes the value 15:
```
i := bitset(12, 5)
j := bitset(13, 128)
k := bitset(13, 10)
```

**Further information**
This function computes the bitwise OR of its operands.

**Related topics**
- bittest
- bitshift
- bitflip
- bitneg
- bitval
Predefined functions and procedures

**bitshift**

**Purpose**
Shift an integer by a number of bits.

**Synopsis**
```plaintext
function bitshift(i:integer, n:integer):integer
```

**Arguments**
i  Integer to be shifted
n  Number of bits: >0 for shifting to the left and <0 for shifting to the right

**Return value**
Shifted integer.

**Example**
In the following, i takes the value 160, j takes the value 32, and k takes the value 1:
```plaintext
i := bitshift(5, 5)
j := bitshift(1, 5)
k := bitshift(128, -7)
```

**Further information**
Shifting of 1 bit to the right is the same as dividing it by 2 and shifting of 1 bit to the left is the same as multiplying by 2.

**Related topics**
bitset, bittest, bitflip, bitneg, bitval
bittest

Purpose
Test bit settings (bitwise AND).

Synopsis
function bittest(i:integer, mask:integer):integer

Arguments
i  Integer to be tested
mask  Bit mask

Return value
Bits selected by the mask.

Example
In the following, i takes the value 4, j takes the value 5, and k takes the value 8:

i:= bittest(12, 5)
j:= bittest(13, 5)
k:= bittest(13, 10)

Further information
This function compares a given number with a bit mask and returns those bits selected by the mask that are set in the number (bit 0 has value 1, bit 1 has value 2, bit 2 has value 4, and so on - use function bitval to get the value of a bit).

Related topics
bitset, bitshift, bitflip, bitneg, bitval
Predefined functions and procedures

**bitval**

**Purpose**
Compute the value corresponding to a bit number.

**Synopsis**
function bitval(i:integer):integer

**Argument**
i Bit number (between 0 and 31)

**Return value**
Value of the selected bit.

**Example**
In the following, i takes the value 1, j takes the value -2147483648, and k takes the value 16:

i := bitval(0)
j := bitval(7)
k := bitval(4)

**Further information**
This function computes the value corresponding to a bit number. The evaluation of bitval(i) corresponds to bitshift(i,i)

**Related topics**
bitset, bitshift, bitflip, bitneg, bittest
Predefined functions and procedures

ceil

Purpose
Round a number to the next largest integer.

Synopsis
function ceil(r:real):integer

Argument
r Real number to be rounded

Return value
Rounded value.

Example
In the following, i takes the value 6, j takes the value -6, and k takes the value 13:

i := ceil(5.6)
j := ceil(-6.7)
k := ceil(12.3)

Related topics
floor, round.
**COS**

**Purpose**
Get the cosine of a value.

**Synopsis**

```plaintext
function cos(r:real):real
```

**Argument**
r
Real number to which to apply the trigonometric function

**Return value**
Cosine value of the argument.

**Example**
The function tangent can be implemented as follows:

```plaintext
function tangent(x:real):real
    returned:=sin(x)/cos(x)
end-function
```

**Related topics**
arctan, sin.
create

Purpose
Create explicitly a cell of a dynamic array.

Synopsis
procedure create(x: array reference)

Argument
x Cell to be created

Example
The following declares a dynamic array of variables, creating only those corresponding to the odd indices. Finally, it defines the linear expression $x(1) + x(3) + x(5) + x(7)$:

\[
declarations
x: dynamic array(1..8) of mpvar
end-declarations

forall(i in 1..8| isodd(i)) create(x(i))
c := sum(i in 1..8) x(i)
\]

Further information
Usually cells of dynamic arrays are created by means of assignments. This procedure can be used as a replacement for an assignment especially when the type of a dynamic array does not provide any assignment operator (like mpvar for instance).

Related topics
Section 2.6.4, delcell.
currentdate

Purpose
Return the current date as a Julian Day Number (JDN).

Synopsis
function currentdate:integer

Return value
The number of days elapsed since 1/1/1970 as an integer.

Further information
1. The control parameter "UTC" indicates whether this function returns a date in local or UTC time.
2. Refer to the module mmsystem for a set of dedicated types for handling date and time.

Related topics
setparam, timestamp, currenttime
currenttime

Purpose
Return the current time as the number of milliseconds since midnight.

Synopsis
function currenttime:integer

Return value
The number of milliseconds since midnight as an integer.

Further information
1. The control parameter "UTC" indicates whether this function returns a time in local or UTC time.
2. Refer to the module mmsystem for a set of dedicated types for handling date and time.

Related topics
setparam, timestamp, currentdate
cuthead

**Purpose**
Cut the first elements of a list.

**Synopsis**
```plaintext
procedure cuthead(l:list, o:integer)
```

**Arguments**
- `l` A list
- `o` Number of elements to remove if >0 or number of elements to keep if <0

**Example**
```plaintext
L:=[1,2,3,4,5]
cuthead(L,2) ! => L=[3,4,5]
cuthead(L,-1) ! => L=[5]
```

**Further information**
If the second parameter is 0, the list is unchanged. If the same parameter is larger than the size of the list, all elements are deleted.

**Related topics**
cuttail
cuttail

**Purpose**
Cut the last elements of a list.

**Synopsis**
procedure cuttail(l:list, o:integer)

**Arguments**
- `l` A list
- `o` Number of elements to remove if >0 or number of elements to keep if <0

**Example**

```
L:=[1,2,3,4,5]
cuttail(L,2) ! => L=[1,2,3]
cuttail(L,-1) ! => L=[1]
```

**Further information**
If the second parameter is 0, the list is unchanged. If the same parameter is larger than the size of the list, all elements are deleted.

**Related topics**
cuthead
**delcell**

**Purpose**
Delete a cell or all cells of a dynamic array.

**Synopsis**

```plaintext
procedure delcell(x: array reference)
procedure delcell(a: array)
```

**Arguments**

- `x`  Cell to be deleted
- `a`  An array

**Further information**

1. Only cells of arrays explicitly declared dynamic can be deleted. This function has no effect with other types of array.

2. Deleting a cell of an array of referenced objects (like `mpvar`) may not effectively release that object. Actually, a referenced object is released only when all its references have been removed. For instance, if an object appears in a set, deleting its main reference using `delcell` will not remove this object from the set.

**Related topics**

Section 2.6.4, `create`. 

**exists**

**Purpose**
Check if a given entry in a dynamic array has been created.

**Synopsis**
function exists(x):boolean

**Argument**
x Array reference (e.g. t(1))

**Return value**
true if the entry exists, false otherwise.

**Example**
The following, a dynamic array of decision variables only has its even elements created, which is checked by displaying the existing variables:

```plaintext
declarations
S=1..8
x: dynamic array(S) of mpvar
end-declarations

forall(i in S| not isodd(i)) create(x(i))
forall(i in S| exists(x(i)))
writeln("x(", i, ") exists")
```

**Further information**
1. If an array is declared dynamic its elements are not created at its declaration. This function indicates if a given element has been created.
2. Under certain conditions, the exists function call is optimized by the compiler when used for filtering an aggregate operator: the loop is only performed for the existing entries instead of enumerating all possible tuples of indices for finding the relevant ones.

**Related topics**
Section 2.7.2, create.
**exit**

**Purpose**
Terminate the program.

**Synopsis**

```plaintext
procedure exit(code:integer)
```

**Argument**

`code` Value to be returned by the program

**Further information**
This procedure terminates the current program and returns the given value. Models exit by default with a value of 0 unless this is changed using `exit`. The Mosel command line interpreter uses this value as exit status.

**Related topics**
Section 1.3.
**exp**

**Purpose**
Get the natural exponent of a value.

**Synopsis**
```plaintext
function exp(r:real):real
```

**Argument**
r
Real value the function is applied to.

**Return value**
Natural exponent ($e^r$) of the argument.

**Related topics**
abs, exp, ln, log, sqrt.
**exportprob**

**Purpose**
Export a problem to a file.

**Synopsis**

```plaintext
procedure exportprob(options:integer, filename:string, obj:linctr)
procedure exportprob(options:integer, filename:string)
procedure exportprob(filename:string, obj:linctr)
procedure exportprob(filename:string)
procedure exportprob
```

**Arguments**

- **options**  
  File format options:
  - EP_MIN minimization (default)
  - EP_MAX maximization
  - EP_MPS MPS format
  - EP_STRIP Use scrambled names
  Several options may be combined using +.
- **filename**  
  Name of the output file. If the empty string "" is given, output is printed to the standard output (the screen)
- **obj**  
  Objective function constraint

**Example**

The following prints the current problem to the screen using the default format and with MinCost as objective function. The second statement exports the problem in LP-format and with scrambled names to the file prob1.lp maximizing the constraint Profit:

```plaintext
declarations
    MinCost, Profit:linctr
end-declarations

exportprob(0, "", MinCost)
exportprob(EP_MAX+EP_STRIP, "prob1", Profit)
```


Predefined functions and procedures

Further information

1. If the given filename uses the default IO driver (no driver specified) and has no extension, Mosel appends `.lp` to it for LP format files and `.mps` for MPS format.

2. Normally, local symbols (i.e. defined in a procedure or function) are replaced by generated names in the exported matrix. However, if the model has been compiled with option `-G`, names defined locally to the routine calling `exportprob` are used in the exported matrix. Moreover, if a local symbol hides a global one, this symbol is prefixed by `'`

3. If the model is compiled with `-G` and the control parameter `recloc` is set to `true` (see `setparam`), missing constraint names are replaced by the source location of the constraint definition (i.e. a combination of the row number, source file name and line number in the file).

4. If no option is provided, the default format is LP for a minimization; if no constraint is given, the current objective (if available) is exported. The matrix is printed to the standard output when this function is used without parameter.

5. This function exports only the LP/MIP problem directly handled by the Mosel core libraries. It cannot report problem extensions managed by external modules. For instance, quadratic constraints or indicator constraints provided by the Xpress Optimizer are not shown by this routine: for this type of problems, the module-specific `writeprob` routine has to be used instead of `exportprob`.

Related topics

`setname`.
fclose

**Purpose**
Close the active input, output or error stream.

**Synopsis**
procedure fclose(stream:integer)

**Argument**
stream  The stream to close:
- F_INPUT  Input stream
- F_OUTPUT  Output stream
- F_ERROR  Error stream

**Further information**
This procedure closes the file that is currently associated with the given stream. The file preceding the closed file (in the order of opening) is then assigned to the corresponding stream. A file that is closed with this procedure must previously have been opened with fopen. This function has no effect if the corresponding stream is not associated with any explicitly opened file (i.e. it is not possible to close the default input, output or error streams). All open streams are automatically closed when the program terminates.

**Related topics**
flush, fopen, fselect, getfid, iseof.
fflush

**Purpose**
Force the operating system to write buffered data.

**Synopsis**
procedure fflush

**Further information**
This procedure forces a write of all buffered data of the default output stream. fflush is automatically called when the stream is closed either with fclose or when the program terminates.

**Related topics**
fclose, fopen.
**finalize**

**Purpose**
Finalize the definition of a set or list.

**Synopsis**

```plaintext
procedure finalize(s: set)
procedure finalize(l: list)
```

**Arguments**

- **s**: Dynamic set
- **l**: Dynamic list

**Example**

In the following, an indexing set is defined, on which depends a dynamic array of decision variables. The set is subsequently defined to have three elements and is finalized. A static array is then defined:

```plaintext
declarations
    Set1: set of string
    x: array(Set1) of mpvar  ! Declare a dynamic array of variables
      ! (entries need to be created
      !    subsequently)
end-declarations
Set1:= {"first", "second", "fifth"}
finalize(Set1)  ! Finalize the set definition

declarations
    y: array(Set1) of mpvar  ! Declare a static array of variables
      ! (entries are created immediately)
end-declarations
```

**Further information**

This procedure finalizes the definition of a set (or list), that is, it turns a dynamic set into a constant set consisting of the elements that are currently in the set. All subsequently declared arrays that are indexed by this set will be created as static (= fixed size). Any arrays indexed by this set that have been declared prior to finalizing the set retain the status dynamic but their set of elements cannot be modified any more.
**findfirst**

**Purpose**
Find the first occurrence of an element in a list.

**Synopsis**
function findfirst(l:list, e:type_of_l):integer

**Arguments**
- l A list
- e The element to look for (it must be of the type of l)

**Return value**
The position of the element or 0 if the element is not included in the list.

**Example**
L=['a','b','c','d','b']
i:=findfirst(L,'b') ! => i=2
i:=findlast(L,'f') ! => i=0

**Related topics**
findlast
findlast

**Purpose**
Find the last occurrence of an element in a list.

**Synopsis**
```plaintext
function findlast(l:list, e:type_of_l):integer
```

**Arguments**
- `l` A list
- `e` The element to look for (it must be of the type of `l`)

**Return value**
The position of the element or 0 if the element is not included in the list.

**Example**
```plaintext
L:=[‘a’,’b’,’c’,’d’,’b’]
i:=findlast(L,’b’) ! => i=5
i:=findlast(L,’f’) ! => i=0
```

**Related topics**
- findfirst
floor

Purpose
Round a number to the next smallest integer.

Synopsis
function floor(r:real):integer

Argument
r Real number to be rounded

Return value
Rounded value.

Example
In the following, i takes the value 5, j the value -7, and k the value 12:

i := floor(5.6)
j := floor(-6.7)
k := floor(12.3)

Related topics
ceil, round.
Predefined functions and procedures

**fopen**

**Purpose**
Open a file and make it the active input or output stream.

**Synopsis**
procedure fopen(f:string, mode:integer)

**Arguments**
f The name of the file to be opened
mode Open mode (may be combined):
- F_INPUT Open for reading
- F_OUTPUT Empty the file and open it for writing
- F_ERROR Empty the file and open it for writing as the error stream
- F_APPEND Open for writing, appending new data to the end of the file
- F_TEXT Open in text mode (the default)
- F_BINARY Open in binary mode
- F_LINBUF If open for writing, flushes buffer after end of each line (default when writing to a console or for an error stream)
- F_SILENT Do not display IO error messages

**Further information**
1. This procedure opens a file for reading or writing. If the operation succeeds, depending on the opening mode, the file becomes the active input, output or error stream. The procedures write and writeln are used to write data to the default output stream and the functions read, readln, and fskipline are used to read data from the default input stream. Error messages are sent to the error stream.

2. The behavior of this function in case of an IO error (i.e. the file cannot be opened) is directed by the control parameter ioctrl: if the value of this parameter is ‘false’ (default value), the interpreter stops. Otherwise, the interpreter ignores the error and continues. The error status of an IO operation is stored in the control parameter iostatus which is 0 when the last operation has been executed successfully. Note that this parameter is automatically reset once its value has been read using the function getparam. The behavior of IO operations after an unhandled error is not defined.

3. The *binary mode* disables character encoding conversion (see section 2.15).

**Related topics**
fclose, fselect, getfid.
fselect

Purpose
Select the active input, output or error stream.

Synopsis
procedure fselect(stream:integer)

Argument
stream   The stream number

Example
The following saves the file ID of the default output before switching output to the file
mylog.txt. Subsequently, the file ID of the current output stream is saved and the default output
is again selected.

    def_out:= getfid(F_OUTPUT)
    fopen("mylog.txt", F_OUTPUT)
    ...
    my_out:= getfid(F_OUTPUT)
    fselect(def_out)

Further information
1. This procedure selects the given stream as the active input, output or error stream. The
   concerned stream is designated by the opening status of the given stream (that is, if the given
   stream has been opened for reading, it will be assigned to the default input stream). The stream
   number can be obtained with the function getfid.

2. The default input, output and error streams have respectively numbers 0, 1 and 2.

Related topics
fclose, fopen, getfid, fwrite, writeln.
**fskipline**

**Purpose**
Advance in the default input stream as long as comment lines are found.

**Synopsis**
```pascal
procedure fskipline(filter:string)
```

**Argument**
- `filter` List of comment signs

**Example**
In the following, the first statement skips all lines beginning with either ‘#’ or ‘!’.
The second statement skips any following blank lines:
```pascal
fskipline("!#")
fskipline("\n")
```

**Further information**
This procedure advances in the input stream using the given list of comment signs as a filter. Each character of the given string is considered to be a symbol that marks the beginning of a comment line. Note that the character ‘\n’ designates lines starting with nothing, that is, empty lines. During the parsing, spaces and tabulations are ignored.

**Related topics**
- `read`, `readln`. 
fwrite, fwriteln

Purpose
Send an expression or list of expressions to the specified output stream.

Synopsis
procedure fwrite(fd:integer, e1:expr[, e2:expr...])
procedure fwriteln(fd:integer)
procedure fwriteln(fd:integer, e1:expr[, e2:expr...])

Arguments
fd An output stream number
e1, e2,... Expression or list of expressions

Further information
1. These procedures are equivalent to calling fselect before using the corresponding output procedure and then restore the initial current stream with a second call to fselect.
2. The selected stream may also be an error stream.

Related topics
write, writeln, fselect, getfid.
getact

**Purpose**
Get the activity value of a constraint.

**Synopsis**
function getact(c:linctr):real

**Argument**
c    A linear constraint

**Return value**
Activity value or 0.

**Further information**
This function returns the activity value of a constraint if the problem has been solved successfully, otherwise 0 is returned.

**Related topics**
getdual, getslack, getsol.
getcoeff

Purpose
Get a constraint coefficient or constant term.

Synopsis
function getcoeff(c:linctr):real
function getcoeff(c:linctr, x:mpvar):real

Arguments
 c  A linear constraint
 x  A decision variable

Return value
Coefficient of the variable or the constant term.

Example
In this example a single constraint with three variables is defined. The calls to getcoeff result in r taking the value -1 and s taking the value -12.

    declarations
     x,y,z:mpvar
    end-declarations

    c:= 4*x + y -z <= 12
    r:= getcoeff(c, z)
    r:= getcoeff(c)

Further information
This function returns the coefficient of a given variable in a constraint, or if no variable is given, the constant term (= -RHS) of the constraint. The returned values correspond to a normalised constraint representation with all variable and constant terms on the left side of the (in)equality sign.

Related topics
getcoeffs, getvars, setcoeff.
**getcoeffs**

**Purpose**
Get all variable coefficients of a constraint.

**Synopsis**
procedure getcoeffs(c:linctr, a:array(set of mpvar) of real, s:set of mpvar)

**Arguments**
c  A linear constraint
a  An array of reals indexed by decision variables
s  A set of decision variables

**Further information**
1. This procedure returns in the parameter a the coefficients of all variables of a constraint. After calling this procedure, the coefficient of variable v of constraint c is a(v). The set s is used to specify for which variables the coefficients have to be retrieved (if this set is empty all variables are considered).

2. If set s is empty all cells of array a are updated (i.e. cells corresponding to variables not included in constraint c are set to 0). Otherwise only cells corresponding to elements of s are modified.

**Related topics**
getcoeffs, getcoeff
getdual

**Purpose**
Get the dual value of a constraint.

**Synopsis**
function getdual(c:linctr):real

**Argument**
c A linear constraint

**Return value**
Dual value or 0.

**Further information**
This function returns the dual value of a constraint if the problem has been solved successfully and the constraint is contained in the problem, otherwise 0 is returned.

**Related topics**
getcost, getslack, getsol.
getfid

Purpose
Get the stream number of the active input, output or error stream.

Synopsis
function getfid(stream:integer):integer

Argument

stream  The stream to query:

F_INPUT  Input stream
F_OUTPUT  Output stream
F_ERROR  Error stream

Return value
Stream number.

Further information
The returned value can be used as parameter for the function fselect.

Related topics
fselect.
**getfirst**

**Purpose**
Get the first element of a range set or a list.

**Synopsis**

```plaintext
function getfirst(r:range):integer
function getfirst(l:list):type_of_l
```

**Arguments**

- `r` A range set
- `l` A list

**Return value**

The first element of the set or list.

**Example**

In this example the range set `r` is defined before its first and last elements are retrieved and displayed:

```plaintext
declarations
    r=2..8
end-declarations
...
writeln("First element of r: ", getfirst(r),
    "\nLast element of r: ", getlast(r))
```

**Further information**

When applied to a list, the type of the function is the type of the list. An error is generated if the argument of the function is empty.

**Related topics**

- `getlast`. 
gethead

Purpose
Get a copy of the first elements of a list.

Synopsis
function gethead(l:list, o:integer):list

Arguments
- l: A list
- o: Number of elements to copy if >0 or number of elements to ignore if <0

Return value
A (partial) copy of the list.

Example
L:=\[1,2,3,4,5\]
L2:=gethead(L,2) ! => L2=\[1,2\]
L2:=gethead(L,-1) ! => L2=\[1,2,3,4\]

Further information
This function does not alter its input list. If the second parameter is 0 an empty list is returned. If the same parameter is larger than the size of the list the function returns a copy of the original list.

Related topics
gettail
**getfname**

**Purpose**
Get the file name associated to the active input, output or error stream.

**Synopsis**
function getfname(stream:integer):string

**Argument**
- stream The stream to query:
  - F_INPUT Input stream
  - F_OUTPUT Output stream
  - F_ERROR Error stream

**Return value**
File name.
getlast

**Purpose**
Get the last element of a range set or a list.

**Synopsis**

```plaintext
function getlast(r:range):integer
function getlast(l:list):type_of_l
```

**Arguments**

- `r` A range set
- `l` A list

**Return value**

The last element of the set or list.

**Example**

In this example the range set `r` is defined before its first and last elements are retrieved and displayed:

```plaintext
declarations
  r=2..8
end-declarations
...
writeln("First element of r: ", getfirst(r), 
  "\nLast element of r: ", getlast(r))
```

**Further information**

When applied to a list, the type of the function is the type of the list. An error is generated if the argument of the function is empty.

**Related topics**

- `getfirst`
getobjval

**Purpose**
Get the objective function value.

**Synopsis**
function getobjval:real

**Return value**
Objective function value or 0.

**Further information**
This function returns the objective function value if the problem has been solved successfully. If integer feasible solution(s) have been found, the value of the best is returned, otherwise the value of the last LP solved.

**Related topics**
getsol.
getparam

**Purpose**  
Get the current value of a control parameter.

**Synopsis**  
function getparam(name:string):integer|string|real|boolean

**Argument**  
name  
Name of the control parameter whose value is to be returned (case insensitive).

**Return value**  
Current setting of the control parameter.
Further information

1. Parameters whose values may be returned by this function include the settings of Mosel as well as those of any loaded module. The module may be specified by prefixing the parameter name with the name of the module (e.g. mmxprs.XPRS_verbose). The type of the return value corresponds to the type of the parameter.

2. This function can be applied only to control parameters whose value can be accessed.

3. The name argument must be a constant string: a model parameter, variable or string expression cannot be used as a control parameter name.

4. The following control parameters are supported by Mosel:
   - `realfmt`  Default C printing format for real numbers (string)
   - `zerotol`  zero tolerance in comparisons between reals (real)
   - `ioctrl`  the interpreter ignores IO errors (Boolean)
   - `iostatus`  status of the last IO operation (integer). This parameter is automatically reset once its value has been read
   - `nbread`  number of items recognized by the last read procedure or read in by the last initializations block (integer)
   - `readcnt`  generate per label counting when executing ‘initializations from’ blocks (Boolean)
   - `UTC`  indicate whether the time functions return time expressed in local (false) or UTC (true) time (Boolean)
   - `autofinal`  indicate whether initialisation blocks are finalizing sets (Boolean)
   - `tmpdir`  the Mosel temporary directory (string)
   - `workdir`  the current working directory of the model (string)
   - `restrict`  active restrictions (integer). See Section 1.3.3 for further details.
   - `modelname`  internal unique name of the model being executed.
   - `modelnumber`  order number of the model being executed.
   - `recloc`  indicate whether automatic recording of source location of constraints definitions is active (Boolean)
   - `localedir`  directory where message catalogs are stored (string)
   - `lang`  current language (string)

5. Function `getparam` may also be used to retrieve parser parameters. As opposed to the other parameters whose value is computed at run time, these parameters are evaluated as soon as they are parsed:
   - `parser_line`  number of the line being parsed (integer)
   - `parser_file`  current source file name (string)
   - `parser_date`  current local date (string)
   - `parser_time`  current local time (string)
   - `parser.UTCdate`  current UTC date (string)
   - `parser.UTCtime`  current UTC time (string)
   - `parser_version`  Mosel version (string)
   - `model_version`  Version of the model as given by the version directive (string)

Related topics

- `setparam`
**getrcost**

**Purpose**
Get the reduced cost value of a variable.

**Synopsis**
function getrcost(v:mpvar):real

**Argument**
v A decision variable

**Return value**
Reduced cost value or 0.

**Further information**
This function returns the reduced cost value of a variable if the problem has been solved successfully and the variable is contained in the problem, otherwise 0 is returned.

**Related topics**
getsllack, getsol, getdual.
getreadcnt

Purpose
Get the number of items read in during last ‘initializations from’ for a given label.

Synopsis
function getreadcnt(l:string):integer

Argument
l A label

Return value
Number of items read in for label l.

Further information
Value 0 is returned if the given string does not correspond to a label or if control parameter readcnt has not been set to true before execution of the initializations block.
**getreverse**

**Purpose**
Duplicate and reverse a list.

**Synopsis**
function getreverse(l:list):list

**Argument**
l A list

**Return value**
A reversed copy of the provided list.

**Example**
L:=[1,2,3,4,5]
L2:=L.reverse ! => L=[5,4,3,2,1]

**Related topics**
reverse.
**getsize**

**Purpose**
Get the size of an array, set, list, constraint or string.

**Synopsis**

```plaintext
function getsize(a:array):integer
function getsize(s:set):integer
function getsize(l:list):integer
function getsize(t:string):integer
function getsize(c:linctr):integer
```

**Arguments**

- `a` An array
- `s` A set
- `l` A list
- `t` A string
- `c` A linear constraint

**Return value**
Number of effective entries for an array, number of elements for a set or a list, number of characters for a string, number of terms for a constraint.

**Example**
In the following, a dynamic array is declared holding eight elements, of which only two are actually defined. Calling `getsize` on this array returns 2 rather than 8. The length `lw` of the string `w` is 9.

```plaintext
declarations
   a:dynamic array(1..8) of real
   w = "some text"
end-declarations

   a(1):= 4
   a(5):= 7.2
   la:= getsize(a)
   lw:= getsize(w)
```

**Further information**
In the case of a dynamic array that has been declared with a maximal range this number may be smaller than the size of the range, but it cannot exceed it. When used with a string, this function returns the length of the string (i.e. the number of characters it contains). If used with a linear constraint, this function returns the number of terms of the constraint (the constant term is not taken into account).
Predefined functions and procedures

getslack

Purpose
Get the slack value of a constraint.

Synopsis
function getslack(c:linctr):real

Argument
  c  A linear constraint

Return value
Slack value or 0.

Further information
This function returns the slack value of a constraint if the problem has been solved successfully and the constraint is contained in the problem, otherwise 0 is returned.

Related topics
getdual, getrcost, getsol.
getsol

**Purpose**
Get the solution value of a variable or a linear expression (constraint).

**Synopsis**
function getsol(v:mpvar):real
function getsol(c:linctr):real

**Arguments**
c A linear constraint
v A decision variable

**Return value**
Solution value or 0.

**Further information**
This function returns the (primal) solution value of a variable if the problem has been solved successfully and the variable is contained in the problem (otherwise 0). If used with a constraint, it returns the evaluation of the corresponding linear expression using the current solution.

**Related topics**
getdual, getrcost, getobjval.
**gettail**

**Purpose**
Get a copy of the last elements of a list.

**Synopsis**
function gettail(l:list, o:integer):list

**Arguments**
l  A list
o  Number of elements to copy if >0 or number of elements to ignore if <0

**Return value**
A (partial) copy of the list.

**Example**

L:= [1,2,3,4,5]
L2:=gettail(L,2) ! => L2=[4,5]
L2:=gettail(L,-1) ! => L2=[2,3,4,5]

**Further information**
This function does not alter its input list. If the second parameter is 0 an empty list is returned. If the same parameter is larger than the size of the list the function returns a copy of the original list.

**Related topics**
gethead
gettype

Purpose
Get the type of a constraint.

Synopsis
function gettype(c:linctr):integer

Argument
c A linear constraint

Return value
Constraint type. Values applicable to any type of linear constraint are:
CT_EQ Equality, ‘=’
CT_GEQ Greater than or equal to, ‘≥’
CT_LEQ Less than or equal to, ‘≤’
CT_UNB Non-binding constraint
CT_SOS1 Special ordered set of type 1
CT_SOS2 Special ordered set of type 2

Values applicable for unary constraints are:
CT_CONT Continuous
CT_INT Integer
CT_BIN Binary
CT_PINT Partial integer
CT_SEC Semi-continuous
CT_SINT Semi-continuous integer
CT_FREE Free

Related topics
settype.
**getvars**

**Purpose**
Get the set of variables of a constraint.

**Synopsis**

```plaintext
procedure getvars(c:linctr,s:set of mpvar)
```

**Arguments**
- **c** A linear constraint
- **s** A set of decision variables

**Example**
The following returns the set of variables in a linear constraint to the set variable `vset`, and then loops through them to find their solution values:

```plaintext
declarations
c:linctr
vset: set of mpvar
end-declarations

getvars(c,vset)
forall(x in vset) writeln(getsol(x))
```

**Further information**
This procedure returns in the parameter `s` the set of variables of a constraint. Note that this procedure replaces the content of the set.

**Related topics**
- `getcoeffs`, `getcoeff`
isdynamic

Purpose
Check whether an array, set, or list is dynamic.

Synopsis
function isdynamic(a:array):boolean
function isdynamic(s:set):boolean
function isdynamic(l:list):boolean

Arguments
a    An array
s    A set
l    A list

Return value
ture if the provided entity is dynamic.
iseof

**Purpose**
Test whether the end of the default input stream has been reached.

**Synopsis**

```plaintext
function iseof:boolean
```

**Return value**

true if the end of the default input stream has been reached, false otherwise.

**Example**
The following opens a datafile of integers, reads one from each line and prints it to the console until the end of the file is reached:

```plaintext
declarations
d:integer
end-declarations

... 
fopen("datafile.dat", F_INPUT)
while(not iseof) do
  readln(d)
  writeln(d)
end-do
fclose(F_INPUT)
```

**Further information**
This function returns the “end of file” status of the active input stream.

**Related topics**

fclose, fopen.
**isfinite**

**Purpose**
Test whether a real value is finite.

**Synopsis**
function isfinite(r: real): boolean

**Argument**
r The value to test

**Return value**
true if the value is neither (-)INFINITY nor NAN.

**Further information**
The call isfinite(v) is equivalent to (not isnan(v) and not isinf(v)).

**Related topics**
setmatherr, isnan, isinf.
**ishidden**

**Purpose**
Test whether a constraint is hidden.

**Synopsis**
```plaintext
function ishidden(c:linctr):boolean
```

**Argument**
c
A linear constraint

**Return value**
true if the constraint is hidden, false otherwise.

**Further information**
This function tests the current status of a constraint. At its creation a constraint is added to the current problem, but using the function `sethidden` it may be hidden. This means, the constraint will not be contained in the problem that is solved by the optimizer but it is not deleted from the definition of the problem in Mosel.

**Related topics**
`sethidden`. 

**isinf**

**Purpose**
Test whether a real value is an infinity.

**Synopsis**
function isinf(r: real):boolean

**Argument**
r The value to test

**Return value**
true if the value is INFINITY or -INFINITY.

**Further information**
When the parameter matherr is set to true (see setparam) mathematical functions return the constant NAN or INFINITY instead of failing. This function can be used to identify incorrect results (direct comparison to NAN or INFINITY always fails).

**Related topics**
setmatherr, isnan, isfinite.
isnan

Purpose
Test whether a real value is valid.

Synopsis
function isnan(r: real): boolean

Argument
r The value to test

Return value
true if the value is not valid (i.e. it corresponds to Not A Number).

Further information
When the parameter matherr is set to true (see setparam) mathematical functions return the constant NAN or INFINITY instead of failing. This function can be used to identify incorrect results (direct comparison to NAN or INFINITY always fails).

Related topics
setmatherr, isinf, isfinite.
isodd

Purpose
Test whether an integer is odd.

Synopsis
function isodd(i:integer):boolean

Argument
i An integer number

Return value
true if the given integer is odd, false if it is even.
**ln**

**Purpose**
Get the natural logarithm of a value.

**Synopsis**
function ln(r:real):real

**Argument**
r
Real value the function is applied to. This value must be positive.

**Return value**
Natural logarithm of the argument.

**Example**
The following example provides a function for calculating logarithms to any (positive) base:

```plaintext
function logn(base,number: real):real
    if (number > 0 and base > 0) then
        returned := ln(number)/ln(base)
    else
        exit(1)
    end-if
end-function
```

**Related topics**
exp, log, sqrt.
Predefined functions and procedures

log

**Purpose**
Get the base 10 logarithm of a value.

**Synopsis**
```
function log(r:real):real
```

**Argument**
r       
Real value the function is applied to. This value must be positive.

**Return value**
Base 10 logarithm of the argument.

**Related topics**
exp, ln, sqrt.
makesos1, makesos2

Purpose

Creates a special ordered set (SOS) using a set of decision variables and a linear constraint.

Synopsis

procedure makesos1(cs:linctr, s:set of mpvar, c:linctr)
procedure makesos2(cs:linctr, s:set of mpvar, c:linctr)
procedure makesos1(s:set of mpvar, c:linctr)
procedure makesos2(s:set of mpvar, c:linctr)

Arguments

cs   A linear constraint
s    A set of decision variables
c    A linear constraint

Example

The following generates the SOS1 set my sos based on the linear constraint rr. The resulting set contains the variables x, y, and z with the weights 0, 2, and 4.

declarations
x,y,z: mpvar
rr,mysos: linctr
end-declarations

rr:= 2*y+4*z
makesos1(mysos, {x,y,z}, rr)

Further information

These procedures generate a SOS set containing the decision variables of the set s with the coefficients of the linear constraint c. The resulting set is assigned to cs if it is provided. Note that these procedures simplify the generation of SOS with weights of value 0.
maxlist

**Purpose**
Get the maximum value of a list of integers or reals.

**Synopsis**
function maxlist(i1:integer, i2:integer[, i3:integer...]):integer
function maxlist(r1:real, r2:real[, r3:real...]):real

**Arguments**
i1,i2,... List of integer numbers
r1,r2,... List of real numbers

**Return value**
Largest value in the given list.

**Example**
In the following r is assigned the value 7 by maxlist:

\[ r := \text{maxlist}(-1, 4.5, 2, 7, -0.3) \]

**Further information**
The returned type corresponds to the type of the input.

**Related topics**
minlist.
minlist

**Purpose**
Get the minimum value of a list of integers or reals.

**Synopsis**

```plaintext
function minlist(i1:integer, i2:integer[, i3:integer...]):integer
function minlist(r1:real, r2:real[, r3:real...]):real
```

**Arguments**

- `i1,i2,...` List of integer numbers
- `r1,r2,...` List of real numbers

**Return value**
Smallest value in the given list.

**Example**
In the following `r` is assigned the value `-1` by `minlist`:

```plaintext
r := minlist(-1, 4.5, 2, 7, -0.3)
```

**Further information**
The returned type corresponds to the type of the input.

**Related topics**
maxlist.
**newmuid**

**Purpose**
Generate a unique identifier.

**Synopsis**

```plaintext
function newmuid: string
```

**Return value**
An identifier string.

**Further information**
This function returns a string of the form `muid#_xxx` where # is an execution number in hexadecimal (specific to this model execution) and xxx a random hexadecimal number. It is guaranteed that each generated value does not correspond to any symbol of the model and that it will never be returned again.
**publish**

**Purpose**
Publish a symbol.

**Synopsis**
procedure publish(name:string, ref:external or structured)

**Arguments**
- name  Symbol to identify the object
- ref    A reference to an object of an external type or a structure (e.g. set, list or array)

**Further information**
1. This procedure can be used to publish an object in the model dictionary such that it can found by native code using name. Any entity (including local and private) can be exposed with this routine as long as it is of a referenced type (basically any type except integer, real, boolean and string).
2. The provided name must be a valid identifier that is not yet being used by the model as symbol name (including entity and subroutine names). In case of error the procedure raises an IO error.

**Related topics**
unpublish, newmuid.
**random**

**Purpose**
Generate a random number.

**Synopsis**
function random:real

**Return value**
A randomly generated number in the interval \([0,1)\).

**Example**
In the following \(i\) is assigned a random integer value between 1 and 10:

\[i := \text{integer}(\text{round}((10 \times \text{random}) + 0.5))\]

**Further information**
Each model uses its own generator which is randomly initialized when the model execution starts. The sequence may also be reset using procedure `setrandseed`.

**Related topics**
`setrandseed`.
Predefined functions and procedures

read, readln

Purpose
Read in formatted data from the active input stream.

Synopsis
procedure read(e1:expr[, e2:expr...])
procedure readln
procedure readln(e1:expr[, e2:expr...])

Argument
e1, e2,...  Expression or list of expressions of basic type

Example
The following reads (possible split over several lines) 12 45 word, followed by toto(12 and 45)=word:

dclarations
i,j:integer
s:string
ts:array (range,range) of string
end-declarations
read(i, j, s)
readln("toto(" i, "and", j, ")=", ts(i,j))

Further information
1. These procedures assign the data read from the active input stream to the given symbols or try to match the given expressions with what is read from the input stream. If ei is a symbol that can be assigned a value, the procedure tries to recognise from the input stream a constant of the required type and, if successful, assigns the resulting value to ei. If ei is a constant or a symbol that cannot be reassigned, the procedure tries to read in a constant of the required value and succeeds if the resulting value corresponds to ei. These procedures do not fail but set the control parameter nbread to the number of items actually recognized.

2. Note that the read procedures are based on the lexical analyser of Mosel: items are separated by spaces and a string that contains spaces must be quoted using either single or double quotes (the quotes are automatically removed once the string has been identified).

3. The procedure readln expects all the items to be recognized to be contained in single line. The function read ignores changes of line. If the procedure readln is used without parameters it skips the end of the current line.

Related topics
write, writeln.
**reset**

**Purpose**
Reset an external type object or problem.

**Synopsis**
procedure reset(x: external or problem)

**Argument**
- x A reference to an object of an external type or problem

**Further information**
Only types supporting the ‘copy’ operation (i.e. they can be assigned a value) can be reset. The effect of this routine depends on the type of the object, typically the object returns to its state just after being created. For instance, applying it to an mpproblem will clear the problem by detaching all constraints it contains.
reverse

Purpose
   Reverse a list.

Synopsis
   procedure reverse(l:list)

Argument
   l    A list

Example
   L:=[1,2,3,4,5]
   reverse(L)   ! => L=[5,4,3,2,1]
   reverse(L)   ! => L=[1,2,3,4,5]

Related topics
   getreverse.
**round**

**Purpose**
Round a number to the nearest integer.

**Synopsis**
function round(r:real):integer

**Argument**
r Real number to be rounded

**Return value**
Rounded value.

**Example**
In the following, i takes the value 6, j the value -7, and k the value 12:

```
i := round(5.5)
j := round(-6.7)
k := round(12.3)
```

**Related topics**
ceil, floor.
setcoeff

**Purpose**
Set the coefficient of a variable or the constant term.

**Synopsis**
procedure setcoeff(c:linctr, x:mpvar, r:real)
procedure setcoeff(c:linctr, r:real)

**Arguments**
c A linear constraint
x A decision variable
r Coefficient or constant term

**Example**
The following declares a constraint c and then changes some of its terms:

```
declarations
x,y,z: mpvar
end-declarations

c:= 4*x + y -z <= 12

setcoeff(c, y, 2)
setcoeff(c, 8.1)
```

The constraint is now \(4 \cdot x + 2 \cdot y - z \leq -8.1\).

**Further information**
If a variable is given then this procedure sets the coefficient of this variable in the constraint to the given value. Otherwise, it sets the constant term of the constraint.

**Related topics**
getcoeff.
sethidden

Purpose
Hide or unhide a constraint.

Synopsis
procedure sethidden(c:linctr, b:boolean)

Arguments
- c: A linear constraint
- b: Constraint status:
  - true: Hide the constraint
  - false: Unhide the constraint

Example
The following defines a constraint and then sets it as hidden:

```mosel
declarations
  x,y,z: mpvar
end-declarations

c := 4*x + y -z <= 12
sethidden(c, true)
```

Further information
At its creation a constraint is added to the current problem, but using this procedure it may be hidden. This means that the constraint will not be contained in the problem that is solved by the optimizer but it is not deleted from the definition of the problem in Mosel. Function istridden can be used to test the current status of a constraint.

Related topics
- istridden
Predefined functions and procedures

setioerr

**Purpose**
Raise an IO error.

**Synopsis**
procedure setioerr(msg:string)

**Argument**
msg Error message to display (or an empty string)

**Further information**
This function sets the control parameter `iostatus` (see `getparam`) such that an IO error is raised. If IO errors are not handled by the model (see `setparam`), the execution is interrupted.

**Related topics**
setmatherr.
**setmatherr**

**Purpose**
Raise a Math error.

**Synopsis**
```pascal
procedure setmatherr(msg: string)
```

**Argument**
- `msg` Error message to display (or an empty string)

**Further information**
If mathematical errors are not handled by the model (see `setparam`), the execution is interrupted. A function ending with a call to this routine may set its return value to `NAN` or `INFINITY` in order to indicate its error status.

**Related topics**
- `setioerr`, `isnan`, `isinf`, `isfinite`. 
setname

**Purpose**
Associate a matrix name to a constraint or variable.

**Synopsis**
procedure setname(c:linctr, n:string)
procedure setname(v:mpvar, n:string)

**Arguments**
c A linear constraint
v A decision variable
n Name given to the constraint or variable

**Further information**
1. When exporting a problem to a matrix file, constraint/variable names are deduced from the global public symbols: anonymous and local entities are usually named after their row/column number in the matrix. This procedure makes it possible to give a name to these entities.

2. If the given name starts with the '#' character, the generated matrix name will include the order number of the constraint or variable in the matrix.

**Related topics**
exportprob.
Predefined functions and procedures

setparam

Purpose
Set the value of a control parameter.

Synopsis
procedure setparam(name:string,val:integer|string|real|boolean)

Arguments
name Name of a control parameter (case insensitive).
val New value for the control parameter

Example
See example of function getparam.

Further information
1. Control parameters include the settings of Mosel as well as those of any loaded module. The module may be specified by prefixing the parameter name with the name of the module (e.g. mmxprs.XPRS_verbose). The type of the value must correspond to the type expected by the parameter.
2. This procedure can be applied only to control parameters the value of which can be modified.
3. The name argument must be a constant string: a model parameter, variable or string expression cannot be used as a control parameter name.
4. The following control parameters, supported by Mosel, can be altered with this procedure:
   realfmt Default C printing format for real numbers (string, default: "%.g")
   zerotol zero tolerance in comparisons between reals (real, default: 1.0e-13)
   ioctrl the interpreter ignores IO errors (Boolean, default: false)
   mathctrl the interpreter ignores Maths errors (Boolean, default: false)
   readcnt generate per label counting when executing ‘initializations from’ blocks (Boolean, default: false)
   UTC indicate whether the time functions return time expressed in local (false) or UTC (true) time (Boolean, default: false)
   autofinal indicate whether initialisation blocks are finalizing sets (Boolean, default: true or false if compiler option noautofinal is used)
   workdir specify the current working directory of the model (string, initialised with the current working directory of the Mosel instance). The provided value can be a relative path (e.g. "../../../somedir")
   recloc enable (or disable) automatic recording of source location of constraints definitions (Boolean, default: false). This parameter can be set to true only if the model has been compiled with option -G; it makes it possible the creation of meaningful constraint names when exporting a matrix (see exportprob)
   localedir directory where message catalogs are stored (string, default: "/locale")

Related topics
getparam.
setrandseed

**Purpose**
Initialize the random number generator.

**Synopsis**
procedure setrandseed(s:integer)

**Argument**
s Seed value

**Further information**
This procedure sets its argument as the seed for a new sequence of pseudo-random numbers to be returned by the function random.

**Related topics**
random.
settype

**Purpose**
Set the type of a constraint.

**Synopsis**
```
procedure settype(c:linctr, type:integer)
```

**Arguments**
- `c` A linear constraint
- `type` Constraint type

**Further information**
The type (`type`) of a linear constraint may be set to one of:
- **CT_EQ** Equality, ‘=’
- **CT_GEQ** Greater than or equal to, ‘≥’
- **CT_LEQ** Less than or equal to, ‘≤’
- **CT_UNB** Non-binding constraint
- **CT_SOS1** Special ordered set of type 1
- **CT_SOS2** Special ordered set of type 2

Values applicable for unary constraints only are:
- **CT_CONT** Continuous
- **CT_INT** Integer
- **CT_BIN** Binary
- **CT_PINT** Partial integer
- **CT_SEC** Semi-continuous
- **CT_SINT** Semi-continuous integer
- **CT_FREE** Free

**Related topics**
- `gettype`
Predefined functions and procedures

sin

**Purpose**
Get the sine of a value.

**Synopsis**
function sin(r:real):real

**Argument**
r Real number to which to apply the trigonometric function

**Return value**
Sine value of the argument.

**Related topics**
arctan, cos.
**splithead**

**Purpose**
Split a list returning the first elements.

**Synopsis**
function splithead(l:list, o:integer):list

**Arguments**
- l  A list
- o  Number of elements to remove if >0 or number of elements to keep if <0

**Return value**
The list of elements removed.

**Example**
L:=[1,2,3,4,5]
L2:=splithead(L,2)  ! => L=[3,4,5] L2=[1,2]

**Further information**
If the second parameter is 0, the list is unchanged and an empty list is returned. If the same parameter is larger than the size of the list, all elements are deleted and the function returns a copy of the original list.

**Related topics**
- splittail
slipttail

Purpose
Split a list returning the last elements.

Synopsis
function slipttail(l:list, o:integer):list

Arguments
l      A list
o      Number of elements to remove if >0 or number of elements to keep if <0

Return value
The list of elements removed.

Example
L:=[1,2,3,4,5]
L2:=slipttail(L,2)  ! => L=[1,2,3] L2=[4,5]
L2:=slipttail(L,-1) ! => L=[1] L2=[2,3]

Further information
If the second parameter is 0, the list is unchanged and an empty list is returned. If the same
parameter is larger than the size of the list, all elements are deleted and the function returns a
copy of the original list.

Related topics
slipthead
sqrt

**Purpose**
Get the positive square root of a value.

**Synopsis**
function sqrt(r:real):real

**Argument**
r  Real value the function is applied to. This value must be non-negative.

**Return value**
Square root of the argument.

**Related topics**
abs, exp, ln, log.
**strfmt**

**Purpose**
Create a formatted string from a string or a number.

**Synopsis**
```plaintext
function strfmt(str:string, len:integer):string
function strfmt(i:integer, len:integer):string
function strfmt(r:real, len:integer):string
function strfmt(r:real, len:integer, dec:integer):string
```

**Arguments**
- `str` String to be formatted
- `i` Integer to be formatted
- `r` Real to be formatted
- `len` Reserved length (may be exceeded if given string is longer, in this case the string is always left justified).
  - `<0` Left justified within reserved space
  - `>0` Right justified within reserved space
  - `0` Use defaults
- `dec` Number of digits after the decimal point

**Return value**
Formatted string.

**Example**
The following:
```plaintext
writeln("text1", strfmt("text2",8), "text3")
writeln("text1", strfmt("text2",-8), "text3")
r:=789.123456
writeln(strfmt(r,0)," ", strfmt(r,4,2), strfmt(r,8,0))
```
produces this output:
```plaintext
text1  text2text3
text1text2  text3
789.123 789.12 789
```

**Further information**
1. This function creates a formatted string from a string or an integer or real number. It can be used at any place where strings may be used. Its most likely use is for generating printed output (in combination with `write` and `writeln`).
2. If the resulting string is longer than the reserved space it is not cut but printed in its entirety, overflowing the reserved space to the right.

**Related topics**
`write`, `writeln`.
**Predefined functions and procedures**

**substr**

**Purpose**
Get a substring of a string.

**Synopsis**
function substr(str:string, i1:integer, i2:integer):string

**Arguments**
- **str** String
- **i1** Starting position of the substring
- **i2** End position of the substring

**Return value**
Substring of the given string.

**Example**
write(substr("Example text", 3, 10))

This outputs the text: ample te

**Further information**
This function returns the substring from the \(i_1^{th}\) to the \(i_2^{th}\) character of a given string (the counting starts from 1). This function returns an empty string if the bounds are not compatible with the string (e.g. starting position larger than the length of the string) or inconsistent (e.g. starting position after end position).
**timestamp**

**Purpose**  
Generate a timestamp by combining the current UTC date and time.

**Synopsis**  
function timestamp:real

**Return value**  
The number of seconds since 1/1/1970 at midnight as a real.

**Further information**

1. This function corresponds to the expression (using UTC time):
   real(currentdate)*86400+currenttime/1000

2. Refer to the module mmsystem for a set of dedicated types for handling date and time.

**Related topics**

currentTime, currentDate
unpublish

**Purpose**
Unpublish a symbol.

**Synopsis**
procedure unpublish(name:string)

**Argument**
name  Symbol to be removed from the dictionary

**Further information**
This procedure has the opposite effect of publish. If the given name does not correspond to a previously published symbol no operation is performed.

**Related topics**
publish.
write, writeln

Purpose
Send an expression or list of expressions to the active output stream.

Synopsis
procedure write(e1:expr[, e2:expr…])
procedure writeln
procedure writeln(e1:expr[, e2:expr…])

Argument
e1, e2,...   Expression or list of expressions

Example
The following lines

Set1:={"first", "second", "fifth"}
write(Set1)          ! Print set contents without return
writeln             ! Print an empty line
b:=true
writeln("A real:", strfmt(7.1234, 4, 2), ", a Boolean:",b)
                          ! Output followed by return

produce this output:

 {'first', 'second', 'fifth'}
A real:7.12, a Boolean:true

Further information
These procedures write the given expression or list of expressions to the active output stream. The procedure writeln adds the return character to the end of the output. Numbers may be formatted using function strfmt. Basic types are printed "as is". For elementary but non-basic types (linctr, mpvar) only the address is printed. If the expression is a set or array, all its elements are printed.

Related topics
fwrite, fwriteln, read, readln, strfmt.
II. Modules
CHAPTER 4

mmetc

This compatibility module just defines the diskdata procedure required to use data files formatted for mp-model from Mosel and provides a commercial discounting function. To use this module, the following line must be included in the header of the Mosel model file:

```mosel
uses 'mmetc'
```

4.1 Procedures and functions

- **disc**
  - Annual discount.
  - p. 155

- **diskdata**
  - Read in or write an array or set of strings to a file.
  - p. 156
disc

Purpose
Annual discount.

Synopsis
function disc(a:real, t:real)

Arguments
a Discount factor, real number greater than -1
 t Time, real number

Return value
Annual discount value: 1 / (1 + a)^t−1.

Further information
This function calculates the annual discount for the given period of time and discount factor.

Module
mmetc
diskdata

Purpose
Read in or write an array or set of strings to a file.

Synopsis
procedure diskdata(format:integer, file:string, a:array)
procedure diskdata(format:integer, file:string, s:set of string)

Arguments
format Format options:
ETC_DENSE dense data format
ETC_SPARSE sparse data format
ETC_SGLQ strings quoted with single quotes
ETC_NOQ strings are not quoted in the file
ETC_OUT write to a file
ETC_APPEND append output to the end of an existing file
ETC_TRANS tables are transposed
ETC_IN read from file (default)
ETC_NOZEROS skip zero values
Several options may be combined using ‘+’.

file Extended file name
a Array with elements of a basic type
s Set of strings

Example
The following example declares two sets and two dynamic arrays. The array ar1 is read in from
the file in.dat. Then both arrays, ar1 and ar2, are saved to the file out.dat (in sparse format) and
finally the contents of the set Set1 is appended to the file out.dat.

declarations
Set1: set of string
R: range
ar1,ar2: array(Set1,R) of real
end-declarations

diskdata(ETC_SPARSE, "in.dat", ar1)
diskdata(ETC_OUT, "out.dat", [ar1, ar2])
diskdata(ETC_OUT+ETC_APPEND, "out.dat", Set1)

Further information
1. This procedure reads in data from a file or writes to a file, depending on the parameter settings.
The file format used is compatible with the command DISKDATA of the modeler mp-model.

2. Only arrays of integer, real, string, mpvar and linctr (indexed by ranges or set of strings) can be
   used with this procedure.

Module mmetc
4.2 I/O drivers

This module provides the diskdata IO driver designed to be used as an interface for initializations blocks for both reading and writing files formatted for the diskdata procedure.

4.2.1 Driver diskdata

\[ \text{diskdata: [dense, sparse, sglq, noq, append, trans, nozeros]} \]

The driver can only be used in ‘initializations’ blocks. In the opening part of the block, no file name has to be provided, but general options can be stated at this point: they will be applied to all labels. In the block, each label entry is understood as the file name to use for the actual processing. Note that, before the file name, one can add further options separated by commas, that are effective to the particular entry. The file name given can use extended notation.

The diskdata driver takes the following options:

- dense: dense data format
- sparse: sparse data format
- sglq: strings quoted with single quotes
- noq: strings are not quoted in the file
- append: append output to the end of an existing file
- trans: tables are transposed
- nozeros: skip zero values

Example:

\begin{verbatim}
declarations
Set1: set of string ! Declare a set of strings
ar1, ar2: array(Set1, range) of real ! Declare two dynamic arrays
r: real ! Declare a real value
end-declarations

initializations from "diskdata:"
  ar1 as "sparse, ind.dat" ! Use 'diskdata' format for reading
  r as "r_init.dat" ! Read 'ar1' from 'in.dat' in sparse format
end-initializations

initializations to "diskdata:append"
  [ar1, ar2] as "out.dat" ! Use 'diskdata' format for output
  Set1 as "out.dat" ! Save two arrays in sparse format
end-initializations
\end{verbatim}
The module mmhttp makes it possible to communicate with external components via HTTP requests. Both modes, client or server side, can be used in a Mosel model: the client routines allow a Mosel model to send the HTTP requests GET, POST, PUT or DELETE to a web service. A model may also act as a web service by starting the integrated HTTP server. In this mode, the model gets notified about connections from remote clients via specific mmjobs events. The model can then reply to these requests using a set of dedicated routines.

To use this module, the following line must be included in the header of the Mosel model file:

```mosel
uses 'mmhttp'
```

### 5.1 Control parameters

The following parameters are defined by mmhttp:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>http_async</code></td>
<td>HTTP requests processing mode.</td>
<td>159</td>
</tr>
<tr>
<td><code>http_browser</code></td>
<td>Path to the web browser.</td>
<td>159</td>
</tr>
<tr>
<td><code>http_defpage</code></td>
<td>Default page of the web server.</td>
<td>159</td>
</tr>
<tr>
<td><code>http_defport</code></td>
<td>Default server TCP port.</td>
<td>160</td>
</tr>
<tr>
<td><code>http_expire</code></td>
<td>Expiration delay of open connections.</td>
<td>160</td>
</tr>
<tr>
<td><code>http_keephdr</code></td>
<td>Whether to keep HTTP headers in results.</td>
<td>160</td>
</tr>
<tr>
<td><code>http_listen</code></td>
<td>Interface used by server.</td>
<td>161</td>
</tr>
<tr>
<td><code>http_maxasync</code></td>
<td>Maximum number of pending asynchronous requests.</td>
<td>162</td>
</tr>
<tr>
<td><code>http_maxconn</code></td>
<td>Maximum number of open connections.</td>
<td>161</td>
</tr>
<tr>
<td><code>http_maxcontime</code></td>
<td>Maximum time for a connection.</td>
<td>161</td>
</tr>
<tr>
<td><code>http_maxreq</code></td>
<td>Maximum number of waiting connections.</td>
<td>162</td>
</tr>
<tr>
<td><code>http_maxreqtime</code></td>
<td>Maximum time for a connection.</td>
<td>162</td>
</tr>
<tr>
<td><code>http_port</code></td>
<td>Server TCP port.</td>
<td>163</td>
</tr>
<tr>
<td><code>http_proxy</code></td>
<td>Proxy address.</td>
<td>163</td>
</tr>
<tr>
<td><code>http_proxyport</code></td>
<td>Proxy TCP port.</td>
<td>163</td>
</tr>
<tr>
<td><code>http_srvconfig</code></td>
<td>Server options.</td>
<td>163</td>
</tr>
</tbody>
</table>
http_async

Description  This parameter selects the processing mode of the HTTP request functions (httpget|put|post|del). These functions return immediately after the connection to the server has been established (without waiting for the reply by the server) when this parameter is set to true. The model is notified about the completion of the request via an event of class EVENT_HTTPEND.

Type       Boolean, read/write
Default value   false
Affects routines   httpdel, httpget, httppost, httpput.
See also     http_maxasync.
Module     mmhttp

http_browser

Description  The path to a web browser to be executed when the parameter http_startwb is active.

Type       String, read/write
Values     Path to a web browser
Affects routines   httpstartsrv.
See also     http_startwb.
Module     mmhttp

http_defpage

Description  The default page is selected when the server receives a request not specifying any path (e.g. "http://server/*").

Type       String, read/write
Values     The label to be used as the default page. Selecting an empty string restores the default value
Default value   "index.html"
Affects routines   httpstartsrv.
Module     mmhttp
### http_defport

**Description**  
This is the port number used by the web server upon its startup. If this parameter is 0, the port number is selected automatically (the actual port number can be retrieved through parameter `http_port`).

**Type**  
Integer, read/write

**Default value**  
0

**Affects routines**  
`httpstartsrv`

**See also**  
`http_port`

**Module**  
`mmhttp`

### http_expire

**Description**  
Connections held in the connection pool are automatically closed if they are not used for more than the amount of time (in seconds) defined by this parameter.

**Type**  
Integer, read/write

**Values**  
Between 5 and 60 by steps of 5 seconds

**Default value**  
5

**See also**  
`http_maxconn`

**Module**  
`mmhttp`

### http_keephdr

**Description**  
By default results of HTTP queries do not include the HTTP header lines. This parameter can be used to retrieve these header lines in addition to the result document (use `httpgetheader` to separate the header from the effective result document).

**Type**  
Boolean, read/write

**Default value**  
false

**Affects routines**  
`httpdel, httpget, httppost, httpput`

**Module**  
`mmhttp`
### http_listen

**Description**
This is the interface used by the web server upon its startup. The default value implies binding to all available interfaces.

**Type**
String, read/write

**Default value**
0.0.0.0

**Affects routines**
httpstartsrv.

**Module**
mmhttp

### http_maxconn

**Description**
This parameter defines the size of the connection pool: whenever an HTTP request is emitted mmhttp tries to use one of the already open connections. After the end of the operation the connection is saved into the pool (if the server supports this functionality). Setting this parameter to 0 disables the pool (i.e. each query is executed on a new connection). When this parameter is changed all connections of the pool are closed.

**Type**
Integer, read/write

**Values**
Between 0 and 8

**Default value**
4

**See also**
http_expire.

**Module**
mmhttp

### http_maxcontime

**Description**
Maximum amount of time (in seconds) allowed for connecting to a HTTP server and send a request. If the operation is longer than the specified duration the request is cancelled and the procedure results in an IO error. A value of 0 disables this time limit.

**Type**
Integer, read/write

**Default value**
0

**See also**
http_maxreqtime.

**Affects routines**
httpdel, httpget, httppost, httpput.

**Module**
mmhttp
http_maxreq

Description: The maximum number of active concurrent connections the server is maintaining. Above this limit, connections are rejected and clients are notified with the HTTP code 500.

Type: Integer, read/write

Values: At least 1

Default value: 16

Affects routines: httpstartsrv.

Module: mmhttp

http_maxreqtime

Description: Maximum amount of time (in seconds) allowed for processing a request. If the operation is longer than the specified duration the request is cancelled and the procedure results in an IO error. A value of 0 disables this time limit.

Type: Integer, read/write

Default value: 0

See also: http_maxcontime.

Affects routines: httpdel, httpget, httppost, httpput.

Module: mmhttp

http_maxasync

Description: This parameter defines the maximum number of active asynchronous requests that can be sent by a Mosel model. Setting this parameter also resets http_async.

Type: Integer, read/write

Values: Between 4 and 56

Default value: 8

Affects routines: httpdel, httpget, httppost, httpput.

See also: http_async.

Module: mmhttp
### http_port

**Description**  
This parameter reports the port number currently used by the web server.

**Type**  
Integer, read only

**Affects routines**  
http_startsrv.

**See also**  
http_defport.

**Module**  
mmhttp

---

### http_proxy

**Description**  
When this parameter is defined, HTTP connections are sent through this proxy server (instead of establishing direct connections).

**Type**  
Integer, read/write

**Default value**  
"

**Affects routines**  
httpdel, httpget, httppost, httpput.

**See also**  
http_proxyport.

**Module**  
mmhttp

---

### http_proxyport

**Description**  
The value of this parameter corresponds to the connection port of the proxy server (when defined).

**Type**  
Integer, read/write

**Default value**  
80

**Affects routines**  
httpdel, httpget, httppost, httpput.

**See also**  
http_proxy.

**Module**  
mmhttp

---

### http_srvconfig

**Description**  
This parameter specifies which request types are accepted by the HTTP server started from a Mosel model. For instance, if the application will only process HTTP GET queries the value of this parameter should be HTTP_GET. Moreover, if the flag HTTP_SSL is set, the server will also listen for HTTPS connections and, if the flag HTTP_SSLONLY is used, only
the HTTPS server will be started (i.e. normal HTTP queries will be rejected). When an HTTPS server is started, the flag HTTP_CLTAUTH enables client authentication: clients are accepted only if they present a known certificate.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Default value</th>
<th>Affects routines</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_CLTAUTH</td>
<td>Integer, read/write</td>
<td>HTTP_DELETE+HTTP_GET+HTTP_POST+HTTP_PUT</td>
<td>httpstartsrv.</td>
<td>mmhttp</td>
</tr>
</tbody>
</table>

### http_startwb

**Description**
If this parameter is true a web browser pointing to the default page is launched just after the web server starts.

<table>
<thead>
<tr>
<th>Type</th>
<th>Boolean, read/write</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>false</td>
</tr>
<tr>
<td>Affects routines</td>
<td>httpstartsrv.</td>
</tr>
<tr>
<td>See also</td>
<td>http_browser.</td>
</tr>
<tr>
<td>Module</td>
<td>mmhttp</td>
</tr>
</tbody>
</table>

### https_defport

**Description**
This is the port number used by the web secure server upon its startup. If this parameter is 0, the port number is selected automatically (the actual port number can be retrieved through parameter https_port).

<table>
<thead>
<tr>
<th>Type</th>
<th>Integer, read/write</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default value</td>
<td>0</td>
</tr>
<tr>
<td>Affects routines</td>
<td>httpstartsrv.</td>
</tr>
<tr>
<td>See also</td>
<td>https_port.</td>
</tr>
<tr>
<td>Module</td>
<td>mmhttp</td>
</tr>
</tbody>
</table>

### https_listen

**Description**
This is the interface used by the secure web server upon its startup. The default value implies binding to all available interfaces.

<table>
<thead>
<tr>
<th>Type</th>
<th>String, read/write</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default value</td>
<td>0.0.0.0</td>
</tr>
<tr>
<td>Affects routines</td>
<td>httpstartsrv.</td>
</tr>
<tr>
<td>Module</td>
<td>mmhttp</td>
</tr>
</tbody>
</table>


**https_port**

**Description**  
This parameter reports the port number currently used by the secure web server.

**Type**  
Integer, read only

**Affects routines**  
httpstartsrv.

**See also**  
https_defport.

**Module**  
mmhttp

### 5.2 Procedures and functions

#### 5.2.1 HTTP client

The HTTP requests GET, HEAD, POST, PUT and DELETE can be sent to a web service using functions `httpget`, `httphead`, `httppost`, `httpput` and `httpdel` respectively. Each of these functions takes at least two parameters: the URL of the resource and a file name where to store the result of the operation. POST and PUT requests require an additional file, namely the data source to be sent to the web service.

HTTP requests are either processed synchronously or asynchronously.

When a request is sent in **synchronous mode** (the default), the HTTP function call returns after the processing has completed and the return value corresponds to the status of the request (a successful request will have a status value between 200 and 299). The following example uses www.bing.com to search for 'FICO' using a synchronous request:

```pascal
status:=httpget("http://www.bing.com/search?q=FICO", "result.html")
if status div 100=2 then
  writeln("Found FICO!")
else
  writeln("Request failed with code ", status, " (", httpreason(status), ")")
end-if
```

If the **asynchronous mode** is active (that is, the parameter `http_async` is set to `true`) the HTTP functions return just after the request has been sent, without waiting for the reply by the server. The processing continues in a separate thread of execution (up to `http_maxasync` requests can be handled at the same time) and the function returns a request identifier (or an error code in case of failure during the connection phase). Once the request has completed (i.e. the server has replied) an event of class `EVENT_HTTPEND` is raised (please refer to the documentation of the module `mmjobs` for further explanation on how to handle events). The associated value of this event is `request_id+status/1000`. For instance if request number 1300 succeeded with status 204 (‘no data’) the corresponding event value is 1300.204. An asynchronous request can be cancelled using `httpcancel`: in this case an event is still generated but its status is 998.

In the following example, search for ‘FICO’ is sent to BING, Yahoo and Ask at the same time. A loop is then started to wait for answers from each of the search engines.

```pascal
setparam("http_async",true)  ! Switch to asynchronous mode
reqyahoo:=httpget("http://us.search.yahoo.com/search/?p=FICO", "resyahoo.html")
writeln("Request ", reqyahoo, " sent to Yahoo")
writeln("Request ", reqbing, " sent to BING")
writeln("Request ", reqask, " sent to Ask")
```
if reqbing<1000 or reqyahoo<1000 or reqask<1000 then  
  writeln("A request has failed!")
else
  nbdone:=0
  repeat
    wait ! Wait for an event
    evt:=getnextevent
    if evt.class = EVENT_HTTPEND then ! One of the requests completed
      reqnum:=integer(evt.value) ! Get request number
      write("Request ", reqnum, " done: ")
      status:=integer((evt.value-reqnum)*1000) ! Get HTTP status
      if status div 100=2 then ! 200<=status<300 is success
        writeln("Found FICO!")
      else ! Any other value is an error code
        writeln("Failed with code ", status, " ", httpreason(status), ")")
    end-if
    nbdone+=1
  end-if
  wait ! Finished when all requests are done
until nbdone=3

By default the module performs direct TCP connections to the servers but a proxy may be specified using the http_proxy and http_proxyport parameters.

It is possible to set a limit on the time spent for connecting to a server by using http_maxcontime. The parameter http_maxreqtime defines a time limit on the entire request (i.e. connection and retrieval of result). mmhttp will wait undefinitely for each request if none of these parameters is defined.

When requests are sent to a secure server (i.e. URL starting with "https://") the trusted certificates file https_cacerts must be available such that authenticity of servers can be verified. If this verification is not required, the control parameter https_trustsrv has to be set to true. If the requested secure server requires client authentication, client certificate https_cltcrt and associated private key https_cltkey must be defined. Note that these parameters are published by mmssl: this module has to be used when a secure requests have to be sent.

HTTP client functions:

- **httpcancel** Cancel an asynchronous request. p. 167
- **httpdel** Perform an HTTP DELETE request. p. 168
- **httpget** Perform an HTTP GET request. p. 169
- **httpgetheader** Extract the HTTP header of a result file. p. 175
- **httphead** Perform an HTTP HEAD request. p. 170
- **httppost** Perform an HTTP POST request. p. 171
- **httpput** Perform an HTTP PUT request. p. 172
- **httpreason** Generate the text representation of an HTTP status code. p. 173
- **urlencode** Encode a text string for a URL. p. 174
httpcancel

Purpose
Cancel an asynchronous request.

Synopsis
procedure httpcancel(id: integer)

Argument
id    Number of the request to cancel

Further information
This procedure has no effect if the request number cannot be found (e.g. the request has completed in the meantime). If the request is effectively cancelled an event of class EVENT_HTTPEND is raised with a request status of value 998.

Related topics
httppost, httpput, httpget, httphead, httpdel.

Module
mmhttp
httpdel

**Purpose**
Perform an HTTP DELETE request.

**Synopsis**
function httpdel(url:string|text, result:string):integer
function httpdel(url:string|text, result:string, xhdr:string|text):integer

**Arguments**
- **url** URL to process
- **result** File to store the result of the request
- **xhdr** Additional headers to add to the request

**Return value**
HTTP status of the request (e.g. 200 for success, see HTTP standard specifications for a full list; value 999 indicates that an I/O error occurred during the operation) or the request number (≥ 1000) if the asynchronous mode is active.

**Further information**
The function returns after the request has been processed when synchronous mode is active (see http_async). Otherwise, using asynchronous mode, the function returns immediately after having sent the request and the model is notified about the completion of the operation by an event of class EVENT_HTTPEND. In this mode the result file result must be a physical file (although drivers "tmp:" and "null:" can still be used).

**Related topics**
- httppost, httpput, httpget, httphead, httpcancel.

**Module**
mmhttp
httpget

Purpose
Perform an HTTP GET request.

Synopsis
function httpget(url:string|text, result:string):integer
function httpget(url:string|text, result:string, xhdr:string|text):integer

Arguments
url URL to process
result File to store the result of the request
xhdr Additional headers to add to the request

Return value
HTTP status of the request (e.g. 200 for success, see HTTP standard specifications for a full list; value 999 indicates that an I/O error occurred during the operation) or the request number (≥ 1000) if asynchronous mode is active

Example
Retrieve the default entry page of the FICO website in French and store it in the file "fico.html":

status:=httpget('http://www.fico.com/fr/Pages/default.aspx', 'fico.html')

Further information
1. The function returns after the request has been processed when synchronous mode is active (see http_async). Otherwise, using asynchronous mode, the function returns immediately after having sent the request and the model is notified about the completion of the operation by an event of class EVENT_HTTPEND. In this mode the result file result must be a physical file (although drivers "tmp:" and "null:" can still be used).

2. When building a query it is important to encode data to be sent using urlencode

3. By default the header "Accept-Encoding: gzip" is inserted into the request and the result data is automatically decompressed if the server supports compression. This behaviour is disabled if this optional header is already specified (e.g. the parameter xhdr includes "Accept-Encoding: identity").

Related topics
httppost, httpput, httpdel, httphead, urlencode, httpcancel.

Module
mmhttp
**httphead**

**Purpose**
Perform an HTTP HEAD request.

**Synopsis**
```plaintext
function httphead(url:string|text, result:string):integer
function httphead(url:string|text, result:string, xhdr:string|text):integer
```

**Arguments**
- **url** URL to process
- **result** File to store the result of the request
- **xhdr** Additional headers to add to the request

**Return value**
HTTP status of the request (e.g. 200 for success, see HTTP standard specifications for a full list; value 999 indicates that an I/O error occurred during the operation) or the request number (≥ 1000) if asynchronous mode is active

**Further information**
The HEAD request is equivalent to a GET request except that no result is returned by the server, only the header can be retrieved (see httpgetheader).

**Related topics**
- `httppost`, `httpput`, `httpdel`, `httpget`, `urlencode`, `httpcancel`.

**Module**
- `mmhttp`
httppost

**Purpose**
Perform an HTTP POST request.

**Synopsis**

```plaintext
function httppost(url:string|text, data:string, result:string):integer
function httppost(url:string|text, data:string, result:string, xhdr:string|text):integer
```

**Arguments**
- **url**: URL to process
- **data**: Data file to be sent to the server
- **result**: File to store the result of the request
- **xhdr**: Additional headers to add to the request

**Return value**
HTTP status of the request (e.g. 200 for success, see HTTP standard specifications for a full list; value 999 indicates that an I/O error occurred during the operation) or the request number (≥ 1000) if asynchronous mode is active

**Further information**

1. The function returns after the request has been processed when synchronous mode is active (see http_async). Otherwise, using asynchronous mode, the function returns immediately after having sent the request and the model is notified about the completion of the operation by an event of class EVENT_HTTPEND. In this mode the result file result must be a physical file (although drivers "tmp:" and "null:" can still be used).

2. The parameter xhdr is typically used when the data type has to be specified. For instance, when the data sent is URL-encoded it may be necessary to use "Content-Type: application/x-www-form-urlencoded" as the value for xhdr in order to indicate to the server how to decode and process the data.

3. By default the header "Accept-Encoding: gzip" is inserted into the request and the result data is automatically decompressed if the server supports compression. This behaviour is disabled if this optional header is already specified (e.g. the parameter xhdr includes "Accept-Encoding: identity").

**Related topics**

httpget, httphead, httpput, httpdel, urlencode, httpcancel.

**Module**

mmhttp
httpput

Purpose
Perform an HTTP PUT request.

Synopsis
function httpput(url:string|text, data:string, result:string):integer
function httpput(url:string|text, data:string, result:string, xhdr:string|text):integer

Arguments
url URL to process
data Data file to be sent to the server
result File to store the result of the request
xhdr Additional headers to add to the request

Return value
HTTP status of the request (e.g. 200 for success, see HTTP standard specifications for a full list; value 999 indicates that an I/O error occurred during the operation) or the request number (≥ 1000) if asynchronous mode is active

Further information
1. The function returns after the request has been processed when synchronous mode is active (see http_async). Otherwise, using asynchronous mode, the function returns immediately after having sent the request and the model is notified about the completion of the operation by an event of class EVENT_HTTPEND. In this mode the result file result must be a physical file (although drivers "tmp:" and "null:" can still be used).

2. The parameter xhdr is typically used when the data type has to be specified. For instance, when the data sent is URL-encoded it may be necessary to use "Content-Type: application/x-www-form-urlencoded" as the value for xhdr in order to indicate to the server how to decode and process this data.

Related topics
httpget, httphead, httppost, httpdel, urlencode, httpcancel.

Module
mmhttp
httpreason

Purpose
Generate the text representation of an HTTP status code.

Synopsis
function httpreason(code: integer): string

Argument
code   HTTP status code

Return value
Text associated to the provided status code or an empty string if the code is unknown

Example
The following displays "Bad Request":

    writeln(httpreason(400))

Further information
The HTTP standard specifies a set of predefined status codes. This function returns the text associated with a given code. For instance, upon success a request will reply with code 200 ("OK") or 204 ("No Content").

Module
mmhttp
mmhttp

urlencode
Purpose
Encode a text string for a URL.
Synopsis
function urlencode(data:string|text):text
Argument
data

Text to encode

Return value
Encoded text suitable for building a URL
Example
The following request sends query "qry" to the server "srv" requiring parameters "a" and "b".
The values associated with these parameters are URL-encoded:
status:=httpget("http://srv/qry?a="+urlencode(a)+
"&b="+urlencode(b), "result.txt")
Further information
1. This function converts a text string into a format that is compatible with URL conventions. The
conversion consists in replacing characters with a special meaning by a portable representation
based on the character code. For example, the character "&" is replaced by "%26".
2. Typically, query parameters have to be encoded when sending them via an HTTP GET request,
data sent via POST may also have to be encoded.
Module
mmhttp

Fair Isaac Corporation Confidential and Proprietary Information

174


**httpgetheader**

**Purpose**
Extract the HTTP header of a result file.

**Synopsis**

```plaintext
function httpgetheader(sfile:string|text):text
function httpgetheader(sfile:string|text, dfile:string|text):text
```

**Arguments**
- **sfile** Name of the file to process
- **dfile** Destination file (can be the same as **sfile**)

**Return value**
Header of the result document

**Further information**
1. Result files of queries include the HTTP header when the parameter `http.keephdr` is set to `true`: this function returns the header of a result file when this setting is active.
2. The optional destination file **dfile** receives a copy of the original result file after the header has been removed.

**Related topics**
- `httpget`, `httppost`, `httpput`, `httpdel`.

**Module**
- `mmhttp`
5.2.2 HTTP server

The `mmhttp` module integrates an HTTP server that is started using the procedure `httpstartsrv` and stopped with `httpstopsrv` (the server is stopped in any case when the execution of the model terminates). The server behaviour may be changed using these module parameters: `http_defport` defines the TCP port on which the server is listening (by default a random port is selected); `http_defpage` indicates which page or label the server has to consider when no path is specified in a request (by default this is "index.html"); `http_srvconfig` defines the set of request types supported by the model (for instance only GET and POST) as well as whether a secure server is to be started; `http_maxreq` sets a limit on the number of simultaneous connections that are kept active.

When a secure server (HTTPS) is requested (the server config includes flags `HTTP_SSL` or `HTTP_SSLONLY`) besides the optional basic settings similar to those used for the standard server (like `https_defport`) additional parameters have to be set. The server certificate `https_srvcert` as well as its private key `https_srvkey` are required. Moreover, if the clients are requested to authenticate themselves (server option `HTTP_CLTAUTH`), the authorised certificate file `https_cacerts` must include the expected certificates. Note that these parameters are published by `mmssl`: this module has to be used when a secure server is started.

The server runs in the background and notifies the model of incoming connections through events of class `EVENT_HTTPNEW` (please refer to the documentation of `mmjobs` for further explanation on how to handle events). The value associated with this event is a request number: the connection to the client is kept open and the model has to reply to the request in order to complete the operation. Any data associated with the incoming request (query in the case of a GET or data sent via POST or PUT) is saved into a temporary file before the event is sent. URL encoded information is automatically decoded and converted to a format compatible with initialisations from blocks. The function `httppending` may also be used to retrieve the list of requests currently waiting for a reply.

Request properties can be obtained through a set of dedicated routines: `httpreqfrom` is the IP address of the client; `httpreqtype` is the request type (i.e. GET, POST, PUT or DELETE); `httpreqheader` is the request header; `httpreqstat` reports the status associated to a request number (for instance whether it is active, or has associated data); `httpreqlabel` is the label of the request. The label of a request is its URL after having removed server reference and the query data (for example, the label returned for "http://srv/some/path?a=10" is "some/path"); `httpreqfile` is the name of the temporary file holding data associated to the request.

Three different methods can be used to reply to a request: `httpreplycode` will only return a status code associated with a short (error) message; `httpreply` takes as input a file to be sent back to the client (with a success status code) and `httpreplyjson` converts its input parameter into JSON data that is sent back to the client.

The following example shows how to implement a simple file server. This program expects GET (download a file) and PUT (upload a file) requests sent to the port 2533. The URI of the request is interpreted as the file name: for example, the URL "http://srv:2533/myfile.txt" could be used to access file "myfile.txt" stored on host "srv" running this example.

```plaintext
setparam("http_defport", 2533) ! Set server port (2533)
setparam("http_srvconfig", HTTP_GET+HTTP_PUT) ! Only GET and PUT requests
setparam("workdir", getparam("tmpdir")) ! Move to temporary directory
httpstartsrv ! Server now running
repeat
wait
  ev:=getnextevent
  if ev.class=EVENT_HTTPNEW then ! Request pending
    r:=integer(ev.value) ! Get request ID
    fname:=httpreqlabel(r)
    if httpreqtype(r)=HTTP_GET then ! Client wants to get the file
      if bittest(getfstat(fname), SYS_TYP) = SYS_REG then
        ! Set server port (2533)
        ! Only GET and PUT requests
        ! Move to temporary directory
        ! Server now running
        ! Wait for an event
        ! Request pending
        ! Get request ID
        ! File name will be the URI
        ! Client wants to get the file
```
httpreply(r,fname) ! If available: send it
else
httpreplycode(r,404) ! Otherwise: reply "Not Found"
end-if
elif httpreqtype(r)=HTTP_PUT and
httpreqstat(r)>=2 then
fmove(httpreqfile(r), fname)
if getsysstat=0 then
httpreplycode(r,204) ! If success: reply "No Content"
else
httpreplycode(r,403) ! Otherwise: reply "Forbidden"
else
httpreplycode(r,400) ! Empty files are refused
end-if
end-if
until false

HTTP server functions:

httppending Get a list of requests waiting for a reply. p. 178
httpreply Reply to an HTTP request with a file. p. 179
httpreplycode Reply to an HTTP request only with a status code. p. 180
httpreplyjson Reply to an HTTP request with JSON data. p. 181
httpreqfile Get the data file associated to a request. p. 182
httpreqfrom Get the IP address of the sender of a request. p. 183
httpreqheader Get the header associated to a request. p. 184
httpreqlabel Get the label associated to a request. p. 185
httpreqstat Get the status associated with a request. p. 186
httpreqtype Get the type of a request. p. 187
httpstartsrv Start the HTTP server. p. 188
httpstopsrv Stop the HTTP server. p. 189
jsonwrite Generate a JSON representation of a Mosel entity. p. 190
httppending

Purpose
Get a list of requests waiting for a reply.

Synopsis
function httppending(lp:list of integer):integer
function httppending:integer

Argument
lp List of request numbers

Return value
Number of requests in the waiting queue

Further information
This function returns in lp the list of requests currently waiting for a reply in the server queue (the content of the list is replaced).

Module
mmhttp
httpreply

Purpose
Reply to an HTTP request with a file.

Synopsis
procedure httpreply(reqid:integer)
procedure httpreply(reqid:integer, fname:string)
procedure httpreply(reqid:integer, fname:string|text, xhdr:string|text)

Arguments
reqid Request number
fname Name of the file holding the response data
xhdr Additional headers to include in the response

Further information
1. This procedure replies to the specified request sending the provided file and using 200 ('OK') as the HTTP status code.
2. The first form of the procedure is the same as providing an empty file name with the second form: in this case no data is sent to the client and the returned status code becomes 204 ('No Content').
3. If the specified request is of type HEAD (see httphead) this procedure sends only the header part of the result.

Related topics
httpreplycode, httpreplyjson.

Module
mmhttp
Purpose
Reply to an HTTP request only with a status code.

Synopsis
procedure httpreplycode(reqid:integer, code:integer)
procedure httpreplycode(reqid:integer, code:integer, msg:string)
procedure httpreplycode(reqid:integer, code:integer, msg:string, xhdr:string|text)

Arguments
reqid  Request number
code   HTTP status code to be returned
msg    Explanation text
xhdr   Additional headers to include in the response

Further information
1. This procedure replies to the specified request using the provided code that should be a valid HTTP status (i.e. 3 digit number).
2. Unless the provided code is 204 (No Content) a basic HTML page is generated as the data associated to the response including the standard reason (e.g. Bad Request for code 400) as well as the given explanation text.
3. If the specified request is of type HEAD (see httphead) this procedure sends only the header part of the result.

Related topics
httpreply, httpreplyjson.

Module
mmhttp
**httpreplyjson**

**Purpose**

Reply to an HTTP request with JSON data.

**Synopsis**

```plaintext
procedure httpreplyjson(reqid:integer)
procedure httpreplyjson(reqid:integer, mosobj:*)
```

**Arguments**

- `reqid` Request number
- `mosobj` Mosel object to use for the reply

**Further information**

1. This procedure replies to the specified request by sending the provided Mosel object encoded as a JSON object.
2. When the first form is used, the returned data is the JSON constant `null`.
3. If the specified request is of type HEAD (see `httphead`) this procedure sends only the header part of the result.

**Related topics**

`httpreply`, `httpreplycode`.

**Module**

`mmhttp`
httpreqfile

**Purpose**
Get the data file associated to a request.

**Synopsis**
function httpreqfile(reqid:integer):string

**Argument**
reqid Request number

**Return value**
Full path to the data file

**Further information**
1. Each request is associated with a data file located in the temporary directory. This function returns the full path to this file.
2. The data file is specific to the given request number and can be used (for instance, to store the response to the request) even if no data is associated with the request.

**Module**
mmhttp
httpreqfrom

Purpose
Get the IP address of the sender of a request.

Synopsis
function httpreqfrom(reqid:integer):string

Argument
reqid Request number

Return value
IP of the sender of the request as a string

Related topics
httpreqtype, httpreqfile, httpreqstat, httpreqlabel, httpreqheader.

Module
mmhttp
httpreqheader

Purpose
Get the header associated to a request.

Synopsis
function httpreqheader(reqid:integer):text

Argument
reqid Request number

Return value
Header of the request

Further information
The header of the request is a block of text consisting of lines of the form
fieldname:value (e.g. Content-Type: application/json).

Related topics
httpreqfrom, httpreqfile, httpreqstat, httpreqlabel, httpreqtype.

Module
mmhttp
httpreqlabel

**Purpose**
Get the label associated to a request.

**Synopsis**
function httpreqlabel(reqid:integer):string

**Argument**
reqid  Request number

**Return value**
Label of the request

**Further information**
The *label* of the request is the URL after having removed server reference and query data (for instance the label returned for "http://srv/some/path?a=10" is "some/path").

**Related topics**
httpreqfrom, httpreqfile, httpreqstat, httpreqtype, httpreqheader.

**Module**
mmhttp
httpreqstat

**Purpose**
Get the status associated with a request.

**Synopsis**
function httpreqstat(reqid:integer):integer

**Argument**
reqid Request number

**Return value**
Request status:
<0 Invalid request number
0 Request not active
1 No associated data
2 Raw data available
3 ‘initialisations from’ data available

**Further information**
If the return value is 2 or 3 a data file is available (see httpreqfile). If this function returns 3 then the file can be read using an `initialisations from` block: data that was originally URL-encoded has been decoded by the server and stored using Mosel’s `initialisations` format.

**Related topics**
httpreqfrom, httpreqfile, httpreqlabel, httpreqtype, httpreqheader.

**Module**
mhttp
**httpreqtype**

**Purpose**
Get the type of a request.

**Synopsis**
function httpreqtype(reqid:integer):integer

**Argument**
reqid Request number

**Return value**
- Request type:
  - <0 Invalid request number
  - 0 Request not active
  - HTTP_GET GET (1)
  - HTTP_POST POST (2)
  - HTTP_PUT PUT (4)
  - HTTP_DELETE DELETE (8)
  - HTTP_HEAD HEAD (128)

**Related topics**
- httpreqfrom, httpreqfile, httpreqstat, httpreqlabel, httpreqheader.

**Module**
mmhttp
httpstartsrv

**Purpose**
Start the HTTP server.

**Synopsis**
```
procedure httpstartsrv
procedure httpstartsrv(srvdir:string, moslab:string)
```

**Arguments**
- `srvdir` Server directory
- `moslab` Label identifying commands

**Further information**
2. Only one server can be run by a model: if the server is already running, no operation is performed.
3. The server processes only authorised request types (see `http_srvconfig`): the model is notified of every valid request by an event of class `EVENT_HTTPNEW`. Malformed or unauthorised requests are automatically rejected.
4. When the function is used with arguments, `srvdir` designates a directory: `mmhttp` will act as a file server for the files stored in this directory (via `GET` queries). The argument `moslab` is a prefix that identifies requests that are to be handled by the model.
5. An IO error is raised if the server cannot start because of a network setting (typically the TCP port is already used or requires higher privileges).
6. If the parameter `http_startwb` is set to `true` a web browser (as defined by `http_browser`) is launched just after the server has started.

**Related topics**
- `httpstopsrv`

**Module**
- `mmhttp`
httpstopsrv

**Purpose**
Stop the HTTP server.

**Synopsis**
procedure httpstopsrv

**Further information**
1. This procedure has no effect if no server is running.
2. During its shutting down procedure the server closes all waiting requests (with a response code 410) such that it is no longer possible for the model to reply to these requests (however, events corresponding to these requests may still be in the event queue).

**Related topics**
httpstartsrv.

**Module**
mmhttp
**jsonwrite**

**Purpose**
Generate a JSON representation of a Mosel entity.

**Synopsis**
```
procedure jsonwrite(fname:string, mosobj:*)
```

**Arguments**
- `fname` Name of a file to store the generated text
- `mosobj` A Mosel object

**Further information**
This procedure generates a JSON representation of a Mosel entity. If the file name is an empty string, the generated string is sent to the current output stream (by default this is the console).

**Module**
`mmhttp`
5.3 I/O drivers

The mmhttp module publishes the url driver with which a URL can be used as a file. Thanks to this facility it is possible to use files stored on an HTTP enabled file server just as if they were located on the local file system. For example, the following command downloads and executes the Mosel file "hello.mos" stored on the web server mysrv:

    > mosel exec mmhttp.url:http://mysrv/hello.mos

5.3.1 Driver url

url:URL

The file name for this driver is a URL. Currently only HTTP URLs are supported (i.e. the name must begin with "http://"). The behaviour of the driver depends on the file operation:

- **reading**: A GET request is sent to the specified URL at the time of opening the file. The following read operations are executed directly from the result stream generated by the server.

- **writing**: The written data is first saved into a temporary file and then sent to the specified URL via a PUT request when the file is closed.

- **deleting**: When deleting a file (e.g., using fdelete) through this driver a DELETE request is sent to the specified URL.
The *mmjava* module for Mosel is intended for users who integrate their Mosel models into Java applications. This module can only be used from a Java enabled application.

### 6.1 I/O drivers

This module provides the *java* and *jraw* IO drivers. The first one can be used to link a Mosel output (input) stream to a Java OutputStream (InputStream) or a Java ByteBuffer. The second driver is a modified version of the *raw* driver suitable for Java: instead of an address, this driver takes as input a reference to an object.

For both drivers, file names are replaced by references to objects. These references are of two kinds: direct references to public static objects (e.g. "java.lang.System.out") and names defined using the XPRM.bind method. The second technique will be used with non static objects: the method XPRM.bind establishes a link between a name and an object. This name can then be used as an object reference for *mmjava* drivers.

When using Java object from Mosel, it is important to make sure objects and related fields can be accessed: in particular the class and its fields must be public.

#### 6.1.1 Driver *java*

```
java:[rewind,]static object|named object
```

With this driver a Java stream (OutputStream or InputStream) as well as a ByteBuffer can be used in place of a file in Mosel. This facility is specially useful for redirecting default Mosel streams to Java objects. Note that the Mosel Java interface uses this driver for redirecting default streams (in, out, and error) to the corresponding Java streams (System.in, System.out and System.err). When the file is open for reading and the referenced object is a ByteBuffer, the option *rewind* can be used in order to rewind the buffer before starting to read.

Example:

```java
mosel=new XPRM();
mosel.bind("out", myout);    /* Associate 'myout' object with string "out" */
/* Redirect default output to 'myout' */
mosel.setDefaultStream(XPRM.F_OUTPUT|XPRM.F_LINBUF, "java:out")
/* Redirect error stream to Java output stream */
mosel.setDefaultStream(XPRM.F_ERROR, "java:java.lang.System.out"
```

If the driver is used in an initializations from block (resp. initializations to block) and the provided object implements interface XPRMInitializationFrom (resp. XPRMInitializationTo) then the corresponding Java methods are used to process the initialization (refer to the Mosel Library JavaDoc for further explanation).
This driver supports the delete operation: deleting a Java file name from the Mosel code (e.g. fdelete("java:out")) corresponds to executing unbind on the corresponding identifier. The operator first tries to unbind the identifier associated to the running model (XPMMModel.unbind) and then uses the global reference (XP.unbind) if the first attempt fails.

6.1.2 Driver jraw

\texttt{jraw: [noindex, all]}

The driver can only be used in 'initializations' blocks. In the opening part of the block, no file name has to be provided, but general options can be stated at this point: they will be applied to all labels. Two options are supported:

- **all**: forces output of all cells of an array even if it is dynamic (by default only existing cells are considered).
- **noindex**: indicates that only data (no indices) are transferred between the Java objects and Mosel. By default, the first fields of each object are interpreted as index values for the array to be transferred. This behavior is changed by this option.

In the block, each label entry is understood as an object reference to use for the actual processing. Note that, before the object reference, one can add further options separated by commas, that are effective to the particular entry.

If the Model object to be initialized (or saved) is a scalar or an array with option noindex, the driver expects a Java object of a corresponding type (i.e. same basic type and scalar or one dimension array). If the option noindex is not used and the Mosel object is an array, the label must specify which fields of the class have to be taken into account for the mapping. This is indicated by a list of field names separated by commas and noted in brackets (e.g. "myobj(f1,f2,f3)").

In the following example the jraw driver is used to initialize an array of reals, \texttt{a}, and an array of integers, \texttt{ia}, with data held in the Java application that executes the model.

Java part:

```java
public class MyData { /* A class to store an 'array(string, int) of real' */
    public String s; public int r; public double v;
    MyData(String i1, int i2, double v0) { s=i1; r=i2; v=v0; }
}
...
MyData[] data;
int[] intarr;
...
mosel=new XPMM();
mosel.bind("data", data); /* Associate 'data' object with string "data" */
mosel.bind("ia", intarr); /* Associate 'intarr' object with string "ia" */
```

Mosel part:

```mosel
declarations
a:array(string, range) of real
ia:array(range) of integer
end-declarations
...
initializations from "jraw:"
aa as "data(s,r,v)" ! Initialize 'aa' with fields s,r,v of object 'data'
ia as "noindex,ia" ! Initialize 'ia' with array 'ia'; no index (only values)
end-initializations
```
Thanks to this module it is possible to load several models in memory and execute them concurrently. In addition, other instances of Mosel might be started (either locally to the running system or remotely on another machine through the network) and used to run additional models controlled by the model that has started them. This means that the computing capacity of the running model is not restricted to the executing process. A general synchronization mechanism based on event queues as well as two specialized IO drivers are also provided in order to ease the implementation of parallel algorithms in Mosel.

To use this module, the following line must be included in the header of the Mosel model file:

```
uses 'mmjobs'
```

### 7.1 Example

The following example shows how to compile, load, and then run a model from another model. After having started the execution, it waits for 60 seconds before stopping the secondary model if the latter has not yet finished.

```
model "mmjobs example"
uses "mmjobs","mmsystem"

declarations
mymod: Model
event: Event
end-declarations

! Compile 'mymod.mos' to memory
if compile("","mymod.mos","shmem:bim")<>0
then
  exit(1)
end-if

load(mymod,"shmem:bim") ! Load bim file from memory...
ffdelete("shmem:bim") ! ... and release the memory block

! Disable model output
setdefstream(mymod,"","null:","null:")
run(mymod) ! Start execution and
wait(60) ! wait 1 min for an event

if waitexpired then ! No event has been sent...
  writeln("Model too long: stopping it!"
stop(mymod) ! ... stop the model then wait
wait
end-if

! An event is available: model finished
event:=getnextevent
writeln("Exit status: ", getvalue(event))
```
writeln("Exit code : ", getexitcode(mymod))
unload(mymod)
end-model

## 7.2 Control parameters

The following parameters are defined by *mmjobs*:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>conntmpl</td>
<td>Default connection template.</td>
<td>mmjobs</td>
</tr>
<tr>
<td>defaultnode</td>
<td>Default node number used by driver 'rmt'.</td>
<td>mmjobs</td>
</tr>
<tr>
<td>fsrvdelay</td>
<td>Maximum wait time for findxsrvs.</td>
<td>mmjobs</td>
</tr>
<tr>
<td>fsrvnbiter</td>
<td>Number of iterations performed by findxsrvs.</td>
<td>mmjobs</td>
</tr>
<tr>
<td>fsrvport</td>
<td>UDP port used by findxsrvs.</td>
<td>mmjobs</td>
</tr>
<tr>
<td>jobid</td>
<td>ID of the current model.</td>
<td>mmjobs</td>
</tr>
<tr>
<td>keepalive</td>
<td>Keepalive timer setting.</td>
<td>mmjobs</td>
</tr>
<tr>
<td>nodenumber</td>
<td>ID of the current instance.</td>
<td>mmjobs</td>
</tr>
<tr>
<td>parentnumber</td>
<td>ID of the parent of the current instance.</td>
<td>mmjobs</td>
</tr>
<tr>
<td>sshcmd</td>
<td>SSH command for xssh driver.</td>
<td>mmjobs</td>
</tr>
</tbody>
</table>

### conntmpl

**Description**

The connection template is used by the `connect` function to generate a valid host specification from an identifier (typically corresponding to a host name). The generation is performed by replacing in the template each occurrence of the `%h` marker by the original identifier.

**Type**

String, read/write

**Values**

A string containing "%h" at least once

**Default value**

"xsrv:%h"

**Affects routines**

connect.

**Module**

mmjobs

### nodenumber

**Description**

The ID (or node number) of the current instance as returned by the function `getid`. The ID of the initial (root) instance is 0

**Type**

Integer, read only

**Module**

mmjobs
**defaultnode**

Description: This parameter is used by the IO driver "rmt:" when it is not given any node reference (see Section 7.4.6). By default its value is 0 (the initial node) but it may be changed by a parent model using the instance parameter defaultnode (see Annex B).

Type: Integer, read only

Module: mmjobs

**jobid**

Description: The ID of the current model as returned by the function getid. The ID of the initial (root) model is 0

Type: Integer, read only

Module: mmjobs

**parentnumber**

Description: The ID (or node number) of the parent (i.e., creator) of the current instance. The ID of the initial instance is 0 and its parent is -1.

Type: Integer, read only

Module: mmjobs

**keepalive**

Description: When using a Mosel remote instance (see connect), the server sends to its client a keepalive message at fixed interval. A connection is considered broken if more than maxfail*interval seconds have elapsed since the last message received. Setting 0 for maxfail disables this mechanism. This parameter can only be changed if no remote Mosel instance is connected.

Type: String, read/write

Values: A string of the form "maxfail/interval"

Default value: "2/60"

Module: mmjobs
**fsrvport**

**Description**
This parameter defines the UDP port to be used by the `findxsrvs` routine for its broadcast messages.

**Type**
Integer, read/write

**Default value**
2514

**Affects routines**
`findxsrvs`.

**See also**
`fsrvnbiter`, `fsrvdelay`.

**Module**
mjjobs

---

**fsrvdelay**

**Description**
After it has sent its broadcast message, the `findxsrvs` routine waits for up to `fsrvdelay` milliseconds for answers before aborting.

**Type**
Integer, read/write

**Default value**
1000

**Affects routines**
`findxsrvs`.

**See also**
`fsrvnbiter`, `fsrvport`.

**Module**
mjjobs

---

**fsrvnbiter**

**Description**
This control parameter specifies the number of times the procedure `findxsrvs` sends a broadcast message.

**Type**
Integer, read/write

**Default value**
1

**Affects routines**
`findxsrvs`.

**See also**
`fsrvdelay`, `fsrvport`.

**Module**
mjjobs
### sshcmd

**Description**

When connecting to a remote host via the "xssh:" I/O driver, an external program is used to establish the SSH tunnel: this parameter specifies which program to use. The arguments of the program are identified with the symbol "%h" for the target host, "%p" for the TCP port and "%f" for the known host file (which is "-" when no file is provided). Note that this control parameter is read-only when Mosel is running under restriction NoExec (see Section 1.3.3).

**Type**

String, read/write

**Affects routines**

connect.

**Values**

A string including at least "%h"

**Default value**

"xprmsrv -sshclt %h -p %p -kh %f"

**Module**

mmjobs

---

### 7.3 Procedures and functions

#### 7.3.1 Mosel instance management

The type *Mosel* is used to reference a Mosel instance. Before an instance can execute commands (like loading or running a model), it must be *connected*. Connecting an instance consists in starting an additional operating system process running Mosel: this is done by the `connect` function. To improve readability of the model source, one can use host aliases (defined by means of the `sethostalias` routine) to designate connection targets. Once work with a particular instance has been finished, the instance can be disconnected (`disconnect`): this terminates the process running Mosel (and releases all associated resources).

- `clearaliases` Delete all defined aliases.  
  [p. 201]
- `connect` Connect a Mosel instance.  
  [p. 199]
- `disconnect` Disconnect a Mosel instance.  
  [p. 200]
- `findxsrvs` Search xprmsrv servers on the local network.  
  [p. 206]
- `getaliases` Retrieve the list of all defined aliases.  
  [p. 204]
- `getbanner` Get the banner displayed by an instance on startup.  
  [p. 202]
- `gethostalias` Get the value of a host alias.  
  [p. 203]
- `sethostalias` Define a host alias.  
  [p. 205]
connect

Purpose
Connect a Mosel instance.

Synopsis
function connect(mi:Mosel, host:string):integer

Arguments
mi The instance to connect
host A host specification

Return value
0 if successful, a positive value otherwise

Example
Start instance inst1 on a separate process:

   r:=connect(inst1,"")

With default settings, the 2 following statements are equivalent:

   r:=connect(inst2,"ariane")
   r:=connect(inst3,"xsrv:ariane")

Further information
1. Any Mosel instance has to be connected before it can be used for executing commands.

2. If the host provided is an empty string (""), it is replaced by "rcmd:" (instance started on the same machine in a separate process). Otherwise, the string host is searched in the list of defined aliases (see sethostalias) and, if found, it is replaced by the associated text. If the resulting specification does not contain any IO driver reference, a valid specification is generated using the current connection template (see conntmpl): each occurrence of the %h marker in the template is replaced by the value of host.

3. The host argument (or the string resulting from the transformations described above) is expected to be an extended file name using an IO driver the task of which is to start a process running the mosel program in remote mode and create/manage the communication streams between the processes. The mmjobs module provides three drivers supporting this service (see Section 7.4): "rcmd:" to start a Mosel instance on a separate process on the same machine, "xsrv:" to start a Mosel instance on a host running the Mosel Remote Launcher (see Section 7.5) and "xssh:" to use a secure connection with an xprmsrv server.

Related topics
sethostalias, findxsrvs, disconnect.

Module
mmjobs
disconnect

**Purpose**
Disconnect a Mosel instance.

**Synopsis**
procedure disconnect(mi:Mosel)

**Argument**
mi The instance to disconnect

**Further information**
This routine should be used to terminate a Mosel instance started by connect.

**Related topics**
connect.

**Module**
mmjobs
clearaliases

Purpose
Delete all defined aliases.

Synopsis
procedure clearaliases

Further information
This routine deletes all host aliases previously defined by sethostalias.

Related topics
sethostalias, getaliases, gethostalias, connect.

Module
mmjobs
**getbanner**

**Purpose**
Get the banner displayed by an instance on startup.

**Synopsis**
```plaintext
function getbanner(mi:Mosel):string
```

**Argument**
- `mi` A connected instance

**Return value**
The text displayed by Mosel when it started the instance

**Further information**
When a new instance is started, the text displayed by Mosel is saved (this includes typically copyright notice and version information): this function returns this startup banner.

**Related topics**
- `connect`

**Module**
- `mmjobs`
gethostalias

**Purpose**
Get the value of a host alias.

**Synopsis**
function gethostalias(alias:string):string

**Argument**
alias   Internal identifier

**Return value**
The host specification corresponding to the alias or an empty string if the alias is not defined

**Related topics**
sethostalias, clearaliases, getaliases, connect.

**Module**
mmjobs
getaliases

**Purpose**
Retrieve the list of all defined aliases.

**Synopsis**
procedure getaliases(aliases:list of string)

**Argument**
aliases A list to return the aliases

**Example**
The following procedure displays all aliases:

```plaintext
procedure showaliases
  declarations
  l:list of string
  end-declarations
  getaliases(l)
  forall(h in l)
    writeln(h,"->",gethostalias(h))
end-procedure
```

**Further information**
This procedure resets its aliases argument.

**Related topics**
sethostalias, clearaliases, gethostalias, connect.

**Module**
mmjobs
sethostalias

Purpose
Define a host alias.

Synopsis
procedure sethostalias(alias:string,host:string)

Arguments
alias Internal identifier
host Corresponding host specification

Example
The first statement defines "localhost" as a separate process on the same machine and "win" for a remote access to the machine "winpc":

    sethostalias("localhost","rcmd:"
    sethostalias("win","xsrv:winpc")

Further information
Host aliases are used by connect to start Mosel instances. If the argument host is the empty string, the corresponding alias is removed from the list (or nothing is done if the alias was not defined before).

Related topics
gethostalias,clearaliases,getaliases,connect.

Module
mmjobs
findxsrvs

Purpose
Search xprmsrv servers on the local network.

Synopsis
procedure findxsrvs(group: integer, maxip: integer, addrs: set of string)

Arguments
- group: Group number of the request
- maxip: Maximum number of addresses to collect
- addrs: Set to store the addresses found

Further information
1. This procedure sends a broadcast message over the local network and waits for replies from running xprmsrv servers (see Section 7.5). A given server will reply only to selected group numbers: the group argument specifies this property.

2. The IP addresses of the hosts having replied to the request are returned via the last argument of the procedure in the form of strings. The maximum size of this set is fixed by maxip. Note that the provided set is not cleared: if it already contains maxip elements the routine returns immediately.

3. Control parameters fsrvnbiter, fsrvdelay and fsrvport can be used to tune the behaviour of findxsrvs. This routine repeats fsrvnbiter times the following procedure: it sends a broadcast message to the fsrvport UDP port and then waits for up to fsrvdelay milliseconds for replies.

Related topics
connect, disconnect.

Module
mmjobs
7.3.2 Model management

The type Model is used to reference a Mosel model. This section describes the procedures and functions available for model management: compilation of source model files, loading of bim files, execution and retrieval of model information. Note that before it can be used, a model has to be initialized by loading a bim file (load).

- **compile**: Compile a source model. p. 208
- **detach**: Detach the current model from its parent node. p. 210
- **getannidents**: Get model identifiers for which annotations are available. p. 230
- **getannotations**: Get model annotations associated to a given symbol. p. 231
- **getdsoprop**: Get module information. p. 218
- **getexitcode**: Get the exit code of a model. p. 226
- **getgid**: Get the group ID of a model. p. 219
- **getid**: Get the ID of a model or Mosel instance. p. 220
- **getmodprop**: Get model information. p. 221
- **getnode**: Get the ID (node number) of the Mosel instance of a model. p. 222
- **getrmtid**: Get the ID of a model on a remote instance. p. 223
- **getstatus**: Get the status of a model. p. 224
- **getuid**: Get the user ID of a model. p. 225
- **load**: Load a Binary Model file. p. 228
- **reset**: Reset a model. p. 229
- **resetmodpar**: Remove a parameter from a model parameter string. p. 213
- **run**: Run a model. p. 217
- **setcontrol**: Set an instance control parameter on a remote instance. p. 214
- **setdefstream**: Set default input/output streams of a model. p. 212
- **setmodpar**: Add or change the value of a parameter in a model parameter string. p. 215
- **setworkdir**: Set the initial working directory of a model. p. 216
- **stop**: Stop a running model. p. 217
- **unload**: Unload a model. p. 229
compile

**Purpose**
Compile a source model.

**Synopsis**

```plaintext
function compile(src:string):integer
function compile(opt:string, src:string):integer
function compile(opt:string, src:string, dst: string):integer
function compile(mi:Mosel, opt:string, src:string, dst: string):integer
```

**Arguments**

- **opt**  
  Compilation options (may be separated by spaces or '-' symbols):
  - "g"  Include debugging information
  - "G"  Include tracing information
  - "s"  Strip symbols
  - "p"  parse only: stop after the syntax analysis of the source file, do not compile (no file generated)
  - "bx=prefix"  Package prefix (can be quoted with single or double quotes)
  - "ix=prefix"  Include source prefix (can be quoted with single or double quotes)
  - "S"  Sign the bim file
  - "E"  Encrypt the bim file
  - "F"  The argument pass is a file name (not the password itself)
  - "Y"  Accept to load signed packages only if their signature can be verified
  - "T"  Accept to load only signed packages with a valid signature

- **src**  Source file name
- **dst**  Destination file name
- **com**  Comment to store in the bim file
- **mi**  The Mosel instance to perform the compilation
- **pass**  Password or password file (for encryption with a password)
- **pke**  Private key file (for bim file signing)
- **kls**  File of public keys (for encryption with public keys)

**Return value**

- 0  Function executed successfully
- 1  Parsing phase has failed (syntax error or file access error)
- 2  Error in compilation phase (a semantic error has been detected)
- 3  Error writing the output file
- 4  License error (compiler not authorized)

**Example**

Compile the local file "src.mos" stored on the current directory using the instance inst1 and store the resulting BIM file on the current directory of this instance:

```plaintext
r:=compile(ins1,"","rmt:src.mos","dst.bim")
```
Further information

1. This function compiles a given model source file into a binary model file (bim file) that is required as input to function load for executing the model.

2. If no destination file name is provided, the output file takes the same name as the source file with the extension .bim.

3. When sending a compilation request to a separate Mosel instance, it is important to keep in mind that the operation is performed in the environment of this instance (in particular its current working directory) and file names should be specified appropriately (the rmt: IO driver can be particularly helpful in this context).

4. The argument kls is a list of public key files (i.e. each line of the file is a key file name): when encrypting a file, the encryption is performed for each of the listed public keys such that the bim file can be decrypted by any of the corresponding private keys.

5. When prefixes provided via bx or ix are quoted with double quotes, backslashes are interpreted such that special characters can be included in the string. It is therefore required to double this symbol when it has to be included (e.g. 'bx="C:\mydir"').

Related topics

load.

Module

mmjobs
**detach**

**Purpose**
Detach the current model from its parent node.

**Synopsis**
procedure detach

**Further information**

1. This procedure *detaches* the model calling it from its parent model such that it becomes a *master* model running on a *root node*. As a consequence the connection to its parent model is closed, its model number is set to 0 and the node number of its instance becomes 0 (root node). The parent node is notified of the detachment by means of a termination event for the model which gets the status RT_DETACHED.

2. The operation is possible only if no file is open between the model and its parent (in particular the default streams have to be set to "null:" ) and the hosting instance is running only this model (*i.e.* no submodel is loaded).

3. After a model is detached, it can no longer communicate with its parent using events or access files through the "rmt:" driver. The HTTP protocol (available through the module *mmhttp*) might be used as an alternative to the facilities provided by *mmjobs* in this case.

4. The instance running the detached model terminates automatically after the end of execution of the model.

5. This routine can only be called by a submodel running on a remote instance. It has no effect if used by a master model.

**Related topics**
getstatus, run.

**Module**
*mmjobs*
load

**Purpose**
Load a Binary Model file.

**Synopsis**

- `procedure load(mo:Model, bimf:string)`
- `procedure load(mo:Model, bimf:string, opt:string, pwd:string, pke:string, kls:string)`
- `procedure load(mi:Mosel, mo:Model, bimf:string)`
- `procedure load(mi:Mosel, mo:Model, bimf:string, opt:string, pwd:string, pke:string, kls:string)`

**Arguments**

- `mo`: Model object to be initialized
- `bimf`: Bim file name
- `mi`: The instance on which the model will be run
- `opt`: Loading options (may be separated by spaces or '-' symbols):
  - "c": Check signature (if the file is signed)
  - "V": If the file is signed, load it only if the signature is valid
  - "T": Load only signed files with a valid signature
  - "F": The argument pass is a file name (not the password itself)
- `opt`: Options
- `pass`: Password or password file (for encrypted bim files)
- `pke`: Private key file (for encrypted bim files)
- `kls`: File of public keys

**Further information**

1. This procedure initializes the model `mo` with the bim file `bimf`. If `mo` has already been initialized, the model it references is unloaded before trying to load the new file (note that this operation fails if the model is running). If the file `bimf` cannot be accessed or one of the required modules cannot be loaded, the procedure generates an IO error (which may be intercepted if the control parameter `ioctrl` is true).

2. When loading a model from a separate Mosel instance, it is important to keep in mind that the operation is performed in the environment of this instance (in particular its current working directory) and file names should be specified appropriately (the `rmt`: IO driver can be particularly helpful in this context).

3. The argument `kls` is a list of public key files (i.e. each line of the file is a key file name): when a signed bim file is loaded, its signature is checked with the keys listed in this file. If this argument is not specified, the signing key is searched in the default public keys directory located at `getparam("ssl_dir")="/pubkeys"`.

**Related topics**
`compile`, `setdefstream`, `run`, `unload`.

**Module**
`mmjobs`
**setdefstream**

**Purpose**
Set default input/output streams of a model.

**Synopsis**
procedure setdefstream(mo:Model, wmd:integer, fname:string)
procedure setdefstream(mo:Model, input:string, output:string, error:string)
procedure setdefstream(mi:Mosel, wmd:integer, fname:string)
procedure setdefstream(mi:Mosel, input:string, output:string, error:string)

**Arguments**
mo A Model
mi A Mosel instance
wmd Stream to set. Possible values:
  F_INPUT Default input stream
  F_OUTPUT Default output stream
  F_ERROR Default error stream
  F_LINBUF Use line buffering
fname Extended file name to be used for the stream.
input Extended file name to be used for the input stream.
output Extended file name to be used for the output stream.
error Extended file name to be used for the error stream.

**Further information**
1. This function sets default IO streams to be used by a model. Model streams can be changed only when the model is not running. Each stream is associated to an extended file name (i.e. IO drivers can be used). For output streams, F_LINBUF may be specified (e.g. F_WRITE+F_LINBUF) in order to enable line buffering for the corresponding stream (the error stream is always open using line buffering).

2. For input and output streams, the filename is stored and streams are actually open when execution of the model starts: in case of an invalid file name, the error is not reported by this function. The error stream is immediately opened so in the case of an invalid file name it is detected by this function.

3. Using an empty string as the file name implies resetting to the original default stream.

4. When applied to a Mosel instance, this routine sets the default streams for this instance. These streams can only be changed if the instance has not yet loaded any model.

5. When using this routine on a separate Mosel instance or on a model loaded on a separate Mosel instance, it is important to keep in mind that the operation is performed in the environment of this instance (in particular its current working directory) and file names should be specified appropriately (the rmt: IO driver can be particularly helpful in this context).

**Module**
mmjobs
resetmodpar

Purpose
Remove a parameter from a model parameter string.

Synopsis
procedure resetmodpar(plist:text, pname:string)

Arguments
plist   Text object storing the parameters
pname   Parameter name

Further information
1. This function helps in building the model parameter string to be passed to the run procedure by removing a parameter definition (previously set with setmodpar) from a parameter string. The plist text is left unchanged if the requested parameter cannot be found.
2. It is expected that the provided text string is either empty or composed of a list of assignments of the form "pname=val,pname2=val2...".

Related topics
setmodpar, run.

Module
mmjobs
setcontrol

Purpose
Set an instance control parameter on a remote instance.

Synopsis
procedure setcontrol(mi:Mosel, ctrl:string, val:string)
procedure setcontrol(mo:Model, ctrl:string, val:string)

Arguments
mi      A Mosel instance
mo      A model reference (it must be loaded onto a remote instance)
ctrl    Control name
val     Control value

Further information
1. This procedure is used to change an instance control parameter in the context of the Remote Invocation Protocol (see Annex B).
2. An IO error is raised in case of error.

Module
mmjobs
setmodpar

**Purpose**
Add or change the value of a parameter in a model parameter string.

**Synopsis**
procedure setmodpar(plist:text, pname:string, val:integer|real|boolean|string|text)

**Arguments**
- **plist**  Text object storing the parameters
- **pname**  Parameter name
- **val**  Value assigned to the parameter.

**Further information**
1. This function helps in building the model parameter string to be passed to the run procedure. As input it takes a text object that it modifies by either adding an assignment of the form `pname=val` or by replacing an existing assignment. The routine adds the necessary quoting as necessary.
2. It is expected that the provided text string is either empty or composed of a list of assignments of the form "pname=val,pname2=val2...".

**Related topics**
resetmodpar, run.

**Module**
mmjobs
setworkdir

**Purpose**
Set the initial working directory of a model.

**Synopsis**
procedure setworkdir(mo:Model, cwd:string)

**Arguments**
- **mo**: A model reference
- **cwd**: Initial working directory

**Example**
The following statement sets the initial working directory of submodel `sub` to the current directory of its master model:

```plaintext
setworkdir(sub,'.')
```

**Further information**
1. This procedure defines the initial working directory to be used when the execution of the model (re)starts. As a consequence it cannot be used to change the environment of a running model.
2. For a local execution the provided path is expanded just before the beginning of the execution relatively to the current working directory of the caller. For a remote execution the path is relative to the directory of the instance running the model.

**Related topics**
- run.

**Module**
- mmjobs
run

Purpose
Run a model.

Synopsis
procedure run(mo:Model)
procedure run(mo:Model, plist:string|text)

Arguments
mo Model to be executed
plist String composed of model parameter initializations separated by commas

Further information
1. This procedure starts the execution of a model in a new thread: when the procedure returns, the model is not necessarily started (this may be delayed depending on the operating system load) and not necessarily terminated (the second model is executing concurrently to the caller).

2. By default the execution starts in the working directory of the Mosel instance (that might be different from the working directory of the calling model). A different initial path can be setup using setworkdir.

3. When the execution of the model is completed (normal termination, interruption after calling stop, or runtime error) or could not be started, an event of class EVENT_END is sent to the caller. The execution status is returned via the event value but it may also be obtained using getstatus. The exit code related to the last execution may be retrieved using getexitcode.

4. An event EVENT_END is also received after a model has detached itself although its execution may continue (see detach). In this case the model status is RT_DETACHED and its associated instance is disconnected.

5. The specified model must have been previously initialized with load and must not be running. If the same model has to be executed several times concurrently, it must be loaded several times in different model objects.

6. The parameter string plist may be built and modified using setmodpar and resetmodpar. These routines handle transparently the protection of parameter values by adding the appropriate quotes when required.

Related topics
load, wait, setmodpar, stop, getstatus, getexitcode, reset.

Module
mmjobs
**getdsoprop**

**Purpose**
Get module information.

**Synopsis**

```plaintext
function getdsoprop(dso:string, prop:integer):string
```

**Arguments**

- **dso**
  The name of a module currently loaded into memory

- **prop**
  The property to retrieve. Possible values:
  - `PROP_NAME`: Module name
  - `PROP_VERSION`: Module version
  - `PROP_PATH`: Path to the module file

**Return value**

The property as a string or an empty string in case of error (invalid property or the module was not found)

**Related topics**

- `getmodprop`

**Module**

- `mmjobs`
getgid

**Purpose**
Get the group ID of a model.

**Synopsis**
function getgid(mo:Model):integer

**Argument**
mo A model

**Return value**
Group ID of the model

**Further information**
A model can be associated with a group ID using setgid. This group ID may be used to identify the origin of an event (see getfromgid) or as a filter for a wait (see waitfor).

**Related topics**
getuid, getid, setgid

**Module**
mmjobs
getid

Purpose
Get the ID of a model or Mosel instance.

Synopsis
function getid(mo:Model):integer
function getid(mi:Mosel):integer

Arguments
mo A model
mi A Mosel instance

Return value
ID of the model or instance as an integer

Further information
1. Each model object has a unique ID number that can be obtained with this function. This ID may be used to identify the origin of an event (see getfromid) or as a filter for a wait (see waitfor).
2. The ID number of a Mosel instance is its node number. The initial instance has node number 0.

Related topics
getuid, getgid

Module
mmjobs
getmodprop

Purpose
Get model information.

Synopsis
function getmodprop(mo:Model, prop:integer):string
function getmodprop(prop: integer):string

Arguments
mo A model
prop The property to retrieve. Possible values:
PROP_NAME Model name (cf. model statement)
PROP_ID Order number
PROP_VERSION Model version
PROP_SYSCOM System comment
PROP_USRCOM User comment
PROP_SIZE Amount of memory (in bytes) used by the model
PROP_DATE Compilation date
PROP_UNAME Unique model name

Return value
The property as a string or an empty string in case of error

Further information
The second form of the function reports information for the calling model.

Related topics
getdsoprop

Module
mmjobs
**getnode**

**Purpose**
Get the ID (node number) of the Mosel instance of a model.

**Synopsis**

```plaintext
function getnode(mo:Model):integer
function getnode(mi:Mosel):integer
```

**Arguments**

- `mo` A model
- `mi` A Mosel instance

**Return value**

ID of the instance on which the model is loaded as an integer or -1 if the model has not been loaded.

**Further information**

1. This function returns the node number of the current instance if the provided model is local.
2. When applied to a Mosel instance this function returns the same information as `getid`.

**Module**

`mmjobs`
getrmtid

Purpose
Get the ID of a model on a remote instance.

Synopsis
function getrmtid(mo: Model): integer

Argument
mo A model

Return value
ID of the model on the remote instance as an integer or -1 if the model has not been loaded or is local to the running instance.

Further information
This ID corresponds to the model number assigned to the model by Mosel when it is loaded (i.e. the value of the control parameter modelnumber). This function can only be used on models handled by remote instances.

Module
mmjobs
getstatus

Purpose
Get the status of a model.

Synopsis
function getstatus(mo:Model):integer
function getstatus(mi:Mosel):integer

Argument
mo      A model

Return value
The status of a Mosel instance is 0 if it is connected, any other value indicates that it is not ready.
The model status can be:
RT_NOTINIT  Model has not been initialized or has been unloaded
RT_RUNNING  Model is running
RT_OK      Model is ready for execution and/or no error occurred during last execution
RT_MATHERR A mathematical error occurred
RT_ERROR   A runtime error occurred
RT_IOERR   An IO error occurred
RT_NULL    A NULL reference error occurred
RT_LICERR  Execution could not start because no license was available
RT_FDCLOSED Execution on a separate instance has been interrupted
RT_DETACHED Execution on a separate instance continues although the instance has been disconnected (see detach)
RT_STOP    Execution has been interrupted by a call to stop

Related topics
connect, stop, getexitcode.

Module
mmjobs
**getuid**

**Purpose**
Get the user ID of a model.

**Synopsis**

```haskell
function getuid(mo:Model):integer
```

**Argument**

- `mo`: A model

**Return value**

User ID of the model

**Further information**
A model can be associated with a user ID using `setuid`. This user ID may be used to identify the origin of an event (see `getfromuid`) or as a filter for a wait (see `waitfor`).

**Related topics**

- `getgid`, `getid`, `setuid`

**Module**

`mmjobs`
**getexitcode**

**Purpose**
Get the exit code of a model.

**Synopsis**
`function getexitcode(mo:Model):integer`

**Argument**
mo A model

**Return value**
Exit code of the last execution or 0

**Further information**
The exit code of the last execution corresponds to the value stated via a call to the procedure *exit*. The default exit value (*i.e.* procedure *exit* has not been called) is 0.

**Related topics**
getstatus.

**Module**
mmjobs
**stop**

**Purpose**
Stop a running model.

**Synopsis**
procedure stop(mo:Model)

**Argument**
mo Model to interrupt

**Further information**
If the model is not currently running, no operation is performed. Note that the effect of this call may not be immediate and the corresponding model may continue running a few seconds before its effective interruption (for instance the time required to complete an IO operation).

**Related topics**
run.

**Module**
mmjobs
reset

Purpose
Reset a model.

Synopsis
procedure reset(mo:Model)

Argument
mo Model to reset

Further information
This procedure resets a model after its execution: all resources it has allocated are released. The model returns to its state just after it has been loaded into memory. Note that this function is automatically called before a model is unloaded or run.

Related topics
run, unload.

Module
mmjobs
unload

**Purpose**
Unload a model.

**Synopsis**
procedure unload(mo:Model)

**Argument**
mo Model to unload

**Further information**
This procedure unloads the given model. All resources used by this model, including modules, are released. The function fails if the model is running.

**Related topics**
load.

**Module**
mmjobs
**getannidents**

**Purpose**
Get model identifiers for which annotations are available.

**Synopsis**
procedure getannidents(mo:Model, si:set of string)
procedure getannidents(si:set of string)

**Arguments**
- **mo** A model reference
- **si** Set receiving the identifiers

**Further information**
1. When used with a single argument this procedure returns information for the calling model.
2. This routine cannot be used with remote models.

**Related topics**
getannotations.

**Module**
mmjobs
getannotations

**Purpose**
Get model annotations associated to a given symbol.

**Synopsis**
```
procedure getannotations(mo:Model, id:string, prefix:string, si:set of string,
    ann:array(string) of string)
procedure getannotations(mo:Model, id:string, prefix:string, lsa:list of string)
procedure getannotations(id:string, prefix:string, si:set of string, ann:array(string)
    of string)
procedure getannotations(id:string, prefix:string, lsa:list of string)
```

**Arguments**
- `mo`: A model reference
- `id`: Symbol for which annotations are requested (an empty string will report global declarations)
- `prefix`: Prefix filter: only annotations with a name starting by the specified prefix will be returned
- `si`: Set receiving the annotation names
- `ann`: Array receiving the annotation values (indexed by names)
- `lsa`: List receiving the annotation names and values

**Further information**
1. With the version taking a list, each annotation is represented by 2 entries: the first one is the annotation name and the second one its value. Note that the version returning information via an array will only report partial information in the case of annotations defined several times.
2. When used without a model reference these procedures return information for the calling model.
3. These routines cannot be used with remote models.

**Related topics**
getannidents.

**Module**
mmjobs
7.3.3 Synchronization

Synchronization between running models can be implemented using events. Events are characterized by a class and a value and may be exchanged between a model and its parent model. The model from which an event has been sent is identified by its unique ID, its user ID and its group ID. An event queue is attached to each model to collect all events sent to this model and is managed with a FIFO policy (First In – First Out). Depending on the needs, a model may check whether its queue is empty or simply suspend its execution until it has been sent an event.

The type Event represents an event in the Mosel language. Objects of type Event may be compared with = or <> and assigned with :=. The function nullevent returns an event without class and value: this is the initial value of a newly created event and no model can send an event of this kind (i.e. the class is necessarily not null).

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>canceltimer</td>
<td>Cancel an active timer.</td>
<td>233</td>
</tr>
<tr>
<td>dropnextevent</td>
<td>Drop the next event in the event queue of the model.</td>
<td>243</td>
</tr>
<tr>
<td>getclass</td>
<td>Get the class of an event.</td>
<td>249</td>
</tr>
<tr>
<td>getfromgid</td>
<td>Get the group ID of the sender of an event.</td>
<td>247</td>
</tr>
<tr>
<td>getfromid</td>
<td>Get the ID of the sender of an event.</td>
<td>246</td>
</tr>
<tr>
<td>getfromuid</td>
<td>Get the user ID of the sender of an event.</td>
<td>248</td>
</tr>
<tr>
<td>getnextevent</td>
<td>Get the next event in the event queue of the model.</td>
<td>242</td>
</tr>
<tr>
<td>gettimer</td>
<td>Get the amount of time remaining before a timer expires.</td>
<td>250</td>
</tr>
<tr>
<td>getvalue</td>
<td>Get the value associated with an event.</td>
<td>251</td>
</tr>
<tr>
<td>isqueueempty</td>
<td>Check whether there are events waiting in the event queue.</td>
<td>244</td>
</tr>
<tr>
<td>nullevent</td>
<td>Return a 'null' event.</td>
<td>245</td>
</tr>
<tr>
<td>peeknextevent</td>
<td>Peek the next event in the event queue of the model.</td>
<td>252</td>
</tr>
<tr>
<td>send</td>
<td>Send an event to a running model.</td>
<td>234</td>
</tr>
<tr>
<td>setgid</td>
<td>Set the group ID of a model.</td>
<td>237</td>
</tr>
<tr>
<td>settimer</td>
<td>Create or update a timer.</td>
<td>235</td>
</tr>
<tr>
<td>setuid</td>
<td>Set the user ID of a model.</td>
<td>236</td>
</tr>
<tr>
<td>wait</td>
<td>Wait for an event.</td>
<td>238</td>
</tr>
<tr>
<td>waitexpired</td>
<td>Indicate whether the previous ‘wait’ or ‘waitfor’ expired.</td>
<td>239</td>
</tr>
<tr>
<td>waitfor</td>
<td>Wait for specific events.</td>
<td>240</td>
</tr>
</tbody>
</table>
canceltimer

Purpose
Cancel an active timer.

Synopsis
procedure canceltimer(tid:integer)

Argument
tid A timer identifier

Further information
1. This procedure has no effect if it cannot find the requested timer. However it will delete from the event queue the event EVENT_TIMER corresponding to a timer that is no longer active.

2. If the provided timer identifier tid is negative or null all timers are cancelled.

Related topics
settimer, gettimer.

Module
mmjobs
send

Purpose
Send an event to a running model.

Synopsis

procedure send(mo:Model, class:integer, value:real)
procedure send(class:integer, value:real)

Arguments

mo Model to send the event to
class Event class (must be >1)
value Event value

Further information

1. Events can be sent to models started by the caller (the child models) by using the first form of the procedure and to the model having started the caller (the parent model) with the second form of the procedure. An event can be received only by a running model using the mmjobs module: sending an event to a model that is not running or not using mmjobs is a no-operation.

2. Events are characterized by a class and a value. Event class values can be used to indicate the cause of the event (for instance, 2 could mean ‘a new solution has been found’) and the associated value may specify a property of the given instance (for example an objective value). Except for the special value 1 (EVENT_END) class values have no predefined meaning.

3. An event of class EVENT_END (=1) and model status as the event value is automatically sent by each model to its parent model when it terminates its execution.

Related topics
wait, waitfor.

Module
mmjobs
settimer

Purpose
Create or update a timer.

Synopsis
function settimer(tid:integer, dur:integer, rep:boolean):integer
function settimer(dur:integer, rep:boolean):integer

Arguments
- **tid**: A timer identifier
- **dur**: A duration in milliseconds
- **rep**: Decides whether the timer will be armed one time only or automatically repeated

Return value
Timer identifier as a positive integer

Further information
1. This function creates or updates an *interval timer*: after a timer has been armed by a call to this routine an event of class EVENT_TIMER is scheduled for being sent to the model after the specified amount of time has elapsed. The value of such an event is the timer identifier *tid*. Note that the system will not emit a new event if an identical event is already in the queue.

2. If the option *rep* is set to *false* the timer is released after its termination, otherwise it is immediately re-armed with the same interval after each expiration until it is explicitly cancelled (see canceltimer).

3. If the provided identifier *tid* is not positive a new timer is created with a newly generated identifier, this corresponds to the behaviour of the second form of this function.

4. When the provided identifier corresponds to an existing timer, this one is first cancelled with a call to canceltimer before being re-created with the new properties.

Related topics
canceltimer, gettimer.

Module
**mmjobs**
**setuid**

**Purpose**
Set the user ID of a model.

**Synopsis**
procedure setuid(mo:Model,uid:integer)

**Arguments**
- **mo** A model
- **uid** New user ID

**Further information**
This function defines the *user ID* associated to a model (by default it is 0). This user ID may be used to identify the origin of an event (see *getfromuid*).

**Related topics**
- setgid, getuid

**Module**
- mmjobs
**setgid**

**Purpose**
Set the group ID of a model.

**Synopsis**
```plaintext
procedure setgid(mo:Model, gid:integer)
```

**Arguments**

- **mo** A model
- **gid** New group ID

**Further information**
This function defines the *group ID* associated to a model (by default it is 0). This group ID may be used to identify the origin of an event (see `getfromgid`).

**Related topics**
`setuid, getgid`

**Module**
`mmjobs`
**wait**

**Purpose**
Wait for an event.

**Synopsis**

```plaintext
procedure wait
deco r procedure wait(dur:integer)
```

**Argument**
- **dur**  
  A duration in seconds

**Further information**
This procedure suspends the execution of the caller until an event is available. The second form specifies a time limit: the processing is suspended for at most `dur` seconds.

**Related topics**
- send, waitfor, waitexpired, isqueueempty, getnextevent, dropnextevent.

**Module**
- mmjobs
**waitexpired**

**Purpose**
Indicate whether the previous 'wait' or 'waitfor' expired.

**Synopsis**
function waitexpired: boolean

**Return value**
true if the last call to *wait* or *waitfor* terminated after expiration of a time limit

**Related topics**
*wait, waitfor.*

**Module**
*mmjobs*
waitfor

Purpose
Wait for specific events.

Synopsis
procedure waitfor(mask:integer)
procedure waitfor(mask:integer,dur:integer)
procedure waitfor(mask:integer,dur:integer,opt:integer)
procedure waitfor(mask:integer,id:integer,dur:integer,opt:integer)

Arguments
mask Bit mask of expected events
id ID of model for which events are expected
dur A duration in seconds, the constant WAIT_INFINITE or a timer identifier as a negative integer
opt Options:
   WAIT_EXACT Mask must be exactly matched
   WAIT_KEEP Keep unexpected events
   WAIT_UID Wait for a particular user ID
   WAIT_GID Wait for a particular group ID

Example
The following statement waits for an event of class 3 coming from a model of group 100 without dropping any event:

waitfor(3,100,WAIT_INFINITE,WAIT_KEEP+WAIT_EXACT+WAIT_GID)

Further information
1. This procedure suspends the execution of the caller until an event of a particular class is available. The second form specifies a time limit: the processing is suspended for at most \(\text{dur}\) seconds. If this duration is 0, the execution is not suspended but the queue of events is processed once. The special value WAIT_INFINITE is interpreted as an infinite duration.

2. The parameter \(\text{dur}\) may also take a negative value: in this case it is interpreted as the opposite of a timer identifier (see settimer) and the function will wait until this timer expires if no valid event arrives. When the routine interrupts its monitoring due to the expiration of a timer the first event in the queue is the event EVENT_TIMER associated to this timer. Note that if no timer corresponds to the given value the routine will terminate only when an expected event is available as if WAIT_INFINITE had been used.

3. By default, the parameter \(\text{mask}\) is interpreted as a bit mask to select the expected events: all events sent to the model are automatically dropped until an event \(\text{ev}\) satisfies the following condition:

\[ \text{bittest(getclass} (\text{ev}), \text{mask}) \neq 0 \]

4. If the parameter \(\text{opt}\) includes option WAIT_EXACT, the parameter \(\text{mask}\) becomes the target event class: the wait will end when an event of a class equal to \(\text{mask}\) is found.

5. If the parameter \(\text{opt}\) includes option WAIT_KEEP, unexpected events are not dropped but the first event satisfying the condition is moved to the top of the queue such that it is returned by the next call to getnextevent.

6. With the last form of the function an \(\text{ID}\) is specified: it characterises events coming from a particular model or a group of models. By default the argument \(\text{id}\) is interpreted as the unique model ID (see getid), if option WAIT_UID is used, the ID is interpreted as a user ID (see getuid) and with option WAIT_GID the argument is a group ID (see getgid).
related topics
send, wait, waitexpired, isqueueempty, getnextevent, dropnextevent.

module
mmjobs
getnextevent

Purpose
Get the next event in the event queue of the model.

Synopsis
function getnextevent: Event

Return value
The next event or nullevent if the queue is empty

Further information
The returned event is removed from the queue after it has been retrieved with this function.

Related topics
peeknextevent, dropnextevent, isqueueempty.

Module
mmjobs
dropnextevent

Purpose
Drop the next event in the event queue of the model.

Synopsis
procedure dropnextevent

Further information
This procedure has no effect if the event queue is empty.

Related topics
peeknextevent, getnextevent, isqueueempty.

Module
mmjobs
isqueueempty

**Purpose**
Check whether there are events waiting in the event queue.

**Synopsis**
function isqueueempty:boolean

**Return value**
false if at least one event is available in the queue, true otherwise.

**Related topics**
dropnextevent, peeknextevent, getnextevent.

**Module**
mmjobs
null event

**Purpose**
Return a ‘null’ event.

**Synopsis**
function nullevent:Event

**Return value**
An event of class and value equal to 0

**Further information**
Variables of type Event are initialized with this function.

**Related topics**
getnextevent.

**Module**
mmjobs
getfromid

Purpose
Get the ID of the sender of an event.

Synopsis
function getfromid(ev:Event):integer

Argument
   ev     An event

Return value
The ID of the sender of the event. 0 is returned for a nullevent

Further information
1. Each model has a unique ID that is attached to each event it sends. With this function one can identify the sender of a given event.
2. The ID of an event sent from the parent model is always 0.

Related topics
getid, getfromgid, getfromuid, getvalue, getclass.

Module
mmjobs
getfromgid

Purpose
Get the group ID of the sender of an event.

Synopsis
function getfromgid(ev:Event):integer

Argument
ev An event

Return value
The group ID of the sender of the event. 0 is returned for a nullevent.

Further information
1. Each model can be associated with a group ID that is attached to each event it sends. With this function one can identify the sender of a given event.
2. The group ID of an event sent from the parent model is always 0.

Related topics
getgid, setgid, getvalue, getfromid, getfromuid, getclass.

Module
mmjobs
getfromuid

Purpose
Get the user ID of the sender of an event.

Synopsis
function getfromuid(ev:Event):integer

Argument
    ev An event

Return value
The user ID of the sender of the event. 0 is returned for a nullevent.

Further information
1. Each model can be associated with a user ID that is attached to each event it sends. With this function one can identify the sender of a given event.
2. The user ID of an event sent from the parent model is always 0.

Related topics
getuid, setuid, getvalue, getfromid, getfromgid, getclass.

Module
mmjobs
**getclass**

**Purpose**
Get the class of an event.

**Synopsis**
function getclass(ev:Event):integer

**Argument**
ev An event

**Return value**
The class of the event (>0) or 0 for a nullevent

**Further information**
A model sends automatically an event of class EVENT_END (=1) when it terminates its processing. Other values are application specific.

**Related topics**
getvalue, getfromid, getfromgid, getfromuid.

**Module**
mmjobs
**gettimer**

**Purpose**
Get the amount of time remaining before a timer expires.

**Synopsis**

```plaintext
function gettimer(tid:integer):integer
```

**Argument**

| tid | A timer identifier |

**Return value**
Remaining time in milliseconds before the timer expires or 0 if the corresponding event is already available in the queue or -1 if no timer corresponds to the provided identifier.

**Further information**
This function will return 0 if an event corresponding to the specified timer is waiting in the queue of event even if this timer has been automatically re-armed.

**Related topics**

canceltimer, settimer.

**Module**

mmjobs
**getvalue**

**Purpose**
Get the value associated with an event.

**Synopsis**

```plaintext
function getvalue(ev:Event):real
```

**Argument**

- `ev`  
  An event

**Return value**

The value of the event

**Further information**

In the case of an event of class `EVENT_END(=1)`, this value corresponds to the model status.

**Related topics**

`getclass, getfromid, getfromgid, getfromuid`

**Module**

`mmjobs`
**peeknextevent**

**Purpose**
Peek the next event in the event queue of the model.

**Synopsis**

```
function peeknextevent:Event
```

**Return value**
A copy of the next event or `nullevent` if the queue is empty

**Further information**
The returned event is a copy of the first available event of the queue. The event queue is not changed by this function.

**Related topics**
`getnextevent`, `dropnextevent`, `isqueueempty`.

**Module**
`mmjobs`
7.4 I/O drivers

The *mmjobs* module provides a modified version of the *mem* IO driver designed to be used in a multithreaded environment: memory blocks allocated by the *shmem* IO driver are persistent (i.e. they are not released after the model terminates) and can be used by several models. Thanks to this facility, models running concurrently may exchange data through memory by means of initialization blocks for instance.

The driver *mempipe* offers another communication mechanism between models: a *memory pipe* may be open by two models simultaneously. One of them for writing and the other one for reading. This driver also supports initialization blocks through which data is transferred in binary form.

The drivers *rcmd*, *xsrv* and *xssh* allow to start additional Mosel instances: they have to be used to build host specifications as expected by the *connect* function. Finally, thanks to the *rmt* driver a Mosel instance can access files available from the environment of another instance.

7.4.1 Driver *shmem*

```
shmem:label[/minsize[/incstep]]
```

The file name for this driver is a *label*: this is the identifier (the first character must be a letter) of the memory block. A label is not local to a particular model and remains valid after the end of the execution of the model having created it. All memory blocks are released when the module *mmjobs* is unloaded but a given memory block may also be deleted explicitly by calling the *fdelete* procedure of module *mmsystem* or by using the *fremove* C-function of the Native Interface. Note also that deleting the special file "shmem:*" has the effect of releasing all memory blocks handled by the driver.

Several models may open a given label at the same time and several read operations may be performed concurrently. However, writing to a memory block can be done by only one model at a time: if several models try to read and write from/to the same label, only one (it becomes the *owner* of the memory block) performs its IO operations for writing and the others are suspended until the owner closes its file descriptor to the specified label. Then, one of the waiting models is restarted and becomes the new owner: this process continues until all file descriptors to the label are closed.

The memory block is allocated dynamically and resized as necessary. By default the size of the memory block is increased by pages of 4 kilobytes: the optional parameter *incstep* may be used to change this page size (i.e. the default setting is "label/0/4k"). The special value 0 modifies the allocation policy: instead of being increased of a fixed amount, the block size is doubled. In all cases unused memory is released when the file is closed.

7.4.2 Driver *mempipe*

```
mempipe:name
```

A *memory pipe* is characterized by its *name*. Only one model may open a pipe for reading but several models may open the same pipe for writing. However, if several models try to write to the same pipe, only one (it becomes the *owner* of the memory pipe) performs its IO operations and the others are suspended until the owner closes its file descriptor to the specified pipe. Then, one of the waiting models is restarted and becomes the new owner: this process continues until all file descriptors to the pipe are closed.

Pipe operations are possible only if the two ends of the pipe are open: one model for reading and at least one model for writing. There is no notion of “end of file” in a pipe: if a model tries to
read from an empty pipe (i.e. no model is writing to the other end) no error is raised and the model is suspended until something is available. Similarly trying to write to a pipe for which no model is reading from the other end is a blocking operation. In order to avoid lock ups, it is usually good practice to synchronize the models using events. For instance a model waits for a specific event before trying to read from a pipe; before starting to write to the same pipe, the other model sends the expected event.

Memory pipes may be used with initialization blocks. This driver does not use labels for each record of the initialization block: it is assumed (but not checked) that both ends of the pipe are using the same sequence of records. For instance, if the writer sends an integer, a string and then an array of reals, the reader must expect an integer, a string and an array of reals: it is not allowed to skip records or change order as it is usually possible with these blocks.

7.4.3 Driver rcmd

rcmd:[command]

This driver starts the specified command in a new process and connects its standard input and output streams to the calling Mosel instance. The created process is executed in the same current working directory as the controlling model and inherits the environment variables defined using setenv. The default command is "mosel -r". A typical use for this driver is to start an instance on the current machine or on a remote computer through an external program. For instance:

rcmd:rsh sunbox mosel -r

When Mosel is running in restricted mode (see Section 1.3.3), the restriction NoExec disables this driver.

7.4.4 Driver xsrv

xsrv:hostname[(port)]/[ctx[/pass]][|var=val...]

This driver connects to the host hostname running the Mosel Remote Launcher (see Section 7.5) through a TCP socket on port port (default value: 2513) asking for the context ctx (default: xpress) using the password pass (default: no password). Additional environment variables can be specified: assignments of the form var=val must be separated by the symbol | and variable values may include variable references noted ${varname} (expansion is performed on the remote host in the context of its environment). The special environment variable MOSEL_CWD defines the current working directory for the newly created instance.

xsrv:winbox(3211)/xpr64|MOSEL_CWD=C:\workdir|MYDATA=${MOSEL_CWD}\data

7.4.5 Driver xssh

xssh:hostname[(port,kwf)]/[ctx[/pass]][|var=val...]

This driver is the secure version of the xsrv driver described above: it establishes the connection to the xprmsrv server through an encrypted SSH tunnel (using 2515 as the default TCP port number). In addition to the port number, the driver can also take a file name (kwf) used as the known host file for server authentication: this file contains the list of known hosts with their corresponding public keys. When the connection is established to the remote host, the public key stored in this file is compared with the key provided by the server. The connection is canceled if keys do not match. Generating this known hosts file requires running the command xprmsrv -key public on the remote server in order to retrieve its public key (see Section 7.5.1).
For instance, the following command will include the server mysun in the knownhosts.txt file (the command must be run on the server):

```bash
xprmsrv -key public -hn mysun >>knownhosts.txt
```

Then after having moved the file to the machine(s) from where connections are initiated, the following connection string may be used to open secure connections with server authentication:

```bash
xssh:mysun(knownhosts.txt)
```

The remote connection is handled by a separate process. By default the program xprmsrv is used as the helper program but it can be replaced by another SSH client by changing the control parameter sshcmd.

### 7.4.6 Driver rmt

rmt:[node]filename

This driver can be used with any routine expecting a physical file for accessing files on remote instances. By default, the file is located on the instance running on the node identified by the parameter defaultnode but a particular instance may be specified by prefixing the file name by its node number enclosed in square brackets. The special node number -1 designates the parent node of the current instance.

```bash
load(mi,mo,"rmt:[-1]model.bim")
```

In addition to physical files, this driver also emulates the behaviour of drivers cb, sysfd, tmp, shmem and java such that it can transfer streams from one instance to another. For instance, "rmt:sysfd:2" is the standard error stream on the initial instance.

### 7.5 The Mosel Remote Launcher xprmsrv

The xprmsrv program is the server part of the "xsrv:" and "xssh:" IO drivers: it must be running on each computer on which instances will be started using these drivers. The communication between two Mosel instances is achieved through a single TCP stream. Mosel instances are started in the context of execution environments: such an environment consists in a set of environment variables as well as the name of the program to start with its initial working directory. The server can manage different execution environments which are identified by a name and optionally protected by a password. Thanks to this feature a single server can offer several versions of Xpress or dedicated settings for particular distributed applications.

This program is also used as an SSH client by mmjob and XPRD when connecting to an xprmsrv server through a secure tunnel. Therefore it must be available when using the "xssh:" IO driver even if no server is to be run on the host machine.

#### 7.5.1 Running the xprmsrv command

##### 7.5.1.1 Main command line options

The first argument of the command that is not identified as an option is used as the name for a configuration file. The following options are accepted:

- `-h`
  - Display a short help message and terminate.
-V  Display the version number and terminate.

-tc Display the current configuration and terminate.

-f  Force automatic setting of environment variable XPRESSDIR even if it is already defined.

-v [#]  Set the verbosity level of the communication protocol. The default value is 1 (report only errors) when the server is running in background (service/daemon) and 2 (report activity) when the server is run from a console.

-l fname  Set a logfile to record all messages.

-li addr  Set the address of the interface to use (default: 0.0.0.0 for all interfaces).

-p port  Set the TCP port to listen to (default port is 2513, -1 to disable).

-bp port  Set the UDP port for broadcast (default port is 2514, -1 to disable).

-pf pfname  Define a file name for recording the process number of the server. This file is removed when the server exits.

-d  Start the server in background (or as a daemon on Posix systems).

The following options are used by the Windows version of the server:

-service install  Install the server as a service. All other provided options (including configuration file) are recorded and will be used by the server. If the corresponding service has already been installed, its execution settings are updated with the provided options.

-service remove  Remove the previously installed service.

-service start  Start the previously installed service.

-service reload  Reload configuration.

-service stop  Stop the previously started service.

-service status  Check whether the service is already running.

-u user  This option is used only when installing the service: it selects the user running the service.

-pwd pwd  This option specifies the password required for the user indicated by the -u option.

The following options are used by all other platforms:

-u user  User that should be running the server.

-g group  Group that should be running the server.

When the server is run as a service (under Windows) or as a daemon (on Posix systems) that are usually started by a privileged user, it is recommended to use the appropriate option to run the process as an unprivileged user for security reasons. For instance, under Windows, installing the service can be done using the following command in order to use the network service account:

```
xprmsrv -service install -u "NT AUTHORITY\NetworkService" conffile
```

Similarly on a Posix system, the server can be run as the nobody user:

```
xprmsrv -d -u nobody conffile
```
7.5.1.2 Secure server

xprmsrv can also accept secure connections through SSH tunnels: this is the protocol used by the xssh IO driver. The following options are used to setup the secure server:

- **-sp** port  Set the TCP port for SSH connections (default port is 2515, -1 to disable).
- **-k** fname  Private key file name.
- **-sc** cilst  Set the list of accepted ciphers in order of preference (default: "aes256-ctr aes192-ctr aes128-ctr aes256-cbc aes192-cbc aes128-cbc blowfish-cbc 3des-cbc").

The secure server requires a private key to authenticate itself (see following section). By default it will use the file "xprmsrv_rsa.pem" located in the same directory as the xprmsrv executable. It is important to store this file in a secure location as it identifies the server, in particular it must not be readable by Mosel models started by the server. If this file is missing or the provided file name cannot be accessed the secure server will be disabled.

7.5.1.3 Private key management

A new private key can be generated with the following command:

```
xprmsrv -key new
```

Additionally, option **-k filename** can be specified to change the default key file location. Note that this procedure does not remove an existing key file.

The following command loads and check the validity of a key file:

```
xprmsrv -key check
```

When executed on a valid key file this command displays the fingerprint of the public part of the key as well as its properties.

The SSH protocol makes possible authentication of a server by a client. This optional feature, supported by the IO driver xssh, requires a known host file on the client side: this text file consists in a list of host server names with their associated public key. The command xprmsrv **-key public** generates the required data for such a file using the hostname reported by the operating system to identify the server. Often this hostname does not correspond to the public name of the machine. In such a case, it is possible to replace the label in the file or use the option **-hn name** to select a different name. For instance, the following command will append to the file knownhosts.txt the public data key for the server using keyfile mykey.pem with host name srvname:

```
xprmsrv -key public -k mykey.pem -hn srvname >>knownhosts.txt
```

7.5.1.4 Mode of operation

The server proceeds as follows:

1. If the environment variable XPRESSDIR is not defined or if the **-f** option is in use, the value of this environment variable is deduced from the location of the program itself. Under Posix operating systems, the environment variable XPRESS is also set up.
2. The environment variables `MOSEL_DSO` (see Section 2.3.1), `MOSEL_EXECPATH` (see `system`), `MOSEL_RWPATH`, `MOSEL_ROPATH` (see Section 1.3.3) and `XPRMSRV_ACCESS` (see Section 7.5.2.1) are cleared and the environment variable `MOSEL_RESTR` is initialised with value "NoReadNoWriteNoExecNoDBWDOnly" (see Section 1.3.3).

3. The default execution environment `xpress` is created: it refers to the Xpress installation detected at the first step.

4. If available, the configuration file is read (see Section 7.5.2): it can be used to define global settings (e.g., defining the logfile) or create and modify execution environments by defining environment variables.

5. The process then starts its main loop listening to the specified TCP and UDP ports.

6. When a connection is requested, a new session is started to process commands from the client. These commands are used to authenticate the client, select an environment and finally start the Mosel program in a separate process. This process inherits all the environment variables defined in the context and starts in the specified working directory (by default: the location pointed by `XPRESSDIR`). In addition, on Posix systems, the path `${XPRESSDIR}/lib` is added to the dynamic library path of the operating system. Once the process is started, `xprmsrv` detaches itself from the client — the communication is established directly between the two Mosel instances.

### 7.5.2 Configuration file

The configuration file consists in a list of variable definitions of one of the following forms:

```
varname=value
varname?=value
```

Each statement is recorded in the `current environment`. The value may contain variable references noted `${varname}`, the expansion is executed when the environment is processed except for self references that are expanded at the time of defining the variable (e.g. `PATH=${PATH}:otherpath`). When the first syntax is used, the variable cannot be changed by a remote host; the second syntax (using `?=`) allows a remote host to modify the corresponding variable before starting the Mosel instance.

Switching to a different environment is done by giving the name of the environment enclosed in square brackets:

```
[newenv]
```

If the environment name has not yet been used, a new environment is created unless the line ends with the symbol `+' (e.g. `[myenv]+`). In this case the following definitions are included only if the environment already exists. If the line ends with the symbol `=' (e.g. `[myenv]=`) the previous definitions for this context are cleared. These markers can be combined (e.g. `[myenv]+=`) such that the definition block replaces the corresponding context only if it exists.

Upon startup, two environments are automatically created: "global" to store general configuration and settings shared by all environments and "xpress" (it can also be referred to as `*` or `default`) the default execution environment. When the reading of the configuration file begins, the `global` environment is selected: in this environment all variable definitions are processed immediately and added to the `xprmsrv` process environment. In this context, some variables have a special meaning and are not handled as ordinary environment variables:

```
LOGFILE   the file to be used for recording all messages. Messages are sent to the standard error stream when this parameter is not set.
```
LISTEN

address of the interface to use (default value: 0.0.0.0 for all interfaces).

TCP_PORT

the port number to use for TCP connections (default value: 2513, -1 to disable).

UDP_PORT

the port number to use for UDP connections (default value: 2514, -1 to disable).
The server listen to this port for broadcast messages (see procedure findxsrvs).

SSH_PORT

the port number to use for SSH connections (default value: 2515, -1 to disable).

KEYFILE

private key file name used by the SSH protocol (default value: xprmsrv_rsa.pem located in the same directory as the xprmsrv executable).

SSH_CIPHERS

the list of accepted ciphers in order of preference (default: "aes256-ctr
aes192-ctr aes128-ctr aes256-cbc aes192-cbc aes128-cbc blowfish-cbc
3des-cbc").

VERBOSITY

verbosity level for the communication protocol (default value: 1 if the server is running in background and 2 if it is run from a console).

GROUPMASK

Bit mask to select what broadcast requests to accept (default value: ANY). The server replies to a request of group grp only if bit test grp & GROUPMASK is not 0 (see procedure findxsrvs). The mask value can be given as an integer (e.g. 3 to allow groups 1 and 2), an hexadecimal number (e.g. 0xFF for groups 1 to 128) or the special keyword ANY (all groups allowed).

MAXAUTHTIME

a connection is closed if the authentication procedure takes more than the specified amount of time in seconds (default value: 30).

MAXSESSIONS

maximum number of concurrent sessions (the default value 0 disables this limitation).

XPRMSRV_ACCESS

access control list (see Section 7.5.2.1).

CONFDIR

a configuration directory path. The server includes each of the files stored in this directory (sorted in alphabetical order) after it has finished reading the main configuration file.

If the corresponding command line options are used (namely options -l, -p, -bp, -sp, -k, -sc and -v) the settings of the configuration file are ignored.

In other contexts, the following variables have a special meaning:

MOSEL_CMD

the command to execute. The default value is "$XPRESSDIR/bin/mosel -r"

MOSEL_CWD

default working directory. The default value is "$XPRESSDIR"

RUN_BEFORE

command to be run before MOSEL_CMD. This command is executed in the same environment as MOSEL_CMD after all variables have been defined.

RUN_AFTER

command to be run after MOSEL_CMD. This command is executed in the environment of the server but the variable itself is expanded in the context of MOSEL_CMD before its execution.

PASS

password required to use this environment (empty by default). If this variable is set to the special value "*", the associated environment is disabled.

MAXSESSIONS

maximum number of concurrent sessions running under this context (by default there is no limit; a maximum of 0 or less disables the environment).

XPRMSRV_ACCESS

context specific access control list (applied after the global access list).
XPRMSRV_SID  session ID: if not explicitly defined this variable is automatically set by the server.

XPRMSRV_PEER  IP address of the remote host: if not explicitly defined this variable is automatically set by the server.

For instance, the following configuration file sets the logfile to "/tmp/logfile.txt"; adds the password "hardone" to the default context and defines an additional context named xptest pointing to a different installation of xpress:

```
# simple xprmsrv config file
LOGFILE=/tmp/logfile.txt

[xpress]
PASS=hardone

[xptest]
XPRESSDIR=/opt/xpressmp/testing
XPRESS=/opt/xpressmp/lic
MOSEL_RESTR=NoWriteNoDBNoExecWDOnly
MOSEL_CWD?=${XPRESSDIR}/workdir
```

Assuming the server using this configuration is running on the machine mypc, the following statements will create two instances on this machine, one for each of the defined execution environments:

```
r1:=connect(m1, "xsrv:mypc/xpress/hardone")
r2:=connect(m2, "xrsv:mypc/xptest")
```

Since MOSEL_CWD has been initialised with the ?= symbol, the remote host can change its working directory. For instance:

```
r2:=connect(m2, "xrsv:mypc/xptest|MOSEL_CWD=/tmp")
```

While the server is running it is possible to request a reload of its configuration: this procedure consists in reading again the configuration file(s) in order to update the definition of the contexts. During this operation only context specific definitions are processed (all global definitions are silently ignored). Under Windows configuration change can only be requested on a running service using the reload command:

```
xprmsrv -service reload
```

On a Unix system configuration change is performed after reception of a signal USR1. For instance if PID is the process ID of a running xprmsrv server:

```
kill -USR1 PID
```

The configuration update can only be executed when the server is not monitoring any Mosel instance. If a request cannot be processed immediately it is delayed until the server is idle. Moreover if an error is detected while reading the configuration an error is reported but the server continues running with its current settings.

7.5.2.1 Access control list

The environment variable XPRMSRV_ACCESS may be defined in each context of the configuration file. This variable defines which hosts are allowed to connect to the server or use a particular context. The restriction applies to the server itself when the variable is defined in the global
context and as a supplementary restriction when it is included in any other context (i.e. a host cannot be allowed in a context if it is rejected by the global context).

The value of the variable must consist in a list of hosts and subnetworks separated by spaces. Each entry of this list can optionally be preceded by the + sign (for accepting the host; this is the default if no policy is specified) or - sign (to reject connection). Order of the list members is important: when checking authorisation for a given host the list is processed from left to right. The first matching entry will decide whether access is allowed or denied. A given host will be rejected if no matching entry can be found.

A host is identified by its name (e.g. myhost) or its IP address (e.g. 192.168.1.1). A subnetwork is defined by a routing prefix that can be expressed as a partial address (e.g. 192.168.1); or using the CIDR notation - the first address of the network followed by the bit-length of the prefix, separated by a slash "/" character (e.g. 192.168.1.0/24). The subnet mask may also be used instead of the bit-length which is a quad-dotted decimal representation like an address (e.g. 192.168.1.0/255.255.255.0). The special identifier ALL is replaced by the subnetwork definition 0.0.0.0/0 (any host) and the identifier SELF is replaced by the hostname of the server.

In the first example below, host uranus is rejected and subnetwork 192.168.1.0/24 is allowed to connect. Note that uranus will be rejected even if it is part of the authorised subnetwork because its reference appears first in the list. In the second example, all hosts are allowed except 2 subnetworks (192.168.1.0/24 and 192.168.2.0/24):

```
XPRMSRV_ACCESS=-uranus 192.168.1
XPRMSRV_ACCESS=-192.168.1.0/255.255.255.0 -192.168.2.0/24 +ALL
```

All defined control lists are preprocessed just after the configuration file has been read in order to resolve host names and check for syntax errors. Unresolved host names are ignored (although a warning is displayed in such a case) but a syntax error on a control list will cause the server to abort its processing.
The \texttt{mmnl} module extends the Mosel language with a new type for representing nonlinear expressions and constraints and also with some additional subroutines. To use this module the following line must be included in the header of the Mosel model file:

\begin{verbatim}
uses 'mmnl'
\end{verbatim}

The first section presents the new functionality for the Mosel language provided by \texttt{mmnl}, namely the new type \texttt{nlctr} and a set of subroutines that may be applied to objects of this type. The following sections give detailed documentation of the subroutines (other than mathematical operators) defined by this module.

## 8.1 New functionality for the Mosel language

### 8.1.1 The problem type \texttt{mpproblem.nl}

This module exposes its functionality through an extension to the \texttt{mpproblem} problem type. As a consequence, all routines presented here are executed in the context of the current problem.

### 8.1.2 The type \texttt{nlctr} and its operators

The module \texttt{mmnl} defines the type \texttt{nlctr} to represent nonlinear constraints in the Mosel Language. As shown in the following example (Section 8.1.4), \texttt{mmnl} also defines the standard arithmetic operations that are required for working with objects of this type. By and large, these are the same operations as for linear expressions (type \texttt{linctr} of the Mosel language) with additionally the possibility to multiply or divide by decision variables and to use the exponential notation $x^r$ (assuming that $x$ is of type \texttt{mpvar}). Nonlinear constraints may also be defined by using overloaded versions of Mosel’s arithmetic and trigonometric functions on expressions involving decision variables (see Section 8.2 for a complete list).

### 8.1.3 Setting initial values

An important feature in Nonlinear Programming is the possibility to set initial values for decision variables. With \texttt{mmnl} this is done by the procedure \texttt{setinitval}. Nonlinear solvers use initial values as starting point for the search. The choice of the initial values may not only have an impact on the time spent by the solver but also, depending on the problem type, on the best (locally optimal) solution found by the solver.

The definitions of initial values can be removed with \texttt{clearinitvals}. It is also possible to employ the solution values obtained from the immediately preceding optimization run as initial values to the next by calling the procedure \texttt{copysoltoinit}. 

---

Fair Isaac Corporation Confidential and Proprietary Information
8.1.4 Example: using mmnl for QCQP

The following example shows how to solve a QCQP (Quadratically Constrained Quadratic Programming) problem with the Xpress-MP QCQP solver. To use this solver we need to load the module mmxprs in addition to mmnl since the module mmnl does not include any solver.

The problem we wish to solve is a classical NLP test problem (source: http://www.orfe.princeton.edu/rvdb/ampl/nlmodels/) that determines the shape of a hanging chain by minimizing its potential energy. The objective function is linear and the problem has convex quadratic constraints.

```plaintext
model "catenary"
uses "mmxprs", "mmnl"

parameters
N = 100  ! Number of chainlinks
L = 1    ! Difference in x-coordinates of endlinks
H = 2*L/N ! Length of each link
end-parameters

declarations
RN = 0..N  ! x-coordinates of endpoints of chainlinks
x: array(RN) of mpvar
y: array(RN) of mpvar  ! y-coordinates of endpoints of chainlinks
end-declarations

forall(i in RN) x(i) is_free
forall(i in RN) y(i) is_free

! Objective: minimise the potential energy
potential_energy:= sum(j in 1..N) (y(j-1)+y(j))/2

! Bounds: positions of endpoints
! Left anchor
x(0) = 0; y(0) = 0
! Right anchor
x(N) = L; y(N) = 0

! Constraints: positions of chainlinks
forall(j in 1..N)
  Link_up(j):= (x(j)-x(j-1))^2+(y(j)-y(j-1))^2 <= H^2

! Setting start values
forall(j in RN) setinitval(x(j), j*L/N)
forall(j in RN) setinitval(y(j), 0)

setparam("XPRS_verbose", true)
minimise(potential_energy)
writeln("Solution: ", getobjval)
forall(j in RN)
  writeln(strfmt(getsol(x(j)),10,5), " ", strfmt(getsol(y(j)),10,5))
end-model
```

A QCQP matrix can be exported to a text file (in MPS or LP format) by adding the following lines to your model after the problem definition:

```plaintext
setparam("XPRS_loadnames", true) ! Enable loading of names
loadprob(potential_energy) ! Load the problem
writeprob("catenary.mat", "") ! Write an MPS matrix ("l" for LP format)
```

Not all problems with quadratic constraints conform with the properties required by QCQP solvers. Xpress-Optimizer therefore performs a convexity check before starting the optimization. This test takes some time and if you know that your problem is convex you may disable it by...
setting the following parameter before starting the optimization.

```
setparam("XPRS_ifcheckconvexity", false)  ! Disable convexity check
```

8.2 Procedures and functions

The module `mmnl` overloads certain mathematical functions of the Mosel language, replacing an argument of type `real` by the types `linctr` and `nlctr`. The return value of these functions is of type `nlctr`. This means they can be used as operators in the definition of nonlinear constraints as shown in the example of Section 8.1.4. The relevant functions are:

- **Arithmetic functions:**
  - `abs`: absolute value
  - `ceil`: rounding to the next largest integer
  - `exp`: natural exponent of the argument
  - `floor`: rounding to the next smallest integer
  - `ln`: natural logarithm of the argument
  - `log`: base 10 logarithm of the argument
  - `round`: rounding to the nearest integer
  - `sqrt`: positive square root of the argument
  - `sign`: sign of an expression (-1 if negative, 1 if positive, 0 otherwise)

- **Trigonometric functions:**
  - `arccos`: arccosine of the argument
  - `arcsin`: arcsine of the argument
  - `arctan`: arctangent of the argument
  - `cos`: cosine of the argument
  - `sin`: sine of the argument
  - `tan`: tangent of the argument

Since these mathematical operators are fairly self-explanatory, we shall forego any more detailed documentation of these functions.

The following list gives an overview of all other functions and procedures defined by `mmnl` for which we give detailed descriptions later.

- `clearinitvals`: Delete all initial value definitions. p. 265
- `copysoltoinit`: Copy solution values to initial values. p. 266
- `getsol`: Get the solution value of a nonlinear constraint. p. 268
- `gettype`: Get the type of a nonlinear constraint. p. 271
- `ishidden`: Test whether a constraint is hidden. p. 269
- `sethidden`: Hide or unhide a nonlinear constraint. p. 270
- `setinitval`: Set an initial value (start value) for a variable. p. 267
- `setname`: Associate a matrix name to a nonlinear constraint. p. 272
- `settype`: Set the type of a nonlinear constraint. p. 273
clearinitvals

Purpose
Delete all initial value definitions.

Synopsis
procedure clearinitvals

Example
The following copies the solution values from an optimization run to the initial values of the
variables involved. Later all initial value definitions are deleted and a new initial value is set for
variable x.

    uses "mmnl"
    declarations
    x,y: mpvar
    end-declarations
    ...
    minimize(sin(x+y))
    copysoltoinit
    ...
    clearinitvals
    setinitval(x, -1)

Further information
This procedure deletes all previously defined initial values for decision variables.

Related topics
copysoltoinit, setinitval.

Module
mmnl
**copysoltoinit**

**Purpose**
Copy solution values to initial values.

**Synopsis**
procedure copysoltoinit

**Example**
The following copies the solution values of all variables in an optimization run to their initial values and then sets a different initial value for variable $x(1)$.

```plaintext
uses "mmnl"
declarations
  x: array(1..10) of mpvar
  y,z: mpvar
end-declarations

maximize(x(1)*x(3) + ln(y+z))
copysoltoinit
setinitval(x(1), 0)
```

**Further information**
This procedure copies the solution values of decision variables in the immediately preceding optimization run to their initial values for the next run. Doing so it overrides any previously set initial values for the involved variables. However, the settings for decision variables that did not occur in the previously solved problem remain unchanged.

**Related topics**
copysoltoinit, clearinitvals, setinitval.

**Module**
mmnl
setinitval

**Purpose**
Set an initial value (start value) for a variable.

**Synopsis**
procedure setinitval(x:mpvar, val:real)

**Arguments**
x A decision variable
val A real number to be used as initial value

**Example**
The following sets an initial value of 0 for variable x. For y its solution from the preceding optimization is set as its new initial value.

```plaintext
uses "mmnl"
declarations
  x, y: mpvar
end-declarations
setinitval(x, 0)
setinitval(y, getsol(y))
```

**Further information**
This procedure sets an initial value for a decision variable. Initial values are used by nonlinear solvers as a (good) starting point for the search. It is in general not required that the initial values be part of a feasible solution to the optimization problem. All previously set initial values can be removed by calling clearinitvals. The procedure copysoltoinit can be used to turn the solution of a previous optimization run into initial values for the next run.

**Related topics**
clearinitvals, copysoltoinit.

**Module**
mmnl
**getsol**

**Purpose**
Get the solution value of a nonlinear constraint.

**Synopsis**
```plaintext
function getsol(c:nlctr):real
```

**Argument**
c
A nonlinear constraint

**Return value**
Solution value or 0.

**Example**
The following prints the solution values of a nonlinear constraint and a nonlinear expression.

```plaintext
uses "mmnl"
declarations
  x,y,z: mpvar
  Ctr: nlctr
end-declarations
... ! (Define and solve the problem)
writeln("Evaluation of Ctr: ", getsol(Ctr))
writeln("Evaluation of an expression: ", getsol(abs(x*y)+5*z^3))
```

**Further information**
This function returns the evaluation of a nonlinear constraint using the current solution values of its variables. Note that the solution value of a variable is 0 if the problem has not been solved or the variable is not contained in the problem that has been solved.

**Related topics**
maximize/minimize, copySolToInit.

**Module**
mmnl
**ishidden**

**Purpose**
Test whether a constraint is hidden.

**Synopsis**
function ishidden(c: nlctr): boolean

**Argument**
c A nonlinear constraint

**Return value**
true if the constraint is hidden, false otherwise.

**Example**
The following tests whether a nonlinear constraint is hidden.

```
uses "mmnl"
declarations
c: nlctr
end-declarations

if ishidden(c) then
  writeln("Constraint 'c' is currently hidden.")
end-if
```

**Further information**
This function tests the current status of a constraint. At its creation a constraint is added to the current problem, but using the function sethidden it may be hidden. This means, the constraint will not be contained in the problem that is solved by the nonlinear solver but it is not deleted from the definition of the problem in Mosel.

**Related topics**
sethidden.

**Module**
mmnl
sethidden

Purpose
Hide or unhide a nonlinear constraint.

Synopsis
procedure sethidden(c:nlctr, b:boolean)

Arguments
  c          A nonlinear constraint
  b Constraint status:
          true    Hide the constraint
          false   Unhide the constraint

Example
The following defines a constraint and then sets it as hidden:

uses "mmnl"
declarations
  x,y,z: mpvar
end-declarations

c:= 4*cos(x) + y - z^2 <= 12
sethidden(c, true)

Further information
At its creation a constraint is added to the current problem, but using this procedure it may be hidden. This means that the constraint will not be contained in the problem that is solved by the nonlinear solver but it is not deleted from the definition of the problem in Mosel. Function ishidden can be used to test the current status of a constraint.

Related topics
ishidden.

Module
mmnl
gettype

**Purpose**
Get the type of a nonlinear constraint.

**Synopsis**
function gettype(c:nlctr):integer

**Argument**
c A nonlinear constraint

**Return value**
Constraint type. Applicable values for nonlinear constraints are:
CT_EQ Equality, ‘=’
CT_GEQ Greater than or equal to, ‘≥’
CT_LEQ Less than or equal to, ‘≤’
CT_UNB Non-binding constraint, i.e. free

**Related topics**
settype.

**Module**
mmnl
**setname**

**Purpose**
Associate a matrix name to a nonlinear constraint.

**Synopsis**
```plaintext
procedure setname(c:nlctr, n:string)
```

**Arguments**
- `c` A nonlinear constraint
- `n` Name given to the constraint

**Further information**
1. When exporting a problem to a matrix file, constraint names are deduced from the global public symbols: anonymous and local constraints are usually named after their row number in the matrix. This procedure makes it possible to give a name to these constraints.
2. If the given name starts with the ‘#’ character, the generated matrix name will include the row number of the constraint in the matrix.
settype

Purpose
Set the type of a nonlinear constraint.

Synopsis
procedure settype(c:nlctr, type:integer)

Arguments
- c: A nonlinear constraint
- type: Constraint type. Applicable values are:
  - CT_EQ: Equality, ‘=’
  - CT_GEQ: Greater than or equal to, ‘≥’
  - CT_LEQ: Less than or equal to, ‘≤’
  - CT_UNB: Non-binding constraint

Further information
This procedure can be used to change the type of a nonlinear constraint, turning it into an equality or inequality or making it unbounded, i.e. free.

Related topics
gettype.

Module
mmnl
The Mosel OCI (Oracle Call Interface) interface provides a set of procedures and functions that may be used to access Oracle databases. To use the OCI interface, the following line must be included in the header of a Mosel model file:

```mosel
uses 'mmoci'
```

This manual describes the Mosel OCI interface and shows how to use some standard PL/SQL commands, but it is not meant to serve as a manual for PL/SQL. The reader is referred to the documentation of Oracle for more detailed information on these topics.

### 9.1 Example

Assume that the Oracle database contains a table “pricelist” of the following form:

<table>
<thead>
<tr>
<th>articlenum</th>
<th>color</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>blue</td>
<td>10.49</td>
</tr>
<tr>
<td>1002</td>
<td>red</td>
<td>10.49</td>
</tr>
<tr>
<td>1003</td>
<td>black</td>
<td>5.99</td>
</tr>
<tr>
<td>1004</td>
<td>blue</td>
<td>3.99</td>
</tr>
</tbody>
</table>

The following small example shows how to logon to a database from a Mosel model file, read in data, and logoff from the database.

```mosel
model 'OCIexample'
uses 'mmoci'
declarations
    prices: array (range) of real
end-declarations
setparam("OCIverbose", true) ! Enable OCI message printing in case of error
OCILogon("scott","tiger","") ! connect to Oracle as the user 'scott/tiger'
writeln("Connection number: ", getparam("OCIconnection"))
OCIExecute("select articlenum,price from pricelist", prices) ! Get the entries of field 'price' (indexed by field 'articlenum') in table 'pricelist'
OCILogoff ! Disconnect from the database
end-model
```
Here the OCI\_verbose control parameter is set to true to enable OCI message printing in case of error. Following the connection, the procedure \texttt{OCIexecute} is called to retrieve entries from the field \texttt{price} (indexed by field \texttt{articlenum}) in the table \texttt{pricelist}. Finally, the connection is closed.

For further examples of working with databases and spreadsheets, the reader is referred to the Xpress whitepaper \textit{Using ODBC and other database interfaces with Mosel}.

### 9.2 Data transfer between Mosel and Oracle

Data transfer between Mosel and Oracle is achieved by calls to the procedure \texttt{OCIexecute}. The value of the control parameter \texttt{OCIndxcol} and the type and structure of the second argument of the procedure decide how the data are transferred between the two systems.

#### 9.2.1 From Oracle to Mosel

Information is moved from Oracle to Mosel when performing a \texttt{SELECT} command for instance. Assuming \texttt{mt} has been declared as follows:

\begin{verbatim}
mt: array(1..10,1..3) of integer
\end{verbatim}

the execution of the call:

\begin{verbatim}
OCIexecute("SELECT c1,c2,c3 from T", mt)
\end{verbatim}

behaves differently depending on the value of \texttt{OCIndxcol}. If this control parameter is true, the columns \texttt{c1} and \texttt{c2} are used as indices and \texttt{c3} is the value to be assigned. For each row \((i,j,k)\) of the result set, the following assignment is performed by \texttt{mmoci}:

\begin{verbatim}
mt(i,j):=k
\end{verbatim}

With a table \texttt{T} containing:

\begin{verbatim}
c1 c2 c3 c4
1 2 5 7
4 3 6 8
\end{verbatim}

We obtain the initialization:

\begin{verbatim}
m2(1,2)=5, m(4,3)=6
\end{verbatim}

If the control parameter \texttt{OCIndxcol} is false, all columns are treated as data. In this case, for each row \((i,j,k)\) the following assignments are performed:

\begin{verbatim}
mt(r,1):=i; mt(r,2):=j; mt(r,3):=k
\end{verbatim}

where \(r\) is the row number in the result set.

Here, the resulting initialization is:

\begin{verbatim}
mt(1,1)=1, mt(1,2)=2, mt(1,3)=5
mt(2,1)=4, mt(2,2)=3, mt(2,3)=6
\end{verbatim}

If the SQL statement selects 4 columns (instead of 3) as in:

\begin{verbatim}
OCIexecute("SELECT c1,c2,c3,c4 from T", mt)
\end{verbatim}

and the control parameter \texttt{OCIndxcol} is false, the first column is used as the first array index while the remaining columns are treated as data. As a consequence, for each row \((i,j,k,l)\) the
following assignments are performed:

\[
mt(i,1):=j; \ mt(i,2):=k; \ mt(i,3):=l
\]

The resulting initialization is therefore:

\[
mt(1,1)=2, \ mt(1,2)=5, \ mt(1,3)=7 \\
mt(4,1)=3, \ mt(4,2)=6, \ mt(4,3)=8
\]

The second argument of \texttt{OCIexecute} may also be a list of arrays. When using this version, the value of \texttt{OCIndxcol} is ignored and the first column(s) of the result set are always considered as indices and the following ones as values for the corresponding arrays. For instance, assuming we have the following declarations:

\[
m1, \ m2: \text{array}(1..10) \text{ of integer}
\]

With the statement:

\[
\text{OCIexecute}("\text{SELECT c1,c2,c3 from T", m1,m2})
\]

for each row (i,j,k) of the result set, the following assignments are performed:

\[
m1(i):=j; \ m2(i):=k
\]

So, if we use the table \( T \) of our previous example, we get the initialization:

\[
m1(1)=2, \ m1(4)=5 \\
m2(1)=3, \ m2(4)=6
\]

9.2.2 From Mosel to Oracle

Information is transferred from Mosel to Oracle when performing an \texttt{INSERT} command for instance. In this case, the way to use the Mosel arrays has to be specified by using parameters in the SQL command. These parameters are identified by their name in the expression. For instance in the following expression 3 parameters (:1, :2 and :3) are used:

\[
\text{INSERT INTO T (c1,c2,c3) VALUES (:1,:2,:3)}
\]

\texttt{mmoci} expects that parameters are always named :n where \( n \) is the parameter number starting at 1 but does not impose any order (i.e. :3,:1,:2 is also valid) and a given parameter may be used several times in an expression. The command is then executed repeatedly as many times as the provided data allows to build new tuples of parameters. The initialization of parameters is similar to what is done for a \texttt{SELECT} statement.

Assuming \( mt \) has been declared as follows:

\[
mt: \text{array}(1..2,1..3) \text{ of integer}
\]

and initialized with this assignment:

\[
mt::[1,2,3, \\
4,5,6]
\]

the execution of the call:

\[
\text{OCIexecute("\text{INSERT INTO T (c1,c2,c3) VALUES (:1,:2,:3)",mt)}
\]

behaves differently depending on the value of \texttt{OCIndxcol}. If this control parameter is true, for each execution of the command, the following assignments are performed by \texttt{mmoci}:
The execution is repeated for all possible values of \( i \) and \( j \) (in our example 6 times). The resulting table \( T \) is therefore:

\[
\begin{array}{ccc}
1 & 1 & 1 \\
1 & 2 & 2 \\
1 & 3 & 3 \\
2 & 1 & 4 \\
2 & 2 & 5 \\
2 & 3 & 6 \\
\end{array}
\]

Note that \textit{mmoci} uses the names of the parameters to perform an initialization and not their relative position. This property is particularly useful for UPDATE statements where the order of parameters needs to be changed. For instance, if we want to update the table \( T \) instead of inserting new rows, we can write:

\[
\text{OCIexecute("UPDATE T c3=:3 WHERE c1=:1, c2=:2",mt)}
\]

This command is executed exactly in the same way as the INSERT example above (\textit{i.e.} we do not have \(':3':=1', ':1':=j, ':2':=mt(i,j)) as the order of appearance in the command suggests but \(':1':=i, ':2':=j, ':3':=mt(i,j)).

The same functionality may also be used to reorder or repeat columns. With the same definition of the array \( mt \) as before and a 4-column table \( S \) in the database the execution of the command

\[
\text{OCIexecute("INSERT INTO S (c1,c2,c3,c4) VALUES (:1,:2,:3,:2)",mt)}
\]

results in the following contents of table \( S \):

\[
\begin{array}{cccc}
1 & 1 & 1 & 1 \\
1 & 2 & 2 & 2 \\
1 & 3 & 3 & 3 \\
2 & 1 & 4 & 1 \\
2 & 2 & 5 & 2 \\
2 & 3 & 6 & 3 \\
\end{array}
\]

If the control parameter \textit{OCIndxcol} is \texttt{false}, only the values of the Mosel array are used to initialize the parameters. So, for each execution of the command of our initial example (with 3 parameters), we have:

\[
':1':=mt(i,1), ':2':=mt(i,2), ':3':=mt(i,3)
\]

The execution is repeated for all possible values of \( i \) (in our example 2 times). The resulting table \( T \) is therefore:

\[
\begin{array}{ccc}
1 & 2 & 3 \\
4 & 5 & 6 \\
\end{array}
\]

However if the SQL query defines 4 parameters (instead of 3) as in:

\[
\text{OCIexecute("INSERT INTO T (c1,c2,c3,c4) VALUES (:1,:2,:3,:4)",mt)}
\]

and the control parameter \textit{OCIndxcol} is \texttt{false}, the first parameter is used as the first array index while the remaining parameters are populated with data. As a consequence, for each execution of the command, the following assignments are performed by \textit{mmoci}:

\[
':1':= i, ':2':= mt(i,1), ':3':= mt(i,2), ':4':=mf(i,3)
\]
The execution is repeated for all possible values of i (in our example 2 times). The resulting table T is therefore:

<table>
<thead>
<tr>
<th>c1</th>
<th>c2</th>
<th>c3</th>
<th>c4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

When **OCIexecute** is used with a list of arrays, the behavior is again similar to what has been described earlier for the **SELECT** command: the first parameter(s) are assigned index values and the final ones the actual array values. For instance, assuming we have the following declarations:

| m1, m2: array(1..3) of integer |

And the arrays have been initialized as follows:

| m1::[1,2,3] |
| m2::[4,5,6] |

Then the following call:

```
OCIexecute("INSERT INTO T (c1,c2,c3) VALUES (:1,:2,:3),[m1,m2])
```

executes 3 times the **INSERT** command. For each execution, the following parameter assignments are performed:

```java
':1' := i, ':2' := m1(i), ':3' := m2(i)
```

The resulting table T is therefore:

<table>
<thead>
<tr>
<th>c1</th>
<th>c2</th>
<th>c3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>

### 9.3 Control parameters

The following parameters are defined by **mmoci**:

- **OCIautocommit**: Enable/disable "commit on success" in OCI. p. 279
- **OCIbufsize**: Data buffer size. p. 279
- **OCIcolsize**: Maximum string length. p. 279
- **OCIconnection**: Identification number of the active OCI connection. p. 280
- **OCIdxcol**: Enable/disable debug mode. p. 280
- **OCIndxcol**: Indicate whether to use first columns as indices. p. 280
- **OCIrowcnt**: Number of lines affected by the last SQL command. p. 281
- **OCIrowxfr**: Number of lines transferred during the last SQL command. p. 281
- **OCIsuccess**: Indicate whether the last SQL command succeeded. p. 281
- **OCIVerbose**: Enable/disable message printing by OCI. p. 281
All parameters can be accessed with the Mosel function `getparam`, and those that are not marked read-only in the list below may be set using the procedure `setparam`.

Example:

```mosel
setparam("OCIverbose", true) ! Enable message printing by OCI
csize:=getparam("OCIcolsize") ! Get the maximum string length
setparam("OCIconnection", 3) ! Select the connection number 3
```

---

**OCIautocommit**

<table>
<thead>
<tr>
<th>Description</th>
<th>Enable/disable &quot;commit on success&quot; in OCI.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Boolean, read/write</td>
</tr>
<tr>
<td>Values</td>
<td><code>true</code> Changes to the database are committed automatically.</td>
</tr>
<tr>
<td></td>
<td><code>false</code> transactions have to be explicitly committed (or rolled back) using <code>OCIcommit</code> (or <code>OCIrollback</code>).</td>
</tr>
<tr>
<td>Default value</td>
<td><code>true</code></td>
</tr>
<tr>
<td>Module</td>
<td><code>mmoci</code></td>
</tr>
</tbody>
</table>

**OCIbufsize**

<table>
<thead>
<tr>
<th>Description</th>
<th>Size in kilobytes of the buffer used for exchanging data between Mosel and Oracle.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Integer, read/write</td>
</tr>
<tr>
<td>Values</td>
<td>At least 1</td>
</tr>
<tr>
<td>Default value</td>
<td>4</td>
</tr>
<tr>
<td>Affects routines</td>
<td><code>OCIexecute, OCIreadstring.</code></td>
</tr>
<tr>
<td>Module</td>
<td><code>mmoci</code></td>
</tr>
</tbody>
</table>

**OCIcolsize**

<table>
<thead>
<tr>
<th>Description</th>
<th>Maximum length of strings accepted to exchange data, anything exceeding this size is cut off.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Integer, read/write</td>
</tr>
<tr>
<td>Values</td>
<td>At least 8</td>
</tr>
<tr>
<td>Default value</td>
<td>64</td>
</tr>
<tr>
<td>Affects routines</td>
<td><code>OCIexecute, OCIreadstring.</code></td>
</tr>
<tr>
<td>Module</td>
<td><code>mmoci</code></td>
</tr>
</tbody>
</table>
### OCIconnection

**Description**  
Identification number of the active OCI connection. By changing the value of this parameter, it is possible to work with several connections simultaneously.

**Type**  
Integer, read/write

**Affects routines**  
OCIlogoff, OCIexecute, OCIreadinteger, OCIreadreal, OCIreadstring.

**Set by routines**  
OCIlogon.

**Module**  
mmoci

### OCIdebug

**Description**  
When this parameter is set to `true`, OCIverbose is also enabled and any SQL request sent to Oracle is displayed to the error stream before execution. This option is ignored if the model is not compiled with debug information.

**Type**  
Boolean, read/write

**Values**  
<table>
<thead>
<tr>
<th>value</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Enable debug mode.</td>
</tr>
<tr>
<td>false</td>
<td>Disable debug mode.</td>
</tr>
</tbody>
</table>

**Default value**  
false

**See also**  
OCIverbose.

**Module**  
mmoci

### OCIndxcol

**Description**  
Indicates whether the first columns of each row must be interpreted as indices in all cases. Setting it to the value `false` might be useful, for example, if one is trying to access a non-relational table, perhaps a dense table. Note this mode can be enabled only if at least the last dimension of each array is of fixed size.

**Type**  
Boolean, read/write

**Values**  
<table>
<thead>
<tr>
<th>value</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Interpret the first columns of each row as indices.</td>
</tr>
<tr>
<td>false</td>
<td>Do not interpret the first columns of each row as indices.</td>
</tr>
</tbody>
</table>

**Default value**  
true

**Affects routines**  
OCIexecute, OCIreadinteger, OCIreadreal, OCIreadstring.

**Module**  
mmoci
# OCIrowcnt

<table>
<thead>
<tr>
<th>Description</th>
<th>Number of lines affected by the last SQL command.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Integer, read only</td>
</tr>
<tr>
<td>Set by routines</td>
<td>OCIexecute, OCIreadinteger, OCIreadreal, OCIreadstring</td>
</tr>
<tr>
<td>See also</td>
<td>OCIrowxfr.</td>
</tr>
<tr>
<td>Module</td>
<td>mmoci</td>
</tr>
</tbody>
</table>

# OCIrowxfr

<table>
<thead>
<tr>
<th>Description</th>
<th>Number of lines transferred during the last SQL command.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Integer, read only</td>
</tr>
<tr>
<td>Set by routines</td>
<td>OCIexecute, OCIreadinteger, OCIreadreal, OCIreadstring</td>
</tr>
<tr>
<td>See also</td>
<td>OCIrowcnt.</td>
</tr>
<tr>
<td>Module</td>
<td>mmoci</td>
</tr>
</tbody>
</table>

# OCIsuccess

<table>
<thead>
<tr>
<th>Description</th>
<th>Indicate whether the last SQL command has been executed successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Boolean, read only</td>
</tr>
<tr>
<td>Values</td>
<td>true Succes. false Error.</td>
</tr>
<tr>
<td>Set by routines</td>
<td>All OCI functions.</td>
</tr>
<tr>
<td>Module</td>
<td>mmoci</td>
</tr>
</tbody>
</table>

# OCIverbose

<table>
<thead>
<tr>
<th>Description</th>
<th>Enable/disable message printing by OCI.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Boolean, read/write</td>
</tr>
<tr>
<td>Values</td>
<td>true Enable message printing. false Disable message printing.</td>
</tr>
<tr>
<td>Default value</td>
<td>true</td>
</tr>
<tr>
<td>Module</td>
<td>mmoci</td>
</tr>
</tbody>
</table>
### 9.4 Procedures and functions

This section lists in alphabetical order the functions and procedures that are provided by the `mmoci` module.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCIcommit</td>
<td>Commit the current transaction.</td>
<td>289</td>
</tr>
<tr>
<td>OCIexecute</td>
<td>Execute an SQL command.</td>
<td>285</td>
</tr>
<tr>
<td>OCIlogoff</td>
<td>Terminate the active database connection.</td>
<td>284</td>
</tr>
<tr>
<td>OCIlogon</td>
<td>Connect to a database.</td>
<td>283</td>
</tr>
<tr>
<td>OCIreadinteger</td>
<td>Read an integer value from a database.</td>
<td>286</td>
</tr>
<tr>
<td>OCIreadreal</td>
<td>Read a real value from a database.</td>
<td>287</td>
</tr>
<tr>
<td>OCIreadstring</td>
<td>Read a string from a database.</td>
<td>288</td>
</tr>
<tr>
<td>OCIrollback</td>
<td>Roll back the current transaction.</td>
<td>290</td>
</tr>
</tbody>
</table>
**OCIlogon**

**Purpose**
Connect to a database.

**Synopsis**
```plaintext
procedure OCIlogon(s:string)
procedure OCIlogon(u:string,p:string,db:string)
```

**Arguments**
- **s** Logon string as "user/password@db"
- **n** User name
- **p** Password
- **db** Database name (may be "" for the default database)

**Example**
The following connects to the database 'test' as the user 'yves' with the password 'DaSH':
```plaintext
OCIlogon("yves/DaSH@test")
```
Open a connection to the default database the user 'scott' with the password 'tiger'
```plaintext
OCIlogon("scott","tiger","")
```

**Further information**
1. This procedure establishes a connection to the database `db` as user `n/p`. It is possible to open several connections but the connection established last becomes active. Each connection is assigned an identification number which can be obtained by getting the value of the parameter `OCIconnection` after this procedure has been executed. This parameter can also be used to change the active connection.
2. When Mosel is running in restricted mode (see Section 1.3.3), the restriction NoDB disables this routine.

**Related topics**
- OCIlogoff.

**Module**
- mmoci
OCIlogoff

**Purpose**
Terminate the active database connection.

**Synopsis**
```
procedure OCIlogoff
```

**Further information**
The active connection can be accessed or changed by setting the control parameter `OCIconnection`.

**Related topics**
OCIlogon.

**Module**
mmoci
OCIexecute

Purpose
Execute an SQL command.

Synopsis
procedure OCIexecute(s:string|text)
procedure OCIexecute(s:string|text, a:array)
procedure OCIexecute(s:string|text, l:list)
procedure OCIexecute(s:string|text, m:set)

Arguments
s       SQL command to be executed
a       An array
l       A list. May be a list of arrays
m       A set

Example
The following example contains four OCIexecute statements performing the following tasks:

■ Get all different values of the column color in the table pricelist.
■ Initialize the arrays colors and prices with the values of the columns color and price of the table pricelist.
■ Create a new table newtab in the active database with 2 columns, ndx and price.
■ Add data entries to table newtab.

declarations
prices: array(1001..1004) of real
colors: array(1001..1004) of string
allcolors: set of string
end-declarations

OCIexecute("select color from pricelist", allcolors)
OCIexecute("select articlenum,color,price from pricelist", [colors,prices])
OCIexecute("create table newtab (ndx integer, price double)"")
OCIexecute("insert into newtab (ndx, price) values (:1,:2)", prices)

Further information
1. This procedure executes the given SQL command. The user is referred to the Oracle documentation for further information on PL/SQL.

2. For output commands (like insert into) this procedure accepts arrays, sets and lists of basic types (integer, real, string or Boolean) as well as module types for which from/to string conversions are available. Record types composed of scalars or other records can also be used (the fields that cannot be handled are silently ignored). It is also possible to use a list of arrays of basic types (all arrays must be indexed by the same sets) or a list of scalar elements of different basic or module types.

3. For input commands (like select from) the same restrictions apply for arrays, lists and list of arrays but sets must be of a basic type.

Related topics
OCIreadinteger, OCIreadreal, OCIreadstring.

Module
mmoci
OCIreadinteger

Purpose
Read an integer value from a database.

Synopsis
function OCIreadinteger(s:string|text):integer

Argument
s  SQL command for selecting the value to be read

Return value
Integer value read or 0.

Example
The following gets the article number of the first data item in table pricelist for which the field color is set to blue:

i:=OCIreadinteger(
   "select articlenum from pricelist where color=blue"
)

Further information
1. 0 is returned if no integer value can be found.
2. If the given SQL selection command does not denote a single value, the first value to which the selection criterion applies is returned.

Related topics
OCIexecute, OCIreadreal, OCIreadstring.

Module
mmoci
**OCIreadreal**

**Purpose**
Read a real value from a database.

**Synopsis**
function OCIreadreal(s:string|text):real

**Argument**
s
SQL command for selecting the value to be read

**Return value**
Real value read or 0.

**Example**
The following returns the price of the data item with index 2 in table newtab:

\[ r := \text{OCIreadreal}("select price from newtab where ndx=2") \]

**Further information**
1. 0 is returned if no real value can be found.
2. If the given SQL selection command does not denote a single value, the first value to which the selection criterion applies is returned.

**Related topics**
OCIexecute, OCIreadinteger, OCIreadstring.

**Module**
mmoci
OCIreadstring

**Purpose**
Read a string from a database.

**Synopsis**
function OCIreadstring(s:string|text):string

**Argument**
s SQL command for selecting the string to be read

**Return value**
String read or empty string.

**Example**
The following retrieves the color of the (first) data item in table pricelist with article number 1004:

s:=OCIreadstring("select color from pricelist where articlenum=1004")

**Further information**
1. The empty string is returned if no real value can be found.
2. If the given SQL selection command does not denote a single entry, the first string to which the selection criterion applies is returned.

**Related topics**
OCIexecute, OCIreadinteger, OCIreadreal.

**Module**
mmoci
**OCIcommit**

**Purpose**
Commit the current transaction.

**Synopsis**
```
procedure OCIcommit
```

**Further information**
This procedure is required only if the control parameter **OCIautocommit** is set to **false**.

**Related topics**
- **OCIrollback**.

**Module**
```
mmoci
```
**OCIrollback**

**Purpose**  
Roll back the current transaction.

**Synopsis**  
procedure OCIrollback

**Further information**  
This procedure can be used only if the control parameter OCIautocommit is set to false.

**Related topics**  
OCIcommit.

**Module**  
mmoci
9.5 I/O drivers

This module provides a driver designed to be used in initializations blocks for both reading and writing data. The oci IO driver simplifies access to Oracle databases.

9.5.1 Driver oci

oci:[debug;][noindex;][colsize=#;][bufsize=#;]logstring

The driver can only be used in 'initializations' blocks. The database to use has to be given in the opening part of the block as user/password@dbname. Before this identifier, the following options may be stated:

- **debug** to execute the block in debug mode (to display what SQL queries are produced). This option is ignored if the model is not compiled with debug information.
- **noindex** to indicate that only data (no indices) are transferred between the data source and Mosel. By default, the first columns of each table are interpreted as index values for the array to be transferred. This behaviour is changed by this option.
- **colsize=c** to set the size of a text column (default 64 characters).
- **bufsize=c** to set the size of the data buffer in kilobytes (default 4).

In the block, each label entry is understood as a table name optionally followed by a list of column names in brackets (e.g. "my_table(col1,col2)"). All columns are used if no list of names is specified. Note that, before the table name, one can add option noindex to indicate that for this particular entry indices are not used.

Example:

initializations from "mmoci.oci:scott/tiger@orcl"
NWeeks as "PARAMS(Weeks)" ! Initialize 'NWeeks' with column 'Weeks'
! of table 'PARAMS'
BPROF as "noindex:BPROFILE" ! Initialize 'BPROF' with table 'BPROFILE'
! all columns being data (no indices)
end-initializations
The Mosel ODBC interface provides a set of procedures and functions that may be used to access databases for which an ODBC driver is available. This module also includes the SQLite database engine that can be directly run without the need for any additional software.

To use the ODBC interface, the following line must be included in the header of a Mosel model file:

```
uses 'mmodbc'
```

This manual describes the Mosel ODBC interface and shows how to use some standard SQL commands, but it is not meant to serve as a manual for SQL. The reader is referred to the documentation of the software he is using for more detailed information on these topics.

### 10.1 Prerequisite

The ODBC technology relies on a **driver manager** that is used as an interface between applications (like *mmodbc*) and a **data source** itself accessed through a dedicated driver. As a consequence, this module requires that both, a driver manager and the necessary drivers (one for each data source to be used), are installed and set up on the operating system.

Under Windows, usually the driver manager is part of the system and most data sources are provided with their ODBC driver (for instance Excel, Access or SQLServer).

On the other supported operating systems it may be necessary to install a driver manager (as well as the necessary drivers). The module *mmodbc* supports two driver managers: **iODBC** ([http://www.iodbc.org](http://www.iodbc.org)) and **unixODBC** ([http://www.unixodbc.org](http://www.unixodbc.org)). Upon startup the module tries to load the dynamic library "libiodbc.so" ("libiodbc.sl" under HP-UX) then, if this fails, tries "libodbc.so" ("libodbc.sl" under HP-UX). If none of these libraries can be found only the SQLite integrated driver will be available, please make sure that one of the driver managers is installed and that the corresponding libraries can be accessed (in general this requires updating some environment variable).

### 10.2 Example

Assume that the data source “mydata” defines a database that contains a table “pricelist” of the following form:
The following small example shows how to connect to a database from an Mosel model file, read in data, and disconnect from the data source.

```mosel
model 'ODBCexample'
uses 'mmodbc'

declarations
  prices: array (range) of real
end-declarations

setparam("SQLverbose", true) ! Enable ODBC message printing in case of error
SQLconnect("DSN=mydata") ! Connect to the database defined by 'mydata'
writeln("Connection number: ", getparam("SQLconnection"))
SQLexecute("select articlenum,price from pricelist", prices)
! Get the entries of field 'price' (indexed by
! field 'articlenum') in table 'pricelist'
SQLdisconnect ! Disconnect from the database
end-model
```

Here the SQLverbose control parameter is set to true to enable ODBC message printing in case of error. Following the connection, the procedure SQLexecute is called to retrieve entries from the field price (indexed by field articlenum) in the table pricelist. Finally, the connection is closed.

For further examples of working with databases and spreadsheets, the reader is referred to the Xpress whitepaper Using ODBC and other database interfaces with Mosel.

### 10.3 Data transfer between Mosel and the database

Data transfer between Mosel and the database is achieved by calls to the procedure SQLexecute. The value of the control parameter SQLndxcol and the type and structure of the second argument of the procedure decide how the data are transferred between the two systems.

#### 10.3.1 From the database to Mosel

Information is moved from the database to Mosel when performing a SELECT command for instance. Assuming mt has been declared as follows:

```mosel
mt: array(1..10,1..3) of integer
```

the execution of the call:

```
SQLexecute("SELECT c1,c2,c3 from T", mt)
```

behaves differently depending on the value of SQLndxcol. If this control parameter is true, the columns c1 and c2 are used as indices and c3 is the value to be assigned. For each row (i,j,k) of the result set, the following assignment is performed by mmodbc:
\[ mt(i,j):=k \]

With a table \( T \) containing:

\[
\begin{array}{cccc}
  c1 & c2 & c3 & c4 \\
  1 & 2 & 5 & 7 \\
  4 & 3 & 6 & 8 \\
\end{array}
\]

We obtain the initialization:

\[
\begin{align*}
  m2(1,2) &= 5, & m(4,3) &= 6 \\
\end{align*}
\]

If the control parameter \( SQLndxcol \) is \textit{false}, all columns are treated as data. In this case, for each row \((i,j,k)\) the following assignments are performed:

\[
\begin{align*}
  mt(r,1) &= i; & mt(r,2) &= j; & mt(r,3) &= k \\
\end{align*}
\]

where \( r \) is the row number in the result set.

Here, the resulting initialization is:

\[
\begin{align*}
  mt(1,1) &= 1, & mt(1,2) &= 2, & mt(1,3) &= 5 \\
  mt(2,1) &= 4, & mt(2,2) &= 3, & mt(2,3) &= 6 \\
\end{align*}
\]

If the SQL statement selects 4 columns (instead of 3) as in:

\[
\text{SQLexecute("SELECT c1,c2,c3,c4 from T", \( mt \)}
\]

and the control parameter \( SQLndxcol \) is \textit{false}, the first column is used as the first array index while the remaining columns are treated as data. As a consequence, for each row \((i,j,k,l)\) the following assignments are performed:

\[
\begin{align*}
  mt(i,1) &= j; & mt(i,2) &= k; & mt(i,3) &= l \\
\end{align*}
\]

The resulting initialization is therefore:

\[
\begin{align*}
  mt(1,1) &= 2, & mt(1,2) &= 5, & mt(1,3) &= 7 \\
  mt(4,1) &= 3, & mt(4,2) &= 6, & mt(4,3) &= 8 \\
\end{align*}
\]

The second argument of \texttt{SQLexecute} may also be a list of arrays. When using this version, the value of \( SQLndxcol \) is ignored and the first column(s) of the result set are always considered as indices and the following ones as values for the corresponding arrays. For instance, assuming we have the following declarations:

\[
m1, m2: \text{array}(1..10) \text{ of integer}
\]

With the statement:

\[
\text{SQLexecute("SELECT c1,c2,c3 from T", \{m1,m2\})}
\]

for each row \((i,j,k)\) of the result set, the following assignments are performed:

\[
m1(i):=j; \quad m2(i):=k
\]

So, if we use the table \( T \) of our previous example, we get the initialization:

\[
\begin{align*}
  m1(1) &= 2, & m1(4) &= 5 \\
  m2(1) &= 3, & m2(4) &= 6 \\
\end{align*}
\]
10.3.2 From Mosel to the database

Information is transferred from Mosel to the database when performing an INSERT command for instance. In this case, the way to use the Mosel arrays has to be specified by using parameters in the SQL command. These parameters are identified by the symbol '?' in the expression. For instance in the following expression 3 parameters are used:

\[
\text{INSERT INTO } T (c1,c2,c3) \text{ VALUES (?,?,?)}
\]

The command is then executed repeatedly as many times as the provided data allows to build new tuples of parameters. The initialization of parameters is similar to what is done for a SELECT statement.

Assuming \( mt \) has been declared as follows:

\[
mt: \text{array}(1..2,1..3) \text{ of integer}
\]

and initialized with this assignment:

\[
mt::[1,2,3, 4,5,6]
\]

the execution of the call:

\[
\text{SQLexecute("INSERT INTO } T (c1,c2,c3) \text{ VALUES (?,?,?)"},mt)
\]

behaves differently depending on the value of \( \text{SQLndxcol} \). If this control parameter is \text{true}, for each execution of the command, the following assignments are performed by \text{mmodebc} \( (?1,?2,?3) \) denote respectively the first second and third parameter):

\[
'?1':=i, '?2':=j, '?3':=mt(i,j)
\]

The execution is repeated for all possible values of \( i \) and \( j \) (in our example 6 times). The resulting table \( T \) is therefore:

\[
c1 \ c2 \ c3 \\
1 \ 1 \ 1 \\
1 \ 2 \ 2 \\
1 \ 3 \ 3 \\
2 \ 1 \ 4 \\
2 \ 2 \ 5 \\
2 \ 3 \ 6
\]

If the control parameter \( \text{SQLndxcol} \) is \text{false}, only the values of the Mosel array are used to initialize the parameters. So, for each execution of the command, we have:

\[
'?1':=mt(i,1), '?2':=mt(i,2), '?3':=mt(i,3)
\]

The execution is repeated for all possible values of \( i \) (in our example 2 times). The resulting table \( T \) is therefore:

\[
c1 \ c2 \ c3 \\
1 \ 2 \ 3 \\
4 \ 5 \ 6
\]

However if the SQL query defines 4 parameters (instead of 3) as in:

\[
\text{SQLexecute("INSERT INTO } T (c1,c2,c3,c4) \text{ VALUES (?,?,?,?)"},mt)
\]

and the control parameter \( \text{SQLndxcol} \) is \text{false}, the first parameter is used as the first array index while the remaining parameters are populated with data. As a consequence, for each execution
of the command, the following assignments are performed by `mmodbcm`:

'\texttt{?1}' = i, '\texttt{?2}' = \texttt{mt(i,1)}, '\texttt{?3}' = \texttt{mt(i,2)}, '\texttt{?4}' = \texttt{mf(i,3)}

The execution is repeated for all possible values of \( i \) (in our example 2 times). The resulting table \( T \) is therefore:

\[
\begin{array}{cccc}
c1 & c2 & c3 & c4 \\
1 & 1 & 2 & 3 \\
2 & 4 & 5 & 6 \\
\end{array}
\]

When `SQLexecute` is used with a list of arrays, the behavior is again similar to what has been described earlier for the `SELECT` command: the first parameter(s) are assigned index values and the final ones the actual array values. For instance, assuming we have the following declarations:

\[
m1, m2: \text{array}(1..3) \text{ of integer}
\]

And the arrays have been initialized as follows:

\[
m1::[1,2,3] \\
m2::[4,5,6]
\]

Then the following call:

\[
\text{SQLexecute("INSERT INTO T (c1,c2,c3) VALUES (?,?,?)",[m1,m2])}
\]

executes 3 times the `INSERT` command. For each execution, the following parameter assignments are performed:

'\texttt{?1}':=i, '\texttt{?2}':=m1(i), '\texttt{?3}':=m2(i)

The resulting table \( T \) is therefore:

\[
\begin{array}{ccc}
c1 & c2 & c3 \\
1 & 1 & 4 \\
2 & 2 & 5 \\
3 & 3 & 6 \\
\end{array}
\]

### 10.4 ODBC and MS Excel

Microsoft Excel is a spreadsheet application. Since ODBC was primarily designed for databases special rules have to be followed to read and write Excel data using ODBC:

- A table of data is referred to as either a named range (e.g. `MyRange`), a worksheet name (e.g. `[Sheet1$]`) or an explicit range (e.g. `[Sheet1$B2:C12]`).
- By default, the first row of a range is used for naming the columns (to be used in SQL statements). The option `FIRSTROWHASNAMES=0` disables this feature and columns are implicitly named F1, F2... However, even with this option, the first row is ignored and cannot contain data.
- The data type of columns is deduced by the Excel driver by scanning the first 8 rows. The number of rows analyzed can be changed using the option `MAXSCANROWS=n` (\( n \) between 1 and 8).

It is important to be aware that when writing to database tables specified by a named range in Excel, they will increase in size if new data is added using an `INSERT` statement. To overwrite
existing data in the worksheet, the SQL statement `UPDATE` can be used in most cases (although this command is not fully supported). Now suppose that we wish to write further data over the top of data that has already been written to a range using an `INSERT` statement. Within Excel it is not sufficient to delete the previous data by selecting it and hitting the Delete key. If this is done, further data will be added after a blank rectangle where the deleted data used to reside. Instead, it is important to use Edit/Delete/Shift cells up within Excel, which will eliminate all traces of the previous data, and the enlarged range.

Microsoft Excel tables can be created and opened by only one user at a time. However, the "Read Only" option available in the Excel driver options allows multiple users to read from the same .xls files.

When first experimenting with acquiring or writing data via ODBC it is tempting to use short names for column headings. This can lead to horrible-to-diagnose errors if you inadvertently use an SQL keyword. We strongly recommend that you use names like "myParameters", or "myParams", or "myTime", which will not clash with SQL reserved keywords.

### 10.5 Control parameters

The following parameters are defined by `mmodbc`:

- **SQLautocommit**: Enable/disable auto commit mode. p. 298
- **SQLbufsize**: Data buffer size. p. 298
- **SQLcolsize**: Maximum string length. p. 298
- **SQLconnection**: Identification number of the active ODBC connection. p. 299
- **SQLdebug**: Enable/disable debug mode. p. 299
- **SQLdm**: Driver manager currently used. p. 299
- **SQLextn**: Enable/Disable extended syntax. p. 300
- **SQLndxcol**: Indicate whether to use first columns as indices. p. 300
- **SQLrowcnt**: Number of lines affected by the last SQL command. p. 300
- **SQLrowxfr**: Number of lines transferred during the last SQL command. p. 301
- **SQLsuccess**: Indicate whether the last SQL command succeeded. p. 301
- **SQLverbose**: Enable/disable message printing by the ODBC driver. p. 301

All parameters can be accessed with the Mosel function `getparam`, and those that are not marked read-only in the list below may be set using the procedure `setparam`.

Example:

```mosel
csetparam("SQLverbose", true)  ! Enable message printing by the ODBC driver
csize:=getparam("SQLcolsize") ! Get the maximum string length
csetparam("SQLconnection", 3) ! Select the connection number 3
```
**SQLautocommit**

**Description**
When this parameter is set to `true` (the default), any change to the database is sent immediately. Otherwise, if transactions are supported by the database, changes are retained until a call to `SQLcommit` (commit changes) or `SQLrollback` (discard changes) is issued. The value of this parameter is used at the time the database is open with `SQLconnect`: once connection is established, changing this parameter has no impact on the existing connections (i.e. they remain in their initial transaction mode).

**Type**
Boolean, read/write

**Values**
- `true`: Enable auto commit mode.
- `false`: Disable auto commit mode (i.e. transactions).

**Default value**
`true`

**Affects routines**
`SQLconnect`

**Module**
mmodbc

---

**SQLbufsize**

**Description**
Size in kilobytes of the buffer used for exchanging data between Mosel and the ODBC driver.

**Type**
Integer, read/write

**Values**
At least 1

**Default value**
4 on Posix systems and 8 on Windows

**Affects routines**
`SQLexecute`, `SQLreadstring`

**Module**
mmodbc

---

**SQLcolsize**

**Description**
Maximum length of strings accepted to exchange data, anything exceeding this size is cut off.

**Type**
Integer, read/write

**Values**
At least 8

**Default value**
64

**Affects routines**
`SQLexecute`, `SQLreadstring`

**Note**
The column size is expressed in bytes when using an ANSI interface (with a multibyte encoding a single character may occupy more than one byte) and in characters when using a Unicode interface.

**Module**
mmodbc
**SQLconnection**

**Description**
Identification number of the active ODBC connection. By changing the value of this parameter, it is possible to work with several connections simultaneously.

**Type**
Integer, read/write

**Affects routines**
SQLdisconnect, SQLexecute, SQLreadinteger, SQLreadreal, SQLreadstring.

**Set by routines**
SQLconnect.

**Module**
mmodbc

---

**SQLdebug**

**Description**
When this parameter is set to true, SQLverbose is also enabled and any SQL request sent to ODBC is displayed to the error stream before execution. This option is ignored if the model is not compiled with debug information.

**Type**
Boolean, read/write

**Values**
- true: Enable debug mode.
- false: Disable debug mode.

**Default value**
false

**See also**
SQLverbose.

**Module**
mmodbc

---

**SQLdm**

**Description**
Driver manager currently used.

**Type**
Integer, read only

**Values**
- <0: No driver manager available (Unix/Linux).
- 0: Unspecified (manager not loaded dynamically).
- 1: iODBC.
- 2: unixODBC.

**Note**
A negative value for this parameter indicates that no driver manager could be found on the system. As a consequence only the integrated SQLite driver can be accessed.

**Module**
mmodbc
**SQLextn**

**Description**: Enable/Disable extended syntax.

**Type**: Boolean, read/write

**Values**
- **true**: Enable extended syntax.
- **false**: Disable extended syntax.

**Default value**: true

**Affects routines**: SQLconnect, SQLexecute.

**Module**: mmodbc

---

**SQLndxcol**

**Description**: Indicates whether the first columns of each row must be interpreted as indices in all cases. Setting it to the value false might be useful, for example, if one is trying to access a non-relational table, perhaps a dense spreadsheet table. Note this mode can be enabled only if at least the last dimension of each array is of fixed size.

**Type**: Boolean, read/write

**Values**
- **true**: Interpret the first columns of each row as indices.
- **false**: Do not interpret the first columns of each row as indices.

**Default value**: true

**Affects routines**: SQLexecute, SQLreadinteger, SQLreadreal, SQLreadstring.

**Module**: mmodbc

---

**SQLrowcnt**

**Description**: Number of lines affected by the last SQL command.

**Type**: Integer, read only

**Set by routines**: SQLexecute, SQLreadinteger, SQLreadreal, SQLreadstring.

**See also**: SQLrowxfr.

**Module**: mmodbc
### SQLrowxfr

**Description**  Number of lines transferred during the last SQL command.

**Type**  Integer, read only

**Set by routines**  SQLexecute, SQLreadinteger, SQLreadreal, SQLreadstring.

**See also**  SQLrowcnt.

**Module**  mmodbc

### SQLsuccess

**Description**  Indicate whether the last SQL command has been executed successfully.

**Type**  Boolean, read only

**Values**  
- true  Success.
- false  Error.

**Set by routines**  All ODBC functions.

**Module**  mmodbc

### SQLverbose

**Description**  Enable/disable message printing by the ODBC driver.

**Type**  Boolean, read/write

**Values**  
- true  Enable message printing.
- false  Disable message printing.

**Default value**  true

**Module**  mmodbc

### 10.6 Procedures and functions

This section lists in alphabetical order the functions and procedures that are provided by the *mmodbc* module.

- **SQLcolumns**  Get the columns of a given table.  p. 303
- **SQLcommit**  Terminate the current transaction by committing any pending changes.  p. 304
- **SQLconnect**  Connect to a database.  p. 305
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLdisconnect</td>
<td>Terminate the active database connection.</td>
<td>307</td>
</tr>
<tr>
<td>SQLexecute</td>
<td>Execute an SQL command.</td>
<td>308</td>
</tr>
<tr>
<td>SQLgetparam</td>
<td>Get the value of an SQL parameter.</td>
<td>311</td>
</tr>
<tr>
<td>SQLindices</td>
<td>Get the list of indices of a given table.</td>
<td>312</td>
</tr>
<tr>
<td>SQLparam</td>
<td>Generate an SQL parameter.</td>
<td>310</td>
</tr>
<tr>
<td>SQLprimarykeys</td>
<td>Get the list of primary keys of a given table.</td>
<td>313</td>
</tr>
<tr>
<td>SQLreadinteger</td>
<td>Read an integer value from a database.</td>
<td>314</td>
</tr>
<tr>
<td>SQLreadreal</td>
<td>Read a real value from a database.</td>
<td>315</td>
</tr>
<tr>
<td>SQLreadstring</td>
<td>Read a string from a database.</td>
<td>316</td>
</tr>
<tr>
<td>SQLrollback</td>
<td>Terminate the current transaction by discarding any pending changes.</td>
<td>317</td>
</tr>
<tr>
<td>SQLtables</td>
<td>Get the list of tables available in the database.</td>
<td>318</td>
</tr>
<tr>
<td>SQLupdate</td>
<td>Update the selected data with the provided array(s).</td>
<td>319</td>
</tr>
</tbody>
</table>
**SQLcolumns**

**Purpose**
Get the columns of a given table.

**Synopsis**

```plaintext
function SQLcolumns(t:string,cname:array(range) of string,cstype:array(range) of string):integer
function SQLcolumns(t:string,cname:array(range) of string,citype:array(range) of integer):integer
function SQLcolumns(t:string,cname:array(range) of string):integer
```

**Arguments**
- `t` The table name
- `cname` An array of strings to return the column names
- `cstype` An array of strings to return the column types (textual representation)
- `citype` An array of integers to return the column types (type codes)

**Return value**
Number of columns.

**Example**
The following example displays the names and types of columns of table ‘dtt’:

```plaintext
declarations
  CR:range
  cname:dynamic array(CR) of string
  ctype:dynamic array(CR) of string
end-declarations

nbc:=SQLcolumns("dtt",cname,ctype)
write("Table 'dtt' has columns:"
forall(c in 1..nbc) write(' ',cname(c),'(',ctype(c),')')
writeln
```

**Related topics**
- SQLtables, SQLprimarykeys, SQLindices.

**Module**
- `mmodbc`
SQLcommit

**Purpose**
Terminate the current transaction by committing any pending changes.

**Synopsis**
procedure SQLcommit

**Further information**
If the database supports transactions and the connection has been created in manual commit mode (see SQLautocommit), all changes to the database are recorded as a transaction. This procedure commits all pending changes corresponding to the current transaction and starts a new transaction.

**Related topics**
SQLrollback.

**Module**
mmodbc
**SQLconnect**

**Purpose**
Connect to a database.

**Synopsis**
procedure SQLconnect(s:string|text)

**Argument**
s Connection string

**Example**
The following connects to the MySQL database ‘test’ as the user ‘yves’ with the password ‘DaSH’:

```
SQLconnect("DSN=mysql;DB=test;UID=yves;PWD=DaSH")
```

Open the database mydata.sqlite with the integrated SQLite engine:

```
SQLconnect("mydata.sqlite")
```

**Further information**

1. This procedure establishes a connection to the database defined by the given connection string. If extended mode is in use (default) and the ODBC driver manager publishes its driver list, the connection string may be reduced to a file name as long as this name allows identification of the required driver (by using the filename extension).

2. Both Unicode and ANSI ODBC interfaces are supported. By default the Unicode interface is used on Windows and the ANSI interface is selected on Posix systems. It is possible to choose the interface by using the "enc:" file name prefix: any of the UTF encodings (except UTF-8) will enable the Unicode interface. Otherwise the ANSI interface is selected using the specified encoding. For instance for using the ANSI interface under Windows with an Access database: "enc:sys,mydb.mdb". Similarly, to use the Unicode interface of MySQL on a Unix machine, the connection strings looks like: "enc:wchar,DSN=mysql;DB=test".

3. It is possible to open several connections but the connection established last becomes active. Each connection is assigned an identification number which can be obtained by getting the value of the parameter SQLconnection after this procedure has been executed. This parameter can also be used to change the active connection.

4. ODBC drivers are not necessarily executed from the same working directory as the model. As a consequence, a driver expecting a file as data source may not be able to locate the file if its name is relative to the current directory (e.g. "DSN=Microsoft Access Driver; DBQ=mydb.mdb"). The use of the function expandpath from mmsystem allows to avoid this problem by generating an absolute path name for the given name (e.g. "DSN=Microsoft Access Driver; DBQ="+expandpath("mydb.mdb")).

5. When Mosel is running in restricted mode (see Section 1.3.3), connections using a file name are not possible if restriction NoRead or NoWrite is active and connections using a DSN are disabled by restriction NoDB.

6. The embedded SQLite database engine is selected when specifying a file name with extension ".db", ".db3", ".sqlite" or ".sqlite3". The driver may also be selected with the help of an extended connection string starting with the DRIVER keyword and using "mmsqlite" as the driver name. In this case the option DB has to be set in order to select the database file and readonly may also be added to open the database read-only. A typical connection string for this SQLite driver is therefore of the form: "DRIVER=mmsqlite;READONLY=FALSE;DB=mydata.db" (that is the same as "mydata.db"). When using this syntax a temporary database can be created by using an empty file name and an in-memory database is generated if the file name is ":memory:".
Related topics

SQLdisconnect.

Module

mmodbc
**Purpose**
Terminate the active database connection.

**Synopsis**
procedure SQLdisconnect

**Further information**
The active connection can be accessed or changed by setting the control parameter SQLconnection.

**Related topics**
SQLconnect.

**Module**
mmodbc
**SQLexecute**

**Purpose**
Execute an SQL command.

**Synopsis**

procedure SQLexecute(s:string|text)
procedure SQLexecute(s:string|text, a:array)
procedure SQLexecute(s:string|text, l:list)
procedure SQLexecute(s:string|text, m:set)
procedure SQLexecute(s:string|text, lp:list, a:array)
procedure SQLexecute(s:string|text, lp:list, l:list)
procedure SQLexecute(s:string|text, lp:list, m:set)

**Arguments**

- **s**  SQL command to be executed
- **a** An array
- **l** A list
- **m** A set
- **lp** A list of parameters

**Example**
The following example contains four `SQLexecute` statements performing the following tasks:

- Get all different values of the column `color` in the table `pricelist`.
- Initialize the arrays `colors` and `prices` with the values of the columns `color` and `price` of the table `pricelist`.
- Create a new table `newtab` in the active database with 2 columns, `ndx` and `price`.
- Add data entries to table `newtab`.

```delphi
declarations
  prices: array(1001..1004) of real
  colors: array(1001..1004) of string
  allcolors: set of string
end-declarations

SQLexecute("select color from pricelist", allcolors)
SQLexecute("select articlenum,color,price from pricelist", [colors,prices])
SQLexecute("create table newtab (ndx integer, price double)")
SQLexecute("insert into newtab (ndx, price) values (?,?)", prices)
```
Further information

1. This procedure executes the given SQL command. The user is referred to the documentation of the database driver he is using for more information about the commands that are supported by it. Note that if extended syntax is in use (default), parameters usually noted '?' in normal SQL queries may be numbered (like '?1','?2',...) in order to control in which order are mapped columns of data source table to Mosel arrays. This feature is especially useful when writing 'update' queries for which indices must appear after values (e.g. "update mytable set datacol=?2 where ndxcol=?1").

2. For output commands (like insert into) this procedure accepts arrays, sets and lists of basic types (integer, real, string or Boolean) as well as module types for which from/to string convertions are available. Record types composed of scalars or other records can also be used (the fields that cannot be handled are silently ignored). It is also possible to use a list of arrays of basic types (all arrays must be indexed by the same sets) or a list of scalar elements of different basic or module types.

3. For input commands (like select from) the same restrictions apply for arrays, lists and list of arrays but sets must be of a basic type.

4. The form using an extra list argument will be used with input commands requiring parameters: the list defines the value of the parameters.

Related topics

SQLupdate, SQLreadinteger, SQLreadreal, SQLreadstring.

Module

mmodbc
SQLparam

Purpose
Generate an SQL parameter.

Synopsis
function SQLparam(i:integer):SQLparameter
function SQLparam(r:real):SQLparameter
function SQLparam(s:string):SQLparameter

Arguments
i The initial value as an integer
r The initial value as a real
s The initial value as a string

Return value
SQL parameter suitable for SQL routines.

Example
The following calls a procedure named myproc using 3 parameters. The first one is an input string parameter ('hello'), the second is an input/output integer parameter (10) and the last one is an output string parameter. The procedure returns a result set that mmodbc will use to initialise result. After execution of the query, the new values of the 2 input/output parameters set by the procedure may be displayed using the appropriate SQLgetparam routines.

```
SQLexecute("CALL myproc(?,?,?)",
    ['hello',SQLparam(10),SQLparam('')],result)
writeln("P1=",SQLgetiparam(1))
writeln("P2=",SQLgetsparam(2))
```

Further information
1. This routine can only be used in a list of parameters for an SQL query: it defines an input/output parameter. The input value of the parameter is provided via the argument function (an integer, a real or a string) and the output value (set by the database during the execution of the query) can be retrieved using one of the SQLgetparam functions.
2. SQL parameters are typed: the type of the parameter is deduced from its initial values (passed to the SQLparam function).

Related topics
SQLexecute, SQLreadreal, SQLreadstring, SQLreadinteger, SQLgetparam.

Module
mmodbc
SQLgetparam

Purpose
Get the value of an SQL parameter.

Synopsis
function SQLgetparam(n:integer):integer
function SQLgetrparam(n:integer):real
function SQLgetsparam(n:integer):string

Argument
n Parameter number (≥ 1)

Return value
The value of the corresponding parameter.

Further information
1. This routine can be used after a query using input/output SQL parameters has been executed to retrieve the values of the parameters.
2. Each of the 3 functions is associated to a specific type: for instance SQLgetiparam will return values only for integer parameters.

Related topics
SQLparam.

Module
mmodbc
SQLindices

**Purpose**
Get the list of indices of a given table.

**Synopsis**
procedure SQLindices(t:string,ls:list of string)

**Arguments**
t     The table name
ls    A list of strings to return the index names

**Further information**
The provided list is reset.

**Related topics**
SQLtables, SQLcolumns, SQLprimarykeys.

**Module**
mmodbc
**SQLprimarykeys**

**Purpose**
Get the list of primary keys of a given table.

**Synopsis**
```
procedure SQLprimarykeys(t:string,ls:list of string)
procedure SQLprimarykeys(t:string,li:list of integer)
```

**Arguments**
- **t** The table name
- **ls** A list of strings to return the column names
- **li** A list of strings to return the column numbers

**Further information**
The provided list is reset.

**Related topics**
- SQLtables, SQLcolumns, SQLindices.

**Module**
mmodb
**SQLreadinteger**

**Purpose**
Read an integer value from a database.

**Synopsis**

```plaintext
function SQLreadinteger(s:string|text):integer
function SQLreadinteger(s:string|text,p:list):integer
```

**Arguments**

- `s` SQL command for selecting the value to be read
- `p` A list of SQL parameters

**Return value**
Integer value read or 0.

**Example**
The following gets the article number of the first data item in table pricelist for which the field color is set to blue:

```plaintext
i:=SQLreadinteger(
   "select articlenum from pricelist where color=blue"
)
```

**Further information**

1. 0 is returned if no integer value can be found.
2. If the given SQL selection command does not denote a single value, the first value to which the selection criterion applies is returned.
3. The second argument can be used to specify SQL parameter values if the SQL query contains parameter markers.

**Related topics**
SQLexecute, SQLreadreal, SQLreadstring.

**Module**
mmodbc
SQLreadreal

**Purpose**
Read a real value from a database.

**Synopsis**
function SQLreadreal(s:string|text):real
function SQLreadreal(s:string|text,p:list):real

**Arguments**
s SQL command for selecting the value to be read
p A list of SQL parameters

**Return value**
Real value read or 0.

**Example**
The following returns the price of the data item with index 2 in table newtab:
r:=SQLreadreal("select price from newtab where ndx=2")

**Further information**
1. 0 is returned if no real value can be found.
2. If the given SQL selection command does not denote a single value, the first value to which the selection criterion applies is returned.
3. The second argument can be used to specify SQL parameter values if the SQL query contains parameter markers.

**Related topics**
SQLexecute, SQLreadinteger, SQLreadstring.

**Module**
mmodb
**SQLreadstring**

**Purpose**
Read a string from a database.

**Synopsis**

```plaintext
function SQLreadstring(s:string|text):string
function SQLreadstring(s:string|text,p:list):string
```

**Arguments**

- `s` SQL command for selecting the string to be read
- `p` A list of SQL parameters

**Return value**
String read or empty string.

**Example**

The following retrieves the color of the (first) data item in table pricelist with article number 1004:

```plaintext
s:=SQLreadstring(
    "select color from pricelist where articlenum=1004"
)
```

**Further information**

1. The empty string is returned if no real value can be found.
2. If the given SQL selection command does not denote a single entry, the first string to which the selection criterion applies is returned.
3. The second argument can be used to specify SQL parameter values if the SQL query contains parameter markers.

**Related topics**

- SQLexecute, SQLreadinteger, SQLreadreal.

**Module**

mmodbc
SQLrollback

Purpose
Terminate the current transaction by discarding any pending changes.

Synopsis
procedure SQLrollback

Further information
If the database supports transactions and the connection has been created in manual commit mode (see SQLautocommit), all changes to the database are recorded as a transaction. This procedure discards all pending changes corresponding to the current transaction and starts a new transaction.

Related topics
SQLcommit.

Module
mmodbc
SQLtables

**Purpose**
Get the list of tables available in the database.

**Synopsis**
procedure SQLtables(l:list of string)

**Argument**
l A list of strings to return the table names

**Further information**
This procedure retrieves the list of tables available in the current database. The provided list is reset.

**Related topics**
SQLcolumns, SQLprimarykeys, SQLindices.

**Module**
mmodb
SQLupdate

Purpose
Update the selected data with the provided array(s).

Synopsis
procedure SQLupdate(s:string|text, a:array)
procedure SQLupdate(s:string|text, la:list)

Arguments
s    An SQL ‘SELECT’ command
a    An array of one of the basic types (integer, real, string or Boolean)
la   A list of arrays of basic types (integer, real string or Boolean)

Example
The following example initializes the array prices with the values of the table pricelist, changes some values in the array and finally, updates the date in the table pricelist.

    declarations
    prices: array(1001..1004) of real
    end-declarations
    SQLexecute("select articlenum,price from pricelist", prices)
    prices(1002):=prices(1002)*0.9; prices(1003):=prices(1003)*0.8
    SQLupdate("select articlenum,price from pricelist", prices)

Further information
This procedure updates the data selected by an SQL command (usually ‘SELECT’) with an array or tuple of arrays. This procedure is available only if the data source supports positioned updates (for instance, MS Access does but MS Excel does not).

Related topics
SQLexecute.

Module
mmodbc
10.7 I/O drivers

In order to simplify access to ODBC enabled data sources, this module provides a driver designed to be used in initializations blocks for both reading and writing data.

10.7.1 Driver odbc

`odbc:[debug;][noindex;][colsize=#;][bufsize=#;]DSN`

The driver can only be used in 'initializations' blocks. The Data Source Name to use has to be given in the opening part of the block. Before the DSN, the following options may be stated:

- `debug` to execute the block in debug mode (to display what SQL queries are produced). This option is ignored if the model is not compiled with debug information,
- `noindex` to indicate that only data (no indices) are transferred between the data source and Mosel. By default, the first columns of each table are interpreted as index values for the array to be transferred. This behaviour is changed by this option,
- `colsize=c` to set the size of a text column (default 64 characters),
- `bufsize=c` to set the size of the data buffer in kilobytes (default 4).

In the block, each label entry is understood as a table name optionally followed by a list of column names in brackets (e.g. "my_table(col1,col2)"). All columns are used if no list of names is specified. Note that, before the table name, one can add option `noindex` to indicate that for this particular entry indices are not used.

Example:

```plaintext
initializations from "mmodbc.odbc:auction.db3"
NWeeks as "PARAMS(Weeks)" ! Initialize 'NWeeks' with column 'Weeks'
! of table 'PARAMS'
BPROF as "noindex:BPROFILE" ! Initialize 'BPROF' with table 'BPROFILE'
! all columns being data (no indices)
end-initializations
```
CHAPTER 11

mmquad

The mmquad module extends the Mosel language with a new type for representing quadratic expressions. To use this module, the following line must be included in the header of the Mosel model file:

```mosel
uses 'mmquad'
```

The first section presents the new functionality for the Mosel language that is provided by mmquad, namely the new type qexp and a set of subroutines that may be applied to objects of this type.

Via the inter-module communication interface, the module mmquad publishes several of its library functions. These are documented in the second section. By means of an example it is shown how the functions published by mmquad can be used in another module for accessing quadratic expressions and working with them.

11.1 New functionality for the Mosel language

11.1.1 The type qexp and its operators

The module mmquad defines the type qexp to represent quadratic expressions in the Mosel Language. As shown in the following example, mmquad also defines the standard arithmetic operations that are required for working with objects of this type. By and large, these are the same operations as for linear expressions (type linct of the Mosel language) with in addition the possibility to multiply two decision variables or one variable with itself. For the latter, the exponential notation \( x^2 \) may be used (assuming that \( x \) is of type mpvar).

11.1.1.1 Example: using mmquad for Quadratic Programming

Quadratic expressions as defined with the help of mmquad may be used to define quadratic objective functions for Quadratic Programming (QP) or Mixed Integer Quadratic Programming (MIQP) problems. The Xpress-Optimizer module mmxprs for instance accepts expressions of type qexp as arguments for its optimization subroutines minimize and maximize, and for the procedure loadprob (see also the mmxprs Reference Manual). The following

```mosel
model "Small MIQP example"
uses "mmxprs", "mmquad"

declarations
  x: array(1..4) of mpvar
  Obj: qexp
end-declarations
```
! Define some linear constraints
\[x(1) + 2x(2) - 4x(4) \geq 0 \]
\[3x(1) - 2x(3) - x(4) \leq 100 \]
\[x(1) + 3x(2) + 3x(3) - 2x(4) \geq 10 \]
\[x(1) + 3x(2) + 3x(3) - 2x(4) \leq 30 \]
\[2 \leq x(1); x(1) \leq 20 \]
\[x(2) \text{ is integer}; x(3) \text{ is integer} \]
\[x(4) \text{ is free} \]

! The objective function is a quadratic expression
\[\text{Obj} := x(1) + x(1)^2 + 2x(1)x(2) + 2x(2)^2 + x(4)^2 \]

! Solve the problem and print its solution
\[\text{minimize(Obj)} \]

\[\text{writeln("Solution: ", getobjval)} \]
\[\forall (i \in 1..4) \text{ writeln( getsol(x(i)) )} \]
\[\text{end-model} \]

11.1.2 Procedures and functions

The module \textit{mmquad} overloads certain subroutines of the Mosel language, replacing an argument of type \textit{linctr} by the type \textit{qexp}.

\begin{align*}
\textit{exportprob} & \quad \text{Export a quadratic problem to a file.} & \text{p. 323} \\
\textit{getsol} & \quad \text{Get the solution value of a quadratic expression.} & \text{p. 324}
\end{align*}
**exportprob**

**Purpose**
Export a quadratic problem to a file.

**Synopsis**

```plaintext
procedure exportprob(options:integer, filename:string, obj:qexp)
procedure exportprob(filename:string, obj:qexp)
```

**Arguments**

- **options**
  File format options:
  - EP_MIN LP format, minimization
  - EP_MAX LP format, maximization
  - EP_MPS MPS format
  - EP_STRIP Use scrambled names

- **filename**
  Name of the output file; if empty, output printed to standard output (screen)

- **obj**
  Objective function (quadratic expression)

**Example**

The following example prints the problem to screen using the default format, and then exports the problem in LP-format to the file `prob1.lp` maximizing constraint Profit:

```plaintext
uses "mmquad"
declarations
 Profit:qexp
end-declarations
...
exportprob(0, "", Profit)
exportprob(EP_MAX, "prob1", Profit)
```

**Further information**

This procedure overloads the `exportprob` subroutine of Mosel to handle quadratic objective functions. It exports the current problem to a file, or if no file name is given (empty string ""), prints it on screen. If the given filename has no extension, Mosel appends `.lp` to it for LP format files and `.mat` for MPS format.

**Module**

`mmquad`
**getsol**

**Purpose**
Get the solution value of a quadratic expression.

**Synopsis**
function getsol(q:qexp):real

**Argument**
q A quadratic expression

**Return value**
Solution value or 0.

**Example**
uses "mmquad"
declarations
x,y,z: mpvar
Profit:qexp
end-declarations
...! (Define and solve the problem)
writeln("Profit value: ", getsol(Profit))
writeln("Evaluation of an expression: ", getsol(x*y+5*z^2))

**Further information**
This function returns the evaluation of a given quadratic expression using the current (primal) solution values of its variables. Note that the solution value of a variable is 0 if the problem has not been solved or the variable is not contained in the problem that has been solved.

**Module**
mmquad
11.2 Published library functions

The module *mmquad* publishes some of its library functions via the service IMCI for use by other modules (see the Mosel Native Interface Reference Manual for more detail about services). The list of published functions is contained in the interface structure *mmquad_imci* that is defined in the module header file *mmquad.h*.

From another module, the context of *mmquad* and its communication interface can be obtained using functions of the Mosel Native Interface as shown in the following example.

```c
static XPRMnifct mm;
XPRMcontext mctx;
XPRMdsolib dso;
mmquad_imci mq;
void **quadctx;

dso=mm->finddso("mmquad"); /* Retrieve the mmquad module*/
quadctx=*(mm->getdsoctx(mctx, dso, (void **)&mq)); /* Get the module context and the communication interface of mmquad */
```

Typically, a module calling functions that are provided by *mmquad* will include this module into its list of dependencies in order to make sure that *mmquad* will be loaded by Mosel at the same time as the calling module. The “dependency” service of the Mosel Native Interface has to be used to set the list of module dependencies:

```c
static const char *deplist[]={"mmquad", NULL}; /* Module dependency list */
static XPRMdsoserv tabserv[] = /* Table of services */
{
    {XPRM_SRV_DEPLST, (void *)deplist}
};
```

11.2.1 Complete module example

If the Mosel procedures `write`/`writeln` are applied to a quadratic expression, they print the address of the expression and not its contents (just the same would happen for types `mpvar` or `linctr`). Especially for debugging purposes, it may be useful to be able to display some more detailed information. The module example printed below defines the procedure `printqexp` that displays all the terms of a quadratic expression (for simplicity’s sake, we do not retrieve the model names for the variables but simply print their addresses).

```c
model "Test printqexp module"
uses "printqexp"

declarations
x: array(1..5) of mpvar
q: qexp
end-declarations

printqexp(10+x(1)*x(2)-3*x(3)^2)
q:= x(1)*(sum(i in 1..5) i*x(i))
printqexp(q)
end-model
```

Note that in this model it is not necessary to load explicitly the *mmquad* module. This will be done by the `printqexp` module because *mmquad* appears in its dependency list.

```c
#include <stdlib.h>
```
#include "xprm_ni.h"
#include "mmquad.h"

/**** Function prototypes ****/
static int printqexp(XPRMcontext ctx, void *libctx);

/**** Structures for passing info to Mosel ****/
/* Subroutines */
static XPRMdsofct tabfct[] =
{
    {"printqexp", 1000, XPRM_TYP_NOT, 1, "|qexp|", printqexp}
};

static const char *deplist[] = {"mmquad", NULL}; /* Module dependency list */

/* Services */
static XPRMdsoserv tabserv[] =
{
    {XPRM_SRV_DEPLST, (void *)deplist}
};

/* Interface structure */
static XPRMdsointer dsointer =
{
    0, NULL, sizeof(tabfct)/sizeof(XPRMdsofct), tabfct,
    0, NULL, sizeof(tabserv)/sizeof(XPRMdsoserv), tabserv
};

/**** Structures used by this module ****/
static XPRMnifct mm; /* For storing Mosel NI function table */

/**** Initialize the module library just after loading it ****/
DSO_INIT printqexp_init(XPRMnifct nifct, int *interver, int *libver, XPRMdsointer **interf)
{
    mm = nifct; /* Save the table of Mosel NI functions */
    *interver = MM_NIVERS; /* Mosel NI version */
    *libver = MM_MKVER(0, 0, 1); /* Module version */
    *interf = &dsointer; /* Pass info about module contents to Mosel */

    return 0;
}

/**** Implementation of "printqexp" ****/
static int printqexp(XPRMcontext ctx, void *libctx)
{
    XPRMdsolib dso;
    mmquad_imci mq;
    mmquad_qexp q;
    void **quadctx;
    void *prev;
    XPRMmpvar v1, v2;
    double coeff;
    int nlin, i;

    dso = mm->finddso("mmquad"); /* Retrieve reference to the mmquad module */
    quadctx = *(mm->getdsoctx(ctx, dso, (void **)(&mq)));
        /* Get the module context and the communication interface of mmquad */

    q = XPRM_POP_REF(ctx); /* Get the quadratic expression from the stack */

    /* Get the number of linear terms */
    mq->getqexpstat(ctx, quadctx, q, &nlin, NULL, NULL, NULL);

    /* Get the first term (constant) */
    prev = mq->getqexpnextterm(ctx, quadctx, q, NULL, &v1, &v2, &coeff);
    if (coeff != 0)
        mm->printf(ctx, "%g ", coeff);
    for (i = 0; i < nlin; i++)
        /* Print all linear terms */
        {
11.2.2 Description of the library functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>clearqexpstat</td>
<td>Free the memory allocated by getqexpstat.</td>
<td>330</td>
</tr>
<tr>
<td>getqexpnextterm</td>
<td>Enumerate the terms of a quadratic expression.</td>
<td>331</td>
</tr>
<tr>
<td>getqexpsol</td>
<td>Evaluate a quadratic expression.</td>
<td>328</td>
</tr>
<tr>
<td>getqexpstat</td>
<td>Get information about a quadratic expression.</td>
<td>329</td>
</tr>
</tbody>
</table>
getqexpso1

**Purpose**
Return an evaluation of a quadratic expression based on the current solution.

**Synopsis**
```c
double getqexpso1(XPRMctx ctx, void *quadctx, mmquad_qexp q);
```

**Arguments**
- `ctx`: Mosel's execution context
- `quadctx`: Context of `mmquad`
- `q`: Reference to a quadratic expression

**Return value**
An evaluation of the expression on the current solution.

**Further information**
This function returns an evaluation of a quadratic expression based on last solution obtained from the optimizer. This is the function called when using `getsol` on a quadratic expression from a Mosel program.

**Module**
`mmquad`
getqexpstat

Purpose
Get information about a quadratic expression.

Synopsis
int getqexpstat(XPRMctx ctx, void *quadctx, mmquad_qexp q, int *nblin, int *nbqd, int *changed, XPRMmpvar **lsvar);

Arguments
- ctx Mosel's execution context
- quadctx Context of mmquad
- q Reference to a quadratic expression
- nblin Pointer to which the number of linear terms is returned (may be NULL)
- nbqd Pointer to which the number of quadratic terms is returned (may be NULL)
- changed Pointer to which the change flag is returned (may be NULL). Possible values of this flag:
  - 1 The expression q has been modified since the last call to this function
  - 0 Otherwise
- lsvar Pointer to which is returned the table of variables that appear in the quadratic expression q (may be NULL)

Return value
Total number of terms in the expression.

Further information
This function returns in its arguments information about a given quadratic expression. Any of these arguments may be NULL to indicate that the corresponding information is not required. The last entry of the table lsvar is NULL to indicate its end. This table is allocated by the module mmquad, it must be freed by the next call to this function or with function clearqexpstat.

Module
mmquad
clearqexpstat

**Purpose**
Free the memory allocated by getqexpstat.

**Synopsis**
void clearqexpstat(XPRMctx ctx, void *quadctx);

**Arguments**
- **ctx** Mosel’s execution context
- **quadctx** Context of `mmquad`

**Further information**
A call to this function frees the table of variables that has previously been allocated by a call to function `getqexpstat`.

**Related topics**
- `getqexpstat`

**Module**
- `mmquad`
getqexpnextterm

**Purpose**
Enumerate the list of terms contained in a quadratic expression.

**Synopsis**
```c
void *getqexpnextterm(XPRMctx ctx, void *quadctx, mmquad_qexp q, void *prev, XPRMmpvar *v1, XPRMmpvar *v2, double *coeff);
```

**Arguments**
- **ctx**: Mosel's execution context
- **quadctx**: Context of *mmquad*
- **q**: Reference to a quadratic expression
- **prev**: Last value returned by this function. Should be **NULL** for the first call
- **v1, v2**: Pointers to return the decision variable references for the current term
- **coeff**: Pointer to return the coefficient of the current term

**Return value**
The value to be used as **prev** for the next call or **NULL** when all terms have been returned.

**Example**
The following displays the terms of a quadratic expression:
```c
void dispqexp(XPRMcontext ctx, mmquad_qexp q)
{
    void *prev;
    XPRMmpvar v1,v2;
    double coeff;
    int nlin,ct;

    mq->getqexpstat(ctx, quadctx, q, &nlin, NULL, NULL, NULL);
    ct=0;
    prev=mq->getqexpnextterm(ctx, quadctx, q, NULL, &v1, &v2, &coeff);
    mm->printf(ctx, "%g ", coeff);
    while(prev!=NULL) {
        prev=mq->getqexpnextterm(ctx, quadctx, q, prev, &v1, &v2, &coeff);
        if(ct<nlin) { mm->printf(ctx,"%g %p", coeff, v2); ct++; }
        else mm->printf(ctx,"%g %p * %p", coeff, v1, v2);
    }
    mm->printf(ctx,"\n");
}
```

**Further information**
This function can be called repeatedly to enumerate all terms of a quadratic expression. For the first call, the parameter **prev** must be **NULL** and the function returns the constant term of the quadratic expression (for v1 and v2 the value **NULL** is returned and **coeff** contains the constant term). For the following calls, the value of **prev** must be the last value returned by the function. The enumeration is completed when the function returns **NULL**.
If this function is called repeatedly, after the constant term it returns next all linear terms and then the quadratic terms.

**Module**
*mmquad*
The *mmrobust* module extends the Mosel language with new types for representing robust constraints and describe the associated uncertainty sets. To use this module the following line must be included in the header of the Mosel model file:

```
uses 'mmrobust'
```

This is the reference manual of *mmrobust*. It is highly recommended to study the Xpress white paper on robust optimization found under docs/robust in the Xpress installation.

The first section presents the new functionality for the Mosel language provided by *mmrobust*, namely the new types `uncertain`, `robustctr` and `uncertainctr` and a set of subroutines that may be applied to objects of these types.

The following sections give detailed documentation of the subroutines (other than mathematical operators) defined by this module.

### 12.1 New functionality for the Mosel language

#### 12.1.1 The problem type `mpproblem.xprs.robust`

This module exposes its functionality through an extension to the `mpproblem.xprs` problem type. As a consequence, all routines presented here are executed in the context of the current problem.

#### 12.1.2 The type `uncertain`

An `uncertain` is a quantity whose value is not known, but carries a level of uncertainty. The type `uncertain` is used in the robust constraints of type `robust` to express constraints that are subject to uncertainty, and in `uncertainctr` constraints that describe the set of values that the `uncertain` can take. The values of the `uncertain` quantity will take the possible worst case against the optimality and feasibility of the problem. An `uncertain` can be intuitively thought of as a variable that is not under our control, but which has a value defined by an opponent to be the worst with respect to the model.

It is important to note that an `uncertain` does not have a default lower bound of zero imposed by Mosel, in contrast to `mpvars`. This difference in default behavior is to reflect the most typical use cases.

An `uncertain` can be assigned a nominal value using the assignment operator `:=`. The working of the nominal value is discussed in the Xpress robust optimization white paper found under docs/robust in the Xpress installation.
The actual value of uncertain\( s \) and robust constraints can be obtained after the solution of the robust problem through \texttt{getsol} and \texttt{getact}. The usage of \texttt{getsol} is extended as explained below.

If an uncertain \( u \) is used in a single robust constraint or only in the objective function, then \texttt{getsol(u)} returns one of the possible realizations of the uncertainty set that induced the optimal solution found by Mosel.

If the same uncertain is used in two robust constraints named \texttt{RCon1} and \texttt{RCon2} respectively, the optimal solution of the problem may imply that the uncertain has different values for \texttt{RCon1} and \texttt{RCon2}. Then its value can be obtained for the two constraints via the command \texttt{getsol(u,RCon1)} and \texttt{getsol(u,RCon2)}.

Finally, the left-hand side of a robust constraint (e.g. \texttt{RCon1}) can simply be obtained via the command \texttt{getact(RCon1)}, whereas \texttt{getsol(RCon1)} returns the evaluation of left-hand side - right-hand side.

12.1.3 The type \texttt{robustctr} and its operators

The module \texttt{mmrobust} defines the type \texttt{robustctr} to represent robust constraints in the Mosel Language. It also defines the standard arithmetic operations that are required for working with objects of this type. By and large, these are the same operations as for linear expressions (type \texttt{linctr} of the Mosel language) with additionally the possibility to include uncertain terms (i.e. of type \texttt{uncertain}).

12.1.4 The type \texttt{uncertainctr} and its operators

An uncertainty constraint \texttt{uncertainctr} describes the possible values of the \texttt{uncertain} data, or in other words defines the feasible set of the \texttt{uncertain}s. Intuitively, if we visualize the role of an \texttt{uncertain} as a value under the control of an opponent, then the set of \texttt{uncertainctr}s defines the limitations under which the opponent is operating when choosing the worst possible values in respect of the optimality and feasibility of the model.

12.1.5 Example: using \texttt{mmrobust} for solving a robust problem

Consider the following example

```mosel
model BaseModel
uses "mmrobust";
declarations
x, y, z : mpvar
end-declarations
x + 2*y + 3*z <= 10
maximize(x+y+z)
writeln("x = ", getsol(x), " y = ", getsol(y), " z = ", getsol(z))
end-model
```

This problem will solve to "\(x = 10\) \(y = 0\) \(z = 0\)".

Let us now assume that we only know that the sum of the first two coefficients is 3, and we need a solution that is valid for all realizations within this assumption.

```mosel
model RobustModel
uses "mmrobust";
declarations
end-model
```

Fair Isaac Corporation Confidential and Proprietary Information
This problem will solve to "x = 2.5 y = 2.5 z = 0; u = 2 v = 2".
It is easy to check that any realization of the uncertain u and v will keep the solution vector feasible, and that it is optimal within this assumption.

12.2 Control parameters

The following parameter is defined by *mmrobust*:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>robust_check_feas_original_problem</td>
<td>Check if original, non-robust problem is feasible.</td>
<td>mmrobust</td>
</tr>
<tr>
<td>robust_check_feas_uncertainty_set</td>
<td>Check if uncertainty sets are non-empty.</td>
<td>mmrobust</td>
</tr>
<tr>
<td>robust_uncertain_overlap</td>
<td>Use of uncertain data in multiple robust constraints.</td>
<td>mmrobust</td>
</tr>
</tbody>
</table>

**robust_uncertain_overlap**

- **Description**: This parameter allows for models where more than one robust constraint can use an uncertain. Because each robust constraint is dealt with independently in the robust problem, the optimal solution implicitly may associate different values of the uncertain quantities with each robust constraint.
- **Type**: Boolean, read/write
- **Default value**: false
- **Module**: mmrobust

**robust_check_feas_uncertainty_set**

- **Description**: This parameter allows for checking whether the uncertainty sets contain at least one feasible vector of uncertain. In other words, with this parameter set to true Mosel will check if the opponent actually has a choice of uncertain. If at least one uncertainty set is empty, a warning will be issued. Uncertainty sets should *not* be empty as otherwise a robust problem cannot be created.
- **Type**: Boolean, read/write
- **Default value**: false
robust_check_feas_original_problem

Description
This parameter allows for checking whether the problem where all uncertain are set to their default value is feasible or not. This is off by default but can be useful to check correctness of one’s model without uncertainty before solving the robust problem.

Type
Boolean, read/write

Default value
false

Module
mmrobust

12.3 Procedures and functions

The module mmrobust overloads certain mathematical operators making possible the expression of linear and quadratic expressions involving the type uncertain in order to create both robustctr and uncertainctr objects. Since these mathematical operators are fairly self-explanatory, we shall forego any more detailed documentation of these functions.

The following list gives an overview of all other functions and procedures defined by mmrobust for which we give detailed descriptions later.

- cardinality: Create a cardinality uncertain constraint. p. 336
- getact: Get the activity value of a robust constraint. p. 338
- getnominal: Get the nominal value of an uncertain. p. 342
- getsol: Get the realisation of an uncertain or robust constraint. p. 337
- gettype: Get the type of a constraint. p. 343
- ishidden: Test whether a constraint is hidden. p. 339
- scenario: Create a scenario uncertain constraint. p. 340
- sethidden: Hide or unhide a constraint. p. 341
- setnominal: Set the nominal value of an uncertain. p. 344
- settype: Set the type of a constraint. p. 345
**cardinality**

**Purpose**
Create a cardinality uncertain constraint.

**Synopsis**

```plaintext
function cardinality(su:set of uncertain,m:integer):uncertainctr
```

**Arguments**

- `su` Uncertains to be added to the constraint
- `m` Maximum number of uncertainsthat can be different from their nominal value

**Return value**
The new cardinality uncertain constraint.

**Further information**
A cardinality uncertain constraint limits the number of uncertainsthat can take a non-zero value, or be different from their nominal value.

**Module**

`mmrobust`
getsol

Purpose
Get the realisation of an uncertain or robust constraint.

Synopsis
function getsol(u:uncertain, rc:robustctr):real
function getsol(u:uncertain):real
function getsol(rc:robustctr):real

Arguments
rc    A robust constraint
u     An uncertain

Return value
Solution value or 0.

Further information
This function returns the realization of uncertain u for the robust optimization problem solved. The value of u is only available after solving the robust optimization problem. The value of u is 0 if the problem has not been solved or the uncertain or constraint is not contained in the problem that has been solved.

If the uncertain u appears in more than one constraint, it is necessary to specify the constraint with function call getsol(u,rc): this is a consequence of robust optimization, for which the same uncertain can assume different values in different constraints. If the uncertain u only appears in one constraint, then it suffices to call getsol(u).

The function getsol(rc) returns the evaluation of a constraint with the current realization of the solution and the uncertainties. Therefore, if a constraint is of the form u*x + v*y + z ≤ 3 and x,y,z are variables while u,v are uncertain, the current realization of x,y,z,u,v will be used to return u*x + v*y + z - 3.

Note that robust equality constraints (for instance, u*x + v*y + z = 3) have a special status in Mosel. The value of uncertain u and v is, in general, related to an inequality constraint and can be safely obtained in this case only. In order to use getsol for equality robust constraints as well, it would be best to decompose these constraints into two inequality constraints (i.e. u*x + v*y + z ≤ 3 and u*x + v*y + z ≥ 3) and then request u and v from each of the two constraints. Note that both uncertain might differ in value when requested from either inequality constraint.

Related topics
getact.

Module
mmrobust
getact

**Purpose**
Get the activity value of a robust constraint.

**Synopsis**
function getact(rc:robustctr):real

**Argument**
rc A robust constraint

**Return value**
Solution value or 0.

**Further information**
This function returns the value of the left-hand side of a constraint with the current realization of the solution and the uncertain. Therefore, if a constraint is of the form $u \cdot x + v \cdot y + z \leq 3$ and $x,y,z$ are variables while $u,v$ are uncertain, the current realization of $x,y,z,u,v$ will be used to return $u \cdot x + v \cdot y + z$.

Note that robust equality constraints (for instance, $u \cdot x + v \cdot y + z = 3$) have a special status in Mosel. The value of uncertain $u$ and $v$ is, in general, related to an inequality constraint and can be safely obtained in this case only. In order to use getact for equality robust constraints as well, it would be best to decompose these constraints into two inequality constraints (i.e. $u \cdot x + v \cdot y + z \leq 3$ and $u \cdot x + v \cdot y + z \geq 3$) and then request $u$ and $v$ from each of the two constraints. Note that both uncertain might differ in value when requested from either inequality constraint.

**Related topics**
getsol.

**Module**
mmrobust
**ishidden**

**Purpose**
Test whether a constraint is hidden.

**Synopsis**

```plaintext
function ishidden(rc:robustctr):boolean
function ishidden(uc:uncertainctr):boolean
```

**Arguments**
- `rc`: A robust constraint
- `uc`: An uncertain constraint

**Return value**
- `true` if the constraint is hidden, `false` otherwise.

**Further information**
This function tests the current status of a constraint. At its creation a constraint is added to the current problem, but using the function `sethidden` it may be hidden. This means, the constraint will not be contained in the problem that is solved by the solver but it is not deleted from the definition of the problem in Mosel.

**Related topics**
- `sethidden`

**Module**
- `mmrobust`
scenario

Purpose
Create a scenario uncertain constraint.

Synopsis
function scenario(data:array (range,set of uncertain) of real):uncertainctr

Argument
data Scenario data

Return value
The new scenario uncertain constraint.

Further information
A scenario uncertain constraint takes historical data of the possible realizations of the uncertain data. In effect, the introduced uncertain constraint enforced that for any solution to the robust optimization problem, any robust constraint robustctr is satisfied for all realizations of the uncertain as defined by the data array.

This function stores a reference to the provided array (i.e. it does not make a copy of it). As a consequence any modification to the array will imply modifications to the constraint even after the constraint has been built. Invalid data is only reported at the time of loading the problem into the optimiser.

Module
mmrobust
sethidden

Purpose
Hide or unhide a constraint.

Synopsis
procedure sethidden(rc:robustctr, b:boolean)
procedure sethidden(uc:uncertainctr, b:boolean)

Arguments
rc A robust constraint
uc An uncertain constraint
b Constraint status:
true Hide the constraint
false Unhide the constraint

Further information
At its creation a constraint is added to the current problem, but using this procedure it may be hidden. This means that the constraint will not be contained in the problem that is solved by the solver but it is not deleted from the definition of the problem in Mosel. Function ishidden can be used to test the current status of a constraint.

Related topics
ishidden.

Module
mmrobust
getnominal

**Purpose**
Get the nominal value of an uncertain.

**Synopsis**
function getnominal(u: uncertain): real

**Argument**
u An uncertain

**Return value**
The nominal value of the uncertain

**Related topics**
setnominal.

**Module**
mmrobust
**gettype**

**Purpose**
Get the type of a constraint.

**Synopsis**
function gettype(rc:robustctr):integer
function gettype(uc:uncertainctr):integer

**Arguments**
- rc: A robust constraint
- uc: An uncertain constraint

**Return value**
Constraint type. Applicable values for nonlinear constraints are:
- CT_EQ: Equality, ‘=’
- CT_GEQ: Greater than or equal to, ‘≥’
- CT_LEQ: Less than or equal to, ‘≤’
- CT_CARD: Cardinality
- CT_SCEN: Scenario
- CT_UNB: Non-binding constraint, *i.e.* free

**Related topics**
settype.

**Module**
mmrobust
setnominal

Purpose
Set the nominal value of an uncertain.

Synopsis
procedure setnominal(u:uncertain,n:real)

Arguments
u An uncertain
n A real constant

Further information
Calling this procedure has the same effect as assigning a value to the uncertain using the operator :=.

Related topics
getnominal.

Module
mmrobust
settype

Purpose
Set the type of a constraint.

Synopsis
procedure settype(rc:robustctr, type:integer)
procedure settype(uc:uncertainctr, type:integer)

Arguments
rc A robust constraint
uc An uncertain constraint
type Constraint type. Applicable values are:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CT_EQ</td>
<td>Equality, ‘=’</td>
</tr>
<tr>
<td>CT_GEQ</td>
<td>Greater than or equal to, ’≥’</td>
</tr>
<tr>
<td>CT_LEQ</td>
<td>Less than or equal to, ’≤’</td>
</tr>
<tr>
<td>CT_UNB</td>
<td>Non-binding constraint</td>
</tr>
</tbody>
</table>

Further information
This procedure can be used to change the type of a constraint, turning it into an equality or inequality or making it unbounded, i.e. free.

Related topics
gettype.

Module
mmrobust
The Mosel module *mmsheet* implements several I/O drivers for accessing and modifying spreadsheet files in different formats from ‘initializations’ blocks. The I/O drivers rely on different technologies for accessing spreadsheets.

### 13.1 I/O drivers

The I/O drivers provided by *mmsheet* are all designed to be used in ‘initializations’ blocks and expect the same type of information regarding file names and record references. The common form of a file specification for all the *mmsheet* drivers is:

```
mmsheet.*:[noindex;][grow;][skiph;][bufsize=#;]filename
```

The spreadsheet file name must be a physical file (with its extension), except for the "csv:" driver that accepts extended file names. The driver options (stated before the file name) shared by all *mmsheet* drivers are:

- **noindex**
  - Indicates that only data (no indices) are transferred between the spreadsheet and Mosel. By default, the first columns of each table are interpreted as index values for the array to be transferred. This behaviour is changed by this option.

- **partndx**
  - Indicates that the first $nbdim-1$ columns are interpreted as indices ($nbdim$ being the number of dimensions of the array to process) and remaining ones are used as data for the last dimension.

- **grow**
  - When writing data, the driver uses the provided range ignoring the end of the data if there is not enough space. When this option is specified, the driver extends the range by adding lines if necessary.

- **skiph**
  - With this option, the driver skips the first line (or header) of the provided range. If the range contains only one line, the following line is selected.

- **bufsize=c**
  - To set the size of the data buffer in kilobytes (default c=2).

The driver-specific options are documented separately for each driver in the following sections.

In the initializations block, each label entry is understood as a range in the workbook: named ranges are represented by their name (e.g. "MyRange") and explicit ranges are noted using square brackets (e.g. "[sheet1$a1:c2]"). For explicit ranges, the sheet is identified by its name or number and separated from the cell selection with the $ sign. The first sheet of the workbook is selected if no indication is given. Similarly, the used cells of the selected sheet are assumed if no selection is provided. The cell selection can be stated either using the usual format with a letter to select the column followed by a line number (e.g. "a1:c1") or by specifying row and column.
numbers by prefixing the row number by the letter "R" and the column number by the letter "C"
(e.g. "R1C1:R1C3"). It is also possible to select some of the columns from the specified range: this

 can be done either with a list of names or a list of column numbers (relative to the beginning of
the range) noted in parentheses after the range description. To use names, the option skiph must
be used and the column names are taken from the header row that is skipped through this

 option. When using skiph, column numbers need to be stated by prefixing the column number
by #. Note that, before the range selection, one can add options as for the file opening. For
instance, "skiph;grow;" can be used for writing data to a named range formatted for an ODBC

connection.

Example:

initializations from "mmsheet.excel:skiph;auction.xls"
NWeeks as "[b1:d12]" ! Initialize 'NWeeks' with data in b2:d12
BPROF as "noindex;BPROFILE" ! Initialize 'BPROF' with named range 'BPROFILE'
  ! all columns being data (no indices)
mycols as "[b1:h12](3,5,7)" ! Initialize 'mycols' with columns d2:d12,
  ! f2:f12 and h2:h12
mycol2 as "[b1:h12](nam1,#5,nam3)"
  ! Initialize 'mycol2' with the column named
  ! 'nam1', the column f2:f12 and the column
  ! named 'nam3'

end-initializations

The mapping between the selected cells of the workbook and the Mosel data structures is similar
to the one used for databases (options noindex and partndx correspond to setting parameter
mmodbc.SQLndxcol to false): refer to the section Data transfer between Mosel and the database
of the mmodbc chapter for further explanation.

Although direct read and write operations are not supported by these drivers, a spreadsheet may
be open using fopen: this allows to keep the document open across several 'initializations' blocks
and avoid the cost of loading and unloading the file (that may be expensive particularly with the
"excel:" driver).

For further examples of working with databases and spreadsheets, the reader is referred to the
Xpress whitepaper Using ODBC and other database interfaces with Mosel.

13.1.1 Driver excel

mmsheet.excel:[noindex;][grow;][skiph;][newxl;][bufsize=#;]filename

This driver uses directly the application Excel for accessing the file (relying on COM/OLE as the
communication channel): as a consequence it is available only under the Windows platform and
requires Excel to be installed on the host executing the Mosel model. All file formats handled by
the version of Excel can be used but this driver does not support creation of new files (i.e. it can
only modify existing files). In addition to the options described in the introductory section, the
option newxl may be used: by default the driver does not open the file if it can find a running
instance of Excel having the required file open: it works directly with the application and
modifications made to the workbook are not saved when the file is closed in Mosel. If this option
is specified a new instance of Excel is started in all cases and the workbook is saved before
quitting the application when the file is closed in Mosel.

13.1.2 Driver xls/xlsx

mmsheet.xls:[noindex;][grow;][skiph[*];][bufsize=#;]filename

mmsheet.xlsx:[noindex;][grow;][skiph[*];][bufsize=#;]filename
These two drivers rely on the **libxl** library to access the spreadsheet file: they are available on the Windows, Linux and MacOS platforms and do not require any additional software. The first driver handles xls files while the second deals with xlsx and xlsm format Excel files. These drivers can be used to create new files: when used for writing (through an 'initializations to' block) non-existing sheets are automatically added to the workbook and the file is created if necessary. When the option **skiph+** is used instead of **skiph** when writing to a file, the necessary header row is created if this row is empty (this option behaves like **skiph** when reading a file and when no column name is provided).

### 13.1.3 Driver **CSV**

This driver works on spreadsheets saved in ascii CSV format (Comma Separated Values). It is available on all platforms that are supported by Mosel and can open or create files using extended format file names (*i.e.* combining several I/O drivers). A CSV file contains a single sheet (number 1 identified as "Sheet1") and does not support named ranges, that is, cell references must use the explicit notation. When the option **skiph+** is used instead of **skiph** when writing to a file, the necessary header row is created if this row is empty (this option behaves like **skiph** when reading a file and when no column name is provided). The following driver-specific options may be used to specify the properties of the format to handle:

- **fsep=c** character used to separate fields. The default value is ",", tabulation or ";" are also often employed
- **dsep=c** character used as decimal separator (default: ".")
- **true=s** text representing the **true** value of a Boolean (default: "true")
- **false=s** text representing the **false** value of a Boolean (default: "false")

For example, the following statements will read data from a file formatted for the French language and that has been compressed with gzip:

```
initializations from "mmsheet.csv:fsep=;dsep=.;true=vrai;false=faux;zlib.gzip:mydata.csv.gz"
A as "[a1:c12]"
end-initializations
```
CHAPTER 14

mmssl

The Mosel module mmssl is an interface to the OpenSSL cryptographic library (http://www.openssl.org). It brings most of the functionality of this library to the Mosel language and serves as the cryptographic component in other parts of Mosel. In particular, it provides support for the HTTPS protocol in mmhttp and implements the encryption and signing mechanisms used by the Mosel core libraries when secure BIM files are used.

14.1 Overview

14.1.1 Document encryption in Mosel

Encryption and decryption of documents are achieved by cipher algorithms. Ciphers can be of two kinds: symmetric ciphers use the same encryption key to perform the encryption and decryption tasks while asymmetric ciphers require one key to execute the encryption and another one for the decryption. In mmssl, symmetric ciphers are made available through the crypt I/O driver (Section 14.4.3): the encryption key (the size of which depends on the cipher) is automatically generated based on some given passphrase (either input from an external file or directly in the file name specification). The implementation of the crypt driver allows the user to select which specific cipher algorithm it should use (for instance AES, DES or IDEA).

For asymmetric encryption, mmssl relies on the RSA cryptographic system. For the RSA algorithm, a key (RSAGenkey) consists of two components: a public part that is usually distributed to the individuals with whom documents are to be exchanged and a private part that must be kept secret by the owner of the key (this private key also includes the public key). In this framework, a document encrypted using a public key (RSApubencrypt) can only be decrypted with the corresponding private key (RSAPrivdecrypt). Moreover, the key pair can also be used for signing documents: the electronic signature of a document is created with a private key (msgsign) and the corresponding public key is used to verify this signature (msgverify). Since only the owner of the private key can create the signature, the recipient has a guarantee on the origin of the document.

RSA keys are commonly stored as text files in the OpenSSL PEM standard format, this is also the most convenient representation for exchanging key information (RSAsavekey). In addition to this file format, mmssl can store a key in the form of a Mosel array of integers (RSAloadkey). By using this encoding a model may embed keys or retrieve them from any of the usual model data sources.

14.1.2 The mmssl command

The module mmssl is distributed together with a command line tool of the same name as the module: mmssl. This program helps setting up an initial working environment and performs basic key and certificate operations directly from a shell (Unix) or command window (Windows).
Running the mmssl program without any arguments will display a short help message, otherwise the following commands can be used:

```
setup
  Check the configuration directory of mmssl and create it if necessary (see parameter ssl_dir)

genkey keyfile [size]
  Generate a new RSA key pair of the specified size (default: 1024) and save it into keyfile.

getpub keyfile keyfilepub
  Extract the public key of the private RSA key file keyfile and save it into keyfilepub

chkkey keyfile [keyfile...]
  Check the validity of the provided key file(s)

gencert certfile [prod=value...] 
  Generate an X509 certificate using the provided properties (see x509newcrt for further detail)

chkcert certfile [keyfile]
  Check the validity of the provided X509 certificate. If an additional private key file is provided, its compatibility with the certificate is also checked.

list [digest|cipher]
  Display the list of supported message digests (digest) or cipher algorithms (cipher). Both lists are reported with the short form of the command.
```

Many procedures of the mmssl module require the availability of a configuration directory. To create and populate an initial setup it is recommended to run the following command before starting to use the module:

```
> mmssl setup
```

Note that the setup procedure is not destructive: if the configuration directory has already been created the command will only check its validity, add any missing components and suggest how to proceed in case of incorrect settings.

### 14.2 Control parameters

Via the getparam function and the setparam procedure it is possible to access the following control parameters of module mmssl (the reader is reminded that parameters may be spelled with lower or upper case letters or a mix of both):

- **https_cacerts**: List of trusted certification authorities. (p. 351)
- **https_ciphers**: Ciphers accepted for SSL communication. (p. 351)
- **https_cltcrt**: HTTPS client certificate. (p. 352)
- **https_cltkey**: HTTPS client private key. (p. 352)
- **https_srvcrt**: HTTPS server certificate. (p. 352)
- **https_srvkey**: HTTPS server private key. (p. 353)
- **https_trustsrv**: Whether to trust server certificates. (p. 353)
### https_cacerts

**Description**
Location of the file containing the certificates of the trusted certification authorities.

**Type**
String, read/write

**Note**
The file identified by this parameter consists of a list of certificates of trusted certification authorities (in order to be able to check the validity of servers they have certified) and certificates of servers trusted by the application (typically using self-signed certificates that could not be certified by an external authority, see x509newcrt). This file is used when HTTPS client connections are established to check the identity of the server unless the control parameter https_trustsrv is set to true. It is also required by servers that perform client authentication (see option HTTP_CLAUUTH of server configuration http_srvconfig): in this case the certificates are used to identify the clients. When this parameter has not been initialised, the default location getparam("ssl_dir")+"/ca-bundle.crt" is used. This default file collecting the certificates of the major certification authorities is installed by the mmssl setup command (Section 14.1.2).

**Affects routines**
httpget, httppost, httpdel, httpput, httpstartsrc

**See also**
https_trustsrv

**Module**
mmssl

### https_ciphers

**Description**
This parameter is used during the algorithm negotiation of an HTTPS session initialisation to select which cryptographic algorithm to use.

**Type**
String, read/write

**Default value**
"TLSv1+HIGH:!SSLv2:!aNULL:!eNULL:!3DES:@STRENGTH"

**Note**
This parameter is employed by both the server and the client in an HTTPS session. Please refer to the OpenSSL documentation for a detailed explanation on how to build this selection string.

**Affects routines**
httpstartsrv, httpget, httppost, httpdel, httpput

**Module**
mmssl
**https_cltcrt**

**Description** Location of the client certificate (for HTTPS queries).

**Type** String, read/write

**Note** This parameter specifies the location of the client certificate. Such a certificate (and its associated private key `https_cltkey`) is required when sending HTTPS requests to a server that requires client authentication (see option `HTTP_CLTAUTH` of server configuration `http_srvconfig`).

**Affects routines** `httpget`, `httppost`, `httpdel`, `httpput`

**See also** `https_cltkey`

**Module** `mmssl`

---

**https_cltkey**

**Description** Location of the client private key (for HTTPS queries).

**Type** String, read/write

**Note** This parameter specifies the location of the client private key. Such a key (and its associated certificate `https_cltcrt`) is required when sending HTTPS requests to a server that requires client authentication (see option `HTTP_CLTAUTH` of server configuration `http_srvconfig`).

**Affects routines** `httpget`, `httppost`, `httpdel`, `httpput`

**See also** `https_cltcrt`

**Module** `mmssl`

---

**https_srvcrt**

**Description** Location of the server certificate (required by an HTTPS server).

**Type** String, read/write

**Note** Running an HTTPS server requires a server certificate and its associated private key. This parameter defines the location of the certificate file; to create a certificate you can either use the `mmssl` command (Section 14.1.2) or the Mosel function `x509newcrt`. If no value has been assigned to this parameter the default certificate file `getparam("ssl_dir")+/server.crt` will be used by the server.

**Affects routines** `httpstartsrv`

**See also** `https_srvkey`

**Module** `mmssl`
### https_srvkey

**Description**
Location of the server private key (required by an HTTPS server).

**Type**
String, read/write

**Note**
Running an HTTPS server requires a server certificate and its associated private key. This parameter defines the location of the private key file; to create a certificate use either the mmssl command (Section 14.1.2) or the function x509newcrt. If no value has been assigned to this parameter the default key file getparam("ssl_dir")+"/server.key" will be used by the server.

**Affects routines**
https_startsrv

**See also**
https_srv.crt, mmssl

**Module**
mmssl

### https_trustsrv

**Description**
This parameter decides whether the HTTPS client should trust servers without checking their certificates.

**Type**
Boolean, read/write

**Default value**
false

**Note**
When this parameter is false (the default) whenever an HTTPS connection is opened (via httpget for instance) the authenticity of the remote server is checked using the list of trusted certification authorities (as defined by the control parameter https_cacerts) and the operation is aborted if the verification fails. Changing the value of this parameter disables this test.

**Affects routines**
httpget, httppost, httpdel, httpput

**See also**
https_cacerts

**Module**
mmssl

### ssl_cipher

**Description**
Name of symmetric cipher to use when no algorithm is specified.

**Type**
String, read/write

**Default value**
"AES-128-CBC"

**Note**
This parameter defines the default symmetric cipher used by the crypt I/O driver. The name of a cipher consists in up to 3 components separated by the "-" symbol: the algorithm name (e.g. aes, bf, des), the key size (when the algorithm may be used with different sizes of keys) and the block chaining mode (e.g. cbc, cfb1, cfb8, ecb, ofb). For instance, "des-ofb" designates DES with Output Feedback chaining. Use the command mmssl list cipher to get a full list of the supported cipher names.
ssl_digest

**Description**
Name of message digest to use when no algorithm is specified.

**Type**
String, read/write

**Default value**
"SHA256"

**Note**
This parameter defines the default message digest algorithm used by the crypt I/O driver, msgdigest, msgsing and msgverify.
Use the command mmssl list digest to get a full list of the supported names.

**Affects routines**
msgdigest, msgsing, msgverify, I/O driver "crypt:" (Section 14.4.3)

**Module**
mmssl

ssl_dir

**Description**
This parameter is the path to the configuration directory of mmssl. Its content is used by both the mmssl routines and the Mosel core libraries for handling signed and encrypted bim files.

**Type**
String, read only

**Note**
By default this location is the path "/$HOME/.mmssl" (on Unix systems) or "%USERPROFILE%\.mmssl" (on Windows). Assuming the active restrictions do not prevent the operation, this directory will be created if it does not exist at the time of loading the module. It is also possible to select a different location by defining the environment variable MOSEL_SSL (in this case, the directory is not automatically created and must be available at loading time).

The configuration directory should contain the following entries:

- **personal.key**
  RSA private key of the user: it is used for signing documents to be published and for decrypting documents that have been encrypted with the corresponding public key.

- **personal**
  RSA public key of the user: to be provided with documents signed with personal.key such that recipients can check the signature. The public key is also used to encrypt documents to be decrypted with personal.key.

- **pubkeys**
  public keys repository: this directory is the default location where public keys are searched for checking the signature of a document.

- **ca-bundle.crt**
  trusted certificates file: mmhttp uses this file when checking authenticity of servers (HTTPS client) or clients (HTTPS server).

- **server.crt**
  HTTPS server certificate: this file is required by the HTTPS server of mmhttp together with the corresponding private key.
server.key HTTPS server private key: this file is required by the HTTPS server of mmhttp together with the corresponding certificate.

The program mmssl can be used to create and populate this directory (Section 14.1.2).

Even if Mosel is run under restrictions, mmssl can still access its configuration directory for getting public keys stored under the pubkeys directory, read the file of trusted certificates ca-bundle.crt and load the private key personal.key to decrypt a document. However, the module requires explicit read access to use the private key personal.key for signing tasks and load the HTTPS server configuration (files server.key and server.crt).

Module mmssl

ssl_privkey

Description Name of the file holding the user’s private key.

Type String, read/write

Note The key identified by this parameter is used when a required private key is not provided. If no value has been assigned to this parameter the default key file getparam("ssl_dir")+/personal.key" will be used.

Affects routines msgsign, x509newcrt, RSAprivdecrypt, BIM file signing and encryption

Module mmssl

14.3 Procedures and functions

msgdigest Compute the message digest of a document.  p. 365
msgsign Compute the digital signature of a document.  p. 366
msgverify Verify the digital signature of a document.  p. 367
RSAfingerprint Generate the fingerprint of an RSA key.  p. 357
RSAgenkey Create a new RSA key pair.  p. 358
RSAgetkeysizé Get the size of an RSA key.  p. 359
RSAisprivate Check whether an RSA key is private.  p. 360
RSAloadkey Load an RSA key file into memory.  p. 361
RSAprivdecrypt Decrypt a document using an RSA private key.  p. 362
RSApubencrypt Encrypt a document using an RSA public key.  p. 363
RSASavekey Save an RSA key to a file.  p. 364
sslivsize Get the size of the initialisation vector of a cipher.  p. 368
sslkeysizé Get the size of the key required by a symmetric cipher.  p. 369
sslmdsize Get the size of a message digest.  p. 370
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>sslrandom</td>
<td>Generate a random number.</td>
<td>371</td>
</tr>
<tr>
<td>sslrandomdata</td>
<td>Generate a random data file.</td>
<td>372</td>
</tr>
<tr>
<td>x509check</td>
<td>Check the compatibility of a private key with an X509 certificate.</td>
<td>373</td>
</tr>
<tr>
<td>x509getinfo</td>
<td>Retrieve information stored in an X509 certificate.</td>
<td>374</td>
</tr>
<tr>
<td>x509newcrt</td>
<td>Create a new self-signed X509 certificate.</td>
<td>375</td>
</tr>
</tbody>
</table>
RSAfingerprint

Purpose
Generate the fingerprint of an RSA key.

Synopsis
function RSAfingerprint(key:array(range) of integer):text
function RSAfingerprint(kfile:string):text

Arguments
key RSA key in the form of an array of integer
kfile File containing the key

Return value
Fingerprint as a text string of hexadecimal digits.

Further information
1. *mmssl* uses an MD5 hash of the public part of the RSA key as its fingerprint. The return value of this function is therefore a text string of 32 hexadecimal digits that characterises a given key.
2. The function can process both public and private keys either directly from a key file or from an array of integers (as produced by *RSAloadkey* or *RSAgenkey*).

Related topics
*RSAloadkey*

Module
*mmssl*
RSAgencode

Purpose
Create a new RSA key pair.

Synopsis
function RSAgenkey(size:integer, key:array(range) of integer):integer
function RSAgenkey(size:integer, kfile:string):integer

Arguments
size Size of the key to generate (in bits, must be at least 1024)
key Array to store the new key
kfile File where to save the key

Return value
Number of integers (first syntax) or size of the file (second syntax) or -1 in case of I/O error and -2 if the provided array is not suitable to store the key.

Further information
1. The generated key can be retrieved either as an array of integers or directly saved into a file. In both cases, the public key may be extracted using RSAsavekey.
2. The function creates keys of at least 1024 bits: a request for a key of a smaller size will result in a 1024 bits key.

Related topics
RSAloadkey, RSAsavekey

Module
mmssl
RSAGetkeysize

**Purpose**

Get the size of an RSA key.

**Synopsis**

```pascal
function RSAGetkeysize(key:array(range) of integer):integer
function RSAGetkeysize(kfile:string, ispriv:boolean):integer
```

**Arguments**

- `key` RSA key in the form of an array of integer
- `kfile` File containing the key
- `ispriv` Must be `true` if the key file contains a private key

**Return value**

Size of the key (number of bits) or `-1` in case of an error.

**Further information**

A return value of `-1` indicates an error condition. Typically this will occur if the file cannot be accessed or the `ispriv` parameter is not correct (e.g. `ispriv` is `true` and the file is a public key).

**Related topics**

- RSAPrivate, RSAloadkey

**Module**

`mmssl`
RSAisprivate

Purpose
Check whether an RSA key is private.

Synopsis
function RSAisprivate(key:array(range) of integer):boolean
function RSAisprivate(kfile:string):boolean

Arguments
key RSA key in the form of an array of integer
kfile File containing the key

Return value
true if the key is an RSA private key, false otherwise.

Further information
A return value of false does not necessarily indicate that the provided data corresponds to a valid public key: this value is also returned in the case of an I/O error (e.g. the file does not exist).

Related topics
RSAloadkey

Module
mmssl
RSALoadkey

Purpose
Load an RSA key file into memory.

Synopsis
function RSALoadkey(key:array(range) of integer, kfile:string, ispriv:boolean):integer
function RSALoadkey(key:array(range) of integer, kfile:string):integer

Arguments
key RSA key in the form of an array of integer
kfile File containing the key
ispriv Must be true if the key file contains a private key

Return value
The number of integers saved into the array, or -1 in the case of an I/O error, or -2 if the provided array is not suitable to store the key.

Further information
If the ispriv parameter is not provided, the function calls first RSAisprivate to determine its value.

Related topics
RSASavekey

Module
mmssl
RSAprivdecrypt

Purpose
Decrypt a document using an RSA private key.

Synopsis
function RSAprivdecrypt(kfile:string, src:string, dest:string):integer

Arguments
- **kfile**: File containing the private key
- **src**: Name of the file to decrypt
- **dst**: Name of the file to store the decrypted document

Return value
Length of the resulting document or -1 in the case of an error.

Further information
This function is used to decrypt a document that has been encrypted using RSApubencrypt. It requires the private part of the key used for encryption.

Related topics
- RSApubencrypt

Module
- mmssl
RSApubencrypt

**Purpose**
Encrypt a document using an RSA public key.

**Synopsis**
```
function RSApubencrypt(kfile:string, src:string, dest:string):integer
```

**Arguments**
- **kfile**: File containing the public key
- **src**: Name of the file to encrypt
- **dst**: Name of the file to store the encrypted document

**Return value**
Length of the resulting document or \(-1\) in the case of an error.

**Further information**
1. This function can be used to encrypt a document using an RSA public key. Decryption will be done using function `RSAprivdecrypt` with the help of the corresponding RSA private key.

2. The algorithm used here cannot handle documents larger than \((\text{RSAgetkeysize}(\text{kfile})/8-41)\) bytes. Typically, encryption of larger documents will be performed with a symmetric cipher (see crypt I/O driver, Section 14.4.3) using a randomly generated key (that can be produced with `sslrandomdata`), in which case the asymmetric cipher is used to encrypt only this random key. The decryption then also operates in two steps: the key is first decrypted using `RSAprivdecrypt` (with a private key) and after this the document can be restored from the decrypted symmetric key.

3. If the key file name does not include an explicit path (e.g. "somekey"), it is searched for in the default public keys directory located at `getparam("ssl_dir")+/pubkeys` instead of the current working directory. It is required to prefix the key file name with "./" in order to access a key file from the current directory (e.g. "./somekey").

**Related topics**
- `RSAprivdecrypt`

**Module**
- `mmssl`
RSAsavekey

**Purpose**
Save an RSA key to a file.

**Synopsis**

```plaintext
function RSAsavekey(key:array(range) of integer, kfile:string, ispriv:boolean):integer
function RSAsavekey(key:array(range) of integer, kfile:string):integer
```

**Arguments**

- **key**: RSA key in the form of an array of integer
- **kfile**: Destination file
- **ispriv**: Save the private key if `true`, only the public key otherwise

**Return value**
A positive value on success or `-1` in case of error.

**Example**

In the code below a new 2048 bits key is generated and both, private and public parts are saved into different files:

```plaintext
if RSgenkey(2048,k)<=0 then
  writeln("Failed to create RSA key")
elif RSsavekey(k,"perso.key",true)<1 or
  RSsavekey(k,"perso",false)<1 then
  writeln("Failed to save key file")
end-if
```

**Further information**

1. This function saves the RSA key that is provided as an array of string into a file in a textual representation. The `ispriv` parameter can be used to select which part of the key to export.
2. If the `ispriv` parameter is not provided, the function will produce a private key file if the key is private and a public key file otherwise.

**Related topics**

- RSAloadkey, RSgenkey

**Module**

```
mmssl
```
**msgdigest**

**Purpose**
Compute the message digest of a document.

**Synopsis**
```
function msgdigest(mdalg:string, fname:string, mdf:string):integer
function msgdigest(fname:string, mdf:string):integer
```

**Arguments**
- **mdalg** Name of the algorithm to use
- **fname** Name of the file to be processed
- **mdf** File where to store the digest

**Return value**
Size of the message digest in bytes or -1 if case of error.

**Example**
The following procedure implements the command ‘md5sum’:
```
procedure md5sum(f:string)
  if msgdigest("md5",f,"mem:dgst")<>16 then
    writeln("Failed to compute digest")
  else
    fcopy("mem:dgst",F_BINARY,"hex:" ,F_TEXT)
    writeln(" ",f)
  end-if
end-procedure
```

**Further information**
1. This function computes a message digest (MD) using either the algorithm specified by the **mdalg** argument or the default algorithm as defined by the control parameter **ssl_digest**. The produced output takes the form of a binary file the size of which is returned by the function.

2. The set of supported algorithms includes "md5", "sha", and "sha256". For a full list use the command **mmssl list digest**.

**Related topics**
- sslmdsize

**Module**
- mmssl
**msgsigh**

**Purpose**
Compute the digital signature of a document.

**Synopsis**

```plaintext
function msgsigh(mdalg:string, pkey:string, fname:string, sgf:string):integer
function msgsigh(fname:string, sgf:string):integer
```

**Arguments**

- **mdalg** Name of the message digest algorithm to use
- **pkey** Name of the private key file to use for signing
- **fname** File to sign
- **sgf** File where the signature is to be saved

**Return value**
Length of the signature or -1 in the case of an error.

**Further information**

1. This function computes the digital signature of a document by encrypting the message digest of its input file using an RSA private key. The resulting signature can be verified with the function `msgverify` used with the appropriate public key.

2. If no message digest algorithm is specified, the default algorithm defined by the control parameter `ssl_digest` is used. Unless a specific key file is selected, the default private key defined by the control parameter `ssl_privkey` or, (if this parameter is not defined) the key under `getparam("ssl_dir")+"/personal.key"` is used.

**Related topics**

- `msgverify`

**Module**

- `mmssl`
msgverify

**Purpose**
Verify the digital signature of a document.

**Synopsis**
function msgverify(mdalg:string, key:string, fname:string, sgf:string):integer
function msgverify(key:string, fname:string, sgf:string):integer

**Arguments**
- **mdalg**: Name of the message digest algorithm to use
- **key**: Name of the public key file to use
- **fname**: File to verify
- **sgf**: Signature used for the verification

**Return value**
1 if the signature is valid, 0 if the verification failed and -1 in the case of an error.

**Further information**
1. This function verifies the digital signature of a document by comparing the message digest of the document with the information obtained by decrypting the provided signature with a given RSA public key. Typically this signature has been obtained with the function `msgsign` and the appropriate private key.
2. If no message digest algorithm is specified, the default algorithm defined by the control parameter `ssl_digest` is used. Note that the same algorithm has to be used for both signing and verifying.
3. If the key file name does not include an explicit path (e.g. "somekey"), it is searched for in the default public keys directory located at `getparam("ssl_dir")+/pubkeys` instead of the current working directory. It is required to prefix the key file name with "./" in order to access a key file from the current directory (e.g. "./somekey").

**Related topics**
`msgsign`

**Module**
mmssl
**sslivsize**

**Purpose**
Get the size of the initialisation vector (IV) required by a symmetric cipher.

**Synopsis**
function sslivsize(cipalg:string):integer

**Argument**
cipalg Name of the cipher to consider

**Return value**
Size of a IV in bytes or -1 if the cipher is not supported.

**Example**
The following statement generates a random IV for the default cipher algorithm:

```
sslrandomdata("myiv",sslivsize("")
```

**Further information**
Some encryption algorithms require an initialisation vector (IV) in addition to the encryption key. Like the key, the IV is an array of bytes of a fixed size. This function returns the length (in bytes) of the IV required by a given symmetric cipher algorithm. A return value of -1 indicates an unrecognised algorithm name: this property can be used to check whether a given algorithm is available.

**Related topics**
sslkeys

**Module**
mmssl
sslkeyszie

Purpose
Get the size of the key required by a symmetric cipher.

Synopsis
function sslkeyszie(cipalg:string):integer

Argument
cipalg Name of the cipher to consider

Return value
Size of a key in bytes or -1 if the cipher is not supported.

Further information
This function returns the length (in bytes) of an encryption key required by a given symmetric cipher algorithm. A return value of -1 indicates that the algorithm name has not been recognised: this property can be used to check whether a given algorithm is available.

Related topics
sslivsize

Module
mmssl
sslmdsize

Purpose
Get the size of a message digest.

Synopsis
function sslmdsize(mdalg:string):integer

Argument
mdalg Algorithm to consider

Return value
Size of the message digest in bytes or -1 if the algorithm is not supported.

Further information
This function returns the length (in bytes) of a digest produced by the requested message digest algorithm. A return value of -1 indicates that the algorithm name has not been recognised: this property can be used to check whether a given algorithm is available.

Related topics
msgdigest

Module
mmssl
sslrandom

Purpose
Generate a random number.

Synopsis
function sslrandom:integer

Return value
A randomly generated integer.

Further information
This function returns an integer by combining 4 bytes obtained from a cryptographically strong pseudo-random generator.

Related topics
sslrandomdata

Module
mmssl
sslrandomdata

**Purpose**
Generate a random data file.

**Synopsis**
```
procedure sslrandomdata(fname:string, size:integer)
```

**Arguments**
- **fname** Name of the file where to save the generated data
- **size** Number of bytes to generate

**Example**
The following statement generates a random key for the default cipher algorithm:
```
sslrandomdata("mykey", sslkeysize(""))
```

**Further information**
This function generates `size` bytes from a cryptographically strong pseudo-random generator that it saves in the specified file `fname`.

**Related topics**
sslrandom

**Module**
mmssl
x509check

Purpose
Check the compatibility of a private key with an X509 certificate.

Synopsis
function x509check(x509:string, kfile:string):integer

Arguments
x509 File containing the certificate
kfile File containing the private key

Return value
0 if the key is compatible with the certificate, 1 if the key is not compatible and -1 in the case of an error.

Further information
This function checks whether the public key recorded in the specified certificate corresponds to the provided private key (the certificate can only be used by the owner of the public key).

Related topics
x509getinfo

Module
mmssl
x509getinfo

**Purpose**
Retrieve information stored in an X509 certificate.

**Synopsis**
```plaintext
function x509getinfo(x509:string, info:array(string) of text):integer
```

**Arguments**
- `x509` Certificate file
- `info` Array where to store certificate information

**Return value**
Number of items stored in the array or -1 in case of error.

**Example**
The example below shows how to display the properties of a certificate:

```plaintext
declarations
    info:array(S:set of string) of text
end-declarations

if x509getinfo("srv.crt",info)<1 then
    writeln("Failed to load certificate")
else
    forall(s in S | exists(info(s)))
        writeln(" ", s, ":", info(s))
end-if
```

**Further information**
This function retrieves some of the information recorded in an X509 certificate. The data is recorded in the provided array indexed by the labels of the records in the certificate. The possible labels are:
- **Version** Format version of the certificate
- **Serial** Serial number
- **Issuer** Issuer of the certificate
- **Subject** Entity associated to the public key stored in the certificate
- **NotBefore** Valid after this date
- **NotAfter** Valid until this date
- **SgnAlg** Algorithm used to sign the certificate

A self-signed certificate (such as those created with `x509newcrt`) will have identical values for **Issuer** and **Subject**.

**Related topics**
- x509check

**Module**
- mmssl
x509newcrt

Purpose
Create a new self-signed X509 certificate.

Synopsis
function x509newcrt(x509:string, kfile:string, info:array(string) of text):integer

Arguments
x509  Certificate file to create
kfile  File containing the private key
info   Array describing the certificate properties

Return value
0 if success or -1 in the case of an error.

Example
The following example creates a certificate that is valid for 3 years, using a new RSA key:

```
info("Version"):="1"
info("Serial"):="123456789"
info("Duration"):="text(365*3)"
info("C"):="FR"
info("O"):="My Company"
info("CN"):="www.mycomp.com"
if RSAgenkey(1024,"srv.key")<=0 then
    writeln("Failed to create RSA key")
elsif x509newcrt("srv.crt","srv.key",info)<0 then
    writeln("Failed to create certificate")
end-if
```

Further information
1. This function creates a self-signed X509 certificate. Such a certificate can be used to run an HTTPS
server but clients of such a server have to disable server certificate verification (see ssl_trustsrv)
or include this certificate in their trusted certificate file (see https_cacerts).

2. The routine expects an array with indices defining the following entries (a default value applies if
the entry is missing):
   - Version  Format version of the certificate (default: 1)
   - Serial   Serial number (default: 1)
   - Duration  Validity (in days) from the current date (default: 365)
   - C   Country code (default: system country or ‘EU’)
   - O   Organisation name (default: anonymous)
   - CN  Common Name (typically the host name to authenticate, default: localhost)
The entries C, O and CN are used to generate the Issuer and Subject records of the certificate. The
provided key is used both as the certificate key (using the public part of the key) and as the
signing key.

Related topics
x509check, x509getinfo

Module
mmssl
14.4 I/O drivers

The mmssl module publishes two drivers for converting binary documents to textual representation and a driver dedicated to symmetric encryption. These drivers have the same behaviour: encryption or encoding is performed when the driver is used for writing while decryption/decoding is done on a stream that is open for reading.

14.4.1 Driver base64

base64: [nonl,] filename

This driver can be used to handle documents encoded using the base64 standard. When used in an output stream, it generates the base64 encoded version of its binary input and in an input stream it expects a base64 encoded document that it decodes.

For instance the following statement encodes "mydata.bin":

```plaintext
fcopy("mydata.bin",F_BINARY,"mmssl.base64:mydata.b64",F_TEXT)
```

By default the generated text is split into lines of 76 characters but with the option nonl the entire document is output on a single line.

14.4.2 Driver hex

hex: filename

This driver produces a textual representation of a binary document by replacing each byte by its hexadecimal representation (e.g. the value 13 is converted to the string "0d").

The following code extract displays the hexadecimal representation of the binary input file "mem:md5":

```plaintext
fcopy("mem:md5",F_BINARY,"mmssl.hex:",F_TEXT)
writeln
```

14.4.3 Driver crypt

crypt: [[nosalt,][md=a,][cipher=c,][key=kf,][iv=if,][pwd=p|pf]] filename

The crypt driver performs encryption (when writing) or decryption (when reading) of its stream using a symmetric cipher (that is, the same key is used for encryption and decryption). Options are provided enclosed in square brackets, at the least a password has to be provided. For instance, the following statement encrypts the file "mydata" using the password stored in the file "passfile":

```plaintext
fcopy("mydata","mmssl.crypt:[passfile]mydata.enc")
```

The password is read from the first line of the password file (that is opened as a text document). Alternatively, the password may be directly passed through the file name using the pwd= option:

```plaintext
fcopy("mydata","mmssl.crypt:[pwd=mysecret]mydata.enc")
```

Encryption (or decryption) is performed using the default cipher as defined by the control parameter ssl_cipher. Another cipher can be selected using the cipher option.
The encryption (or decryption) process requires a key as well as an initialisation vector. The size of these components depends on the selected cipher and the appropriate data is generated by a key derivation routine using the provided password as input. This procedure employs a message digest algorithm and may use some initial value (or salt). Without any specific option, the driver relies on the default message digest algorithm defined by the control parameter ssl_digest and generates a random salt of 8 bytes. These bytes are then saved at the beginning of the encrypted document so that the decryption process can retrieve them and regenerate the encryption key and initialisation vector from the provided password. This default behaviour can be changed using the nosalt option to avoid using a salt and the option md to select some other message digest algorithm. It is also possible to provide the encryption key and the initialisation vector via dedicated files using options key and iv. In this case no password has to be provided.
The \textit{mmsvg} package provides a set procedures which allow users to display graphs of functions, diagrams, networks, various shapes \textit{etc.} in SVG format. To use this module the following line must be included in the header of the Mosel model file:

\begin{verbatim}
uses "mmsvg"
\end{verbatim}

\textit{mmsvg} requires a webbrowser in order to be able to display graphics. Running a Mosel model that uses the \texttt{svgrefresh} routine provided by this module opens a window in the default browser that is configured on the system. In the absence of a webbrowser, it is still possible to generate graphics and save them to file via \texttt{svgsave}.

\section{SVG graph structure}

The SVG graph format is an XML format, that is, the elements of a graph are organized in a hierarchical tree structure. \textit{mmsvg} structures graphical objects in three levels:

\begin{enumerate}
\item SVG graph
\item object group
\item graphical object
\end{enumerate}

Each individual graphical object (line, polygon, text \textit{etc.}) must be created within an \textit{object group}. By default this is the last group that has been added to the graph, but some other object reference can be stated. A default graph object is always present and object groups are created within this default graph.

\subsection{Object groups}

Object groups are identified via a string ID that is specified by the user at their creation, this ID must be unique. Each object group receives an entry in the \textit{legend} of the graph. Typically a group serves to represent a collection of graphical objects that are logically related. The style defined for a group is applied to all its objects unless it is overwritten by individual settings, meaning that it is usually more efficient to state generally valid style settings for an entire group instead of repeating them for each individual object.

At the creation of a group, optionally a \textit{group color} can be specified. If no color is given, then a default color will be selected from a built-in list of color values.

Graphical objects are displayed in the order of definition of object groups, and within each group in the order of their definition.
15.1.2 SVG styling

Style definitions can be applied to all levels of SVG elements, to the graph, object groups, or for individual objects. `mmsvg` defines a set of property constants but other SVG styling options can equally be used by directly stating their name in the `svgset[graph]style` routines. For a complete list of SVG style properties and their permissible values the reader is referred to the SVG property specifications at https://www.w3.org/TR/SVG/propidx.html.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVG_COLOR</td>
<td>Default color name (for object groups)</td>
</tr>
<tr>
<td>SVG_DECORATION</td>
<td>Text decoration; possible values include ‘none’, ‘underline’, ‘overline’, ‘line-through’, ‘blink’</td>
</tr>
<tr>
<td>SVG_FILL</td>
<td>Fill color name</td>
</tr>
<tr>
<td>SVG_FILLOPACITY</td>
<td>Fill opacity; values between 0.0 and 1.0</td>
</tr>
<tr>
<td>SVG_FONT</td>
<td>Whitespace separated list of font settings</td>
</tr>
<tr>
<td>SVG_FONTFAMILY</td>
<td>Font family definition; this can be generic families (‘serif’, ‘sans-serif’, ‘cursive’, ‘fantasy’, ‘monospace’) or specific font names</td>
</tr>
<tr>
<td>SVG_FONTSIZE</td>
<td>Font size; constants (‘xx-small’, ‘x-small’, ‘small’, ‘medium’, ‘large’, ‘x-large’, ‘xx-large’) or percentage value or length (e.g. in ‘em’, ‘pt’, ‘px’, ‘cm’)</td>
</tr>
<tr>
<td>SVG_FONTSTYLE</td>
<td>Font style; values ‘normal’, ‘italic’, ‘oblique’</td>
</tr>
<tr>
<td>SVG_FONTWEIGHT</td>
<td>Font weight; numbers 100,...900 or constants (‘bold’, ‘bolder’, ‘lighter’, ‘normal’)</td>
</tr>
<tr>
<td>SVG_OPACITY</td>
<td>Generic opacity setting; values between 0.0 and 1.0</td>
</tr>
<tr>
<td>SVG_STROKE</td>
<td>Color for lines and borders</td>
</tr>
<tr>
<td>SVG_STROKEDASH</td>
<td>Line style; comma-separated list of lengths or percentages specifying lengths of alternating dashes and gaps</td>
</tr>
<tr>
<td>SVG_STROKEOPACITY</td>
<td>Stroke opacity, values between 0.0 and 1.0</td>
</tr>
<tr>
<td>SVG_STROKEWIDTH</td>
<td>Stroke width; percentage or length</td>
</tr>
<tr>
<td>SVG_TEXTANCHOR</td>
<td>Vertical alignment of text; possible values include ‘start’, ‘middle’, ‘end’</td>
</tr>
</tbody>
</table>

Other predefined constants are `SVG_CURRENT` for the current color and `SVG_NONE`.

`mmsvg` defines the following color constants (applicable to the properties `SVG_COLOR`, `SVG_FILL`, `SVG_STROKE`) that can be used in place of SVG color keywords or color definitions generated via the `svgcolor` routine:

- `SVG_BLACK`, `SVG_BLUE`, `SVG_BROWN`, `SVG_CYAN`, `SVG_GOLD`, `SVG_GRAY`, `SVG_GREEN`, `SVG_LIME`, `SVG_MAGENTA`, `SVG_ORANGE`, `SVG_PINK`, `SVG_PURPLE`, `SVG_RED`, `SVG_SILVER`, `SVG_WHITE`, `SVG_YELLOW`

For a full list of SVG color keywords and their definitions please see https://www.w3.org/TR/SVG/types.html.

The complete set of style properties specified for a graph, object group or individual objects can be retrieved via the routines `svggetstylesheet` and `svggetgraphstylesheet`, for example in order to copy them to some other object via `svgsetstylesheet` or `svgsetgraphstylesheet` respectively.
15.1.3 Interaction with the graphical display

The command `svgrefresh` sends the current graph and any additional files that might have been added to it (see `svgaddfile`) to the built-in server that handles the coordination with the display and triggers an update of the graphical display. The end of the model execution will also terminate the display, unless a call to the routine `svgwaitclose` is added at the end of the model, in which case the model waits for the closing of the display window.

Inserting a call to the routine `svgpause` into a model will pause its execution at this point until the user hits the 'Continue' button in the graphical display. Typically, this feature will be used to allow the user time for visual inspection of the output if a model iteratively generates graphics or updates to a graphic.

15.1.4 Example

The following example shows how to define a few simple graphical objects, saves the resulting graphic to a file and also displays it in a web browser. The model waits until the browser is closed.

```plaintext
model "svg example"
uses "mmsvg"

! ***** Line objects *****
svgaddgroup("gl", "Lines") ! Group with automatic color
svgaddline(10,10,250,10) ! Simple line with default style
PointList:=sum(i in 1..20)[i*10,40+round(20*random)]
svgaddline(PointList) ! Polyline
l:=svggetlastobj ! Retrieve object reference
svgsetstyle(l, SVG_STROKE, SVG_MAGENTA) ! Change line color
svgsetstyle(l, SVG_STROKEDASH, "1,1") ! Dotted line

! ***** Various shapes *****
svgaddgroup("gs", "Shapes", SVG_GREY) ! Group with user-defined color
svgaddrectangle(275,25,250,250) ! Draw a square
svgsetstyle(svggetlastobj, SVG_STROKEWIDTH, 3) ! Wider border
svgaddcircle(400,150,75) ! Draw a circle
svgaddstyle(svggetlastobj, SVG.FILL, SVG_CURRENT) ! Fill with group color
svgaddrectangle([200,400,200,350,300,300,500,350,600,350,600,400])
svgsetstyle(svggetlastobj, SVG.FILL, SVG_GREEN) ! Fill with user color

! ***** Pie chart *****
forall(i in 1..6) svgaddgroup("gp"+i,"Pie"+i) ! Pie slices with auto-colors
setrandseed(3); ttl:=0.0
forall(i in 1..5) do
  rd:=random/6
  svgaddpie("gp"+i, 150, 525, 100, ttl, ttl+rd)
ttl+=rd
end-do
svgaddpie("gp6", 150, 525, 100, ttl, 1)

! ***** Text objects *****
svgaddgroup("gt", "Text", SVG_BLACK) ! Group with user-defined color
svgaddtext(20, 100, "Text with default formatting")
svgaddtext(20, 120, "Formatted text")
t:=svggetlastobj
svgsetstyle(t, SVG.FONTSIZE, "20pt")
svgsetstyle(t, SVG.FONTSTYLE, "italic")
svgsetstyle(t, SVG.FONTWEIGHT, "bold")
svgsetstyle(t, SVG.COLOR, SVG_BLUE)
svgaddxmltext(20, 150, 'XML formatted text:
  <tspan font-size="large"> large</tspan>
  <tspan text-decoration="underline">underlined</tspan>
  <tspan stroke="red"> red</tspan>')

svgsetgraphviewbox(0,0,610,635) ! Optional: specify graph size
```

Fair Isaac Corporation Confidential and Proprietary Information 380
Figure 15.1: Graphical output produced by the example

15.2 Control parameters

The following parameters are defined by mmsvg:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>p.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMSVGDISPLAY</td>
<td>Enable/disable rendering.</td>
<td>381</td>
</tr>
<tr>
<td>MMSVG{TGZ}</td>
<td>Location of the mmsvg.tgz archive.</td>
<td>382</td>
</tr>
</tbody>
</table>

MMSVGDISPLAY

**Description**

When this parameter is set to true (the default) the first call to `svgrefresh` starts a web browser for displaying the current graph. Changing the value of this parameter disables rendering: after a warning message is reported calls to `svgrefresh` have no effect and the function `svgclosing` always returns true. Note that setting the environment variable `MMSVGDISPLAY` to a non empty string has the same effect as changing this control parameter.

**Type**

Boolean, read/write

**Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Enable rendering.</td>
</tr>
<tr>
<td>false</td>
<td>Disable rendering.</td>
</tr>
</tbody>
</table>

**Default value**

true
Affects routines  svgrefresh, svgclosing.
Module  mmsvg

MMSVGTGZ

Description  The function svgrefresh requires the archive mmsvg.tgz for its processing. By default this file is expected to be located in the same directory as the module mmsystem. This parameter makes it possible to specify an alternate location.

Type  String, read/write
Default value  ""
Affects routines  svgrefresh.
Module  mmsvg

15.3  Procedures and Functions

svgaddarrow  Add an arrow to an object group.  p. 385
svgaddcircle  Add a circle to an object group.  p. 386
svgaddellipse  Add an ellipse to an object group.  p. 387
svgaddfile  Add a file to a graph.  p. 388
svgaddgroup  Add a new object group to the user graph.  p. 384
svgaddimage  Add an image to an object group.  p. 389
svgaddline  Add a line or polyline to an object group.  p. 390
svgaddpie  Add a pie slice.  p. 391
svgaddpoint  Add a small square to mark a point.  p. 392
svgaddpolygon  Add a polygon to an object group.  p. 393
svgaddrectangle  Add a rectangle to an object group.  p. 394
svgaddtext  Add a text to an object group.  p. 395
svgaddxmltext  Add an XML formatted text to an object group.  p. 396
svgclosing  Test whether the display window is being closed.  p. 397
svgcolor  Compute a composite color.  p. 398
svgdelobj  Delete the specified graphical object.  p. 399
svgerase  Erase all object groups or the contents of a specific group.  p. 400
svggetgraphstyle  Retrieve a style property of a graph.  p. 401
svggetgraphstylesheet  Retrieve the style definitions of a graph.  p. 402
svggetgraphviewbox  Retrieve the viewbox definition of a graph.  p. 403
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>svggetlastobj</td>
<td>Retrieve the identifier of a graphical object.</td>
<td>404</td>
</tr>
<tr>
<td>svggetstyle</td>
<td>Retrieve a style property of a graphical object or object group.</td>
<td>405</td>
</tr>
<tr>
<td>svggetstylesheet</td>
<td>Retrieve style definitions of a graphical object or object group.</td>
<td>406</td>
</tr>
<tr>
<td>svgpause</td>
<td>Suspend the execution of a model.</td>
<td>407</td>
</tr>
<tr>
<td>svgrefresh</td>
<td>Refresh the graph display.</td>
<td>408</td>
</tr>
<tr>
<td>svgsave</td>
<td>Save a graph to a file.</td>
<td>409</td>
</tr>
<tr>
<td>svgsetgraphlabels</td>
<td>Set x- and y-axis labels for a graph.</td>
<td>410</td>
</tr>
<tr>
<td>svgsetgraphpointsize</td>
<td>Set point size property for a graph.</td>
<td>411</td>
</tr>
<tr>
<td>svgsetgraphscale</td>
<td>Set scaling value for a graph.</td>
<td>412</td>
</tr>
<tr>
<td>svgsetgraphstyle</td>
<td>Set a style property of a graph.</td>
<td>413</td>
</tr>
<tr>
<td>svgsetgraphstylesheet</td>
<td>Set the style definitions for a graph.</td>
<td>414</td>
</tr>
<tr>
<td>svgsetgraphviewbox</td>
<td>Set the visible area for a user graph.</td>
<td>415</td>
</tr>
<tr>
<td>svgsetreffreq</td>
<td>Set the refresh frequency for a graph.</td>
<td>416</td>
</tr>
<tr>
<td>svgsetstyle</td>
<td>Set a style property for a graphical object or object group.</td>
<td>417</td>
</tr>
<tr>
<td>svgsetstylesheet</td>
<td>Set the style for a graphical object or object group.</td>
<td>418</td>
</tr>
<tr>
<td>svgshowgraphaxes</td>
<td>Force displaying of graph axes.</td>
<td>419</td>
</tr>
<tr>
<td>svgwaitclose</td>
<td>Delay model termination.</td>
<td>420</td>
</tr>
</tbody>
</table>
svgaddgroup

**Purpose**
Add a new object group to the user graph.

**Synopsis**

```plaintext
procedure svgaddgroup(gid: string, desc: text, color: text)
procedure svgaddgroup(gid: string, desc: text)
```

**Arguments**

- **gid**
  Object group ID (must be unique within a graph).
- **desc**
  A text that will appear in the legend.
- **color**
  A color specification obtained using `svgcolor` or one of the predefined constants (see list in section 15.1.2).

**Example**

The following adds two groups ‘g1’ and ‘g2’ to the user graph:

```plaintext
svgaddgroup("g1", "sine", SVG_RED) ! User-specified group color
svgaddgroup("g2", "random numbers") ! Automatically selected color
```

**Further information**

1. A group is identified by its ID whereas the ‘desc’ serves as text for the legend of the graphic. A group contains any number of individual objects (points, lines, arrows, texts etc.) which were added to it.

2. An empty string for the ‘desc’ attribute indicates that the group is not to be included in the legend.

3. If no color is specified at the creation of a group it will be assigned a default color from a built-in list. This setting can be overwritten for individual objects within the group. Note that any style settings that are common to a large number of objects within a group should preferably be specified for the group rather than for the individual objects.

**Related topics**

- `svgsetstyle`
**svgaddarrow**

**Purpose**
Add an arrow to an object group.

**Synopsis**

```plaintext
procedure svgaddarrow(gid: string, x1: real, y1: real, x2: real, y2: real)
procedure svgaddarrow(x1: real, y1: real, x2: real, y2: real)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gid</td>
<td>Object group ID.</td>
</tr>
<tr>
<td>x1</td>
<td>The x coordinate of the first point.</td>
</tr>
<tr>
<td>y1</td>
<td>The y coordinate of the first point.</td>
</tr>
<tr>
<td>x2</td>
<td>The x coordinate of the second point.</td>
</tr>
<tr>
<td>y2</td>
<td>The y coordinate of the second point.</td>
</tr>
</tbody>
</table>

**Example**

The following adds two arrows to a group named ‘thetime’. The arrows suggest three o’clock:

```plaintext
svgaddgroup("arrows", "thetime", SVG_BLACK)
svgaddarrow("arrows", 0, 0, 4.5, 0)
svgaddarrow(0, 0, 0, 5)
svgsetgraphviewbox(-5, -6, 10, 12)
```

**Further information**

1. The arrow connects the two points whose coordinates are given as parameters, pointing to the second one.

2. If no group ID is specified, the arrow is added to the last group that has been created.
svgaddcircle

**Purpose**
Add a circle to an object group.

**Synopsis**
procedure svgaddcircle(gid: string, x: real, y: real, r: real)
procedure svgaddcircle(x: real, y: real, r: real)

**Arguments**
gid  Object group ID.
x    The x coordinate of the center point.
y    The y coordinate of the center point.
r    The length of the radius of the circle.

**Example**
The following code draws a filled, semi-transparent circle centered at the origin with a radius of 10.

```plaintext
declarations
  circ: integer
end-declarations

svgaddcircle(0, 0, 10)
circ:=svggetlastobj
svgsetstyle(circ, SVG_FILL, SVG_CYAN)
svgsetstyle(circ, SVG_OPACITY, 0.5)
```

**Further information**
If no group ID is specified, the circle is added to the last group that has been created.
svgaddellipse

**Purpose**
Add an ellipse to an object group.

**Synopsis**

```plaintext
procedure svgaddellipse(gid: string, x: real, y: real, rx: real, ry: real)
procedure svgaddellipse(x: real, y: real, rx: real, ry: real)
```

**Arguments**

- **gid**  
  Object group ID.
- **x**  
  The x coordinate of the center point of the ellipse.
- **y**  
  The y coordinate of the center point of the ellipse.
- **rx**  
  The horizontal radius.
- **ry**  
  The vertical radius.

**Example**

The following code draws a very "flat" ellipse centered at the origin filled with the group color.

```plaintext
svgaddellipse(0,0,5,0.5)
svgsetstyle(svggetlastobj, SVG_FILL, SVG_CURRENT)
```

**Further information**

If no group ID is specified, the ellipse is added to the last group that has been created.
**svgaddfile**

**Purpose**
Add a file to a graph.

**Synopsis**
```
procedure svgaddfile(fname: string, fid: string)
```

**Arguments**
- `fname`  Filename (including path) of the file to be included.
- `fid` Name for the file used within the SVG graph.

**Example**
The following code adds an image file to the current graph and displays it in an area with corner points at the coordinates (100,100) and (250,250).
```
svgaddfile("./someimage.png", "myimg.png")
svgaddimage("myimg.png", 100, 100, 150, 150)
```

**Further information**
1. This routine is typically used in combination with `svgaddimage` to associate some external file with the graph.
2. Using an empty file name `fname` will remove the corresponding `fid` from the file database.

**Related topics**
```
svgaddimage.
```
svgaddimage

**Purpose**
Add an image to an object group.

**Synopsis**
```plaintext
procedure svgaddimage(gid: string, fid: text, x: real, y: real, w: real, h: real)
procedure svgaddimage(fid: text, x: real, y: real, w: real, h: real)
```

**Arguments**
- **gid**  
  Object group ID.
- **fid**  
  Name for the file used within the SVG graph.
- **x**  
  The x coordinate of the lower left corner.
- **y**  
  The y coordinate of the lower left corner.
- **w**  
  The width of the image.
- **h**  
  The height of the image.

**Example**
The following code adds an image file to the current graph and displays it 3 times at different positions (3 squares forming a row).

```plaintext
svgaddfile("./someimage.png", "myimg.png")
forall(i in 1..3)
    svgaddimage("myimg.png", 100*i, 100, 100, 100)
```

**Further information**
1. Any external file to be displayed within a graph needs to be associated with the graph via a call to `svgaddfile`
2. If no group ID is specified, the image is added to the last group that has been created.

**Related topics**
- `svgaddfile`
svgaddline

Purpose
Add a line or polyline to an object group.

Synopsis
procedure svgaddline(gid: string, x1: real, y1: real, x2: real, y2: real)
procedure svgaddline(x1: real, y1: real, x2: real, y2: real)
procedure svgaddline(gid: string, points: list of integer|real)
procedure svgaddline(points: list of integer|real)

Arguments
- **gid**: Object group ID.
- **x1**: The x coordinate of the first point.
- **y1**: The y coordinate of the first point.
- **x2**: The x coordinate of the second point.
- **y2**: The y coordinate of the second point.
- **points**: A list of points.

Example
The following code draws the outline of a triangle, given the correct aspect ratio of the user graph.

```
svgaddgroup("t", "triangle", SVG_ORANGE)
svgaddline([-2, -2, 0, 2, 2, -2, -2, -2])
svgsetgraphviewbox(-5, -5, 10, 10)
```

If the shape is to be filled (here: using the group color), you need to use polygon drawing instead of a polyline:

```
svgaddpolygon([-2, -2, 0, 2, 2, -2])
svgsetstyle(svggetlastobj, SVG_FILL, SVG_CURRENT)
```

Further information
1. The line connects the two points whose coordinates are given as parameters or the points contained in the specified list in their order of appearance in the list.
2. If no group ID is specified, the line is added to the last group that has been created.

Related topics
- **svgaddpolygon**.
**svgaddpie**

**Purpose**
Add a filled pie slice at the given coordinates.

**Synopsis**

```plaintext
procedure svgaddpie(gid: string, x: real, y: real, r: real, p1: real, p2: real)
procedure svgaddpie(x: real, y: real, r: real, p1: real, p2: real)
```

**Arguments**

- **gid**: Object group ID.
- **x**: The x coordinate of the center point.
- **y**: The y coordinate of the center point.
- **r**: Radius (side length of the pie slice).
- **p1**: Start position on the circle (percentage).
- **p2**: End position on the circle (percentage).

**Example**

This code draws a pie chart with 5 slices of 20% width each around the center point (150,150) with a radius of 100.

```plaintext
forall(i in 1..5) do
  svgaddgroup("gp"+i, "Pie"+i)
  svgaddpie(150, 150, 100, (i-1)*0.2, i*0.2)
end-do
```

**Further information**

1. Pie slices are by default filled with the group color. If they are not to be filled with any color specify value SVG_NONE for the style property SVG_FILL.
2. If no group ID is specified, the pie slice is added to the last group that has been created.
**svgaddpoint**

**Purpose**
Add a small square to mark a point at the given coordinates.

**Synopsis**
procedure svgaddpoint(gid: string, x: real, y: real)
procedure svgaddpoint(x: real, y: real)

**Arguments**
gid Object group ID.
x The x coordinate of the point.
y The y coordinate of the point.

**Example**
This code plots 100 random points:

```
svgaddgroup("cloud", "Random points", SVG_YELLOW)
svgsetgraphviewbox(-5, -5, 10, 10)
forall(i in 1..100)
    svgaddpoint("cloud", -2+4*random, -2+4*random)
```

**Further information**
If no group ID is specified, the point is added to the last group that has been created.

**Related topics**
`svgsetgraphpointsiz`. 
svgaddpolygon

**Purpose**
Add a polygon to an object group.

**Synopsis**
```
procedure svgaddpolygon(gid: string, points: list of integer|real)
procedure svgaddpolygon(points: list of integer|real)
```

**Arguments**
- **gid** Object group ID.
- **points** A list of points.

**Example**
The following code draws two semi-transparent, partially overlapping polygons, the first is filled with the group color, the second with a different color:
```
svggaddgroup("p", "Polygons")
svgsetstyle(SVG_OPACITY, 0.5)
svgsetstyle(SVG_FILL, SVG_CURRENT)
svgaddpolygon([-2, -2, 0, 2, 2, -2])
svgaddpolygon([-1, -2, 1, 2, 3, -2])
svgsetstyle(svggetlastobj, SVG_FILL, SVG_GREY)
```

**Further information**
1. The last point in the list of points is automatically connected to the first point in the list to form a closed shape.
2. If no group ID is specified, the polygon is added to the last group that has been created.

**Related topics**
- [svgaddline](#)
svgaddrectangle

Purpose
Add a rectangle to an object group.

Synopsis
procedure svgaddrectangle(gid: string, x: real, y: real, w: real, h: real)

Arguments
- gid Object group ID.
- x The x coordinate of the lower left corner.
- y The y coordinate of the lower left corner.
- w The width of the rectangle.
- h The height of the rectangle.

Example
The following code draws a rectangle filled with the group color covering an area 10 units long and 1 unit high starting at the origin.

```
svgaddrectangle(0,0,10,1)
svgsetstyle(svggetlastobj, SVG_FILL, SVG_CURRENT)
```

Further information
If no group ID is specified, the rectangle is added to the last group that has been created.
**svgaddtext**

**Purpose**
Add a text to an object group.

**Synopsis**

procedure svgaddtext(gid: string, x: real, y: real, msg: text)
procedure svgaddtext(x: real, y: real, msg: text)

**Arguments**

- **gid**  
  Object group ID.
- **x**  
  The x coordinate of the point.
- **y**  
  The y coordinate of the point.
- **msg**  
  The text that will be displayed at the given point.

**Example**

This code complements the time graph with a dial:

```plaintext
forall(i in 1..12)
    svgaddtext(4.8*cos(1.57-6.28*i/12), 5*sin(1.57-6.28*i/12), text(i))
```

**Further information**

1. By default the specified point denotes the lower left corner of the text display area; the vertical alignment can be changed via the style option `SVG_ANCHOR` (values 'start', 'middle', or 'end').

2. If no group ID is specified, the text is added to the last group that has been created.

**Related topics**

`svgaddxmltext`.
svgaddxmltext

**Purpose**
Add an XML formatted text to an object group.

**Synopsis**

```plaintext
procedure svgaddxmltext(gid: string, x: real, y: real, msg: text)
```

**Arguments**
- `gid` Object group ID.
- `x` The x coordinate of the point.
- `y` The y coordinate of the point.
- `text` The text that will be displayed at the given point.

**Example**
This code displays some text with individual formatting on different words:

```plaintext
svgaddxmltext(20, 150, 'XML formatted text:
  <tspan font-size="20px">large</tspan>,
  <tspan font-style="oblique">oblique</tspan>,
  <tspan font-weight="bold">bold</tspan>,
  <tspan text-decoration="underline">underlined</tspan>,
  <tspan stroke="red">red</tspan>
')
```

**Further information**
1. By default the specified point denotes the lower left corner of the text display area; the vertical alignment can be changed via the style option `SVG_ANCHOR` (values ‘start’, ‘middle’, or ‘end’).
2. If no group ID is specified, the text is added to the last group that has been created.

**Related topics**
- `svgaddtext`
**svgclosing**

**Purpose**
Test whether the display window is being closed.

**Synopsis**
function svgclosing:boolean

**Return value**
‘false’ until the display window is about to be closed, ‘true’ afterwards.

**Example**
The following loop uses the browser window opening status as stopping criterion.

```plaintext
solct:= 0
svgrefresh ! Start graph display before svgclosing test
while (solct<NBSOL and not svgclosing) do
    solct+=1
    draw_solution(solct) ! Draws a graph calling svgrefresh and svgpause
end-do
svgwaitclose
```

**Further information**
This function can be used to intercept the event of the display window being closed in order to adapt the behaviour of the model execution (e.g. to interrupt a loop with repeated graphical displays or an optimization solver run).

**Related topics**
svgwaitclose.
**svgcolor**

**Purpose**
Compute a composite color by combining amounts of red, green and blue.

**Synopsis**

```plaintext
function svgcolor(red, green, blue: integer): text
function svgcolor(red, green, blue: real): text
function svgcolor(red, green, blue: text): text
```

**Arguments**

- **red** 
  Amount of red (integer between 0 and 255, real between 0 and 1, or hexadecimal value between 0 and FF).
- **green** 
  Amount of green (integer between 0 and 255, real between 0 and 1, or hexadecimal value between 0 and FF).
- **blue** 
  Amount of blue (integer between 0 and 255, real between 0 and 1, or hexadecimal value between 0 and FF).

**Return value**
Hexadecimal representation of the composite color.

**Example**
The following definitions mix red with green and store the result in a variable. All three forms result in the same color.

```plaintext
declarations
  a_color: text
end-declarations

  a_color:=svgcolor(255,255,0)
a_color:=svgcolor(1.0,1.0,0.0)
a_color:=svgcolor("FF","FF","0")
```

**Further information**
If the color component values are out of range, *mmsvg* will raise an I/O error.
svgdlobj

Purpose
Delete the specified graphical object.

Synopsis
procedure svgdelobj(obj: integer)

Argument
obj  Object ID as returned by svggetlastobj.

Further information
This procedure serves for deleting a specific graphical object. Use svgerase to delete the whole contents of an object group or all groups.

Related topics
svgerase, svggetlastobj.
svgerase

**Purpose**
Erase all object groups or the contents of a specific group.

**Synopsis**
procedure svgerase
procedure svgerase(gid: string)

**Argument**
gid Object group ID.

**Further information**
1. This procedure can be used together with `svgpause` to explore a number of different user graph versions during the execution of a Mosel model.
2. If a group ID is specified only the objects within this group are removed without deleting the group definition itself.
3. Use `svgdelobj` to delete individual graphical objects.

**Related topics**
`svgdelobj`, `svgpause`. 
svggetgraphstyle

**Purpose**
Retrieve a style property of a graph.

**Synopsis**
function svggetgraphstyle(prop: string): text

**Argument**
prop The desired property (mmsvg constant or SVG property name).

**Return value**
Value of the property or empty string.

**Example**
This code retrieves the font family defined for a graph and applies it to an object group.

```javascript
svgaddgroup("g", "A group")
svgsetstyle("g", SVG_FONTFAMILY, svggetgraphstyle("b", SVG_FONTFAMILY))
```

**Further information**
This function can be used to retrieve a style property of a graph in order to apply it to some object or group of objects. Use svggetgraphstylesheet to retrieve the whole set of style properties of a graph.

**Related topics**
svggetstyle, svgsetstyle, svgsetgraphstyle, svggetgraphstylesheet, svgsetgraphstylesheet, svggetstylesheet, svgsetstylesheet.
**svggetgraphstylesheet**

**Purpose**
Retrieve the style definitions of a graph.

**Synopsis**
```plaintext
function svggetgraphstylesheet: array of text
```

**Return value**
An array of style properties ('stylesheet') with their respective values.

**Example**
This code retrieves the style properties of a graph and applies them to an object group.
```plaintext
svgaddgroup("a", "A group")
svgsetstylesheet("a", svggetgraphstylesheet)
```

**Further information**
This function can be used to retrieve the set of style properties ('stylesheet') of a graph in order to apply it to some object or group of objects. Use `svggetgraphstyle` to retrieve individual style properties of a graph.

**Related topics**
- `svggetstyle`, `svgsetstyle`, `svggetgraphstyle`, `svgsetgraphstyle`, `svgsetgraphstylesheet`, `svggetstylesheet`, `svgsetstylesheet`.
svggetgraphviewbox

**Purpose**
Retrieve the viewbox definition of a graph.

**Synopsis**
function svggetgraphviewbox: svgbox

**Return value**
An object of type ‘svgbox’ that holds the view box defined for the graph.

**Example**
This code displays the viewbox defined for a graph.

writeLn(svggetgraphviewbox)

**Further information**
This function can be used to retrieve the viewbox (=visible area) defined for a graph.

**Related topics**
svgsetgraphviewbox.
svgetlastobj

Purpose
Retrieve the identifier of a graphical object.

Synopsis
function svgetlastobj:integer

Return value
Integer identifier of the last graphical object that has been added.

Example
This code retrieves an object identifier to apply several style settings.

    declarations
    t: integer
    end-declarations

    svgaddgroup("gt", "Text")
    svgaddtext(20, 120, "Formatted text")
    t:=svgetlastobj
    svgsetstyle(t, SVG_COLOR, SVG_GREEN)
    svgsetstyle(t, SVG_FONTSTYLE, "italic")

Further information
This function serves for retrieving the identifier of a graphical object, in particular in order to apply style settings to this object.

Related topics
svgsetstyle.
**svggetstyle**

**Purpose**
Retrieve a style property of a graphical object or object group.

**Synopsis**
function svggetstyle(gid: string, prop: string):text  
function svggetstyle(prop: string):text  
function svggetstyle(obj: integer, prop: string):text

**Arguments**
gid  Object group ID.  
obj  Object ID.  
prop  The desired property (*mmsvg* constant or SVG property name).

**Return value**
Value of the property or empty string.

**Example**
This code retrieves the color of a group and applies it to an object belonging to another group.

```javascript
svgaddgroup("a", "Group A")
svgaddgroup("b", "Group B")
svgaddtext("a", 20, 120, "Formatted text")
svgsetstyle(svggetlastobj, SVG_COLOR, svggetstyle("b", SVG_COLOR))
```

**Further information**
This function can be used to retrieve a style property of some object in order to apply it to some other object or group of objects. Use *svggetstylesheet* to retrieve the whole set of style properties of an object or group of objects.

**Related topics**
*svgsetstyle*, *svggetgraphstylesheet*, *svgsetgraphstylesheet*, *svggetstylesheet*, *svgsetstylesheet*. 
svggetstylesheet

Purpose
Retrieve style definitions of a graphical object or object group.

Synopsis
function svggetstylesheet(gid: integer): array of text
function svggetstylesheet: array of text
function svggetstylesheet(obj: integer): array of text

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gid</td>
<td>Object group ID.</td>
</tr>
<tr>
<td>obj</td>
<td>Object ID.</td>
</tr>
</tbody>
</table>

Return value
An array of style properties (‘stylesheet’) with their respective values.

Example
This code retrieves the style properties of a group and applies them to an object belonging to another group.

```
svgaddgroup("a", "Group A")
svgaddgroup("b", "Group B")
svgaddtext("a", 20, 120, "Formatted text")
svgsetstylesheet(svggetlastobj, svggetstylesheet("b"))
```

Further information
This function can be used to retrieve the set of style properties (‘stylesheet’) of some object in order to apply it to some other object or group of objects. Use svggetstyle to retrieve individual style properties of an object or group of objects.

Related topics
svggetstyle, svgsetstyle, svggetgraphstylesheet, svgsetgraphstylesheet, svgsetstylesheet.
svgpause

Purpose
Suspend the execution of a Mosel model at the line where the call occurs.

Synopsis
procedure svgpause

Further information
While the model run is suspended, the displayed graph or other model output can be inspected. This allows for visualization of intermediate states or solutions. To continue, click on the 'Continue' button in the display window.
**svgrefresh**

**Purpose**
Refresh the graph display.

**Synopsis**

```
procedure svgrefresh
```

**Example**

This code defines some objects and draws the graph, it then adds further objects and updates the display.

```
svgaddgroup("a", "Group A")
svgaddtext(0, 0, "Some text")
svgrefresh ! Display the graph
svgaddgroup("b", "Group B")
svgaddtext("a", 0, 20, "Some more text")
svgaddcircle(10,10, 45)
svgrefresh ! Update the display
```

**Further information**

svgrefresh needs to be called in order to trigger the display of a graph. The subroutine can be called repeatedly in order to update the display—each time it will be completely redrawn. The refresh frequency can be controlled via svgsetreffreq.

**Related topics**

svgsetreffreq.
svgsave

**Purpose**
Save a graph to a file.

**Synopsis**
procedure svgsave(fname: string)

**Argument**
fname  The (extended) filename to be used as output destination.

**Example**
This code saves a graph to the file ‘mygraph.svg’ in the model working directory.

```
svgaddgroup("a", "Group A")
svgaddrectangle(20, 120, 200, 250)
svgsave("mygraph.svg")
```

**Further information**
This procedure can be used independently of the graphical display in order to produce output in SVG format of the current graph definition.
svgsetgraphlabels

**Purpose**
Set x- and y-axis labels for a graph.

**Synopsis**
procedure svgsetgraphlabels(xlabel: text, ylabel: text)

**Arguments**
xlabel Label text for the x-axis.
ylabel Label text for the y-axis.

**Example**
The following line sets the x-axis label text to ‘Time in sec’ and the y-axis label to ‘Solution value’.

    svgsetgraphlabels("Time in sec", "Solution value")

**Further information**
1. By default (no labels specified or empty strings) no label text is displayed.
2. The axes are displayed only if a label is defined (for x or y axis) unless svgshowgraphaxes has been used.

**Related topics**
    svgsetgraphscale, svgsetgraphpointsize, svgsetgraphviewbox, svgshowgraphaxes.
**svgsetgraphpointsize**

**Purpose**
Set point size property for a graph.

**Synopsis**

```plaintext
procedure svgsetgraphpointsize(val: real)
```

**Argument**

`val` The new value for the point size.

**Example**

This code shows how to modify graph scaling properties.

```plaintext
svgsetgraphpointsize(0.5)
svgsetgraphscale(10)
```

**Further information**

This routine is likely to be used in combination with `svgsetgraphscale` in order to resize a graph.

**Related topics**

`svgsetgraphscale, svggetgraphstyle, svgsetgraphstyle, svggetgraphstylesheet, svgsetgraphstylesheet`. 
svgsetgraphscale

**Purpose**  
Set scaling value for a graph.

**Synopsis**  
procedure svgsetgraphscale(val: real)

**Argument**  
val  
The new scaling value.

**Example**  
This code shows how to modify graph scaling properties.  

```plaintext
svgsetgraphpointsize(0.5)
svgsetgraphscale(10)
```

**Further information**  
This routine is likely to be used in combination with `svgsetgraphpointsize` in order to resize a graph for display.

**Related topics**  
`svgsetgraphpointsize`, `svgsetgraphstyle`, `svgsetgraphstylesheet`, `svgsetgraphstylesheet`. 
**svgsetgraphstyle**

**Purpose**
Set a style property of a graph.

**Synopsis**
```plaintext
procedure svgsetgraphstyle(prop: string, val: text|real)
```

**Arguments**
- `prop` The desired property (mmsvg constant or SVG property name).
- `val` The new value for the property (usually a text, but properties like SVG_OPACITY or SVG_STROKEWIDTH also accept numerical values).

**Return value**
Value of the property or empty string.

**Example**
This code retrieves the font family defined for a group and applies it to the entire graph.
```plaintext
svgaddgroup("g", "A group")
svgsetgraphstyle(SVG_FONTFAMILY, svggetstyle("g", SVG_FONTFAMILY))
```

**Further information**
This procedure can be used to define a style property of a graph. Use `svgsetgraphstylesheet` to define the whole set of style properties of a graph.

**Related topics**
- `svggetstyle`, `svgsetstyle`, `svggetgraphstyle`, `svggetgraphstylesheet`, `svgsetgraphstylesheet`, `svggetstylesheet`, `svgsetstylesheet`.
**svgsetgraphstylesheet**

**Purpose**
Set the style definitions for a graph.

**Synopsis**
procedure svgsetgraphstylesheet(stsh: array (svgstyleattrs) of text)

**Argument**
stsh  Style definition.

**Example**
This code retrieves the style properties of a group and applies them to the entire graph.

```plaintext
svgaddgroup("a", "A group")
svgsetgraphstylesheet(svggetstylesheet("a"))
```

**Further information**
This procedure can be used to define the set of style properties ('stylesheet') of a graph. Use `svgsetgraphstyle` to define individual style properties of a graph.

**Related topics**
`svggetstyle`, `svgsetstyle`, `svggetgraphstyle`, `svgsetgraphstyle`, `svggetgraphstylesheet`, `svggetstylesheet`, `svgsetstylesheet`. 
svgsetgraphviewbox

**Purpose**
Set the visible area for a user graph.

**Synopsis**
procedure svgsetgraphviewbox(x: real, y: real, w: real, h: real)
procedure svgsetgraphviewbox(box: svgbox)

**Arguments**
x The x coordinate of the lower left corner.
y The y coordinate of the lower left corner.
w The width of the viewbox.
h The height of the viewbox.
box Viewbox specification as returned by svggetgraphviewbox.

**Further information**
1. The viewable area is determined by its lower left corner, its width and height.
2. *mmsvg* automatically determines a viewbox (enclosing all specified coordinates) that can be retrieved with svggetgraphviewbox.

**Related topics**
svggetgraphviewbox, svgsetgraphlabels, svgsetgraphscale.
**svgsetreffield**

**Purpose**
Set the refresh frequency for a graph.

**Synopsis**
procedure svgsetreffield(val: real)

**Argument**
val   The new refresh frequency (maximum number of refreshs per second).

**Further information**
The refresh frequency indicates how often individual calls to `svgrefresh` are posted to the display. If several refresh occur during the specified time span, only the last one is executed.

**Related topics**
`svgrefresh`
svgsetstyle

Purpose
Set a style property for a graphical object or object group.

Synopsis
procedure svgsetstyle(gid: string, prop: string, val: text|real)
procedure svgsetstyle(prop: string, val: text|real)
procedure svgsetstyle(obj: integer, prop: string, val: text|real)

Arguments
  gid    Object group ID.
  obj    Object ID.
  prop   The desired property (mmsvg constant or SVG property name).
  val    The new value for the property (usually a text, but properties like SVG_OPACITY or SVG_STROKEWIDTH also accept numerical values).

Example
This code retrieves the color of a group and applies it to an object belonging to another group.

    svgaddgroup("a", "Group A")
    svgaddgroup("b", "Group B")
    svgaddtext("a", 20, 120, "Formatted text")
    svgsetstyle(svggetlastobj, SVG_COLOR, svggetstyle("b", SVG_COLOR))

Further information
This procedure can be used to define a style property of some object or group of objects. Use svgsetstylesheet to redefine the whole set of style properties of an object or group of objects.

Related topics
gvgetstyle, svggetgraphstylesheet, svgsetgraphstylesheet, svggetstylesheet, svgsetstylesheet.
**svgsetstylesheet**

**Purpose**
Set the style for a graphical object or object group.

**Synopsis**

- procedure svgsetstylesheet(gid: string, stsh: array (svgstyleattrs) of text)
- procedure svgsetstylesheet(stsh: array(svgstyleattrs) of text)
- procedure svgsetstylesheet(obj: integer, stsh: array(svgstyleattrs) of text)

**Arguments**

- **gid** Object group ID.
- **obj** Object ID.
- **stsh** Style definition.

**Example**
This code retrieves the style properties of a group and applies them to an object belonging to another group.

```
svgaddgroup("a", "Group A")
svgaddgroup("b", "Group B")
svgaddtext("a", 20, 120, "Formatted text")
svgsetstylesheet(svggetlastobj, svggetstylesheet("b"))
```

**Further information**
This procedure can be used to define a set of style properties ('stylesheet') of some object or group of objects. Use `svgsetstyle` to modify individual style properties of an object or group of objects.

**Related topics**

- `svgsetstyle`, `svgsetstylesheet`, `svggetgraphstylesheet`, `svgsetgraphstylesheet`, `svgsetstylesheet`.
svgshowgraphaxes

**Purpose**
Force displaying of graph axes.

**Synopsis**
procedure svgshowgraphaxes(force:boolean)

**Argument**
force  Decide whether graph axes must be shown when no label is defined.

**Further information**
By default the axes are only shown if a label text is defined (for x or y axis). This procedure makes it possible to display the axes even if no label is used.

**Related topics**
svgsetgraphlabels.
svgwaitclose

**Purpose**
Delay model termination.

**Synopsis**
procedure svgwaitclose(msg:text, mode:integer
procedure svgwaitclose(msg:text)
procedure svgwaitclose

**Arguments**
- **msg** Some message to display.
- **mode** Mode of operation:
  - 0 Wait until the browser window is closed
  - 1 Same as above except if running from Workbench: termination occurs after the graph is loaded

**Example**
This code shows a typical call sequence for graphical display.

```plaintext
svgaddgroup("a", "Group A")
svgaddrectangle(20, 120, 200, 250)
svgrefresh ! Display the graphic
svgwaitclose ! Model waits here until display window is closed
```

**Further information**
1. A call to this routine is typically added to the end of any model that includes graphical display (that is, calls to svgrefresh) via mmsvg to allow the user time for inspecting the graphical output. If this subroutine call is not present, then model termination may close the display window or prevent the browser to load the graph.
2. The last form of the routine is equivalent to svgwaitclose("",0).

**Related topics**
- svgrefresh, svgclosing.
The mmsystem module provides a set of procedures and functions related to the operating system. Note that the behavior of these operators may vary between systems. To use this module, the following line must be included in the header of the Mosel model file:

uses 'mmsystem'

16.1 New functionality for the Mosel language

16.1.1 The type text

This module provides the type text for text manipulation. Like the Mosel basic type string, this new type may be generated from all objects that can be converted to a text representation and supports the usual string operations (like concatenation or formatting). In addition, text objects can be generated from structured entities (like arrays or lists); altered (one can get and change a single as well as a sequence of characters in a text); offer a wider set of operations (like insertion/deletion/search of substrings) and, as all module types, are passed by reference to subroutines. Note that this type supports implicit conversion from string: a routine expecting a text as parameter may be used with a string instead (in this case the compiler creates a temporary text from the provided string). When creating a text object from a structured type it is possible to specify a limit on the size of the generated string. For instance if S is a set, text(S,128) will produce a textual representation of S of at most 128 characters.

16.1.2 The type date

As the name suggests, the type date is used to represent a calendar date. Internally, a date is stored as three independent integers for representing the year (-32768 to 32767), the month (-128 to 127) and the day in the month (-128 to 127). The validity of a date can be checked using the function isvalid. A date object can be initialized by a text string, a single or three numerical values. In the first case, the conversion is processed using a predefined date format (see datefmt); in the second case, the integer is interpreted as the number of days elapsed since 1/1/1970; finally, if three integers are used, they are respectively interpreted as the year, month and day for the date. The constant SYS_NOW may also be used to initialize a date: date(SYS_NOW) is the current date. This type also supports assignment, comparison as well as difference (returned in number of days) and addition/subtraction of an integer (number of days).

16.1.3 The type time

The type time is used to represent a time during the day. Internally, a time object is stored as an integer representing a number of milliseconds. A time object can be initialized by a text string or one to four numerical values. In the first case, the conversion is processed using a predefined time format (see timefmt). The type time also supports assignment, comparison as well as difference (returned in number of seconds) and addition/subtraction of an integer (number of seconds).
format (see \texttt{timefmt}); in the second case, the integer is interpreted as a number of milliseconds. When two to four integers are used, they are understood as the hours, minutes, seconds and milliseconds. The constant \texttt{SYS_NOW} may also be used to initialize a time: \texttt{time(SYS_NOW)} is the current time. This type also supports assignment, comparison as well as difference (returned in number of milliseconds) and addition/subtraction of an integer (number of milliseconds).

16.1.4 The type \texttt{datetime}

The type \texttt{datetime} is used to represent a timestamp by combining a date and a time. A datetime object can be initialized by a text string, a pair date and time or a numerical value. In the first case, the conversion is processed using a predefined time format (see \texttt{datetimefmt}); in the third case, the number is interpreted as the number of seconds elapsed since 1/1/1970 at midnight. If the provided number is a real value, the fractional part is stored as a number of milliseconds. The constant \texttt{SYS_NOW} may also be used to initialize a datetime: \texttt{datetime(SYS_NOW)} is the current date and time. This type also supports assignment, comparison as well as difference (returned in number of seconds) and addition/subtraction of a numerical value (number of seconds).

16.1.5 The type \texttt{parsectx}

This module publishes a set of routines for parsing input text strings (for instance \texttt{parseint} or \texttt{nextfield}). These routines use several module parameters for both their configuration and as a way to record their internal state: a variable of type \texttt{parsectx} may be used as a replacement for these module parameters in order to implement parsing procedures independent of the rest of the program. A single \texttt{parsectx} object integrates \texttt{endparse} (see \texttt{sys_endparse}), \texttt{sepchar} (see \texttt{sys_sepchar}), \texttt{trim} (see \texttt{sys_trim}) and \texttt{qtype} (see \texttt{sys_qtype}). The current value of each of these components can be accessed using the corresponding \texttt{set} and \texttt{get} routine (for instance \texttt{getendparse}).

16.1.6 The type \texttt{textarea}

The \texttt{textarea} type is used by the regular expression matching function \texttt{regmatch} to return locations in the input string. Each text area is defined by a \texttt{starting position} (that is an offset in the original string) and an \texttt{ending position} characterised by the offset of the character following the region to be considered. Functions \texttt{getstart} and \texttt{getsucc} can be used to retrieve these properties.

For instance, the following statement displays the region \texttt{ta} of the text \texttt{txt}:

\begin{verbatim}
writeln(copytext(txt,ta.start,ta.succ-1))
\end{verbatim}

This can also be written as follows:

\begin{verbatim}
writeln(copytext(txt,ta))
\end{verbatim}

16.2 Control parameters

Via the \texttt{getparam} function and the \texttt{setparam} procedure it is possible to access the following control parameters of module \texttt{mmsystem} (the reader is reminded that parameters may be spelled with lower or upper case letters or a mix of both):

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{datefmt}</td>
<td>Date text format.</td>
<td>423</td>
</tr>
<tr>
<td>\texttt{datetimefmt}</td>
<td>Date and time text format.</td>
<td>424</td>
</tr>
</tbody>
</table>
**datefmt**

**Description**
Define the text format for both reading and writing a date.

**Type**
String, read/write

**Default value**
"%.y-%0m-%0d"

**Note**
The date format consists in a text string in which the date information (like day number) is specified using tags. A tag begins by the character "%" optionally followed by "." or "0" and a character indicating which specific information must be provided. The possible values are:

- **C** Century
- **Y** Year number in the century
- **y** Year
- **m** Month (1-12)
- **N** Name of month according to parameter `monthnames`
- **d** Day (1-31)
- **%** The symbol ";"

If the second character is used, the corresponding information is produced in fixed format with space (" .") or zero ("0") as the padding character. For instance, the day 1 will be displayed as "1" with the format ";d"; as " 1" with "%.d" and as "01" with "%0d".

**See also**
datetimefmt, monthnames

**Module**
mmsystem

**timefmt**

**Description**
Define the text format for both reading and writing time.

**Type**
String, read/write

**Default value**
"%0H:%0M:%0S,%s"
### datetimefmt

**Description**
Define the text format for both reading and writing a datetime object.

**Type**
String, read/write

**Default value**
```
%.y-%0m-%0dT%0H:%0M:%0S,%s
```

**Note**
The datetime format accepts the syntaxes of the date formant and the time format in the same string.

**See also**
datfmt, timefmt

**Module**
mmsystem

### monthnames

**Description**
Define month names to be used with the %N format.

**Type**
String, read/write

**Default value**
"jan feb mar apr may jun jul aug sep oct nov dec"

**Note**
This parameter is used when converting dates from/to strings with the %N format. The string must contain 12 words separated by spaces. For conversions from strings, the comparison is not case sensitive.

**See also**
datfmt, datetimefmt

**Module**
mmsystem
sys_endparse

Description: Index in the text string where the parsing stopped. This parameter is updated and may be used (as a starting position) by the parse* routines.

Type: Integer, read/write

Set by routines: parseint, parsereal, parseextn, parsetext, nextfield

Module: mmsystem

sys_fillchar

Description: Character code used to fill empty regions generated in text strings when using the function setchar.

Type: Integer, read/write

Values: Between 1 and 127

Default value: 32 (space character)

Affects routines: setchar

Module: mmsystem

sys_pid

Description: System identification (Process ID) of the process running Mosel.

Type: Integer, read only

Default value: assigned by the operating system

Module: mmsystem

sys_qtype

Description: Convention to use when quoting/parsing a text string.

Type: Integer, read/write

Default value: 0
Note

Supported quoting conventions are:

0    Mosel: strings optionally quoted with either single or double quotes. With double quotes, escape sequences starting with the backslash character ("\") are supported
1    C/C++: double quotes with escape sequences starting with the backslash character ("\")
2    CSV: strings are optionally quoted with double quotes. The symbol "double quotes" is doubled when it is included in a quoted string
-1   No quoting

Affects routines parse\text, \quote
Module mmsystem

sys\_regcache

Description Regular expression searches require a compilation procedure to be performed before the actual search. In order to speedup handling of expressions, a number of compiled expressions are saved in a cache pool: this parameter specifies the size of this pool. Note that setting this parameter has the effect of clearing the cache (even if the pool size if kept unchanged).

Type Integer, read/write
Values Between 1 and 25
Default value 3
Affects routines regmatch, regreplace
Module mmsystem

sys\_sepchar

Description Character code used as a field separator for text parsing routines.

Type Integer, read/write
Values Between 1 and 127
Default value 32 (space character)
Affects routines parse\text, \quote, nextfield
Module mmsystem

sys\_trim

Description If this parameter is \texttt{true}, function nextfield skips blank characters around field separators.
sys_txtmem

Description
All text objects are stored in a single block of memory. This parameter corresponds to the size of this block expressed in kilobytes. Changing this value makes it possible either to pre-allocate memory by increasing the size of the block or release unused memory by reducing its size. If the requested size is not large enough to contain the currently defined text objects, the memory block is reduced to the smallest possible size.

Type Integer, read/write
Default value 0 (at program startup)
Module mmsystem

16.3 Procedures and functions

In general, the procedures and functions of mmsystem do not fail but set a status variable that can be read with getsysstat. To make sure the operation has been performed correctly, check the value of this variable after each system call.

- addmonths: Add a number of months to a date or datetime. p. 431
- copytext: Copy a part of a text or string. p. 432
- cuttext: Cut a part of a text returning a copy of the deleted string. p. 433
- deltext: Delete a part of a text. p. 434
- endswith: Check whether a text or string ends with a given string. p. 435
- expandpath: Expand a path or file name. p. 436
- fcopy: Copy a file. p. 437
- fdelete: Delete a file. p. 438
- findfiles: Search for files according to file name patterns. p. 439
- findtext: Search for a string in a text or string. p. 440
- fmove: Rename or move a file. p. 441
- formattext: Create a text from a format string and its parameters. p. 442
- getasnumber: Convert a date, time or datetime into a number. p. 444
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getchar</code></td>
<td>Get a character in a string or text.</td>
<td>445</td>
</tr>
<tr>
<td><code>getcwd</code></td>
<td>Get the current working directory.</td>
<td>446</td>
</tr>
<tr>
<td><code>getdate</code></td>
<td>Get the date part of a datetime.</td>
<td>447</td>
</tr>
<tr>
<td><code>getday</code></td>
<td>Get the day number in the month of a date or datetime.</td>
<td>448</td>
</tr>
<tr>
<td><code>getdaynum</code></td>
<td>Get the day number in the year of a date or datetime.</td>
<td>449</td>
</tr>
<tr>
<td><code>getdays</code></td>
<td>Get the number of days of a month.</td>
<td>450</td>
</tr>
<tr>
<td><code>getdirsep</code></td>
<td>Get the directory separator of the running operating system.</td>
<td>451</td>
</tr>
<tr>
<td><code>getendparse</code>, <code>setendparse</code></td>
<td>Get and set endparse property of a parser context.</td>
<td>452</td>
</tr>
<tr>
<td><code>getenv</code></td>
<td>Get the value of an environment variable.</td>
<td>453</td>
</tr>
<tr>
<td><code>getfsize</code></td>
<td>Get the size of a file.</td>
<td>454</td>
</tr>
<tr>
<td><code>getfstat</code></td>
<td>Get the status of a file or directory.</td>
<td>455</td>
</tr>
<tr>
<td><code>getftime</code></td>
<td>Get time information of a file.</td>
<td>456</td>
</tr>
<tr>
<td><code>gethour</code></td>
<td>Get the hour part of a time or datetime.</td>
<td>457</td>
</tr>
<tr>
<td><code>getminute</code></td>
<td>Get the minute part of a time or datetime.</td>
<td>458</td>
</tr>
<tr>
<td><code>getmonth</code></td>
<td>Get the month number of a date or datetime.</td>
<td>459</td>
</tr>
<tr>
<td><code>getmsec</code></td>
<td>Get the millisecond part of a time or datetime.</td>
<td>460</td>
</tr>
<tr>
<td><code>getpathsep</code></td>
<td>Get the path separator of the running operating system.</td>
<td>461</td>
</tr>
<tr>
<td><code>getqtype</code>, <code>setqtype</code></td>
<td>Get and set qtype property of a parser context.</td>
<td>463</td>
</tr>
<tr>
<td><code>getsecond</code></td>
<td>Get the second part of a time or datetime.</td>
<td>464</td>
</tr>
<tr>
<td><code>getsepchar</code>, <code>setsepchar</code></td>
<td>Get and set sepchar property of a parser context.</td>
<td>465</td>
</tr>
<tr>
<td><code>getsize</code></td>
<td>Get the size of a text.</td>
<td>466</td>
</tr>
<tr>
<td><code>getstart</code>, <code>setstart</code></td>
<td>Get and set start property of a text area.</td>
<td>467</td>
</tr>
<tr>
<td><code>getsucc</code>, <code>setsucc</code></td>
<td>Get and set succ (position of successor character) property of a text area.</td>
<td>462</td>
</tr>
<tr>
<td><code>getsysinfo</code></td>
<td>Get information about the running operating system.</td>
<td>468</td>
</tr>
<tr>
<td><code>getsysstat</code></td>
<td>Get the system status.</td>
<td>469</td>
</tr>
<tr>
<td><code>gettime</code></td>
<td>Get a time measure or the time part of a datetime.</td>
<td>470</td>
</tr>
<tr>
<td><code>gettempdir</code></td>
<td>Get the temporary directory as a text object.</td>
<td>471</td>
</tr>
<tr>
<td><code>gettrim</code>, <code>settrim</code></td>
<td>Get and set trim property of a parser context.</td>
<td>472</td>
</tr>
<tr>
<td><code>getweekday</code></td>
<td>Compute the day of the week for a date or datetime.</td>
<td>473</td>
</tr>
<tr>
<td><code>getyear</code></td>
<td>Get the year part of a date or datetime.</td>
<td>474</td>
</tr>
<tr>
<td><code>inserttext</code></td>
<td>Paste a text or string into a text.</td>
<td>475</td>
</tr>
<tr>
<td><code>isvalid</code></td>
<td>Check whether a date, time or datetime is valid.</td>
<td>476</td>
</tr>
<tr>
<td><code>makedir</code></td>
<td>Create a new directory in the given file system.</td>
<td>477</td>
</tr>
</tbody>
</table>
makepath
Create a new directory including its parents if necessary. p. 478
newtar
Create a Unix tar archive from a list of files. p. 479
newzip
Create a Zip archive from a list of files. p. 480
nextfield
Advance to next field in a structured text string. p. 481
openpipe
Start an external process for bidirectional communication. p. 483
parseextn
Initialise an object of a module type from a text. p. 484
parseint
Convert a text into an integer. p. 485
parsereal
Convert a text into a real. p. 487
parsetext
Extract a text from a text. p. 488
pastetext
Paste a text or string into a text. p. 490
pathmatch
Check whether a file name matches a given pattern. p. 491
pathsplt
Split a path into its components. p. 492
qsort
Sort a list or an array or (a subset of) the indices of an array. p. 493
quote
Quote and encode a text string. p. 494
readtextline
Read a line of text from the current input stream. p. 495
regmatch
Compare text strings using a regular expression. p. 496
regreplace
Replace portions of a text string based on a regular expression. p. 497
removedir
Remove a directory. p. 498
removefiles
Remove files selected using file name patterns. p. 499
setchar
Set a character in a text. p. 500
setdate
Set the date part of a datetime. p. 501
setday
Set the day number of a date or datetime. p. 502
setenv
Set the value of an environment variable. p. 503
sethour
Set the hour part of a time or datetime. p. 504
setminute
Set the minute part of a time or datetime. p. 505
setmonth
Set the month number of a date or datetime. p. 506
setmsec
Set the millisecond part of a time or datetime. p. 507
setsecond
Set the second part of a time or datetime. p. 508
settime
Set the time part of a datetime. p. 509
setyear
Set the year part of a date or datetime. p. 510
sleep
Suspend execution for a fixed amount of time. p. 511
startswith
Check whether a text or string starts with a given string. p. 512
system
Execute an external program. p. 513
tarlist
Get the list of files included in a Unix tar archive. p. 514
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>textfmt</td>
<td>Create a formatted text from a string, a text or a number.</td>
<td>515</td>
</tr>
<tr>
<td>tolower</td>
<td>Generate the lowercase version of the provided text.</td>
<td>517</td>
</tr>
<tr>
<td>toupper</td>
<td>Generate the uppercase version of the provided text.</td>
<td>518</td>
</tr>
<tr>
<td>trim</td>
<td>Remove blank characters at the beginning and/or end of a text string.</td>
<td>519</td>
</tr>
<tr>
<td>untar</td>
<td>Extract files from a Unix tar archive.</td>
<td>520</td>
</tr>
<tr>
<td>unzip</td>
<td>Extract files from a Zip archive.</td>
<td>521</td>
</tr>
<tr>
<td>ziplist</td>
<td>Get the list of files included in a Zip archive.</td>
<td>522</td>
</tr>
</tbody>
</table>
addmonths

Purpose
Add a number of months to a date or datetime.

Synopsis
function addmonths(d:date, nbm:integer):date
function addmonths(dt:datetime, nbm:integer):datetime

Arguments
- d: A date object
- dt: A datetime object
- nbm: The number of months to be added (can be negative)

Return value
The modified date or datetime.

Example
writeln(addmonths(date(2000,1,31),1)) ! displays: 2000-02-29
writeln(addmonths(date(2012,12,12),-12)) ! displays: 2011-12-12

Further information
The day number is preserved unless it is not compatible with the computed month: in this case
the day number is moved to the last day of the month.

Module
mmsystem
copytext

**Purpose**
Copy a part of a text or string.

**Synopsis**
```plaintext
function copytext(t:text|string, i1:integer, i2:integer):text
function copytext(t:text|string, ta:textarea):text
```

**Arguments**
- **t** A string or text object
- **i1** Starting position of the region to copy
- **i2** End position of the region to copy
- **ta** A text area object

**Return value**
A copy of the region.

**Example**
The following:

```plaintext
writeln(copytext("abcdefgh",3,7))
writeln(copytext("abcdefgh",7,10))
```

produces this output:
```
cdefg
gh
```

**Further information**
This function returns an empty text if the bounds are not compatible with the string (e.g. starting position larger than the length of the string) or inconsistent (e.g. starting position after end position).

**Related topics**
deltext, inserttext, pastetext, cuttext

**Module**
mmsystem
cuttext

Purpose
Cut a part of a text returning a copy of the deleted string.

Synopsis
function cuttext(txt: text, i1: integer, i2: integer): text
function cuttext(txt: text, ta: textarea): text

Arguments
- txt: A text object
- i1: Starting position of the region to cut
- i2: End position of the region to cut
- ta: A text area object

Return value
A copy of the region. The input text is modified accordingly.

Example
The following:

t := text("abcdefgh")
writeln(cuttext(t, 3, 7))
writeln(t)

produces this output:
cdefg
abh

Further information
This function returns an empty text if the bounds are not compatible with the string (e.g. starting position larger than the length of the string) or inconsistent (e.g. starting position after end position).

Related topics
deltext, inserttext, pastetext, copytext

Module
mmsystem
deltxt

Purpose
Delete a part of a text.

Synopsis
procedure deltext(txt:text, i1:integer, i2:integer)
procedure deltext(txt:text, ta:textarea)

Arguments
- txt A text object
- i1 Starting position of the region to delete
- i2 End position of the region to delete
- ta A text area object

Example
The following:

```
t:=text("abcdefgh")
deltext(t,3,7)
writeln(t)
```

produces this output:

```
abh
```

Related topics
cuttext, inserttext, pastetext, copytext

Module
mmsystem
endswith

**Purpose**
Check whether a text or string ends with a given string.

**Synopsis**
function endswith(txt: text|string, tofs: text|string): boolean

**Arguments**
txt A string or text object
tofs String to find

**Return value**
true if the ending of txt corresponds to tofs.

**Related topics**
startswith

**Module**
mmsystem
expandpath

Purpose
Expand a path or file name.

Synopsis
function expandpath(fname:string|text):text

Argument
fname  File name to be expanded

Return value
An absolute path to the given file name.

Further information
This function expands a path or file name: it replaces all relative references (like "." or ".") and completes the path such that the returned string is an absolute path to the provided file name.

Module
mmsystem
**fcopy**

**Purpose**
Copy a file.

**Synopsis**
procedure fcopy(namesrc:string|text, namedest:string|text)
procedure fcopy(namesrc:text, opts:integer, namedest:text, optd:integer)

**Arguments**
namesrc The name of the file to be copied
opts Open options for the input file
namedest The destination name
optd Open options for the output file

**Example**
The following statement appends file "src" to file "dst":

```plaintext
fcopy("src",0,"dst",F_APPEND)
```

**Further information**
1. This procedure copies the file namesrc to namedest. The provided names may use extended notation.
2. With the second form of the procedure it is possible to select options used to open the 2 files (as used with the fopen procedure). The first syntax corresponds to: fcopy(src,F_SILENT+F_BINARY,dst,F_SILENT+F_BINARY)

**Related topics**
fopen

**Module**
mmsystem
fdelete

Purpose
Delete a file.

Synopsis
procedure fdelete(filename:string|text)

Argument
filename The extended name of the file to be deleted

Further information
The provided name may use extended notation.

Related topics
removedir, removefiles.

Module
mmsystem
findfiles

Purpose
Search for files according to file name patterns.

Synopsis
procedure findfiles(opt:integer, lsf:list of text,
                      dir:string|text, filters:string)
procedure findfiles(opt:integer, lsf:list of text, filters:string)
procedure findfiles(lsf:list of text, filters:string)
procedure findfiles(lsf:list of text)

Arguments
  opt  Options (several options can be combined):
       SYS_RECURS  Recursive search in subdirectories
       SYS_NODIR   Do not report directories (only files)
       SYS_DIRONLY Report only directories
       SYS_REVORD  Reverse sort order
       SYS_NOSORT  Do not sort resulting list
  lsf   Resulting list of file names
  dir   Base directory for the search (default: current directory)
  filters File name filters (default: all files reported)

Example
The following prints the list of files with extension .mos and .bim of the current directory:

          findfiles(lsf,"*.mos|*.bim")
          writeln(lsf)

Further information

1. The filters argument consists in a list of patterns separated by the symbol ";": for each of these patterns the function executes a search from the specified dir directory. A pattern is composed of a path (using the usual operating system conventions) which last component may include wildcard characters "*" (any text of any length), "?" (any single character) and "|" (logical "or"). For instance "bin/*.*exe;models/*.*mos|*.dat" will select all files with extension ".exe" in the "bin" directory as well as files with extension ".mos" and ".dat" in the "models" directory.

2. File name matching is achieved using function pathmatch and differences may be observed depending on the operating system (e.g. file names are case sensitive under Posix systems but not under Windows).

3. Unless option SYS_NOSORT is used, the resulting list is sorted and duplicate entries are removed. Note also that the provided list lsf is not reset: the result of the search is appended to this list.

Related topics
removefiles

Module
mmsystem
findtext

Purpose
Search for a string in a text or string.

Synopsis
function findtext(txt:text, toft:text, start:integer):integer
function findtext(txt:text, tofs:string, start:integer):integer
function findtext(str:string, tofs:string, start:integer):integer

Arguments

txt A text object
str String
toft Text to find
tofs String to find
start Starting position for the search

Return value
Index of the string or 0 if not found.

Example
The following:

    writeln(findtext("abcdefgh","de",2))
    writeln(findtext("abcdefgh","de",5))

produces this output:

    4
    0

Module
mmsystem
fmove

Purpose
Rename or move a file.

Synopsis
procedure fmove(namesrc:string|text,namedest:string|text)

Arguments
namesrc The name of the file to be moved or renamed
namedest The destination name and/or path

Further information
This procedure renames the file namesrc to namedest. If the second name is a directory, the file is moved into that directory. The provided names may use extended notation.

Module
mmsystem
**formattext**

**Purpose**
Create a text from a format string and its parameters.

**Synopsis**

```plaintext
function formattext(fmt:string, a1, a2...):text
function formattext(fmt:string, l: list):text
```

**Arguments**

- **fmt** Format string
- **ai** Parameters of the format string
- **l** List of parameters of the format string

**Return value**

Formatted text.

**Example**

The following:

```plaintext
writeln(formattext("text1%8stext3", "text2"))
writeln(formattext("text1%-8stext3", "text2"))
r:=789.123456
writeln(formattext("%1$r %1$4.2f%1$8.0f",r))
```

produces this output:

```
text1 text2 text3
text1 text2 text3
789.123 789.12 789
```
Further information

1. This procedure behaves in a similar way as the sprintf function of the C language: the resulting text is generated by inserting each of the parameters $ai$ in the format string at locations identified by a marker. This marker is of the form:

$$%[\text{index}$][\text{flags}][\text{width}][.\text{precision}]\text{conv}$$

Where $\text{index}$ (a non-negative integer), $\text{flags}$ (string of ‘’, ‘’, ‘’, ‘’, ‘’), $\text{width}$ (positive integer) and $\text{precision}$ (non-negative integer) are optional.

The $\text{index}$ indicates which parameter to use for the conversion (first parameter has number 1), when it is not specified the marker position is used instead (e.g. the third marker is used for the third parameter).

The $\text{flags}$ essentially affect numerical conversions: with the flag ‘0’ the value is zero padded; with ‘-’ the value is left justified; with a space a blank is put before positive numbers and with ‘+’ positive numbers are preceded by the ‘+’ sign.

The $\text{width}$ defines a minimum width for the field.

The $\text{precision}$ gives the minimum number of digits to appear for an integer conversion. With a floating point value and a conversion ‘e’, ‘E’ or ‘f’ it states the number of digits to appear after the radix and for a ‘g’ conversion it is the maximum number of significant digits. The precision indicates a maximum number of characters to display with textual conversions.

The conversion specifier $\text{conv}$ is a letter indicating how to process the corresponding parameter and what to output. Possible values for this character are:

- $\text{d, i, o, u, X}$: an integer value is output: the parameter must be an integer or a Boolean. The value is displayed as a decimal number (‘d’ or ‘i’), an octal number (‘o’), an unsigned number (‘u’) or a hexadecimal number (‘x’ or ‘X’)

- $\text{e, E, f, r}$: a real value is output: the parameter must be a real or an integer. When using the ‘r’ conversion the optional part components of the marker are ignored and the value is converted using the current real printing format (see setparam). The conversion ‘e’ and ‘E’ format the number as $[-]d.ddd\pm/dd$; conversion ‘f’ uses a format of the form $[-]dd.dd$ and conversion ‘g’ selects format ‘e’ or ‘f’ depending on the value of the number.

- $\text{b}$: ‘true’ or ‘false’ is output: the parameter must be a Boolean

- $\text{c}$: a character is output: the parameter must be an integer that is interpreted as a Unicode code point

- $\text{s}$: a text string is output: the parameter must be a string or any type supporting conversion to text

- $\text{p}$: a pointer expressed in hexadecimal is output: the parameter can be any referenced entity

2. To include the symbol ‘%’ in the format string use the sequence ‘%%%’.

Related topics

textfmt

Module

mmsystem
getasnumber

Purpose
Convert a date, time or datetime into a number.

Synopsis
function getasnumber(d:date):integer
function getasnumber(t:time):integer
function getasnumber(dt:datetime):real

Arguments
d A date object
t A time object
dt A datetime object

Return value
The numerical representation of the argument.

Further information
A date is converted to an integer Julian Day Number (number of days since 1/1/1970 at midnight). This function returns an integer number of milliseconds for a time and a real number of seconds for a datetime. This number represents the number of seconds and milliseconds (as the fractional part of the number) since 1/1/1970 at midnight.

Module
mmsystem
getchar

Purpose
Get a character in a string or text.

Synopsis
function getchar(txt:text, index:integer):integer
function getchar(str:string, index:integer):integer

Arguments
- txt: A text object
- str: String
- index: Position of the character

Return value
Character code or -1 if the index is not valid.

Related topics
setchar

Module
mmsystem
getcwd

**Purpose**
Get the current working directory.

**Synopsis**

function getcwd:string

**Return value**
The current working directory.

**Further information**

1. This function returns the current working directory, that is the directory where the model is being executed and where files are looked for.

2. The returned value corresponds to getparam("workdir"). The current working directory can also be changed via this control parameter (for instance setparam("workdir", "./../somedir").

**Module**
mmsystem
getdate

**Purpose**
Get the date part of a datetime.

**Synopsis**

```plaintext
function getdate(dt:datetime):date
```

**Argument**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dt</td>
<td>A datetime object</td>
</tr>
</tbody>
</table>

**Return value**

A date object.

**Related topics**

- `gettime`

**Module**

- `mmsystem`
getday

Purpose
Get the day number in the month of a date or datetime.

Synopsis
function getday(d:date):integer
function getday(dt:datetime):integer

Arguments
\[\begin{align*}
    d & \quad \text{A date object} \\
    dt & \quad \text{A datetime object}
\end{align*}\]

Return value
Day number in the month.

Related topics
ggetyear, getmonth, getdaynum

Module
mmsystem
getdaynum

**Purpose**
Get the day number in the year of a date or datetime.

**Synopsis**
function getdaynum(d:date):integer
function getdaynum(dt:datetime):integer

**Arguments**
d A date object
dt A datetime object

**Return value**
Day number in the year.

**Example**
writeLn(getdaynum(date(2010,2,1))) ! displays: 32

**Related topics**
getday

**Module**
mmsystem
getdays

Purpose
Get the number of days of a month.

Synopsis
function getdays(y:integer, m:integer):integer
function getdays(d:date):integer
function getdays(dt:datetime):integer

Arguments
y Year
m Month
d A date object
dt A datetime object

Return value
Number of days for the given month in the specified year.

Example
writeln(getdays(2016, 2)) ! displays: 29

Module
mmsystem
getdirsep

Purpose
Get the directory separator of the running operating system.

Synopsis
function getdirsep: string

Return value
"/" on Posix systems and "\" on Windows.

Related topics
getpathsep

Module
mmsystem
getendparse, setendparse

**Purpose**
Get and set endparse property of a parser context.

**Synopsis**

```plaintext
function getendparse(pctx:parsectx):integer
procedure setendparse(pctx:parsectx, ep:integer)
```

**Arguments**

- `pctx` A parser context
- `ep` New endparse value

**Return value**
Current endparse value stored in the context.

**Related topics**

- `sys_endparse`, `getsepchar`, `gettrim`, `getqtype`

**Module**

`mmsystem`
**getenv**

**Purpose**
Get the value of an environment variable of the operating system.

**Synopsis**
```plaintext
function getenv(name:string|text):string
```

**Argument**
- **name**
  Name of the environment variable

**Return value**
Value of the environment variable (an empty string if the variable is not defined).

**Further information**
This procedure is included in the published interface of mmsystem (see Section 16.5).

**Example**
The value of the environment variable PATH is retrieved as follows:
```plaintext
str:= getenv("PATH")
```

**Related topics**
- setenv

**Module**
- mmsystem
getfsize

Purpose
Get the size of a file.

Synopsis
function getfsize(filename:string|text):integer

Argument
filename Name (and path) of the file

Return value
The size of the file in bytes or -1 in case of error

Further information
The function returns -1 if the file cannot be found or accessed and INT_MAX if the size exceeds the integer capacity (2Gb).

Module
mmsystem
getfstat

Purpose
Get the status (type and access mode) of a file or directory.

Synopsis
function getfstat(filename:string|text):integer

Argument
filename Name (and path) of the file or directory to check

Return value
Bit encoded type and mode of the given file.

Example
The following determines whether ftest is a directory and if it is writable:

    fstat:= getfstat("ftest")
    if bittest(fstat, SYS_TYP)=SYS_DIR
       then writeln("ftest is a directory")
    end-if
    if bittest(fstat, SYS_WRITE)=SYS_WRITE
       then writeln("ftest is writeable")
    end-if

Further information
The returned status type may be decoded using the constant mask SYS_TYP (the types are exclusive). Possible values are:
- SYS_DIR Directory
- SYS_REG Regular file
- SYS_OTH Special file (device, pipe...)

The access mode may be decoded using the constant mask SYS_MOD (the access modes are additive). Possible values are:
- SYS_READ Can be read
- SYS_WRITE Can be modified
- SYS_EXEC Is executable

Module
mmsystem
getftime

Purpose
Get time information of a file.

Synopsis
function getftime(filename:string|text,what:integer):real

Arguments
filename Name (and path) of the file
what Information requested. Possible values:
SYS_FTIM_ACC  Last access
SYS_FTIM_MOD  Last modification

Return value
The time requested as the number of seconds elapsed since 1/1/1970 at midnight or 0 in case of error.

Module
mmsystem
gethour

**Purpose**
Get the hour part of a time or datetime.

**Synopsis**
function gethour(t:time):integer
function gethour(dt:datetime):integer

**Arguments**
t A time object
dt A datetime object

**Return value**
Hour as an integer.

**Related topics**
getminute, getsecond, getmsec

**Module**
mmsystem
getminute

Purpose
Get the minute part of a time or datetime.

Synopsis
function getminute(t:time):integer
function getminute(dt:datetime):integer

Arguments
 t   A time object
 dt  A datetime object

Return value
Minute as an integer.

Related topics
gethour, getsecond, getmsec

Module
mmsystem
**getmonth**

**Purpose**
Get the month number of a date or datetime.

**Synopsis**
```
function getmonth(d:date):integer
function getmonth(dt:datetime):integer
```

**Arguments**
- `d` A date object
- `dt` A datetime object

**Return value**
Month number in the year.

**Related topics**
getyear, getday
**getmsec**

**Purpose**
Get the millisecond part of a time or datetime.

**Synopsis**

```plaintext
function getmsec(t:time):integer
function getmsec(dt:datetime):integer
```

**Arguments**
- `t` A `time` object
- `dt` A `datetime` object

**Return value**
Millisecond as an integer.

**Related topics**
- `gethour`, `getminute`, `getsecond`

**Module**
`mmsystem`
getpathsep

**Purpose**
Get the path separator of the running operating system.

**Synopsis**
function getpathsep:string

**Return value**
"." on Posix systems and ";" on Windows.

**Related topics**
getdirsep

**Module**
mmsystem
**getsucc, setsucc**

**Purpose**
Get and set succ (position of successor character) property of a text area.

**Synopsis**
function getsucc(ta:textarea):integer
procedure setsucc(ta:textarea, st:integer)

**Arguments**
ta  A text area object
st  New succ value

**Return value**
Current succ value stored in the object.

**Related topics**
gestart

**Module**
mmsystem
getqtype, setqtype

**Purpose**
Get and set qtype property of a parser context.

**Synopsis**
```pascal
function getqtype(pctx: parsectx): integer
procedure setqtype(pctx: parsectx, qt: integer)
```

**Arguments**
- `pctx` A parser context
- `qt` New qtype value

**Return value**
Current qtype value stored in the context.

**Related topics**
- `sys_qtype`, `getsepchar`, `gettrim`, `getendparse`

**Module**
- `mmsystem`
getsecond

Purpose
Get the second part of a time or datetime.

Synopsis
function getsecond(t:time):integer
function getsecond(dt:datetime):integer

Arguments
- t  A time object
- dt A datetime object

Return value
Second as an integer.

Related topics
gethour, getminute, getmsec

Module
mmsystem
getsepchar, setsepchar

**Purpose**
Get and set sepchar property of a parser context.

**Synopsis**

function getsepchar(pctx:parsecxt):integer
procedure setsepchar(pctx:parsecxt, sc:integer)

**Arguments**

pctx A parser context
sc New sepchar value

**Return value**
Current sepchar value stored in the context.

**Related topics**

sys_sepchar, getendparse, gettrim, getqtype

**Module**
mmsystem
**Purpose**
Get the size of a text.

**Synopsis**
```plaintext
function getsize(txt:text):integer
function getsize(ta:textarea):integer
```

**Arguments**
- `txt`: A text object
- `ta`: A text area object

**Return value**
The number of characters included in the text or text area.

**Module**
`mmsystem`
getstart, setstart

Purpose
Get and set start property of a text area.

Synopsis
function getstart(ta:textarea):integer
procedure setstart(ta:textarea, st:integer)

Arguments
  ta   A text area object
  st   New start value

Return value
Current start value stored in the object.

Related topics
  getsucc

Module
  mmsystem
**getsysinfo**

**Purpose**
Get information about the running operating system.

**Synopsis**
```
function getsysinfo:string
function getsysinfo(what:integer):string
function getsysinfo(I:Mosel):string
function getsysinfo(I:Mosel,what:integer):string
```

**Arguments**
- **what** What information to collect:
  - SYS_NAME Name of the operating system
  - SYS_VER Version name of the operating system
  - SYS_REL Release number of the operating system
  - SYS_PROC Processor type
  - SYS_ARCH Processor architecture (32 or 64 bit)
  - SYS_NODE Computer name
- **I** A Mosel instance

**Return value**
A text string reporting the requested information.

**Example**
The following prints the computer name and its operating system version:
```
writeln("Node ",getsysinfo(SYS_NODE),
       " is running ",getsysinfo(SYS_NAME+SYS_REL))
```

**Further information**
1. Several information items can be obtained in a single call by summing up the option codes. In such a case, the resulting string consists in the different items separated by commas.
2. When the function is used without the `what` parameter, all information items are returned.
3. This function may also be used with a Mosel instance as its first parameter. In this case the returned information relates to the system running this instance instead of the current system.

**Related topics**
- mmjobs

**Module**
mmsystem
getsysstat

Purpose
Get the system status.

Synopsis
function getsysstat:integer

Return value
0 if the last operation of the module was executed successfully.

Example
In this example we attempt to delete the file randomfile. If this is unsuccessful, a warning message is displayed:

    fdelete("randomfile")
    if getsysstat <> 0 then
        writeln("randomfile could not be deleted.")
    end-if

Module
mmsystem
gettime

Purpose
Get a time measure or the time part of a datetime.

Synopsis
function gettime:real
function gettime(dt:datetime):time

Argument
dt A datetime object

Return value
Time measure in seconds or a time object.

Example
The following prints the program execution time:

starttime := gettime ! Get the start time
... ! Do something
write("Time: ", gettime - starttime)

Further information
1. The measure returned by this function corresponds to the elapsed time since the module has been initialized (just before execution of the model starts).
2. The second form of this function is used to extract the time part of a datetime structure.

Related topics
getdate

Module
mmsystem
gettmpdir

Purpose
Get the temporary directory as a text object.

Synopsis
function gettmpdir:text

Return value
Temporary directory as a text object.

Further information
This function is equivalent to text(getparam("tmpdir")).

Module
mmsystem
gettrim, settrim

Purpose
Get and set trim property of a parser context.

Synopsis
function gettrim(pctx:parsectx):boolean
procedure settrim(pctx:parsectx, t:boolean)

Arguments
pctx A parser context
t New trim value

Return value
Current trim value stored in the context.

Related topics
sys_trim, getsepchar, getendparse, getqtype

Module
mmsystem
getweekday

**Purpose**
Compute the day of the week for a date or datetime.

**Synopsis**

```
function getweekday(d:date):integer
function getweekday(dt:datetime):integer
```

**Arguments**

- **d**: A date object
- **dt**: A datetime object

**Return value**
The number of the day in the week (1-7).

**Further information**
The first day of the week (number 1) is Monday.

**Module**

`mmsystem`
getyear

**Purpose**
Get the year part of a date or datetime.

**Synopsis**
function getyear(d:date):integer
function getyear(dt:datetime):integer

**Arguments**
d  A date object
dt  A datetime object

**Return value**
Year as an integer.

**Related topics**
calendar, getday

**Module**
mmsystem
inserttext

Purpose
Paste a text or string into a text.

Synopsis
procedure inserttext(txt:text, str:string, start:integer)
procedure inserttext(txt:text, src:text, start:integer)

Arguments
<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>txt</td>
<td>A text object</td>
</tr>
<tr>
<td>src</td>
<td>A text object</td>
</tr>
<tr>
<td>str</td>
<td>A string</td>
</tr>
<tr>
<td>start</td>
<td>Insert position</td>
</tr>
</tbody>
</table>

Example
The following:
```pascal
  t:=text("abcdefgh")
  inserttext(t,"123",2)
  writeln(t)
  inserttext(t,"456",8)
  writeln(t)
```
produces this output:
```
a123bcdefgh
a123bcd456efgh
```

Related topics
cuttext, deltext, pastetext, copytext

Module
mmsystem
isValid

Purpose
Check whether a date, time or datetime is valid.

Synopsis
function isValid(d:date):boolean
function isValid(t:time):boolean
function isValid(dt:datetime):boolean

Arguments
  d  A date object
  t  A time object
  dt A datetime object

Return value
True if the argument is valid.

Further information
A date is valid if its month number is in the range 1-12 and its day number is in the range 1-31 and is compatible with its month number (for instance 2006-2-29 is not a valid date). A time is valid if it is positive and smaller than an entire day. A datetime is valid if both its date part and its time part are valid.

Module
mmsystem
makedir

**Purpose**
Create a new directory in the given file system.

**Synopsis**
procedure makedir(dirname:string|text)

**Argument**
dirname          The name and path of the directory to be created

**Related topics**
removedir, makepath.

**Module**
mmsystem
**makepath**

**Purpose**
Create a new directory including its parents if necessary.

**Synopsis**
procedure makepath(dirname:string|text)
procedure makepath(dirname:string|text,last_is_file:boolean)

**Arguments**
dirname The name and path of the directory to be created
last_is_file If true, the last component of the path is ignored

**Further information**
1. This routine creates the directory $\text{dirname}$ as well as intermediate directories in the path if necessary. For instance, `makepath("/tmp/dir1/dir2")` will create "/tmp" then "/tmp/dir1" before "/tmp/dir1/dir2" if these directories are missing.

2. As opposed to `makedir`, this routine does not return an error condition if the path already exists.

3. The second form of this procedure can be used when the argument is a path to a file in order to create the directory in which the file can be created. For instance, `makepath("/tmp/dir1/myfile",true)` will create "/tmp/dir1" such that file "/tmp/dir1/myfile" can be created.

**Related topics**
removedir, makedir.

**Module**
mmsystem
newtar

Purpose
Create a Unix tar archive from a list of files.

Synopsis
procedure newtar(opt:integer, tarfile:text, dir:text, lsf:list of text|string)
procedure newtar(tarfile:text, lsf:list of text|string)

Arguments
opt Options:
SYS_NODIR Do not store directories (only files)
SYS_DIRONLY Store only directories
tarfile File name of the archive
dir Base directory (default: current directory)
lsf List of files to store in the archive (file names are relative to the dir directory)

Example
The following creates an archive of the Xpress installation including only binary files:
findfiles(SYS_RECURS, lsf, getenv("XPRESSDIR"), "bin/*/lib/*/dso/*")
newtar(0, "xpress.tar", getenv("XPRESSDIR"), lsf)

Further information
1. This implementation processes only regular files and directories: other file types (like links) are silently ignored and not included in the archive.
2. By default file names are represented according the current system encoding in the archive. To select a different encoding use the enc: file name prefix (see Section 2.15) on the archive name (e.g. "enc:utf-8,myarc.tar").

Related topics
tarlist, untar, newzip

Module
mmsystem
newzip

*Purpose*
Create a Zip archive from a list of files.

*Synopsis*
procedure newzip(opt:integer, zipfile:text, dir:text,
lsf:list of text|string, password:text)
procedure newzip(opt:integer, zipfile:text, dir:text,
lsf:list of text|string)
procedure newzip(zipfile:text, lsf:list of text|string)

*Arguments*
- **opt**
  - Options:
    - SYS_NODIR: Do not store directories (only files)
    - SYS_DIRONLY: Store only directories
- **zipfile**: File name of the archive
- **dir**: Base directory (default: current directory)
- **lsf**: List of files to store in the archive (file names are relative to the dir directory)
- **password**: Password to generate an encrypted zip file

*Example*
The following creates an archive of the Xpress installation including only binary files:

```plaintext
findfiles(SYS_RECURS, lsf, getenv("XPRESSDIR"), "bin/*;lib/*;dso/*")
newzip(0, "xpress.zip", getenv("XPRESSDIR"), lsf)
```

*Further information*
1. This implementation only supports the standard Zip format (only 32bit and basic encryption algorithm).
2. By default file names are represented according to the current system encoding in the archive. To select a different encoding use the `enc:` file name prefix (see Section 2.15) on the archive name (e.g. "enc:utf-8,myarc.zip").

*Related topics*
ziplist, unzip, newtar

*Module*
mmsystem
nextfield

Purpose
Advance to next field in a structured text string.

Synopsis
function nextfield(txt:text,start:integer,trim:boolean):boolean
function nextfield(txt:text):boolean
function nextfield(txt:text,pctx:parsectx):boolean

Arguments
txt   A text object
pctx  A parser context
start Starting position in the text
trim  Whether to skip blank characters around separators

Return value
true if more data can be parsed.

Example
The following function returns the list of records of a text string using comma as the field separator character:

function split(t:text):list of text
declarations
  pctx:parsectx
end-declarations

  pctx.sepchar:=44 ! ',,'
  while(nextfield(t,pctx)) do
    returned+=[parsetext(t,pctx)]
  end-do
end-function

Further information
1. When start is 0, this routine saves the position of the first character of the text string in control parameter sys_endparse and returns true.
2. When start is greater than 0 and the character located at position start is the separator character sys_sepchar, the position start+1 is saved in control parameter sys_endparse and true is returned. In all other cases false is returned.
3. This function returns false if the provided text txt is empty or the starting position start is not valid.
4. If argument trim is true, blank characters are skipped before and after the separator character. The provided value is saved in parameter sys_trim when start is 0.
5. In the second form of the routine, parameters sys_endparse and sys_trim are used as default values for arguments start and trim.
6. The version using a parser context works with the information contained in this context instead of the global parameters (see Section 16.1.5).

Related topics
parseint, parsereal, parseextn, parsetext
Module

mmsystem
openpipe

Purpose
Start an external process for bidirectional communication.

Synopsis
procedure openpipe(cmd:string|text)

Argument
cmd The command to be executed in the separate process

Example
The following example uses an external program sort (we assume it writes a sorted copy of what it reads) to display a sorted list of the content of set ToSort:

openpipe("sort")
forall(i in ToSort)
  writeln(i)
fclose(F_OUTPUT)
while(not iseof) do
  readln(l)
  writeln(l)
end-do
fclose(F_INPUT)

Further information
1. Pipes required by this procedure are created using the pipe driver of this module (see Section 16.4.2). As a consequence, the string provided as argument must be suitable for the driver (i.e. a program name followed by its options separated by spaces).

2. This procedure opens both an input and output streams that must be closed explicitly using fclose. Note that the output stream must be closed first otherwise the program may lock up.

3. When Mosel is running in restricted mode (see Section 1.3.3), this procedure behaves like the system procedure.

Module
mmsystem
parseextn

Purpose
Initialise an object of a module type from a text.

Synopsis
procedure parseextn(txt:text,start:integer,e:mtype)
procedure parseextn(txt:text,e:mtype)
procedure parseextn(txt:text,pctx:parsectx,e:mtype)
procedure parseextn(txt:text,ta:textarea,e:mtype)

Arguments
  txt  A text object
  pctx A parser context
  ta   A text area object
  start Starting position in the text
  e    An object of an external type

Further information
1. This function can only be used with types supporting initialisation from a string. The parsing begins at the specified starting position and stops as soon as an invalid character is found. This location is then stored in the parameter sys_endparse. This control parameter is used as the starting position when start is not provided.

2. In case of error the system status is set with a non-zero value (see getsysstat).

3. The version using a parser context works with the information contained in this context instead of the global parameters (see Section 16.1.5).

4. If used with a textarea object the routine uses the start property of the object as the starting position and does not modify the parameter sys_endparse.

Related topics
parseint, parsedreal, parsetext, nextfield, sys_endparse

Module
mmsystem
**parseint**

**Purpose**
Convert a text into an integer.

**Synopsis**

```plaintext
function parseint(txt: text,start: integer): integer
function parseint(txt: text,start: integer,base: integer): integer
function parseint(txt: text): integer
function parseint(txt: text,pctx: parsectx): integer
function parseint(txt: text,pctx: parsectx,base: integer): integer
function parseint(txt: text,ta: textarea): integer
function parseint(txt: text,ta: textarea,base: integer): integer
```

**Arguments**

txt A text object
pctx A parser context
ta A text area object
start Starting position in the text
base Base to use for the conversion (between 2 and 36)

**Return value**
The integer represented by the string.

**Example**
The following:

```plaintext
t:=text("a123.4b")
writeln(parseint(t,2))
writeln(getparam("sys_endparse"))
```

produces this output:

```
123
5
```

**Further information**

1. The parsing begins at the specified starting position and stops as soon as an invalid character is found. This location is then stored in the parameter `sys_endparse`. This control parameter is used as the starting position when `start` is not provided.

2. In case of error (no valid character found or overflow) the system status is set with a non-zero value (see `getsysstat`) and, depending on the situation, 0, `MAX_INT` or `-MAX_INT-1` is returned.

3. The optional `base` argument may be used if the text is not expressed in base 10. Valid values for this parameter is 0 and 2 to 36. If base is zero or 16, the string may then include a '0x' prefix, and the number will be read in base 16. Furthermore, if the base is 0, the text will be read in base 8 if the first character is 0 and in base 10 otherwise.

4. The base value may also be negative: in this case the input data is interpreted as an unsigned integer.

5. The version using a parser context works with the information contained in this context instead of the global parameters (see Section 16.1.5).

6. If used with a `textarea` object the routine uses the `start` property of the object as the starting position and does not modify the parameter `sys_endparse`. 
Related topics
parsereal, parseextrn, parsetext, nextfield, sys_endparse

Module
mmsystem
parsereal

Purpose  
Convert a text into a real.

Synopsis  
function parsereal(txt:text,start:integer):real
function parsereal(txt:text):real
function parsereal(txt:text,pctx:parsectx):real
function parsereal(txt:text,ta:textarea):real

Arguments  
txt    A text object
pctx   A parser context
ta     A text area object
start  Starting position in the text

Return value  
The real represented by the string.

Example  
The following:
    t:=text("a123.4b")
    writeln(parsereal(t,2))
    writeln(getparam("sys_endparse"))
produces this output:
            123.4
            7

Further information
1. The parsing begins at the specified starting position and stops as soon as an invalid character is found. This location is then stored in the parameter sys_endparse. This control parameter is used as the starting position when start is not provided.

2. In case of error (no valid character found or overflow) the system status is set with a non-zero value (see getsysstat) and, depending on the situation, 0, MAX_REAL or -MAX_REAL is returned.

3. The version using a parser context works with the information contained in this context instead of the global parameters (see Section 16.1.5).

4. If used with a textarea object the routine uses the start property of the object as the starting position and does not modify the parameter sys_endparse.

Related topics  
parseint, parseextn, parsetext, nextfield, sys_endparse

Module  
mmsystem
**parsetext**

**Purpose**
Extract a text from a text.

**Synopsis**

```plaintext
function parsetext(txt:text,start:integer):text
function parsetext(txt:text):text
function parsetext(txt:text,pctx:parsectx):text
function parsetext(txt:text,ta:textarea):text
```

**Arguments**

- **txt**: A text object
- **pctx**: A parser context
- **ta**: A text area object
- **start**: Starting position in the text

**Return value**
Decoded text.

**Example**
The following:

```plaintext
t:=text("a123.4b")
setparam("sys_sepchar",46) ! ',
writeln(parsetext(t,2))
writeln(getparam("sys_endparse"))
```

produces this output:

```
123
5
```

**Further information**

1. The behaviour of this routine depends on 2 control parameters: *sys_sepchar* defines a field separator that may mark the end of a non-quoted string and the parameter *sys_qtype* specifies the convention to use for quoted strings: if this parameter has value 0 (the default), Mosel quoting convention is used (both single and double quotes may be employed and with double quotes escape sequences are allowed); with value -1 no quoting is expected; with value 1, C/C++ quoting convention applies (only double quotes with escape sequences). Finally, with value 2, CSV convention is expected (double quotes and repetition of double quotes to escape this character). The returned string is decoded: quotes are removed and escape sequences are replaced by their corresponding characters.

2. The parsing begins at the specified starting position and stops as soon as the separator character *sys_sepchar* is found or the quoted string is terminated. This location is then stored in the parameter *sys_endparse*. This control parameter is used as the starting position when start is not provided.

3. In case of error, getsysstat will return a negative value. A positive value indicates that a quoted string is unfinished (i.e. the end of the source text is reached although no matching quote has been found).

4. The version using a parser context works with the information contained in this context instead of the global parameters (see Section 16.1.5).

5. If used with a textarea object (see Section 16.1.6) the routine uses the start property of the object as the starting position and does not modify the parameter *sys_endparse*. 
Related topics
parseint, parsereal, parseextn, nextfield, sys_sepchar, sys_qtype, sys_endparse

Module
mmsystem
**pasteText**

**Purpose**
Paste a text or string into a text.

**Synopsis**

```plaintext
procedure pastetext(txt:text, str:string, start:integer)
procedure pastetext(txt:text, src:text, start:integer)
```

**Arguments**
- **txt**  A text object
- **src**  A text object
- **str**  A string
- **start** Paste position

**Example**
The following:

```plaintext
t := text("abcdefgh")
pastetext(t,"123",2)
writeln(t)
pastetext(t,"456",8)
writeln(t)
```

produces this output:

```
a123efgh
a123efg456
```

**Related topics**
- cuttext, inserttext, deltext, copytext

**Module**
- mmsystem
**pathmatch**

**Purpose**
Check whether a file name matches a given pattern.

**Synopsis**
function pathmatch(filename:string|text,pattern:string|text):boolean

**Arguments**
- filename: The file name to evaluate
- pattern: Matching pattern that may include * (any text of any length) or ? (any single character)

**Return value**
true if the file name matches the pattern.

**Example**
The following function identifies Mosel source file names:

```mosel
function is_mosel_file(f:text):boolean
  returned:=pathmatch(f,"*.mos")
end-function
```

**Further information**
The comparison respects the operating environment conventions and behaviour may differ depending on the operating system. In particular, under Posix systems comparisons are case sensitive; this is not the case on Windows (i.e. file names are not case sensitive).

**Related topics**
- regmatch

**Module**
mmsystem
pathsplit

**Purpose**
Split a path into its components.

**Synopsis**

```
function pathsplit(how:integer,path:text,rem:text):text
function pathsplit(how:integer,path:text):text
```

**Arguments**

- `how` How to split the path:
  - `SYS_DIR` Directory (i.e. part preceding the last directory separator)
  - `SYS_FNAME` File name (i.e. part following the last directory separator)
  - `SYS_EXTN` File name extension (i.e. part following the last dot)
- `path` The path name to split
- `rem` Remaining part of the path after the returned value has been removed

**Return value**
The requested part of the path.

**Example**
The following function returns the base name of a path (file name without directory and extension):

```
function basename(f:text):text
  returned:=pathsplit(SYS_FNAME,f)
  dummy:=pathsplit(SYS_EXTN,returned,returned)
end-function
```

**Further information**
Arguments `path` and `rem` can be the same object.

**Module**
`mmsystem`
qsort

Purpose
Sort a list or an array or (a subset of) the indices of an array.

Synopsis
procedure qsort(sense:boolean, lvals:list of integer|real)
procedure qsort(sense:boolean, lvals:list of string|text)
procedure qsort(sense:boolean, vals:array of integer|real)
procedure qsort(sense:boolean, vals:array of string|text)
procedure qsort(sense:boolean, vals:array of integer|real, ndx:array)
procedure qsort(sense:boolean, vals:array of string|text, ndx:array)
procedure qsort(sense:boolean, vals:array of integer|real, ndx:array, sel:set)
procedure qsort(sense:boolean, vals:array of string|text, ndx:array, sel:set)
procedure qsort(sense:boolean, vals:array of integer|real, lndx:list, sel:set)
procedure qsort(sense:boolean, vals:array of string|text, lndx:list, sel:set)

Arguments
sense  Sense of the sorting:
    SYS_UP   Ascending order
    SYS_DOWN  Descending order
lvals  List to be sorted
vals  One-dimensional array to be sorted
ndx  One-dimensional array of the same type and size as the indexing set of vals
lndx List of the same type as the indexing set of vals
sel  Subset of the indexing set of vals

Example
The following example sorts an array of real numbers:

declarations
    ar: array(1..10) of real
end-declarations

ar:: [1.2, -3, -8, 10.5, 4, 7, 2.9, -1, 0, 5]
qsort(true, ar)
writeln("Sorted array: ", ar)

Further information
1. In the first four versions of the procedure (with two arguments, sense and vals or lvals) the input array (list) vals (lvals) is overwritten by the resulting sorted array (list).
2. When an array ndx is provided, the resulting sorted array is returned in the argument ndx in the form of its sorted index set. If a selection set sel of indices is provided, only the specified indices are processed.
3. When a list lndx is provided, the resulting sorted array is returned in the argument lndx in the form of a list of sorted indices. If a selection set sel of indices is provided, only the specified indices are processed.
4. When applied to a dynamic array this procedure processes all indices of the index set including those not referring to an existing cell (a subset of the indexing set sel can be used to select only the existing entries).

Module
mmsystem
quote

Purpose
Quote and encode a text string.

Synopsis
function quote(txt:text,qtype:integer,sepchar:integer):text
function quote(txt:text):text

Arguments
<table>
<thead>
<tr>
<th>txt</th>
<th>A text object</th>
</tr>
</thead>
<tbody>
<tr>
<td>qtype</td>
<td>Quoting convention</td>
</tr>
<tr>
<td>sepchar</td>
<td>Code of the separator character or 0</td>
</tr>
</tbody>
</table>

Example
The following statement:

    writeln(quote('test CSV "quoted" string',2,44))

displays: "test CSV ""quoted"" string"

Further information
1. This function generates an encoded form of the provided text string according to the given quoting convention qtype (see sys_qtype) and separator character sepchar. The provided text may be returned unchanged if the selected convention does not require quotes and the text does not include any special character or the specified separator character.

2. If argument sepchar is 0, quoting is enforced even if the selected quoting convention would not require quotes.

3. In the second form of the routine, parameters sys_qtype and sys_sepchar are used as default values for arguments qtype and sepchar.

Related topics
parsetext

Module
mmsystem
readtextline

Purpose
Read a line of text from the current input stream.

Synopsis
function readtextline(txt:text):integer

Argument
txt       A text object

Return value
Number of characters read or -1 if end of file.

Module
mmsystem
**regmatch**

**Purpose**
Compare text strings using a regular expression.

**Synopsis**

```
function regmatch(src:text, regex:string):boolean
function regmatch(src:text, regex:string, start:integer, flags:integer,
                   mp:array(range) of textarea):boolean
```

**Arguments**

- **src** Text to process
- **regex** Regular expression
- **start** Position where to start the search
- **flags** Search options:
  - **REG_EXTENDED** Use Extended Regular Expression syntax (ERE), default is to interpret the expression as a Basic Regular Expression (BRE)
  - **REG_ICASE** Comparison is performed case insensitive (by default it is case sensitive)
  - **REG_NEWLINE** The character newline (\n) is treated as the end of line (by default it is handled as an ordinary symbol)
  - **REG_NOTBOL** The beginning of the text string is not the beginning of a line
  - **REG_NOTEOL** The end of the text string is not the end of a line
- **mp** Matching regions as an array of textarea objects

**Return value**

true if a match was found.

**Example**
The following example extracts the value of ‘pars2’ from an input text consisting of lines of the form `name=value`:

```
declarations
  m:array(range) of textarea
  t:text
end-declarations

t:="p1=10\npars2=234\nparam9=56\n"
if regmatch(t,'pars2=\(.*\)$',1,REG_NEWLINE,m) then
  pars2:=parseInt(t,m(1))
  writeln(pars2)
end-if
```

**Further information**

1. This function relies on the TRE library (see [http://laurikari.net/tre](http://laurikari.net/tre)). Please refer to the documentation of this library for a detailed description of the supported expression syntax.

2. When the **mp** argument is provided and the search is successful, the result of the processing is returned via this array as textarea objects (see Section 16.1.6): the array cell 0 refers to the entire matching region and the following ones to each of the subexpressions.

**Related topics**

- pathmatch, regreplace, sys_regcache

**Module**

mmsystem
**regreplace**

**Purpose**
Replace portions of a text string based on a regular expression.

**Synopsis**

```plaintext
function regreplace(src:text, regex:string, repl:string):integer
function regreplace(src:text, regex:string, repl:string, start:integer,
                   flags:integer):integer
```

**Arguments**

- **src** Text to process
- **regex** Regular expression
- **repl** Replacement string expression
- **start** Position where to start the search
- **flags** Search options:
  - **REG_EXTENDED** Use Extended Regular Expression syntax (ERE), default is to interpret the expression as a Basic Regular Expression (BRE)
  - **REG_ICASE** Comparison is performed case insensitive (by default it is case sensitive)
  - **REG_NEWLINE** The character `\n` is treated as the end of line (by default it is handled as an ordinary symbol)
  - **REG_NOTBOL** The beginning of the text string is not the beginning of a line
  - **REG_NOTEOL** The end of the text string is not the end of a line
  - **REG_ONCE** Stop after the first replacement (by default the entire input string is processed)

**Return value**
The number of replacements performed.

**Example**
The following statement transforms dates expressed as `year-month-day` to dates in the form `day/month/year`

```plaintext
nbr:=regreplace(t,
               '(\[[[:digit:]]\]{4})-(\[[01]\][[:digit:]]\)-(\[0-3]\[[[:digit:]]\])',
               '\3/\2/\1',1,REG_EXTENDED)
```

**Further information**
1. This function relies on the TRE library (see [http://laurikari.net/tre](http://laurikari.net/tre)). Please refer to the documentation of this library for a detailed description of the supported expression syntax.
2. In the replacement string `repl` the backslash character (`'\'`) has a special meaning: if followed by another backslash character it is replaced by a single backslash; if followed by a digit it is replaced by the corresponding subexpression defined by the regular expression. The subexpression number 0 corresponds to the entire matching region.

**Related topics**
- `regmatch`, `sys_regcache`

**Module**
- `mmsystem`
removedir

**Purpose**
Remove a directory.

**Synopsis**
```plaintext
procedure removedir(dirname:string|text)
```

**Argument**
dirname The name and path of the directory to delete

**Further information**
For deletion of a directory to succeed, the given directory must be empty.

**Related topics**
fdelete, makedir.

**Module**
mmsystem
removefiles

Purpose
Remove files selected using file name patterns.

Synopsis
procedure removefiles(opt:integer, dir:text,filters:text)
procedure removefiles(filters:text)

Arguments
opt Options (several options can be combined):
SYS_RECURS Recursive search in subdirectories
SYS_NODIR Do not remove directories (only files)
SYS_DIRONLY Remove only directories
dir Base directory for the search (default: current directory)
filters File name filters (default: all files removed)

Example
The following deletes directory "mydir" including its content:

    removefiles(SYS_RECURS,"mydir","*")
    removedir("mydir")

Further information
1. The filters argument consists in a list of patterns separated by the symbol ";". A pattern is composed of a path (using the usual operating system conventions) which last component may include wildcard characters "*" (any text of any length), "?" (any single character) and "|" (logical "or"). For instance "bin/\*.exe;models/\*.mos|*.dat" will select all files with extension ".exe" in the "bin" directory as well as files with extension ".mos" and ".dat" in the "models" directory.

2. File name matching is achieved using function pathmatch and differences may be observed depending on the operating system (e.g. file names are case sensitive under Posix systems but not under Windows).

Related topics
findfiles, fdelete, removedir

Module
mmsystem
setchar

Purpose
Set a character in a text.

Synopsis
procedure setchar(txt:text, index:integer, c:integer)

Arguments
txt    A text object
str    String
index  Position of the character
c      Character code

Further information
If the index requested is after the end of the text, the text is expanded as necessary and the
newly created space is padded with the character which code is the parameter sys_fillchar.

Related topics
getchar, sys_fillchar, pastetext

Module
mmsystem
setdate

Purpose
Set the date part of a datetime.

Synopsis
procedure setdate(dt:datetime,d:date)

Arguments
dt A datetime object
d A date object

Related topics
settime

Module
mmsystem
setday

Purpose
Set the day number of a date or datetime.

Synopsis
procedure setday(d:date,j:integer)
procedure setday(dt:datetime,j:integer)

Arguments
- d    A date object
- dt   A datetime object
- j    Day number

Related topics
setyear, setmonth

Module
mmsystem
setenv

**Purpose**
Set the value of an environment variable of the operating system.

**Synopsis**
```plaintext
procedure setenv(name:string|text, value:string|text)
```

**Arguments**
- **name**   Name of the environment variable
- **value**  New value for the environment variable

**Further information**
1. The environment variable is deleted if it is assigned an empty string.
2. Variables created or modified with this procedure can be retrieved using the `getenv` function and are inherited by processes started by `system` or `openpipe`.
3. The effect of this procedure is local to the running model (i.e. system calls like the C function `getenv` will not work for these variables). However, another module may access the environment maintained by `mmsystem` using the IMCI function `getenv` (see Section 16.5).
4. This procedure is included in the published interface of `mmsystem` (see Section 16.5).

**Related topics**
- `getenv`, `system`, `openpipe`.

**Module**
- `mmsystem`
sethour

**Purpose**
Set the hour part of a time or datetime.

**Synopsis**
```
procedure sethour(t:time,h:integer)
procedure sethour(dt:datetime,h:integer)
```

**Arguments**
- **t** A time object
- **dt** A datetime object
- **h** Hour

**Related topics**
setminute, setsecond, setmsec
setminute

Purpose
Set the minute part of a time or datetime.

Synopsis
procedure setminute(t:time,m:integer)
procedure setminute(dt:datetime,m:integer)

Arguments
\[ t \quad \text{A time object} \]
\[ dt \quad \text{A datetime object} \]
\[ m \quad \text{Minute} \]

Related topics
sethour, setsecond, setmsec

Module
mmsystem
**setmonth**

**Purpose**
Set the month number of a date or datetime.

**Synopsis**
```
procedure setmonth(d:date,m:integer)
procedure setmonth(dt:datetime,m:integer)
```

**Arguments**
- **d**  A date object
- **dt** A datetime object
- **m**   Month number

**Related topics**
- setyear, setday
setmsec

**Purpose**
Set the millisecond part of a time or datetime.

**Synopsis**
procedure setmsec(t:time,ms:integer)
procedure setmsec(dt:datetime,ms:integer)

**Arguments**
- t A time object
- dt A datetime object
- ms Millisecond

**Related topics**
sethour, setminute, setsecond

**Module**
 mmsystem
**setsecond**

**Purpose**
Set the second part of a time or datetime.

**Synopsis**

```plaintext
procedure setsecond(t:time,s:integer)
procedure setsecond(dt:datetime,s:integer)
```

**Arguments**
- `t` A time object
- `dt` A datetime object
- `s` Second

**Related topics**
- sethour, setminute, setmsec

**Module**
- mmsystem
settime

Purpose
Set the time part of a datetime.

Synopsis
procedure settime(dt:datetime,t:time)

Arguments
  dt  A datetime object
  t   A time object

Related topics
setdate

Module
mmsystem
setyear

**Purpose**
Set the year part of a date or datetime.

**Synopsis**

procedure setyear(d:date,y:integer)
procedure setyear(dt:datetime,y:integer)

**Arguments**

- d A date object
- dt A datetime object
- y Year

**Related topics**

setmonth, setday

**Module**

mmsystem
sleep

**Purpose**
Suspend execution for a fixed amount of time.

**Synopsis**
procedure sleep(duration:int)

**Argument**
duration  Sleep time in milliseconds

**Further information**
The model uses no CPU while it is suspended.

**Module**
mmsystem
startswith

**Purpose**
Check whether a text or string starts with a given string.

**Synopsis**
function startswith(txt: text|string, tofs: text|string): boolean
function startswith(txt: text|string, tofs: text|string, start: integer): boolean

**Arguments**
- `txt`: A string or text object
- `tofs`: String to find
- `start`: Starting position for the search

**Return value**
true if the beginning of `txt` corresponds to `tofs`.

**Related topics**
endswith

**Module**
mmsystem
**system**

**Purpose**
Execute an external program.

**Synopsis**
procedure system(command:string|text)

**Argument**
command The command to be executed

**Example**
The following displays the functionality of the mmsystem module using the program mosel:

```plaintext
system('mosel -s -c "exam mmsystem"')
```

**Further information**

1. The given program is executed directly: if the specified expression is a shell command, it is necessary to call the shell explicitly. For instance to get a directory listing under Windows the command will be "cmd /C dir".

2. Using this procedure should be avoided in applications that are to be run on different systems because such a call is always system dependent and may not be portable.

3. The generated process inherits the current system environment plus the environment variables modified/created using the setenv procedure.

4. The default output and error streams of the generated process are redirected to the corresponding Mosel streams. The default input stream is closed.

5. This procedure is included in the published interface of mmsystem (see Section 16.5).

6. When Mosel is running in restricted mode (see Section 1.3.3), the restriction NoExec disables this routine unless the environment variable MOSEL_EXECPATH is defined. This variable, used in a similar way as the PATH environment variable, gives a list of paths than can still be used under the restriction. In addition to directories, the definition of the variable may include paths to executables such that it may directly specify a list of programs. It is also worth noting that no search is performed (i.e. executables must be given with their full path) and that path expansion is performed a the time of loading mmsystem relative to the Mosel initial working directory.

7. The command may be preceded by the prefix "enc:" to specify the encoding of the output streams (see Section 2.15).

**Module**
mmsystem
**tarlist**

**Purpose**
Get the list of files included in a Unix tar archive.

**Synopsis**
procedure tarlist(opt:integer,tarfile:text,lsf:list of text,
filters:string)
procedure tarlist(tarfile:text,lsf:list of text)

**Arguments**
- **opt** Options:
  - SYS_NODIR Do not report directories (only files)
  - SYS_DIRONLY Report only directories
- **tarfile** File name of the archive
- **lsf** Resulting list of file names
- **filters** File name filters (default: all files reported)

**Example**
The following prints the list of files included in the archive myfiles.tar:

```delphi
  tarlist("myfiles.tar",lsf)
  writeln(lsf)
```

**Further information**
1. The `filters` argument has a similar structure as the corresponding argument of procedure `findfiles` except that wildcard characters "*" and "?" may appear anywhere in a path. A file is reported if it matches any of the patterns of this list.
2. When evaluating the filters, file name matching is achieved using function `pathmatch` and differences may be observed depending on the operating system (e.g. file names are case sensitive under Posix systems but not under Windows).
3. This implementation processes only regular files and directories: other file types included in the archive (like links) are silently ignored.
4. By default file names are expected to be represented according the current system encoding in the archive. To select a different encoding use the `enc:` file name prefix (see Section 2.15) on the archive name (e.g. "enc:utf-8,myarc.tar").

**Related topics**
untar, newtar, ziplist

**Module**
mmsystem
**textfmt**

**Purpose**
Create a formatted text from a string, a text or a number.

**Synopsis**

```plaintext
function textfmt(str:string, len:integer):text
function textfmt(txt:text, len:integer):text
function textfmt(i:integer, len:integer):text
function textfmt(i:integer, len:integer, flag:integer, base:integer):text
function textfmt(r:real, len:integer):text
function textfmt(r:real, len:integer, dec:integer):text
```

**Arguments**

- **str**  String to be formatted
- **txt**  Text to be formatted
- **i**    Integer to be formatted
- **r**    Real to be formatted
- **len**  Reserved length (may be exceeded if given string is longer, in this case the string is always left justified).
  - `<0`  Left justified within reserved space
  - `>0`  Right justified within reserved space
  - `0`   Use defaults
- **flag** Bit encoded options:
  - `1`  Left padding with "0" (instead of space)
  - `2`  Use capital letters for bases greater than 10
- **base** Encoding base (between 2 and 36)
- **dec** Number of digits after the decimal point

**Return value**
Formatted text.

**Example**
The following:

```plaintext
writeln("text1", textfmt("text2",8), "text3")
writeln("text1", textfmt("text2",-8), "text3")
r:=789.123456
writeln(textfmt(r,0)," ", textfmt(r,4,2), textfmt(r,8,0))
```

produces this output:

```
  text1    text2text3
text1text2   text3
789.123  789.12   789
```

**Further information**

1. If the resulting string is longer than the reserved space it is not cut but printed in its entirety, overflowing the reserved space to the right.

2. When processing an integer specifying a base, the provided value is treated as an unsigned integer if the base is negative.

**Related topics**

formattext
Module

mmsystem
tolower

Purpose
Generate the lowercase version of the provided text.

Synopsis
function tolower(t:text|string):text
function tolower(c:integer):integer

Return value
The lowercase version of the input string or a character code.

Arguments
t Text to convert
c Character code

Further information
When this function is used with a text string, it returns a copy of its argument converted to lowercase. When it is called with an integer, the returned value corresponds to the character code of the lowercase version of the provided code. In both cases, the function will return an unmodified copy of its argument if no conversion can be done.

Related topics
toupper

Module
mmsystem
**toupper**

**Purpose**
Generate the uppercase version of the provided text.

**Synopsis**
function toupper(t: text|string): text
function toupper(c: integer): integer

**Return value**
The uppercase version of the input string or a character code.

**Arguments**
t Text to convert
c Character code

**Further information**
When this function is used with a text string, it returns a copy of its argument converted to uppercase. When it is called with an integer, the returned value corresponds to the character code of the uppercase version of the provided code. In both cases, the function will return an unmodified copy of its argument if no conversion can be done.

**Related topics**
tolower

**Module**
mmsystem
trim

Purpose
Remove blank characters at the beginning and/or end of a text string.

Synopsis
procedure trim(t: text)
procedure trim(t: text, where: boolean)

Arguments
- t: Text to trim
- where: Part of the text to trim:
  - SYS_LEFT: Beginning of the string
  - SYS_RIGHT: End of the string

Further information
When the function is used with a single argument, both starting and ending blank characters are deleted.

Module
mmsystem
untar

**Purpose**
Extract files from a Unix tar archive.

**Synopsis**

```plaintext
procedure untar(opt:integer,tarfile:text,dir:text,
                filters:string)
procedure untar(tarfile:text,dir:text)
procedure untar(tarfile:text)
```

**Arguments**

<table>
<thead>
<tr>
<th>opt</th>
<th>Options:</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS_OVERWRT</td>
<td>Replace existing files</td>
</tr>
<tr>
<td>SYS_NODIR</td>
<td>Do not extract directories (only files)</td>
</tr>
<tr>
<td>SYS_DIRONLY</td>
<td>Extract only directories</td>
</tr>
</tbody>
</table>

- **tarfile**: File name of the archive
- **dir**: Destination path (default: current directory)
- **filters**: File name filters (default: all files extracted)

**Example**

The following extracts all files included in the archive `myfiles.tar` to directory `mydir`:

```plaintext
untar("myfiles.tar","mydir")
```

**Further information**

1. The `filters` argument has a similar structure as the corresponding argument of procedure `findfiles` except that wildcard characters "*" and "?" may appear anywhere in a path. A file is extracted if it matches any of the patterns of this list.

2. When evaluating the filters, file name matching is achieved using function `pathmatch` and differences may be observed depending on the operating system (e.g. file names are case sensitive under Posix systems but not under Windows).

3. This implementation processes only regular files and directories: other file types included in the archive (like links) are silently ignored.

4. By default file names are expected to be represented according the current system encoding in the archive. To select a different encoding use the `enc:` file name prefix (see Section 2.15) on the archive name (e.g. "enc:utf-8,myarc.tar").

**Related topics**

- `tarlist`
- `newtar`
- `unzip`

**Module**

`mmsystem`
unzip

**Purpose**
Extract files from a Zip archive.

**Synopsis**

```
procedure unzip(opt:integer, zipfile:text, dir:text, filters:string, password:text)
procedure unzip(opt:integer, zipfile:text, dir:text, filters:string)
procedure unzip(zipfile:text, dir:text)
procedure unzip(zipfile:text)
```

**Arguments**

- **opt**: Options:
  - SYS_OVERWRT: Replace existing files
  - SYS_NODIR: Do not extract directories (only files)
  - SYS_DIRONLY: Extract only directories
- **zipfile**: File name of the archive
- **dir**: Destination path (default: current directory)
- **filters**: File name filters (default: all files extracted)
- **password**: Password to access an encrypted archive

**Example**

The following extracts all files included in the archive *myfiles.zip* to directory *mydir*:

```
unzip("myfiles.zip","mydir")
```

**Further information**

1. The `filters` argument has a similar structure as the corresponding argument of procedure `findfiles` except that wildcard characters "*" and "?" may appear anywhere in a path. A file is extracted if it matches any of the patterns of this list.

2. When evaluating the filters, file name matching is achieved using function `pathmatch` and differences may be observed depending on the operating system (e.g. file names are case sensitive under Posix systems but not under Windows).

3. This implementation only supports the standard Zip format (only 32bit and basic encryption algorithm).

4. By default file names are expected to be represented according the current system encoding in the archive. To select a different encoding use the `enc: file name prefix` (see Section 2.15) on the archive name (e.g. "enc: utf-8, myarc.zip").

**Related topics**

- ziplist, newzip, untar

**Module**

mmsystem
**ziplist**

**Purpose**
Get the list of files included in a Zip archive.

**Synopsis**

```pascal
procedure ziplist(opt:integer, zipfile:text, lsf:list of text,
filters:string)
procedure ziplist(zipfile:text, lsf:list of text)
```

**Arguments**
- **opt**  Options:
  - SYS_NODIR  Do not report directories (only files)
  - SYS_DIRONLY  Report only directories
- **zipfile**  File name of the archive
- **lsf**  Resulting list of file names
- **filters**  File name filters (default: all files reported)

**Example**

The following prints the list of files included in the archive `myfiles.zip`:

```pascal
ziplist("myfiles.zip",lsf)
writeln(lsf)
```

**Further information**

1. The `filters` argument has a similar structure as the corresponding argument of procedure `findfiles` except that wildcard characters "*" and "?" may appear anywhere in a path. A file is reported if it matches any of the patterns of this list.

2. When evaluating the filters, file name matching is achieved using function `pathmatch` and differences may be observed depending on the operating system (e.g. file names are case sensitive under Posix systems but not under Windows).

3. By default file names are expected to be represented according the current system encoding in the archive. To select a different encoding use the `enc:` file name prefix (see Section 2.15) on the archive name (e.g. "enc:utf-8,myarc.zip").

**Related topics**
- unzip, newzip, tarlist

**Module**
- mmsystem
16.4 I/O drivers

The mmsystem module provides two IO drivers: the first one allows to use a string or text object as a file and the second connects a Mosel input or output stream to a program started in a different process. Using this driver, it is possible to get the output of an external program (for instance the result of a preprocessor to feed the Mosel compiler) or implement a basic bidirectional inter process communication thanks to the openpipe procedure (which relies on this IO driver).

16.4.1 Driver text

The file name for this driver is an external program with its options. Options are separated by spaces or tabulations and may be quoted using either single or double quotes. A quoted option may contain any kind of character except the quote used to delimit the string.

When the system opens a pipe, a new process is started for executing the given program and default input and output streams are directed to system pipes. If the file is open for reading (resp. writing), the default output stream (resp. input stream) of the new process becomes the current input stream (resp. output stream) of the model. To locate the program to be executed, the system relies on the PATH environment variable. Detection of error (typically the program cannot be found or is not executable) differs depending on the operating system: under Windows, the error is reported immediately and the pipe is not open. With Posix systems, no error is reported but following IO operations fail.

When the file is closed, both input and output streams of the external process are closed then the system waits for its termination: in order to avoid a lock up of the Mosel program one must make sure that the external program ends its execution when default input and output streams are closed.

Example: the following command could be used with Mosel Console for compiling the model mymod.mos after it has been processed by the C preprocessor. Note that we have to provide an output file name since the compiler cannot deduce it from the source file name.

For a Posix systems:

```
compile 'mmsystem.pipe:cpp mymod.mos' '' mymod.bim
```
For Windows (with MSVC):

```
compile 'mmsystem.pipe:cl /E mymod.mos' ' ' mymod.bim
```

When Mosel is running in restricted mode (see Section 1.3.3), this driver behaves like the system procedure.

16.5 Published library functions

The module mmsystem publishes its implementation of `getenv`, `setenv` and `system` as well as the functions `gettxtsize`, `gettxtbuf` and `txtresize` for text access via the service IMCI for use by other modules (see the Mosel Native Interface Reference Manual for more detail about services). The list of published functions is contained in the interface structure `mmsystem_imci` that is defined in the module header file `mmsystem.h`.

From another module, the context of mmsystem and its communication interface can be obtained using functions of the Mosel Native Interface as shown in the following example.

```c
static XPRMnifct mm;
XPRMcontext mmctx;
XPRMdsolib dso;
mmsystem_imci mmsys;
void *sysctx;

dso=mm->finddso("mmsystem"); /* Retrieve the mmsystem module*/
sysctx**=mm->getdsoctx(mmctx, dso, (void **)(&mmsys));
/* Get the module context and the communication interface of mmsystem */
```

Typically, a module calling functions that are provided by mmsystem will include this module into its list of dependencies in order to make sure that mmsystem will be loaded by Mosel at the same time as the calling module. The “dependency” service of the Mosel Native Interface has to be used to set the list of module dependencies:

```c
static const char *deplist[]={"mmsystem",NULL}; /* Module dependency list */
static XPRMdsoserv tabserv[]= /* Table of services */
{
  {XPRM_SRV_DEPLST, (void *)deplist}
};
```

Using these functions a module may access and modify the environment of the calling model and execute an external program with automatic redirection of default streams:

```c
mmsys->setenv(ctx,sysctx,"MYVAR","A_VALUE");
rts=mmsys->system(ctx,sysctx,"myprogram arg1 arg2");
```

16.5.1 Description of the library functions

- **getdate** Get the date of a date object. p. 528
- **getdatetime** Get the date and time of a datetime object. p. 530
- **gettime** Get the time of a time object. p. 526
- **gettxtbuf** Get a reference to the character buffer of a text object. p. 533
- **gettxtsize** Get the size of a text object. p. 532
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>setdate</td>
<td>Set the date of a date object.</td>
<td>529</td>
</tr>
<tr>
<td>setdatetime</td>
<td>Set the date and time of a datetime object.</td>
<td>531</td>
</tr>
<tr>
<td>settime</td>
<td>Set the time of a time object.</td>
<td>527</td>
</tr>
<tr>
<td>txtresize</td>
<td>Resize a text object.</td>
<td>534</td>
</tr>
</tbody>
</table>
gettime

**Purpose**
Get the time of a time object.

**Synopsis**
```c
int gettime(XPRMctx ctx, void *sysctx, void *t, int *h, int *mi, int *s, int *ms);
```

**Arguments**
- `ctx` Mosel's execution context
- `sysctx` Context of `mmsystem`
- `t` Reference to a time object
- `h` Reference to store the hours or NULL
- `mi` Reference to store the minutes or NULL
- `s` Reference to store the seconds or NULL
- `ms` Reference to store the milliseconds or NULL

**Return value**
0 if successful or -1 if `t` is NULL.

**Further information**
Provided references are set even if the function fails.

**Related topics**
- `settime`

**Module**
`mmsystem`
settime

Purpose
Set the time of a time object.

Synopsis
int settime(XPRMctx ctx, void *sysctx, void *t, int h, int mi, int s, int ms);

Arguments
ctx Mosel’s execution context
sysctx Context of mmsystem
t Reference to a time object
h hours
mi minutes
s seconds
ms milliseconds

Return value
0 if successful or -1 if t is NULL.

Related topics
gmtime

Module
mmsystem
getdate

Purpose
Get the date of a date object.

Synopsis
int getdate(XPRMctx ctx, void *sysctx, void *t, int *y, int *m, int *d);

Arguments
ctx Mosel’s execution context
sysctx Context of mmsystem
t Reference to a date object
y Reference to store the years or NULL
m Reference to store the months or NULL
d Reference to store the days or NULL

Return value
0 if successful or -1 if t is NULL.

Further information
Provided references are set even if the function fails.

Related topics
setdate

Module
mmsystem
setdate

Purpose
Set the date of a date object.

Synopsis
int setdate(XPRMctx ctx, void *sysctx, void *t, int y, int m, int d);

Arguments
ctx Mosel's execution context
sysctx Context of mmsystem
t Reference to a date object
y Years
m Months
d days

Return value
0 if successful or -1 if t is NULL.

Related topics
getcode

Module
mmsystem
getdatetime

**Purpose**
Get the date and time of a datetime object.

**Synopsis**
```c
int getdatetime(XPRMctx ctx, void *sysctx, void *t, int *y, int *m, int *d, int *h,
               int *mi, int *s, int *ms);
```

**Arguments**
- `ctx` Mosel's execution context
- `sysctx` Context of `mmsystem`
- `t` Reference to a date object
- `y` Reference to store the years or NULL
- `m` Reference to store the months or NULL
- `d` Reference to store the days or NULL
- `h` Reference to store the hours or NULL
- `mi` Reference to store the minutes or NULL
- `s` Reference to store the seconds or NULL
- `ms` Reference to store the milliseconds or NULL

**Return value**
- 0 if successful or -1 if t is NULL.

**Further information**
Provided references are set even if the function fails.

**Related topics**
- `setdatetime`

**Module**
- `mmsystem`
setdatetime

Purpose
Set the date and time of a datetime object.

Synopsis
int setdatetime(XPRMctx ctx, void *sysctx, void *t, int y, int m, int d, int h, int mi, int s, int ms);

Arguments
ctx Mosel's execution context
sysctx Context of mmsystem
t Reference to a date object
y Years
m Months
d days
h hours
mi minutes
s seconds
ms milliseconds

Return value
0 if successful or -1 if t is NULL.

Related topics
getdatetime

Module
mmsystem
**gettxtsize**

**Purpose**
Get the size of a text object.

**Synopsis**
```c
int gettxtsize(XPRMctx ctx, void *sysctx, void *t);
```

**Arguments**
- `ctx` Mosel’s execution context
- `sysctx` Context of `mmsystem`
- `t` Reference to a text object

**Return value**
The size of the character buffer (excluding the terminating 0 character).

**Related topics**
- `txtresize`, `gettxtbuf`

**Module**
`mmsystem`
**gettxtbuf**

**Purpose**
Get a reference to the character buffer of a text object.

**Synopsis**
```c
char *gettxtbuf(XPRMctx ctx, void *sysctx, void *t);
```

**Arguments**
- `ctx`: Mosel’s execution context
- `sysctx`: Context of mmsystem
- `t`: Reference to a text object

**Return value**
A reference to the character buffer.

**Further information**
1. The buffer returned is terminated by the character 0 (like a C string) and can be modified as long as the size is not changed. If the length of the buffer has to be altered, use the function `txtresize`.
2. Since the memory management of the module may move text buffers when allocating memory, the pointer returned by this function is only valid until the next memory allocation.

**Related topics**
- `txtresize`, `gettxtsize`

**Module**
- mmsystem
**txtresize**

**Purpose**
Resize and get a reference to the character buffer of a text object.

**Synopsis**
```c
char *txtresize(XPRMctx ctx, void *sysctx, void *t, int s);
```

**Arguments**
- `ctx` Mosel’s execution context
- `sysctx` Context of `mmsystem`
- `t` Reference to a text object
- `s` New size of the buffer (terminating 0 is not counted)

**Return value**
A reference to the new character buffer.

**Further information**
1. The buffer returned is terminated by the character 0 (like a C string) and can be modified as long as the size is not changed.
2. Since the memory management of the module may move text buffers when allocating memory, the pointer returned by this function is only valid until the next memory allocation.

**Related topics**
- `gettxtsize`

**Module**
- `mmsystem`
This module provides an XML parser and generator for the manipulation of XML documents from Mosel models. To use this module, the following line must be included in the header of the Mosel model file:

```mosel
uses 'mmxml'
```

`mmxml` relies on the XML parser EXPAT by James Clark (http://www.libexpat.org) for loading documents.

## 17.1 Document representation in mmxml

### 17.1.1 Data model

The XML document is stored as a list of nodes. Different node types are used to represent the document structure:

- element node
- text section
- comment
- CDATA
- processing instruction

In addition to these usual node types, the type `DATA` is used for XML constructs not supported by `mmxml` (for instance a `DOCTYPE` declaration is recorded as a `DATA` section). Although they are not directly recorded in the document tree, attributes are also stored as nodes of a dedicated type.

Each node is characterised by a `name` and a `value`. Nodes of type `text`, `comment`, `CDATA` and `DATA` have a constant name. The name of a processing instruction is the processing instruction’s target and its value the remaining part of the statement (e.g. the name of `<?proc inst>` is `proc` and its value is `inst`). The value of `comment` and `CDATA` sections is the content of the section without its delimiters but the value of a `DATA` block includes the delimiters. Element nodes have also an ordered list of child nodes. The value of an `element` node corresponds to the value of the first child `text` node (if any).

The `root` node is a special element node with no name, no parent and no successor that includes the entire document as its children.
Example of an XML document with node types:

```xml
<?xml version="1.0" encoding="iso-8859-1" standalone="no" ?>
<?xml-stylesheet type="text/css" href="examplestyle.css" ?>
<!DOCTYPE exampleList SYSTEM "examples.dtd" [
  <!ENTITY otherfile SYSTEM "anotherfile.xml">
]>

<!-- List of optimization application examples -->
<exampleList>
  <!-- Example B3 -->
  <model id="book_B_3">
    <modFile date="Mar.2002">
      b3jobshop.mos
    </modFile>
    <modData file="b3jobshop.dat" />
    <modData file="b3jobshop2.dat" />
    <modTitle>
      Job shop scheduling
    </modTitle>
    <modRating>
      3
    </modRating>
    <modFeatures>
      <![CDATA[dynamic array, range, exists, forall-do]]>
    </modFeatures>
  </model>
</exampleList>
```

17.1.2 Paths in a document

Nodes can be retrieved using a path similar to a directory path used to locate a file. An XML path consists in a list of location steps separated by the slash character ("/"): each step selects a set of nodes from the input set resulting from the preceding step (context nodes). The initial set of the path is either the root node (absolute path) or some specified node (relative path).

A step is composed of an optional axis specifier followed by a node test and possibly completed by a predicate. The axis specifies the tree relationship between the nodes selected by the step and the context node. The node test is either an element name (to select elements of the given name) or a node type (to select nodes by their type). The predicate is a Boolean expression the truth value of which decides whether a selected node is kept in the result set of the step.

Examples:

- `/examples/chapter` all element nodes ‘chapter’ under elements ‘examples’
- `/examples/chapter/model/modRating[number()>=4]/..` all ‘model’ nodes under ‘examples/chapter’ for which element ‘modRating’ has a value greater than or equal to 4
- `//*[@attribute1 and @attribute2='value2']` all element nodes of the document having ‘attribute1’ defined and ‘attribute2’ with value ‘value2’
- `/descendant::text()` all text sections of the document
- `./@mytag` all element nodes named ‘mytag’ starting from the current node
17.1.2.1 Axis specifier

An axis specifier consists in an axis name followed the the symbol ::. The supported axes are:

- **child**: children of the context node (this is the default if no axis is given)
- **parent**: parent of the context node
- **self**: the context node itself
- **attribute**: the attributes of the context node
- **following**: following node of the context node
- **descendant-or-self**: the context node as well as all its descendants
- **descendant**: all descendants of the context node

17.1.2.2 Node test

By default only element nodes are considered, the node test is used to select the nodes by their name. The special name "*" will keep all element nodes. Alternatively, the test can be related to the type of the nodes; in this case all nodes are considered and the test is one of the following expressions:

- **text()**: to select text nodes
- **comment()**: to select comment nodes
- **cdata()**: to select CDATA nodes
- **data()**: to select DATA nodes
- **processing-instruction()**: to select processing instruction nodes
- **node()**: to keep all nodes (independently of the type and name)

17.1.2.3 Abbreviated notation

Common combinations of axis-node tests have an abbreviated notation. The supported abbreviations are:

- . is equivalent to self::node()
- .. is equivalent to parent::node()
- // (used in place of /) is the same as descendant-or-self::node()

17.1.2.4 Predicate

A **predicate** is a Boolean expression enclosed in square brackets. The expression evaluator supports Boolean, text and numerical values (encoded as floating point numbers). Type conversions are implicit and implied by the operators: for instance the additive operator "+" operates on numbers, as a consequence its operands are systematically converted to numbers. Constant strings must be quoted using either single or double quotes.
The notation \texttt{@attname} designates the attribute which name is "attname": if used where a Boolean value is expected, it is true if the attribute is defined for the current node. Otherwise, this is the value of the attribute.

Supported arithmetic operators include \texttt{+}, \texttt{-}, \texttt{*}, \texttt{div} (division on floating point numbers, not integral division as in Mosel!), \texttt{mod} (modulo on floating point numbers). Boolean expressions can be composed using \texttt{and} and \texttt{or}; the usual comparators \texttt{<, <=, =, >, <>(or !=)} can be applied to numbers. Note that equality testing (\texttt{=} and \texttt{<>}) is defined for all types. The following predefined functions can also be used in expressions:

\begin{itemize}
  \item \texttt{name()} \quad \text{name of the node}
  \item \texttt{string()} \quad \text{value of the node}
  \item \texttt{number()} \quad \text{value of the node as a number}
  \item \texttt{boolean()} \quad \text{value of the node as a Boolean}
  \item \texttt{position()} \quad \text{position of the current node in the selected set (first node has position 1)}
  \item \texttt{not(booleanexp)} \quad \text{true if \texttt{booleanexp} is false, false otherwise}
  \item \texttt{true()} \quad \text{value \texttt{true}}
  \item \texttt{false()} \quad \text{value \texttt{false}}
  \item \texttt{string-length()}/\texttt{getsize()} \quad \text{length of the node value}
  \item \texttt{string-length(strexp)/\texttt{getsize(strexp)}} \quad \text{length of the text passed as parameter}
  \item \texttt{starts-with(strexp1,strexp2)} \quad \text{true if text \texttt{strexp1} starts with text \texttt{strexp2}}
  \item \texttt{contains(strexp1,strexp2)} \quad \text{true if text \texttt{strexp1} contains text \texttt{strexp2}}
  \item \texttt{round(numexp)} \quad \text{rounded value of \texttt{numexp}}
  \item \texttt{floor(numexp)} \quad \text{floor value of \texttt{numexp}}
  \item \texttt{ceiling(numexp)/\texttt{ceil(numexpr)}} \quad \text{ceil value of \texttt{numexp}}
  \item \texttt{abs(numexp)} \quad \text{absolute value of \texttt{numexp}}
\end{itemize}

If the predicate \texttt{[expr]} is not a Boolean value, the whole expression is interpreted as \texttt{[position()=expr]}.

### 17.1.3 JSON document as an XML tree

In addition to XML documents \texttt{mmxml} can also load and generate JSON documents represented as XML trees such that the information contained in the document can be handled using the routines published by this module. The procedure \texttt{jsonload} parses a JSON file that it maps to the internal XML representation using the following conventions: every JSON syntactic entity is converted to an XML element the value of which corresponds to the associated JSON value. The type of the value is identified via the attribute \texttt{"jst"} that can be \texttt{"str"} (string), \texttt{"num"} (numeric), \texttt{"boo"} (Boolean), \texttt{"nul"} (null object), \texttt{"obj"} (object) or \texttt{"arr"} (array). Names of object components can be mapped to either the name of the XML element or to an attribute (the behaviour of the parser is selected via an option of \texttt{jsonload}).

For instance, consider the following JSON document:
It will be represented by the following XML document when object member names are turned into XML element names:

```xml
<?xml version="1.0" encoding="iso-8859-1"?>
<jsv jst="arr">
  <jsv jst="obj">
    <name jst="str">bob</name>
    <age jst="num">25</age>
    <student jst="boo">true</student>
    <phone jst="arr">
      <jsv jst="obj">
        <type jst="str">home</type>
        <number jst="str">1234567900</number>
      </jsv>
      <jsv jst="obj">
        <type jst="str">work</type>
        <number jst="str">6789012345</number>
      </jsv>
    </phone>
  </jsv>
</jsv>
```

Note that with this representation the generated XML document is not necessarily valid XML (this mapping can for instance produce XML elements that have a number as name) and trying to export a JSON document using the `save` procedure may produce a file that cannot be processed by an XML parser. Using the second mode of operation avoids this problem: all elements are named "jsv" and object names are represented by attributes. The resulting XML document is larger than the one produced with the first mode:

```xml
<?xml version="1.0" encoding="iso-8859-1"?>
<jsv jst="arr">
  <jsv jst="obj" name="name">bob</jsv>
  <jsv jst="num" name="age">25</jsv>
  <jsv jst="boo" name="student">true</jsv>
  <jsv jst="arr" name="phone">
    <jsv jst="obj" name="type">home</jsv>
    <jsv jst="str" name="number">1234567900</jsv>
  </jsv>
  <jsv jst="obj" name="type">work</jsv>
  <jsv jst="str" name="number">6789012345</jsv>
</jsv>
```

Assuming an XML tree has been built using the above conventions, the procedure `jsonsave` can be used to generate a JSON document. The XML document may combine the two representations described above and in most cases the `jst` attribute can be omitted. Therefore, `jsonsave` will produce the same JSON document as the example shown at the start of this section from the following XML file:
17.2 New functionality for the Mosel language

17.2.1 The type xmldoc

The type xmldoc represents an XML document stored in the form of a tree. Each node of the tree is identified by a node number (an integer) that is attached to the document (i.e. a node number cannot be shared by different documents and in two different documents the same number represents two different nodes). The root node of the document has number 0: the content of the document is stored as the children of this root node. In addition to structural properties (e.g. name, value, successor, parent) nodes have 2 formatting properties: vertical (setvspace) and horizontal (sethspace) spacing. These indications are used when the document is saved in text form for controlling how the resulting text has to be organised (see save). The general formatting policy is defined by a set of document settings: indentation mode (setindentmode), indentation skip (setindentskip) and line length (setlinelen). Also used when exporting the documents are the XML version (setxmlversion), standalone status (setstandalone) and encoding (setencoding).

17.3 Procedures and functions

- addnode: Add a node to a document tree. p. 542
- copynode: Copy a node. p. 544
- delattr: Delete an attribute of an element node. p. 545
- delnode: Delete a node in a document tree. p. 546
- getattr: Get the value of an attribute. p. 547
- getencoding: Get the character encoding of the document. p. 549
- getfirstattr: Get the first attribute of an element node. p. 552
- getfirstchild: Get the first child of an element node. p. 554
- gethspace: Get horizontal spacing of a node. p. 562
- getindentmode: Get indent mode of the document. p. 564
- getindentskip: Get the size of an indentation step. p. 565
- getlastchild: Get the last child of an element node. p. 555
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>getlinelen</td>
<td>Get the length of a line.</td>
<td>566</td>
</tr>
<tr>
<td>getmaxnodes</td>
<td>Get the number of nodes currently allocated for a document.</td>
<td>567</td>
</tr>
<tr>
<td>getname</td>
<td>Get the name of a node.</td>
<td>550</td>
</tr>
<tr>
<td>getnext</td>
<td>Get the successor of a node.</td>
<td>553</td>
</tr>
<tr>
<td>getnode</td>
<td>Get the first node returned by a path specification.</td>
<td>556</td>
</tr>
<tr>
<td>getnodes</td>
<td>Get the list of nodes returned by a path specification.</td>
<td>557</td>
</tr>
<tr>
<td>getparent</td>
<td>Get the parent of a node.</td>
<td>558</td>
</tr>
<tr>
<td>getsize</td>
<td>Get the size of a document.</td>
<td>568</td>
</tr>
<tr>
<td>getstandalone</td>
<td>Get the standalone flag of the document.</td>
<td>560</td>
</tr>
<tr>
<td>gettype</td>
<td>Get the type of a node.</td>
<td>559</td>
</tr>
<tr>
<td>getvalue</td>
<td>Get the value of a node.</td>
<td>551</td>
</tr>
<tr>
<td>getvspace</td>
<td>Get vertical spacing of a node.</td>
<td>563</td>
</tr>
<tr>
<td>getxmlversion</td>
<td>Get the XML version of the document.</td>
<td>561</td>
</tr>
<tr>
<td>jsonload</td>
<td>Load a JSON document.</td>
<td>569</td>
</tr>
<tr>
<td>jsonparse</td>
<td>Parse a JSON document.</td>
<td>570</td>
</tr>
<tr>
<td>jsonsave</td>
<td>Save a JSON document.</td>
<td>572</td>
</tr>
<tr>
<td>load</td>
<td>Load an XML document.</td>
<td>573</td>
</tr>
<tr>
<td>save</td>
<td>Save an XML document.</td>
<td>574</td>
</tr>
<tr>
<td>setattr</td>
<td>Set the value of an attribute.</td>
<td>575</td>
</tr>
<tr>
<td>setencoding</td>
<td>Set the character encoding of the document.</td>
<td>576</td>
</tr>
<tr>
<td>sethspace</td>
<td>Set horizontal spacing of a node.</td>
<td>580</td>
</tr>
<tr>
<td>setindentmode</td>
<td>Set indent mode for the document.</td>
<td>583</td>
</tr>
<tr>
<td>setindentskip</td>
<td>Set the size of an indentation step.</td>
<td>584</td>
</tr>
<tr>
<td>setlinelen</td>
<td>Set the size of a line.</td>
<td>585</td>
</tr>
<tr>
<td>setmaxnodes</td>
<td>Set the number of allocated nodes for a document.</td>
<td>577</td>
</tr>
<tr>
<td>setname</td>
<td>Set the name of a node.</td>
<td>578</td>
</tr>
<tr>
<td>setstandalone</td>
<td>Set the standalone flag of the document.</td>
<td>586</td>
</tr>
<tr>
<td>setvalue</td>
<td>Set the value of a node.</td>
<td>579</td>
</tr>
<tr>
<td>setvspace</td>
<td>Set vertical spacing of a node.</td>
<td>582</td>
</tr>
<tr>
<td>setxmlversion</td>
<td>Set the XML version of the document.</td>
<td>587</td>
</tr>
<tr>
<td>testattr</td>
<td>Test existence of an attribute for a given element node.</td>
<td>548</td>
</tr>
<tr>
<td>xmlattr</td>
<td>Get an attribute during parsing of an element.</td>
<td>588</td>
</tr>
<tr>
<td>xmldecode</td>
<td>Decode a text string for XML.</td>
<td>590</td>
</tr>
<tr>
<td>xmlencode</td>
<td>Encode a text string for XML.</td>
<td>589</td>
</tr>
<tr>
<td>xmlparse</td>
<td>Parse an XML document.</td>
<td>591</td>
</tr>
</tbody>
</table>
addnode

Purpose
Add a node to a document tree.

Synopsis
function addnode(doc: xmldoc, n: integer, where: integer, type: integer, name: string,
value: text): integer
function addnode(doc: xmldoc, n: integer, where: integer, type: integer,
nameval: string|text): integer
function addnode(doc: xmldoc, n: integer, name: string, value: text): integer
function addnode(doc: xmldoc, n: integer, type: integer, nameval: string|text): integer
function addnode(doc: xmldoc, n: integer, type: integer): integer
function addnode(doc: xmldoc, n: integer, name: string, value: text|string|boolean|integer|real): integer

Arguments
doc Document to use
n Node number where to attach the new node
where How to attach the new node to the node n:
  XML_FIRST as the first element of the list where node n is located
  XML_LAST as the last element of the list where node n is located
  XML_NEXT after node n
  XML_FIRSTCHILD as the first child of node n (node n must be an element)
  XML_LASTCHILD as the last child of node n (node n must be an element)
When the function is used without this parameter, XML_LASTCHILD is assumed.
type Type of node to add:
  XML_ELT an element
  XML_TXT a text block
  XML_CDATA a CDATA section
  XML_COM a comment
  XML_DATA non interpreted data
  XML_PINST processing instruction
When the function is used without this parameter, XML_ELT is assumed.
name Name associated to the new node. Only element and processing instruction nodes have a name
value Value associated to the new node. An element node does not have any value: if this parameter is provided for a node of this type, an additional text node with the specified value is added as the first child of the new node
nameval If the type is XML_ELT or XML_PINST this parameter is used as the name of this node. Otherwise it is the value of the new node

Return value
Number of the newly created node within the document.

Example
The following code extract appends a new node ‘employee’ as last child to the node APAC. It shows how to use different versions of addnode for the creation of descendants of the new node.

declarations
DB: xmldoc
APAC, NewPers, n, k: integer
end-declarations

APAC:= getnode(DB, "personnelList/region[@id='APAC']")
! Append a new node to 'APAC' and set its attribute 'id':
NewPers:= addnode(DB, APAC, XML_LASTCHILD, XML_ELT, "employee")
setattr(DB, NewPers, "id", "T432")
! Create a comment:
n:= addnode(DB, NewPers, XML_COM, "This is a new employee")
! Add 2 nodes containing the specified text (nodes):
n:= addnode(DB, NewPers, XML_ELT, "startDate", text(2012))
n:= addnode(DB, NewPers, XML_ELT, "name", "Tim")
! Add an empty node, then set its contents:
n:= addnode(DB, NewPers, XML_ELT, "address")
setvalue(DB, n, "Sydney")
! Add an empty node, then create its contents as a text node:
n:= addnode(DB, NewPers, XML_ELT, "language")
k:= addnode(DB, n, XML_TXT, "English")

XML resulting from this code:

<employee id="T432">
  <!--This is a new employee-->
  <startDate>2012</startDate>
  <name>Tim</name>
  <address>Sydney</address>
  <language>English</language>
</employee>

Further information

1. An element or processing instruction node must be named: trying to create a node of these types with an empty name will cause an error.

2. It is not possible to add attributes with this function. Use setattr for this purpose.

Related topics
copynode, delnode

Module
mmxml
copynode

**Purpose**
Copy a node.

**Synopsis**
```plaintext
function copynode(src:xmldoc, s:integer, dst:xmldoc, d:integer, where:integer):integer
```

**Arguments**
- **src** Document of node to be copied
- **s** Number of the node to copy
- **dst** Destination document
- **d** Node number where to attach the new node in the destination document
- **where** How to attach the copy of the source node to the node `d`:
  - `XML_FIRST` as the first element of the list where node `d` is located
  - `XML_LAST` as the last element of the list where node `d` is located
  - `XML_NEXT` after node `d`
  - `XML_FIRSTCHILD` as the first child of node `d` (node `d` must be an element)
  - `XML_LASTCHILD` as the last child of node `d` (node `d` must be an element)

**Example**
The following code extract shows how to move (copy node, then delete original) and edit a node with all its descendants

```plaintext
declarations
DB: xmldoc
   APAC, NewPers, Pers: integer
end-declarations

! Retrieve destination region node
APAC:= getnode(DB, "personnelList/region[@id='APAC']")
! Retrieve employee record (node) for 'Lisa'
Pers:=
   getnode(DB, "personnelList/region/employee/name[string()='Lisa']/..")
! Employee Lisa moves to Delhi: copy node & delete in original location
NewPers:= copynode(DB, Pers, DB, APAC, XML_LASTCHILD)
   delnode(DB, Pers)
! Update the 'address' information
   setvalue(DB, getnode(DB, NewPers, "address"), "Delhi")
```

**Further information**
This routine copies the node as well as all of its descendants if it is an **element node**. Source and destination documents may be the same.

**Related topics**
- addnode, delnode

**Module**
- mmxml
**delattr**

**Purpose**
Delete an attribute of an element node.

**Synopsis**
probecedure delattr(doc:xmldoc, n:integer, name:string)

**Arguments**
- **doc** Document to use
- **n** Element node to modify
- **name** Name of the attribute to remove

**Example**
See testattr.

**Further information**
This routine has no effect if the element does not have any attribute of the specified name.

**Related topics**
setattr

**Module**
mmxml
**delnode**

**Purpose**
Delete a node in a document tree.

**Synopsis**
procedure delnode(doc:xmldoc, n:integer)

**Arguments**
doc Document to use
n Number of the node to delete

**Example**
See copynode.

**Further information**
This routine deletes the node as well as all of its descendants if it is an *element node*.

**Related topics**
addnode, copynode

**Module**
mmxml
getattr

Purpose
Get the value of an attribute.

Synopsis
function getattr(doc:xmldoc, n:integer, name:string):text
function getboolattr(doc:xmldoc, n:integer, name:string):boolean
function getintattr(doc:xmldoc, n:integer, name:string):integer
function getrealattr(doc:xmldoc, n:integer, name:string):real
function getstrattr(doc:xmldoc, n:integer, name:string):string

Arguments
doc    Document to use
n      Node number (must be an element)
name   Name of the attribute

Return value
The value of the attribute or an empty string, 0 or false depending on the expected type.

Example
The following code extract prints the contents of ‘name’ (leftbound in a 10 character space) and
the attributes ‘id’ of all ‘employee’ nodes, and the ‘id’ of their parent node.

declarations
DB: xmldoc
    AllEmployees: list of integer
end-declarations

getnodes(DB, "personnelList/region/employee", AllEmployees)
forall(p in AllEmployees)
    writeln(textfmt(getvalue(DB, getnode(DB, p, "name")), -10),
            "(ID: ", getattr(DB,p,"id"), ") ",
            "region: ", getattr(DB, getparent(DB, p), "id"))

Output produced by this code will look as follows:
Lisa   (ID: L234) region: EMEA
James  (ID: J876) region: APAC
Sarah  (ID: S678) region: AM

Further information
1. Values of attributes are stored as text objects: the first version of the routine returns a reference
to the object containing the attribute value. Modifying this text will also alter the attribute
value. Using one of the alternative versions of this routine allows to avoid having to perform a
type conversion.

2. A default value (empty string, 0 or false) is returned if the requested attribute is not defined. Use
function testattr to check whether a given node has a particular attribute.

Related topics
setattr, testattr, getfirstattr

Module
mmxml
**testattr**

**Purpose**
Test existence of an attribute for a given element node.

**Synopsis**
function testattr(doc: xmldoc, n: integer, name: string): boolean

**Arguments**
doc Document to use
n Node number (must be an element)
name Name of the attribute

**Return value**
true if the requested attribute is defined for the node.

**Example**
This example tests whether the attribute ‘parttime’ is defined for an employee, and if this is the case the attribute gets deleted after printing the name of the employee.

declarations
DB: xmldoc
    AllEmployees: list of integer
end-declarations

getnodes(DB, "personnelList/region/employee", AllEmployees)
forall(p in AllEmployees | testattr(DB, p, "parttime")) do
    writeln(getvalue(DB, getnode(DB, p, "name")))
    delattr(DB, p, "parttime")
end-do

**Related topics**
setattr, getattr

**Module**
mmxml
getencoding

**Purpose**
Get the character encoding of the document.

**Synopsis**
function getencoding(doc: xmldoc): string

**Argument**
doc  Document to use

**Return value**
Character encoding of the document

**Related topics**
getstandalone, getxmlversion

**Module**
mxml
getname

Purpose
Get the name of a node.

Synopsis
function getname(doc: xmldoc, n: integer): string

Arguments
doc Document to use
n Node number

Return value
The name of the node depending on the node type:
XML_ELT name of the element section
XML_TXT "#text"
XML_CDATA "#cdata-section"
XML_COM "#comment"
XML_DATA "#data"
XML_PINST processing instruction target
XML_ATTR name of the attribute

Example
The following example collects the names of all element nodes occurring in a document.

declarations
DB: xmldoc
NodeList: list of integer
NodeNames: set of string
end-declarations

getnodes(DB, "/descendant-or-self::node()", NodeList)
NodeNames:= union(r in NodeList | gettype(DB,r)=XML_ELT) {getname(DB,r)}
writeln("Names of element nodes: ", NodeNames)

Further information
Only element, attribute and processing instruction nodes have a name, for all other node types
the above listed constant name is returned.

Related topics
gettype, getvalue, setname

Module
mmxml
getvalue

Purpose
Get the value of a node.

Synopsis
function getvalue(doc: xmldoc, n: integer): text
function getboolvalue(doc: xmldoc, n: integer): boolean
function getintvalue(doc: xmldoc, n: integer): integer
function getrealvalue(doc: xmldoc, n: integer): real
function getstrvalue(doc: xmldoc, n: integer): string

Arguments
doc Document to use
n Node number

Return value
The value of the node.

Example
This code prints out the name of the employee with attribute id="T345".

declarations
DB: xmldoc
doc = getnode(DB, "personellList/region/employee[@id='T345']/name")
end-declarations

writeln("Person with id='T345': ", getvalue(DB, getnode(DB, "personellList/region/employee[@id='T345']/name") ))

Further information
1. Values of nodes are stored as text objects: the first version of the routine returns a reference to the object containing the value. Modifying this text will also alter the node value. Using one of the alternative versions of this routine allows to avoid having to perform a type conversion.

2. Element nodes have no value: the returned value corresponds to the value of the first child of type text of this element (or an empty string if no such child can be found).

Related topics
getype, getname, setvalue

Module
mmxml
getfirstattr

**Purpose**
Get the first attribute of an element node.

**Synopsis**
function getfirstattr(doc:xmldoc, n:integer):integer

**Arguments**
doc  Document to use
n Node number (must be an element)

**Return value**
The node number of the first attribute of the element node provided or -1 if there is no attribute.

**Example**
The following example displays all attributes of node e:

```plaintext
declarations
  DB: xmldoc
  a,e: integer
end-declarations

  a:=getfirstattr(DB,e)
  while(a>0) do
    writeln(getname(DB,a), "=", getvalue(DB,a))
    a:=getnext(DB,a)
  end-do
```

**Further information**
Attributes are represented by nodes of type XML_ATTR: all node-related routines can be applied to attribute nodes.

**Related topics**
getchild, getfirstchild, getlastchild, getparent

**Module**
mmxml
getnext

**Purpose**
Get the successor of a node.

**Synopsis**
function getnext(doc: xmldoc, n: integer): integer

**Arguments**
doc Document to use
n Node number

**Return value**
The node number of the following node or -1 if the current node is the last of the list.

**Example**
This example enumerates all child nodes within a specific region and displays the ‘id’ for all ‘employee’ nodes on a single line, adding a line break after the last name:

```plaintext
declarations
DB: xmldoc
  APAC, Pers: integer
end-declarations

APAC:= getnode(DB, "personnelList/region[@id='APAC']")
Pers:= getfirstchild(DB, APAC)
LastPers:= getlastchild(DB, APAC)
while(Pers>-1) do
  if getname(DB, Pers)="employee" then
    write(" ", getattr(DB,Pers,"id"))
  end-if
  if Pers=LastPers then writeln; end-if
  Pers:= getnext(DB, Pers)
end-do
```

**Further information**
Node numbers returned by Mosel are not directly related to the order of nodes within the XML document (i.e. a larger node number does not imply that a node succeeds a node with a smaller number).

**Related topics**
getfirstattr, getfirstchild, getlastchild, getparent

**Module**
mmxml
getfirstchild

**Purpose**
Get the first child of an element node.

**Synopsis**
```
function getfirstchild(doc: xmldoc, n: integer): integer
```

**Arguments**
- `doc` Document to use
- `n` Node number (must be an element)

**Return value**
The node number of the first child or -1 if there is no child.

**Example**
See `getnext`.

**Related topics**
- `getfirstattr`, `getnext`, `getlastchild`, `getparent`

**Module**
`mmxml`
getlastchild

Purpose
Get the last child of an element node.

Synopsis
function getlastchild(doc: xmldoc, n: integer): integer

Arguments
- doc Document to use
- n Node number (must be an element)

Return value
The node number of the last child or -1 if there is no child.

Example
See getnext.

Related topics
getfirstattr, getfirstchild, getnext, getparent

Module
mmxml
getnode

Purpose
Get the first node returned by a path specification.

Synopsis
function getnode(doc: xmldoc, n: integer, p: string|text): integer
function getnode(doc: xmldoc, n: integer): integer
function getnode(doc: xmldoc, p: string): integer

Arguments
doc Document to use
n Base node number (0 when not provided)
p Path to the node (*) when not provided

Return value
The node number of the first node selected by the path p; -1 if no node can be found.

Example
The following example shows different forms of the getnode function.

declarations
  DB: xmldoc
  Root, EMEA: integer
end-declarations

! Get the first element that is not a comment or a processing instruction
Root:= getnode(DB,"*")  ! Same as: getnode(DB,0,"*")

! Get the 'region' node with id=EMEA
EMEA:= getnode(DB, "personnelList/region[@id='EMEA']")

! Check for employee record (node) for 'Sam' under 'EMEA'
if getnode(DB, EMEA, "employee/name[string()='Sam']/..")<0 then
  writeln("No employee called 'Sam' in EMEA")
end-if

Further information
1. Refer to section 17.1.2 for a detailed description of the syntax and semantic of XML paths.
2. This function is the same as getfirstchild when used without path specification.

Related topics
getnodes, getfirstchild

Module
mmxml
getnodes

Purpose
Get the list of nodes returned by a path specification.

Synopsis
procedure getnodes(doc: xmldoc, n: integer, p: string|text, l: list of integer)
procedure getnodes(doc: xmldoc, p: string, l: list of integer)
procedure getnodes(doc: xmldoc, n: integer, l: list of integer)

Arguments
doc Document to use
n Base node number (0 when not provided)
p Path to the node ("*" when not provided)
l List where result is returned

Example
Here are a number of examples how to retrieve nodes with specific properties:

declarations
   DB: xmldoc
   Employees, AllEmployees: list of integer
end-declarations

! Get all employees in the Americas
getnodes(DB, "personnelList/region[@id='AM']/employee", Employees)

! All employees who started before 2005
getnodes(DB, "personnelList/region/employee/startDate[number() < 2005]/..", Employees)

! All employees whose names start with "J"
getnodes(DB, "personnelList/region/employee", AllEmployees)
forall(n in AllEmployees) do
   getnodes(DB, n, "./name[starts-with(string(), 'J')]/..", Employees)
forall(p in Employees) save(DB, p, ")")
end-do

! Employees speaking at least 3 languages (= have a third "language" entry)
getnodes(DB, "personnelList/region/employee/language[position() = 3]/..", Employees)

Further information
1. Refer to section 17.1.2 for a detailed description of the syntax and semantic of XML paths.
2. This function resets the list it receives as parameter: the provided list is returned empty if no node can be found.

Related topics
getnode

Module
mmxml
**getparent**

**Purpose**
Get the parent of a node.

**Synopsis**
function getparent(doc: xmldoc, n: integer): integer

**Arguments**
doc Document to use
n Node number

**Return value**
The node number of the parent node or -1 if n=0 (root node has no parent).

**Example**
See getattr.

**Related topics**
getfirstattr, getfirstchild, getlastchild, getnext

**Module**
mmxml
gettype

Purpose
Get the type of a node.

Synopsis
function gettype(doc: xmldoc, n: integer): integer

Arguments
doc Document to use
n Node number

Return value
The type of the node:
XML_ELT an element
XML_TXT a text
XML_CDATA a CDATA section
XML_COM a comment
XML_DATA a data section
XML_PINST a processing instruction
XML_ATTR an attribute
-1 if the node number is not valid

Example
See getname.

Further information
This function returns -1 if the provided node number is not valid: this feature can be used to verify the validity of a node number before using it with other functions.

Related topics
getname, getvalue

Module
mmxml
getstandalone

**Purpose**
Get the standalone flag of the document.

**Synopsis**
```
function getstandalone(doc: xmldoc): integer
```

**Argument**
- `doc` Document to use

**Return value**
- `-1` flag not specified
- `0` standalone=no
- `1` standalone=yes

**Further information**
The value of this flag is not used by **mmxml**. This is just an information to be saved in the header of the XML document. The default value for this flag is -1.

**Related topics**
- `getencoding`, `getxmlversion`

**Module**
**mmxml**
getxmlversion

Purpose
Get the XML version of the document.

Synopsis
function setxmlversion(doc:xmlDoc):string

Argument
doc Document to use

Return value
XML version as a text string

Further information
The XML version number is not used by mmxml. This is just an information to be saved in the header of the XML document. The default value for this option is 1.0.

Related topics
getencoding, getstandalone

Module
mmxml
gethspace

Purpose
Get horizontal spacing of a node.

Synopsis
function gethspace(doc: xmldoc, n: integer): integer

Arguments
    doc    Document to use
    n      Node number

Return value
Number of spaces inserted before the node output

Further information
This spacing indicates the number of spaces to insert before displaying the node from the start of a new line when outputing the document. The horizontal spacing setting is only used when the indentation is in manual mode (see setindentmode).

Related topics
getvspace, getindentmode

Module
mmxml
getvspace

Purpose
Get vertical spacing of a node.

Synopsis
function getvspace(doc:xml_doc, n:integer):integer

Arguments
doc Document to use
n Node number

Return value
Number of carriage returns inserted before the node output

Further information
This spacing indicates the number of empty lines to insert before displaying the node when outputing the document. The vertical spacing setting is only used when the indentation is in manual mode (see setindentmode).

Related topics
gethspace, getindentmode

Module
mmxml
**getindentmode**

**Purpose**
Get indent mode of the document.

**Synopsis**
function getindentmode(doc: xmldoc): integer

**Argument**
doc Document to use

**Return value**
Indent mode:
- **XML_AUTO** automatic indentation
- **XML_NONE** no formatting
- **XML_MANUAL** use vertical/horizontal spacing settings of each node

**Related topics**
setindentmode, getindentskip, getlinelen

**Module**
mmxml
getindentskip

**Purpose**
Get the size of an indentation step.

**Synopsis**
function getindentskip(doc: xmldoc): integer

**Argument**
doc  Document to use

**Return value**
Number of spaces to add for each indentation

**Related topics**
setindentskip, getindentmode, getlinelen

**Module**
mmxml
getlinelen

Purpose
Get the length of a line.

Synopsis
function getlinelen(doc: xmldoc): integer

Argument
doc Document to use

Return value
Length of a line in characters for outputting the XML document

Related topics
setlinelen, getindentmode, getindentskip

Module
mmxml
getmaxnodes

**Purpose**
Get the number of nodes currently allocated for a document.

**Synopsis**

```plaintext
function getmaxnodes(doc: xmldoc): integer
```

**Argument**

- `doc`  Document to use

**Return value**

Number of nodes currently allocated

**Further information**

This function returns the amount of memory (in number of nodes) currently allocated for a given document. This amount may be larger than the amount actually in use.

**Related topics**

- setmaxnodes, getsize

**Module**

mmxml
getsiz

**Purpose**
Get the size of a document.

**Synopsis**
function getsize(doc: xmldoc): integer

**Argument**
doc    Document to use

**Return value**
The number of nodes used by the document.

**Related topics**
getmaxnodes

**Module**
mmxml
jsonload

Purpose
Load a JSON document.

Synopsis
procedure jsonload(doc: xmldoc, fname: text)
procedure jsonload(doc: xmldoc, fname: text, mode: integer)

Arguments
- doc       Document to use
- fname     File name of the document to load
- mode      How to handle JSON object names:
            0       Object names are converted to XML element names (default)
            1       Object names are saved as the attribute "name"

Further information
1. This routine replaces the content of the provided document object with the JSON file given as second argument
2. The parser converts the original JSON document into an XML representation (See Section 17.1.3). Using the version of the procedure without the mode argument is the same as using 0 for this parameter.

Related topics
jsonsave, jsonparse, load

Module
mmxml
jsonpars

**Purpose**

Parse a JSON document.

**Synopsis**

```plaintext
function jsonparse(afct:array(range) of string, ctx:ctxtype): integer
```

**Arguments**

- **afct**
  
  Event function table. Each entry of this array is the name of the function to call when the corresponding event occurs. The expected events are (all of these entries are optional):
  
  - `JSON_FCT_OPEN_OBJ`: Opening of an object
  - `JSON_FCT_CLOSE_OBJ`: Closing of an object
  - `JSON_FCT_OPEN_ARR`: Opening of an array
  - `JSON_FCT_CLOSE_ARR`: Closing of an array
  - `JSON_FCT_TEXT`: A textual value
  - `JSON_FCT_NUM`: A numerical value
  - `JSON_FCT_BOOL`: A Boolean value
  - `JSON_FCT_NULL`: The null value
  
- **ctx**
  
  Value passed as first argument of all event functions

**Return value**

0 if successful, 1 in case of parsing error or a non-zero value returned by an event function

**Example**

This example displays values of object members "name" and "age" of a JSON document:

```plaintext
declarations
  afct:array(range) of string
  s_ctx=record
    cnt:integer
  end-record
  c:s_ctx
end-declarations

public function setvalue_all(ctx:s_ctx, name:text, type:integer, val:text):integer
  if name="name" or name="age" then
    writeln(name,"=",val)
    ctx.cnt+=1
  end-if
end-function

afct(JSON_FCT_TEXT):="setvalue_all" ! A value as a text
fopen("mydoc.json",F_INPUT)
rts:=jsonpars(afct,c)
fclose(F_INPUT)
writeln("line count ",c.cnt)
```
Further information

1. This function is an alternative approach to \texttt{jsonload} for processing JSON documents: instead of loading into memory the entire document this function calls a dedicated routine whenever it identifies a JSON entity. For instance a specific function is called when an object is open and another one when it is closed. It is up to the Mosel program to decide how to handle the document via these \textit{event handling functions}.

2. To each event type corresponds a specific function signature. These functions return an integer that decides whether parsing should continue: a non-zero value will cause the parsing to cancel (this value is used as the return value of \texttt{jsonparse}). The expected function signatures are:

\begin{verbatim}
JSON_FCT_OPEN_OBJ function open_object(ctx:ctxtype, name:text):integer
JSON_FCT_CLOSE_OBJ function close_object(ctx:ctxtype):integer
JSON_FCT_OPEN_ARR function open_array(ctx:ctxtype, name:text):integer
JSON_FCT_CLOSE_ARR function close_array(ctx:ctxtype):integer
JSON_FCT_TEXT function text_val(ctx:ctxtype, name:text, type:integer, val:text):integer
JSON_FCT_NUM function num_val(ctx:ctxtype, name:text, name:text, val:real):integer
JSON_FCT_BOOL function bool_val(ctx:ctxtype, name:text, name:text, val:boolean):integer
JSON_FCT_NULL function null_val(ctx:ctxtype, name:text):integer
\end{verbatim}

In addition to the pre-defined arguments these functions take a \textit{context} as their first parameter. This variable (that can be of any type) is provided to the \texttt{jsonpars}\texttt{e} routine and can be used by the event functions for storing progress information. The \textit{name} argument is not empty only when the value corresponds to an object member: in this case this parameter is the label of this member. The \textit{type} argument passed to the \texttt{text_val} function indicates the type of the data (0 for \texttt{null}, 1 for \texttt{text}, 2 for numerical and 3 for \texttt{Boolean}): this function is used with the textual representation of the value when the required type-specific function is not available. For instance this function will be called with \texttt{type=3} if a Boolean value has been read and the entry \texttt{JSON_FCT_BOOL} is not defined in the function table.

3. An error message indicating the location of the error is displayed when the parsing fails or if an event function returns a negative value (a positive value also interrupts parsing but no message is displayed).

Related topics

- \texttt{jsonload}

Module

- \texttt{mmxml}
jsonsave

**Purpose**
Save a JSON document.

**Synopsis**
`procedure jsonsave(doc: xmldoc, fname: text)`

**Arguments**
- `doc` Document to save
- `fname` Destination file name

**Further information**
1. This routine generates a JSON file from the provided `xmldoc` object. It is assumed that the document is built according to the JSON conventions (See Section 17.1.3). The result is undefined if the conventions are not respected.

2. This procedure does not require that the elements of the tree are typed using the "jst" attribute: the type is deduced from the value of the node when this attribute is missing. Moreover, both object member naming conventions can be used: when outputing an object, the member name can be taken either from the element name or from the attribute "name". If both are available, the attribute takes precedence.

**Related topics**
- `jsonload`, `save`

**Module**
- `mmxml`
load

**Purpose**
Load an XML document.

**Synopsis**
procedure load(doc: xmldoc, fname: text)

**Arguments**
doc Document to use
fname File name of the document to load

**Example**
This code reads in a document and displays its contents on screen applying automatic formatting instead of its original formatting.

```plaintext
declarations
    DB: xmldoc
end-declarations

! Reading data from an XML file
load(DB, "refexample.xml")

! Set indentation mode for XML output (default after load: MANUAL)
setindentmode(DB, XML_AUTO)

! Display document contents on screen
save(DB, "")
```

**Further information**
This routine replaces the content of the provided document object with the XML file given as second argument: all properties of the document are reset to their default value and the indentation mode is set to XML_MANUAL (see setindentmode). Vertical and horizontal spacing of each loaded node are set in order to preserve as much as possible the original formatting of the document.

**Related topics**
save, xmlparse, jsonload

**Module**
mmxml
save

**Purpose**
Save an XML document.

**Synopsis**
procedure save(doc: xmldoc, fname: text)
procedure save(doc: xmldoc, n: integer, fname: text)

**Arguments**
doc Document to save
n Node number to use as root node (default: 0)
fname Destination file name

**Example**
This example shows the two versions of this procedure.

```plaintext
declarations
    DB: xmldoc
    Pers: integer
end-declarations

! Save XML document to file 'results.xml'
save(DB, "results.xml")

! Display a subtree on screen
    Pers:= getnode(DB, "personnelList/region/employee[@id='T345']")
save(DB, Pers, "")
```

**Further information**
1. This routine generates an XML file from the provided `xmldoc` object. The XML header is produced using the properties defined with `setencoding`, `setxmlversion` and `setstandalone`. No header is emitted if either the encoding or the version is an empty string.

2. When providing an alternative root node, only the specified part of the document tree is exported without any XML header.

3. The document is formatted according to the indentation mode and its associated settings (see `setindentmode`); XML control characters are encoded (see `xmlencode`).

**Related topics**
load, save

**Module**
mmxml
**setattr**

**Purpose**
Set the value of an attribute.

**Synopsis**
```plaintext
procedure setattr(doc: xmldoc, n: integer, name: string, v: text|string|boolean|integer|real)
```

**Arguments**
- **doc** Document to use
- **n** Node number (must be an element)
- **name** Name of the attribute
- **v** New value for the attribute

**Example**
See `addnode`.

**Further information**
1. Attribute values are stored as `text` objects: the versions of this procedure accepting other types perform the conversion implicitly.
2. Attributes are nodes of type `XML_ATTR`: procedure `setvalue` may also be used to change the value of an attribute.
3. Setting an empty value to an attribute does not remove this attribute from the attribute list of the element. Use `delattr` for this purpose.

**Related topics**
- `getattr`, `delattr`

**Module**
- `mmxml`
setencoding

**Purpose**
Set the character encoding of the document.

**Synopsis**
procedure setencoding(doc: xmldoc, enc: string)

**Arguments**
doc Document to use
enc Name of the character encoding

**Further information**
The default character encoding is UTF-8.

**Related topics**
save, setstandalone, setxmlversion

**Module**
mmxml
**setmaxnodes**

**Purpose**
Set the number of allocated nodes for a document.

**Synopsis**
procedure setmaxnodes(doc: xmldoc, m: integer)

**Arguments**
doc Document to use  
m Number of nodes to reserve

**Further information**
This procedure sets the amount of memory reserved for a document. Normally, *mmxml* allocates memory on demand but using this procedure it is possible to allocate at once a larger block of memory to possibly speedup the loading of very large documents. If the requested amount is smaller than what is required to represent the document currently held in the doc object, the memory block is reduced as much as possible such that the document can still be stored.

**Related topics**
getmaxnodes

**Module**
*mmxml*
setname

Purpose
Set the name of a node.

Synopsis
procedure setname(doc:xmldoc, n:integer,name:string)

Arguments
doc    Document to use
n      Node number (must be an element or processing instruction)
name   New name for the node

Further information
Only element and processing instruction nodes can be modified with this routine; for all other
node types an error will be raised.

Related topics
setvalue

Module
mmxml
**setvalue**

**Purpose**
Set the value of a node.

**Synopsis**
```plaintext
procedure setvalue(doc: xmldoc, n: integer, v: text|string|integer|real|boolean)
```

**Arguments**
- `doc` Document to use
- `n` Node number
- `v` New value for the node

**Example**
See `copynode`.

**Further information**
1. Node values are stored as `text` objects: the versions of this procedure accepting other types perform the conversion implicitly.

2. Element nodes have no value: this procedure will modify the value of the first child text node of the element. If no such node exists, a new text node will be added to the beginning of the list of children.

**Related topics**
- `setname`

**Module**
- `mmxml`
sethspace

Purpose
Set horizontal spacing of a node.

Synopsis
procedure sethspace(doc: xmldoc, n: integer, s: integer)

Arguments
doc  Document to use
n    Node number
s    Number of spaces to put before the node output

Example
The following example reformats the XML document layout by adding an additional line before 'region' nodes and printing three consecutive tags within 'employee' on a single line. The indentmode is set to 'manual' in order to apply the user formatting (instead of automatic or none).

declarations
    DB: xmldoc
    NodeList, Employees: list of integer
end-declarations

! New line without indentation for Root
setvspace(DB, Root, 1)

! Add extra line in between regions, keeping original indentation
getnodes(DB, "personnelList/region", NodeList)
forall(r in NodeList) setvspace(DB, r, 2)

! Spacing/indentation for 'employee' tag
getnodes(DB, "personnelList/region/employee", Employees)
forall(p in Employees) do
    setvspace(DB, p, 1); sethspace(DB, p, 4)

! Within 'employee', display up to 3 consecutive tags on a single line
getnodes(DB, p, "child::node()[position() mod 3=1]", NodeList)
forall(r in NodeList) do
    setvspace(DB, r, 1); sethspace(DB, r, 6)
end-do
getnodes(DB, p, "child::node()[position() mod 3<>1]", NodeList)
forall(r in NodeList) do
    setvspace(DB, r, 0); sethspace(DB, r, 1)
end-do
end-do

! Set indentation mode to 'manual' to use our own formatting for display
setindentmode(DB, XML_MANUAL)
save(DB, ")

Further information
This spacing indicates the number of spaces to skip from the start of a new line before displaying the node when outputing the document. The horizontal spacing setting is only used when the indentation is in manual mode (see setindentmode).
Related topics
setvspace, setindentmode

Module
mmxml
**setvspace**

**Purpose**
Set vertical spacing of a node.

**Synopsis**
procedure setvspace(doc:xmldoc, n:integer, s:integer)

**Arguments**
doc Document to use
n Node number
s Number of carriage return to put before the node output

**Example**
See sethspace.

**Further information**
This spacing indicates the number of empty lines to add before displaying the node when outputing the document. The vertical spacing setting is only used when the indentation is in manual mode (see setindentmode).

**Related topics**
sethspace, setindentmode

**Module**
mmxml
**setindentmode**

**Purpose**
Set indent mode for the document.

**Synopsis**
procedure setindentmode(doc: xmldoc, imod: integer)

**Arguments**
doc Document to use
imod Indent mode:
  - XML_AUTO automatic indentation
  - XML_NONE no formatting
  - XML_MANUAL use vertical/horizontal spacing of each node

**Example**
See sethspace.

**Further information**
This parameter specifies how the XML document must be formatted by the save routine. Automatic indentation can be tuned by redefining the indent skip (setindentskip) and line length (setlinelen). If the indent mode is set to XML_NONE, the document is exported on a single line without formatting. Finally, with manual indenting, each node is placed according to its horizontal/vertical spacing as specified by setvspace and sethspace.

**Related topics**
save, setindentskip, setlinelen

**Module**
mmxml
setindentskip

Purpose
Set the size of an indentation step.

Synopsis
procedure setindentskip(doc: xmldoc, skip: integer)

Arguments
doc Document to use
skip Number of spaces to add for each indentation (at least 1; default is 2)

Example
This code reads in a document and displays its contents on screen applying automatic formatting with single space indentation and increased line length.

declarations
DB: xmldoc
end-declarations

! Reading data from an XML file
load(DB, "refexample.xml")

! Set indentation mode for XML output (default after load: MANUAL)
setindentmode(DB, XML_AUTO)

! Set smaller indentation skip than default
setindentskip(DB, 1)

! Increase default line length
setlinelen(DB, 80)

! Display document contents on screen
save(DB, "")

Further information
When the document is formatted automatically (see setindentmode) the number of spaces specified by this procedure is added to the current margin each time a new indent step is created.

Related topics
save, setindentmode, setlinelen

Module
mmxml
**setlinelen**

**Purpose**
Set the length of a line.

**Synopsis**
```plaintext
procedure setlinelen(doc:xml_doc, len:integer)
```

**Arguments**
- **doc**  Document to use
- **len**  Length of a line in characters (at least 1; default is 70)

**Example**
See setindentskip.

**Further information**
When outputting the document, a line break is inserted between nodes or while displaying a list of element attributes whenever more than the specified number of characters has been written.

**Related topics**
- save, setindentmode, setindentskip

**Module**
- mmxml
setstandalone

Purpose
Set the standalone flag of the document.

Synopsis
procedure setstandalone(doc:xmlDoc, std:integer)

Arguments
  doc   Document to use
  std   Standalone flag:
        -1 flag not specified
        0 standalone=no
        1 standalone=yes

Further information
The value of this flag is not used by mmxml. This is just an information to be saved in the header of the XML document. The default value for this flag is -1.

Related topics
save, setencoding, setxmlversion

Module
mmxml
**setxmlversion**

**Purpose**
Set the XML version of the document.

**Synopsis**
procedure setxmlversion(doc:xmldoc, xv:string)

**Arguments**
doc Document to use
xv XML version

**Further information**
The XML version number is not used by mmxml. This is just an information to be saved in the header of the XML document. The default value for this option is 1.0.

**Related topics**
save, setencoding, setstandalone

**Module**
mmxml
xmlattr

Purpose
Get an attribute during parsing of an element.

Synopsis
procedure xmlattr(ndx:integer, name;text, val;text)
procedure xmlattr(aname:string, val;text)

Arguments
ndx    Attribute index
name   Attribute name (returned by the procedure)
val    Attribute value (returned by the procedure)
aname  Attribute name (provided to the procedure)

Further information
1. This procedure can only be used from the open element function while parsing an XML document with the xmlparse function.
2. With the first syntax, the attribute index ndx is returned by the procedure (both its name and value). This index value must range between 1 and the last index as passed to the open element function. With the second syntax the name of the attribute to retrieve is given to the procedure. An empty string is returned if this attribute is not defined for the current element.

Related topics
xmlparse

Module
mmxml
xmlencode

Purpose
Encode a text string for XML.

Synopsis
function xmlencode(t:text):text

Argument
t text to encode

Further information
Encode a text string for XML by replacing control characters (<?, &apos, &quot) by their encoded equivalent.

Related topics
xmldecode

Module
mmxml
xmldecode

**Purpose**
Decode a text string for XML.

**Synopsis**
function xmldecode(t: text): text

**Argument**
t    text to decode

**Further information**
Decode a text string from XML by replacing encoded sequences (&lt; &gt; &amp; &apos; &quot;) by the corresponding control characters.

**Related topics**
xmlencode

**Module**
mmxml
xmlparse

Purpose
Parse an XML document.

Synopsis
function xmlparse(afct:array(range) of string,mode:integer,ctx:ctxtype): integer

Arguments
afct Event function table. Each entry of this array is the name of the function to call when the corresponding event occurs. The expected events are (all of these entries are optional):
- XML_FCT_DECL Document declarations
- XML_FCT_TXT A text node (same value as ML_TEXT)
- XML_FCT_CDATA A CDATA node
- XML_FCT_COM A commentary node
- XML_FCT_DATA A DATA node
- XML_FCT_PINST A processing instruction node
- XML_FCT_OPEN_ELT Opening of a new element node
- XML_FCT_CLOSE_ELT Closing of an element node
mode If 0, spaces are preserved and returned as text elements. Otherwise all text elements are trimmed
ctx Value passed as first argument of all event functions

Return value
0 if successful, 1 in case of parsing error or a non-zero value returned by an event function

Example
This example displays the structure of an XML document without loading it into memory.

! Display element name and update indentation
public function start_elt(spce:text,name:text,nba:integer):integer
  writeln(spce,name)
  spce=" 
end-function

! Update indentation when element closes
public function end_elt(spce:text):integer
  spce=" 
end-function

declarations
  afct:array(range) of string
end-declarations

afct(XML_FCT_OPEN_ELT):="start_elt" ! define open element
afct(XML_FCT_CLOSE_ELT):="end_elt" ! define close element
fopen("mydocument.xml",F_INPUT)
rts:=xmlparse(afct,1,text(""))
fclose(F_INPUT)
Further information

1. This function is an alternative approach to load for processing XML documents: instead of loading into memory the entire document this function calls a dedicated routine whenever it identifies an XML entity. For instance a specific function is called when an element is open and another one when it is closed. It is up to the Mosel program to decide how to handle the document via these event handling functions.

2. To each event type corresponds a specific function signature. These functions return an integer that decides whether parsing should continue: a non-zero value will cause the parsing to cancel (this value is used as the return value of xmlparse). The expected function signatures are:

   XML_FCT_DECL function xmldecl(ctx:ctxtype, vers:text, enc:text, std:integer):integer
   XML_FCT_TXT function text_node(ctx:ctxtype, type:integer, data:text):integer
   XML_FCT_CDATA function cdata_node(ctx:ctxtype, type:integer, data:text):integer
   XML_FCT_COM function comment_node(ctx:ctxtype, type:integer, data:text):integer
   XML_FCT_DATA function data_node(ctx:ctxtype, type:integer, data:text):integer
   XML_FCT_PINST function processing_instr(ctx:ctxtype, target:text, data:text):integer
   XML_FCT_OPEN_ELT function open_element(ctx:ctxtype, name:text, nba:integer):integer
   XML_FCT_CLOSE_ELT function close_element(ctx:ctxtype):integer

   In addition to the pre-defined arguments these functions take a context as their first parameter. This variable (that can be of any type) is provided to the xmlparse routine and can be used by the event functions for storing progress information.
   The type passed to the text node functions is the XML type corresponding to the function (namely XML_TXT, XML_CDATA, XML_COM, XML_DATA).
   The open_element function receives the name of the element as well as the number of defined attributes. To retrieve these attributes xmlattr can be used.

3. An error message indicating the location of the error is displayed when the parsing fails or if an event function returns a negative value (a positive value also interrupts parsing but no message is displayed).

Related topics
load

Module
mmxml
The \textit{mmxnlp} module provides access to nonlinear solvers, extending the capabilities provided by the \textit{mmxprs} and \textit{mmnl} modules. In particular, this module allows existing linear or mixed integer (MIP) models to be upgraded to include nonlinearities, without requiring unnecessary changes to the formulation. To use this module, the following line must be included in the header of the Mosel model file:

\begin{verbatim}
uses 'mmxnlp'
\end{verbatim}

Problem type and module hierarchy

Module \textit{mmxprs} provides

- linear models and
- mixed integer linear models.

Module \textit{mmnl} adds support for

- convex quadratic models,
- convex quadratic mixed integer models,
- convex, quadratically constrained models, and
- convex, quadratically constrained mixed integer models.

Module \textit{mmxnlp} adds support for

- general nonlinear problems and
- general nonlinear mixed integer problems.

If the \textit{mmxnlp} module is used for a model which does not require a general nonlinear solver, this should be equivalent to using the appropriate \textit{mmxprs} or \textit{mmnl} module directly.

\section*{18.1 New functionality for the Mosel language}

\subsection*{18.1.1 The userfunc type}

A nonlinear model may employ one or more black box evaluation functions, which can be used to provide function evaluations to the solver. These are represented in \textit{mmxnlp} by the new \texttt{userfunc} type. The implementation of each \texttt{userfunc} must be described by calling one of:
mmxnlp

- **userfuncMosel**: to declare that a user function is implemented as a Mosel function
- **userfuncExcel**: to declare that a user function is implemented in an Excel file
- **userfuncExcelMacro**: to declare that a user function is implemented as a Visual Basic macro in Excel
- **userfuncDLL**: to declare that a user function is implemented in a dynamically linked library (DLL)

Note that user functions returning multiple arguments are supported by the *mmxnlp* module. The $F$ construction allows a `userfunc` to be included in any nonlinear (nlctr) expression, and groups each occurrence of the `userfunc` with its parameters. During the solve, the parameters (which are of type nlctr themselves) will be evaluated at the current solution, and the real-valued results passed to the `userfunc` implementation. The function `userfuncinfo` can be used to find out which parameters the system has deduced it needs to pass to a particular `userfunc`.

### 18.1.2 The tolset type

The module provides a large number of configurable tolerances for users of the Xpress NonLinear SLP solver. A `tolset` describes a convergence tolerance set, which can be used for those nonlinear solvers supporting variable-specific convergence tolerances. The elements of a tolerance set are defined by using `settol`, and assigned to a variable or list of variables using `settolset`. For more details on tolerance sets, please refer to the *Xpress NonLinear Reference Manual*.

- `XNLP_TOL_TC`: The absolute closure tolerance
- `XNLP_TOL_TA`: The absolute delta tolerance
- `XNLP_TOL_RA`: The relative delta tolerance
- `XNLP_TOL_TM`: The absolute matrix tolerance
- `XNLP_TOL_RM`: The relative matrix tolerance
- `XNLP_TOL_TI`: The absolute impact tolerance
- `XNLP_TOL_RI`: The relative impact tolerance
- `XNLP_TOL_TS`: The relative slack impact tolerance
- `XNLP_TOL_RS`: The absolute slack impact tolerance

### 18.1.3 The mpproblem.xprs.xnlp problem type

When using the *mmxnlp* module, the type of the active Mosel problem is changed from mpproblem.xprs to the extended type mpproblem.xprs.xnlp. This means that all of the routines presented in this section operate in the context of the current Mosel problem.

### 18.2 *mmxnlp* and the other Mosel modules

*mmxnlp* is designed to provide seamless integration with other Mosel functionalities. However, the fundamentally different nature of nonlinear problems makes some compromises necessary; these are listed in this section.
18.2.1 Overloaded functions

The following functionality is modified or extended by the *mmxnlp* module:

- Retrieval of solution values with *getsol*, both for variables and nonlinear constraints. A detailed description of the behaviour of this function can be found in the documentation for the *mmaxps* and *mmnl* modules.

- Functions implemented in the *mmnl* module are extended for nonlinear solvers:
  - *setinitval*, *clearinitval* and *copyсолtoinit* to manage initial values.
  - Mathematical functions: *abs*, *exp*, *ln*, *log*, *sqrt*, *cos*, *sin*, *tan*, *arccos*, *arcsin*, *arctan*.
  - Mosel constraint and constraint visibility functions, *gettype*, *settype*, *ishidden* and *sethidden*.

- Functions implemented in the *mmxprs* module are extended for nonlinear solvers:
  - *maximize* and *minimize* to solve the problem.
  - *getprobstat* to return the problem solution status.
  - *fixglobal* for managing integer problems.
  - Exporting the current status (for debugging purposes) with *loadprob*, *writeprob* and *savestate*.

18.2.2 Module compatibility

The *mmquad* module is incompatible with the *mmxnlp* module, and should not be used together with it.

The *mmxprs* and *mmnl* modules are automatically loaded when using the *mmxnlp* module.

The *mmnl* module defines several discontinuous functions for use with decision variables (*mpvar*), which are not supported by the *mmxnlp* module. These constructions should instead be modelled with integer constraints. The functions are: *round*, *ceil*, *floor*, *idiv* and *mod*.

The following standard functionalities are not available for nonlinear problems:

- Functions for working with a basis: *loadbasis*, *readbasis* and *savebasis*.
- Logical constraints of the form *logctr*, and their operators: *implies*, *indicator*, *or*, *xor* and *and*.
- Functions for working with multiple MIP solutions, the solution pool and the solution enumerator: *selectsol*, *XPRS_enumduplpol*, *XPRS_enummaxsol* and *XPRS enumsols*.
- Functions for cut management, including model cuts and delayed rows: *addcut*, *addcuts*, *loadcuts*, *storecut*, *storecuts*, *delcut*, *dropcuts*, *getcnlist* and *getcplist*.
- Functions for determining irreducible infeasible sets, and for repairing infeasibility: *getiis*, *getiissense*, *getiistype*, *isiisvalid*, *resetiis* and *repairinfeas* and *getinfeas*.

18.3 Control parameters

When using *mmxnlp*, *getparam* and *setparam* are extended to additionally provide access to all the control and problem parameters of the Xpress NonLinear SLP solver. The module also provides the following controls of its own:
### XNLP_LOADASNL

**Description**
When set to true, quadratic expressions will be treated as being of general nonlinear type. If they are known to be non-convex, the overhead of attempting to treat the expression as convex initially is avoided.

**Type**
Integer, read/write

**Values**
- 0: Assume that quadratic expressions are convex
- 1: Assume that quadratic expressions are non-convex

**Default value**
0

**Module**
mmxnlp

### XNLP_LOADNAMES

**Description**
When set to true, names from the Mosel file will be passed to the underlying solver to improve the readability of messages it generates. This is an alias for XPRS_LOADNAMES.

**Type**
Integer, read/write

**Values**
- 0: Names are not loaded into the solver
- 1: Names are loaded into the solver

**Default value**
0

**Module**
mmxnlp

### XNLP_NLPSTATUS

**Description**
The solution status of the problem. For a detailed description of this value, please see the documentation for the XSLP_NLPSTATUS attribute in the Xpress NonLinear Reference Manual.

### XNLP_SOLVER

**Description**
Solver selection when available.

### XNLP_VERBOSE

**Description**
When set to true, informative messages from any underlying nonlinear solver will be displayed. This is an alias for XPRS_VERBOSE.
**Type**

Integer, read only

**Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Optimization unstarted</td>
</tr>
<tr>
<td>1</td>
<td>Locally optimal</td>
</tr>
<tr>
<td>2</td>
<td>Optimal</td>
</tr>
<tr>
<td>3</td>
<td>Locally infeasible</td>
</tr>
<tr>
<td>4</td>
<td>Infeasible</td>
</tr>
<tr>
<td>5</td>
<td>Unbounded</td>
</tr>
<tr>
<td>6</td>
<td>Unfinished</td>
</tr>
</tbody>
</table>

**Default value**

0

**Module**

`mmxnlp`

---

### XNLP_SOLVER

**Description**

Solver selection when available.

**Type**

Integer, read/write

**Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Determine automatically, based on problem characteristics and availability of solvers</td>
</tr>
<tr>
<td>0</td>
<td>Xpress NonLinear (SLP)</td>
</tr>
<tr>
<td>1</td>
<td>Knitro</td>
</tr>
</tbody>
</table>

**Default value**

-1

**Module**

`mmxnlp`

---

### XNLP_VERBOSE

**Description**

When set to true, informative messages from any underlying nonlinear solver will be displayed. This is an alias for XPRS_VERBOSE.

**Type**

Integer, read/write

**Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No solver logging</td>
</tr>
<tr>
<td>1</td>
<td>Solver log is displayed</td>
</tr>
</tbody>
</table>

**Default value**

0

**Module**

`mmxnlp`

---

### 18.4 Procedures and functions

This section lists in alphabetical order the functions and procedures that are provided by the `mmxnlp` module.

`addmultistart`    Loads a single or a set of multistart job(s) into the multistart job pool. p. 599
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>chgdeltatype</td>
<td>Changes the type of a delta variable associated to an mpvar.</td>
<td>600</td>
</tr>
<tr>
<td>F</td>
<td>Include a user function in a nonlinear constraint.</td>
<td>601</td>
</tr>
<tr>
<td>generateUFparallel</td>
<td>Generates a parallel version of a Mosel user function that is implemented as a Mosel package.</td>
<td>603</td>
</tr>
<tr>
<td>printmodelmemory</td>
<td>Print a summary of the current memory usage of the nonlinear module.</td>
<td>604</td>
</tr>
<tr>
<td>printmodelscaling</td>
<td>Print a summary of the scaling of the model, as loaded into the solver.</td>
<td>605</td>
</tr>
<tr>
<td>setcomplementary</td>
<td>Set two variables as being complementary.</td>
<td>606</td>
</tr>
<tr>
<td>setdelayedctr</td>
<td>Mark a constraint as delayed.</td>
<td>607</td>
</tr>
<tr>
<td>setdetrow</td>
<td>Set the determining row for a variable.</td>
<td>608</td>
</tr>
<tr>
<td>setenforcedctr</td>
<td>Mark a nonlinear constraint as enforced.</td>
<td>609</td>
</tr>
<tr>
<td>setinitsb</td>
<td>Provide the initial step bound for a variable.</td>
<td>610</td>
</tr>
<tr>
<td>settol</td>
<td>Define a particular tolerance in a tolerance set.</td>
<td>611</td>
</tr>
<tr>
<td>settolset</td>
<td>Assigns a tolerance set to a variable, or list of variables.</td>
<td>612</td>
</tr>
<tr>
<td>userfuncDLL</td>
<td>Create a user function implemented as a dynamic linked library.</td>
<td>613</td>
</tr>
<tr>
<td>userfuncExcel</td>
<td>Create a user function from a Microsoft Excel spreadsheet.</td>
<td>614</td>
</tr>
<tr>
<td>userfuncExcelMacro</td>
<td>Create a user function from a Microsoft Excel macro.</td>
<td>615</td>
</tr>
<tr>
<td>userfuncinfo</td>
<td>Print the inferred prototype of the given user function.</td>
<td>616</td>
</tr>
<tr>
<td>userfuncMosel</td>
<td>Create a user function from a Mosel function.</td>
<td>617</td>
</tr>
<tr>
<td>validate</td>
<td>Print a summary of the feasibility of the current solution.</td>
<td>618</td>
</tr>
</tbody>
</table>
addmultistart

**Purpose**
Loads a single or a set of multistart job(s) into the multistart job pool.

**Synopsis**
```
addmultistart(descr:string)
addmultistart(descr:string, controls:array(set of string) of real)
addmultistart(descr:string, initvalues:array(set of mpvar) of real)
addmultistart(descr:string, initvalues:array(set of mpvar) of real, controls:
    array(set of string) of real)
addmultistart(descr:string, controls:array(set of string) of real,
    initvalues:array(set of mpvar) of real)
addmultistart(descr:string, preset:integer)
addmultistart(descr:string, preset:integer, cnt:integer)
addmultistart(descr:string, preset:integer, initvalues:array(set of mpvar) of real,
    controls:array(set of string) of real)
addmultistart(descr:string, preset:integer, cnt:integer, initvalues:array(set of
    mpvar) of real, controls:array(set of string) of real)
```

**Arguments**
- `descr` Text description of the job. Used in reporting and in callbacks.
- `controls` An array containing the controls to be set for the loaded multistart job.
- `initvalues` An array containing initial values to be set for the loaded multistart job.
- `preset` The multistart preset of jobs to be loaded. Please see the *Xpress NonLinear Reference Manual* for the list of possible presets.
- `cnt` The upper bound on the number of jobs to be created, in case a preset is used.

**Further information**
Adds a job or a preset to the multistart job pool. Multistart jobs are automatically executed on the next minimize/maximize command, unless the `XSLP_MULTISTART` control is set to 0. Please refer to the *Xpress NonLinear Reference Manual* for a detailed description of multistart.

**Module**
```
mmxnlp
```
**chgdeltatype**

**Purpose**  
Changes the type of a delta variable associated to an mpvar.

**Synopsis**  
procedure chgdeltatype(col:mpvar, type:integer, value:real)

**Arguments**  
col The column for which the delta is to be changed.
type The new type of the delta.
value Value associated to the new delta type.

**Further information**  
Please refer to the *Xpress NonLinear Reference Manual* for more details about delta types.

**Related topics**  
setinitval, setinitsb, setdetrow, setenforcedctr.

**Module**  
mmxnlp
F

Purpose
Include a user function in a nonlinear constraint.

Synopsis
function F(UF:userfunc, arg:linctr):nlctr
function F(UF:userfunc, arg:nlctr):nlctr
function F(UF:userfunc, arg:list of nlctr):nlctr
function F(UF:userfunc, arg:array(any sets) of nlctr):nlctr
function F(UF:userfunc, arg:array(any sets) of nlctr, returnarg:integer):nlctr

Arguments
UF A user function of type userfunc
arg Argument to be passed to the user function
returnarg Return argument to be substituted into the formula for multivalued user functions

Return value
A nonlinear expression which may form part of any nlctr.

Example
The following example shows how to implement a negative cosine function.

```
model "SimpleUF"
uses "mmxnlp"
declarations
  obj: nlctr
  x: mpvar
  MinusSine: userfunc
end-declarations

! Creation and assignment of the user function
MinusSine := userfuncMosel("MinusSineImplementation")
! which can then be embedded into any nonlinear expression
obj := F(MinusSine,x)
minimize(obj)

function MinusSineImplementation (x:real): real
  returned := -sin(x)
end-function
end-model
```

Further information
User functions allow extremely complex, recursive or non-algebraic expressions to be included in nonlinear formulae. As such they may make use of simulators or other black box evaluators. The actual parameters to a user function depend upon the way it is bound to the model by the F function. Please see the chapter on user functions for more details. Each user function instance defined by the means of the F function must share the same argument syntax structure, however the actual formula content may differ: e.g. if a function takes an array of nonlinear expressions as input arguments, each instance of the function corresponding to the same definition based on the same F instance must have the same underlying array structure, although the expressions stored in them may differ. If a separate F instance is used using the same function
implementation, this rule does not apply. Also note, that for Mosel to be able to correctly cross reference the sets used in the definition of an array, the sets must be named.

Related topics
userfuncDLL, userfuncMosel, userfuncExcel, userfuncExcelMacro.

Module
mmxnlp
**generateUFparallel**

**Purpose**
Generates a parallel version of a Mosel user function that is implemented as a Mosel package.

**Synopsis**
```plaintext
procedure generateUFparallel(bimname:string, fctname:string)
```

**Arguments**
- `bimname`: Path to the compiled Mosel package implementing the user function.
- `fctname`: The public user function inside the package.

**Further information**
Please refer to the *Xpress NonLinear Reference Manual* for more details about this functionality.

**Module**
`mmxnlp`
printmodelmemory

**Purpose**
Print a summary of the current memory usage of the nonlinear module.

**Synopsis**
procedure printmodelmemory

**Further information**
This procedure has no effect unless XNLP_VERBOSE is set. It is provided solely for the purpose of model analysis and debugging.

**Related topics**
validate, printmodelscaling, userfuncinfo.

**Module**
mmxnlp
printmodelscaling

Purpose
Print a summary of the scaling of the model, as loaded into the solver.

Synopsis
procedure printmodelscaling

Further information
This procedure has no effect unless XNLP_VERBOSE is set. It is provided solely for the purpose of model analysis and debugging.

Related topics
validate, printmodelmemory, userfuncinfo.

Module
mmxnlp
setcomplementary

**Purpose**
Set two variables as being complementary.

**Synopsis**
procedure setcomplementary(var1:mpvar, var2:mpvar)

**Arguments**
- **var1** The first variable of the variable pair to be set as complementing
- **var2** The first variable of the variable pair to be set as complementing

**Further information**
A complementing variable pair implements the constraint that is equivalent with the product of the variables being zero. However, the solvers may be able to treat such constraints in a special, more efficient ways, which may make a difference if the complementarity constraints are the problematic part of the model. Note that Knitro only allows non-overlapping complementary variables, and in the presence of overlaps Xpress will default to use SLP. Complementary variables must have a lower bound of zero.

**Related topics**
- setinitval, setinitsb, setdetrow, setenforcedctr.

**Module**
mmxnlp
setdelayedctr

Purpose
Mark a constraint as delayed.

Synopsis
procedure setdelayedctr(row:nlctr, delay:integer)

Arguments
row The constraint to be delayed
delay Integer value defining the number of iterations in which the constraint should be ignored

Further information
A delayed constraint will be introduced after some number of solver iterations have occurred. This may be useful for constraints that could aid the convergence of a solver, but which are not expected to be binding at an optimal solution and which could make early iterations more expensive.

Related topics
setinitval, setinitsb, setdetrow, setenforcedctr.

Module
mmxnlp
setdetrow

Purpose
Set the determining row for a variable.

Synopsis
procedure setdetrow(var:mpvar, row:linctr)
procedure setdetrow(var:mpvar, row:nlctr)
procedure setdetrow(row:linctr, var:mpvar)
procedure setdetrow(row:nlctr, var:mpvar)

Arguments
var     The variable for which the determining row is provided
row     The row that determines the value of the variable.

Further information
A row which is determining for a variable defines the value of that variable. This means that the variable is a derived value which is calculated in another part of the model. Some solvers will use such designations to refine their search, and in particular in sequential linear programming, a process called cascading makes use of determining rows. Please refer to the Xpress NonLinear Reference Manual (chapter ‘Cascading’) for more information.

Related topics
setinitval, setinitsb, setenforcedctr, setdelayedctr.

Module
mmxnlp
setenforcedctr

Purpose
Mark a nonlinear constraint as enforced.

Synopsis
procedure setenforcedctr(row:nlctr)

Argument
row The constraint to be set enforced

Further information
A constraint which is marked as enforced will not have penalty error vectors introduced upon it by solvers which use such techniques. This may be useful for constraints which are hard to satisfy.

Related topics
setinitval, setinitsb, setdetrow, setdelayedctr.

Module
mmxnlp
**setinitsb**

**Purpose**
Provide the initial step bound for a variable.

**Synopsis**

```plaintext
procedure setinitsb(var:mpvar, value:real)
```

**Arguments**
- `var` The variable for which the step bound is provided
- `value` Value to be used as initial value

**Further information**
The initial step bounds define in turn the size of the initial trust region. Please refer to the *Xpress NonLinear Reference Manual* for more information.

**Related topics**
- `setinitval`, `setdetrow`, `setenforcedctr`, `setdelayedctr`.

**Module**
- `mmxnlp`
settol

Purpose
Define a particular tolerance in a tolerance set.

Synopsis
procedure settol(tset:tolset, which:integer, value:real)

Arguments
tset    The tolerance set to be modified
which    The tolerance which is being defined
value    The new value of the tolerance

Further information
The tolerances which may be defined by this method are:

XNLP_TOL_TC    The absolute closure tolerance
XNLP_TOL_TA    The absolute delta tolerance
XNLP_TOL_RA    The relative delta tolerance
XNLP_TOL_TM    The absolute matrix tolerance
XNLP_TOL_RM    The relative matrix tolerance
XNLP_TOL_TI    The absolute impact tolerance
XNLP_TOL_RI    The relative impact tolerance
XNLP_TOL_TS    The relative slack impact tolerance
XNLP_TOL_RS    The absolute slack impact tolerance

Please refer to the Xpress NonLinear Reference Manual, and particularly the chapter 'Convergence criteria', for more information on these tolerances.

Related topics
setinitval, setinitb, setdetrow, setenforcedctr.

Module
mmxnlp
**settolset**

**Purpose**
Assigns a tolerance set to a variable, or list of variables.

**Synopsis**
procedure settolset(var:mpvar, tset:tolset)
procedure settolset(vars:list of mpvar, tset:tolset)

**Arguments**
- var Variable to which the tolerance set is to be assigned
- vars List of variable to which the tolerance set is to be assigned
- tset The tolerance set to be assigned to the variable(s)

**Related topics**
settol.

**Module**
mmxnlp
userfuncDLL

**Purpose**  
Create a user function implemented as a dynamic linked library.

**Synopsis**  
`function userfuncDLL(libfile:string, fctname:string):userfunc`

**Arguments**  
- `libfile`  
  Name of the dynamically linked library containing the implementation of the user function  
- `fctname`  
  Name of the function inside the dynamic library

**Return value**  
A `userfunc` object that can be used in the `F` functions to be embedded in formulas.

**Further information**  
User functions allow extremely complex, recursive or non-algebraic expressions to be included in nonlinear formulae. As such they may make use of simulators or other black box evaluators. The actual parameters to a user function depend upon the way it is bound to the model by the `F` function. Please see the chapter 'User functions' of the Xpress NonLinear Reference Manual for more details. Dynamically linked libraries are supported on all platforms, and are usually the most computationally efficient way to implement user functions in `mmxnlp`.

**Related topics**  
`F`, `userfuncMosel`, `userfuncExcel`, `userfuncExcelMacro`.

**Module**  
`mmxnlp`
userfuncExcel

Purpose
Create a user function from a Microsoft Excel spreadsheet.

Synopsis
function userfuncExcel(filename:string, sheetname:string):userfunc
function userfuncExcel(filename:string, sheetname:string, macro:string):userfunc

Arguments
filename Name of the Excel file including the function implementation
sheetname Name of the worksheet in the workbook used for input and output
macro Name of the Visual Basic macro to be called to recalculate the spreadsheet.
Optional, if not provided recalculation is done by the standard recalculation request

Return value
A userfunc object that can be used in the F functions to be embedded in formulas.

Further information
User functions allow extremely complex, recursive or non-algebraic expressions to be included in nonlinear formulae. As such they may make use of simulators or other black box evaluators. The actual parameters to a user function depend upon the way it is bound to the model by the F function. Please see the chapter on user functions for more details. Communication with Excel carries significant overhead and performance degradation may result from using functions of this type.

Related topics
F, userfuncMosel, userfuncExcelMacro, userfuncDLL.

Module
mmxnlp
userfuncExcelMacro

Purpose
Create a user function from a Microsoft Excel macro.

Synopsis

function userfuncExcelMacro(filename:string, sheetname:string, macro:string):userfunc

Arguments

filename Name of the Excel file including the function implementation
sheetname Name of the worksheet in the workbook including the Visual Basic macro
macro Name of the Visual Basic macro implementing the user function

Return value

A userfunc object that can be used in the F functions to be embedded in formulas.

Further information

User functions allow extremely complex, recursive or non-algebraic expressions to be included in nonlinear formulae. As such they may make use of simulators or other black box evaluators. The actual parameters to a user function depend upon the way it is bound to the model by the F function. Please see the chapter on user functions for more details. Communication with Excel carries significant overhead and performance degradation may result from using functions of this type.

Related topics

F, userfuncMosel, userfuncExcel, userfuncDLL.

Module

mmxnlp
userfuncinfo

**Purpose**
Print the inferred prototype of the given user function.

**Synopsis**
procedure userfuncinfo(UF:userfunc)

**Argument**
UF The user function to be analyzed

**Further information**
The type and signature of a user function are inferred from its use in calls to the \( F \) function in the current model. This procedure has no effect unless XNLP_VERBOSE is set. It is provided solely for the purpose of model analysis and debugging.

**Related topics**
validate, printmodelmemory, printmodelscaleing.

**Module**
mmxnlp
userfuncMosel

**Purpose**
Create a user function from a Mosel function.

**Synopsis**
function userfuncMosel(fctname:string):userfunc
function userfuncMosel(fctname:string, options:integer):userfunc

**Arguments**
- `fctname` Name of the Mosel function to wrap
- `options` Options describing special properties of the user function

**Return value**
A `userfunc` object that can be used in the F functions to be embedded in formulas.

**Further information**
User functions allow extremely complex, recursive or non-algebraic expressions to be included in nonlinear formulae. As such they may make use of simulators or other black box evaluators. The actual parameters to a user function depend upon the way it is bound to the model by the F function. Please see the chapter on user functions for more details.

There is support for user functions providing their own derivatives. Currently, user functions taking an array of `nlctr` and returning a single function values may provide their own derivatives. To mark a function as returning it's own derivatives, use option `XNLP_DERIVATIVES` or `XNLP_DELTAS` to indicate that the solver should suggest perturbation values for the variables.

**Related topics**
- F, userfuncExcel, userfuncExcelMacro, userfuncDLL.

**Module**
mmxnlp
validate

Purpose
Print a summary of the feasibility of the current solution.

Synopsis
procedure validate

Further information
This procedure has no effect unless XNLP_VERBOSE is set. It is provided solely for the purpose of model analysis and debugging.

Related topics
printmodelmemory, printmodelscale, userfuncinfo.

Module
mmxnlp
### 18.5 Error codes issued by mmxnlp

<table>
<thead>
<tr>
<th>Code</th>
<th>Error Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Out of memory</td>
</tr>
<tr>
<td></td>
<td>The system has run out of memory.</td>
</tr>
<tr>
<td>2</td>
<td>No purchase authorization found</td>
</tr>
<tr>
<td></td>
<td>No license found</td>
</tr>
<tr>
<td>3</td>
<td>Failed to initialize XSLP</td>
</tr>
<tr>
<td></td>
<td>Cannot initialize the XPRS library. There may be a licensing problem</td>
</tr>
<tr>
<td>4</td>
<td>Unsupported XSLP version</td>
</tr>
<tr>
<td></td>
<td>The version of the ‘XSLP’ library is incompatible with the current module. The Xpress installation may be corrupt</td>
</tr>
<tr>
<td>5</td>
<td>Failed to create the XSLP problem object</td>
</tr>
<tr>
<td></td>
<td>Cannot create the XSLP optimizer problem. There may be a licensing problem</td>
</tr>
<tr>
<td>6</td>
<td>Unexpected mmxnlp user function signature</td>
</tr>
<tr>
<td></td>
<td>The provided user functions’ signature does not match any expected format.</td>
</tr>
<tr>
<td>7</td>
<td>Unexpected external token in mmxnlp</td>
</tr>
<tr>
<td></td>
<td>An unexpected external token found by the ‘mmxnlp’ module. Please contact support.</td>
</tr>
<tr>
<td>8</td>
<td>Unsupported operator</td>
</tr>
<tr>
<td></td>
<td>The provided operator is not supported by ‘mmxnlp’.</td>
</tr>
<tr>
<td>9</td>
<td>Failed to load problem</td>
</tr>
<tr>
<td></td>
<td>Could not load the problem into the optimizer.</td>
</tr>
<tr>
<td>10</td>
<td>Variable bound conflict in problem</td>
</tr>
<tr>
<td></td>
<td>Inconsistent bounds provided for the variable.</td>
</tr>
<tr>
<td>11</td>
<td>Failed to load user function</td>
</tr>
<tr>
<td></td>
<td>The user function could not be loaded into the optimizer.</td>
</tr>
<tr>
<td>12</td>
<td>Error evaluating user function</td>
</tr>
<tr>
<td></td>
<td>Error while evaluation the user function. The user function likely to have returned an error code.</td>
</tr>
<tr>
<td>13</td>
<td>Unknown tolerance set</td>
</tr>
<tr>
<td></td>
<td>The provided tolerance set is invalid.</td>
</tr>
<tr>
<td>14</td>
<td>List type error in user function</td>
</tr>
<tr>
<td></td>
<td>The list provided to the user function is not valid for the function.</td>
</tr>
<tr>
<td>15</td>
<td>Failed to create save file</td>
</tr>
<tr>
<td></td>
<td>The savefile could not be created.</td>
</tr>
<tr>
<td>16</td>
<td>Error in optimization</td>
</tr>
<tr>
<td></td>
<td>An error has occurred during optimization.</td>
</tr>
<tr>
<td>17</td>
<td>Cannot reoptimize using a different objective (use named linctr or nlctr)</td>
</tr>
<tr>
<td></td>
<td>The objective has unexpectedly changed</td>
</tr>
</tbody>
</table>
18 **Internal error in mmxnlp. Please contact FICO support**
An internal error has occurred. Please contact support.

20 **Incompatible array definitions for user function arguments**
The user function received incompatible arrays.

21 **Non-Mosel user functions only take ‘list of nlctr’ type arguments**
User functions that are not implemented as a Mosel function can only take list of ‘nlctr’ arguments (no arrays).

22 **Invalid argument list for external function**
The provided argument list is not valid for the external function.

23 **Provided user function is not returning a single real**
The provided user function was expected to return a single real value.

24 **Provided user function is not returning an array indexed by integers**
The provided user function was expected to return an array of reals indexed by integers.

25 **User function must be loaded before it’s properties can be retrieved**
The user function must be loaded before it’s properties are interrogated. Please use ‘loadprob’ to load the model including the user function.

26 **Unexpected variable found. Please reload problem first using ‘loadprob’**
An unexpected variable has been used. Please reload the problem to load the variable.

27 **Operation only supported on the main problem (e.g. not inside multi-start callbacks)**
This operation is only supported in the main problem. It cannot be used on worker problems.

28 **Math error while evaluating expression**
A mathematical error has occurred while evaluating the expression.
The \textit{mmxprs} module provides access to FICO\textsuperscript{®} Xpress Optimizer from within a Mosel model and as such it requires the Xpress Optimizer library (XPRS) to be installed on the system. To use this module, the following line must be included in the header of the Mosel model file:

\begin{verbatim}
uses 'mmxprs'
\end{verbatim}

A large number of optimization-related routines are provided, ranging from those for finding a solution to the problem, to those for setting callbacks and cut manager functions. Whilst a description of their usage is provided in this manual, further details relating to the usage of these may be found by consulting the \textit{Xpress Optimizer Reference Manual}.

\section{New functionality for the Mosel language}

\subsection{The problem type \texttt{mpproblem.xprs}}

This module exposes its functionality through an extension to the \texttt{mpproblem} problem type. As a consequence, all routines presented here are executed in the context of the current problem. In particular, the setting of a control parameter is applied only to the current problem and each problem has its own set of settings and solution information. However, when a new problem instance is created, the value of the control parameters \texttt{XPRS\_colorder}, \texttt{XPRS\_enummaxsol}, \texttt{XPRS\_enumduplpol}, \texttt{XPRS\_loadnames} and \texttt{XPRS\_verbose} are initialised with the settings of the main problem.

\subsection{The type \texttt{basis}}

The module \textit{mmxprs} defines the type \texttt{basis} to represent solution basis in the Mosel Language. This new type is used to store a basis computed by the optimizer during its solution process (\texttt{savebasis}). A basis can then be loaded again into the optimiser with \texttt{loadbasis}, inspected (by getting the basis status of each variable/constraint it includes with \texttt{getbstatus}) or modified (by changing this basis status using \texttt{setbstat}). The type \texttt{basis} supports assignment and test of equality. This comparison only checks whether two basis contain the same information, it does not indicate whether the basis are equivalent.

\subsection{The type \texttt{mpsol}}

The type \texttt{mpsol} characterises a \textit{solution} of an MP problem by associating a value to each decision variable (type \texttt{mpvar}) of the problem. Initialising such an object can be achieved by saving the current solution found by the optimizer (\texttt{savesol} or \texttt{savemipsol}) or by building it one variable at a time (\texttt{setsol}). Various routines requiring solution information support the solution object. For instance \texttt{getsol} may be used to evaluate an expression on a specific solution; \texttt{loadmipsol} and
addmipsol accept this object as input. A solution might be saved into a file using writesol and the resulting file can be loaded into the optimiser with readsol. The type mpsol supports assignment and test of equality.

19.1.4 The type logctr

The type logctr represents either a logical expression over linear constraints or an indicator constraint (see indicator). Logical expressions can be built using standard operators (and, or, not) or with the help of the dedicated functions implies and xor. These logical constructs are handled like linear constraints: they are associated to the current problem, can be (re)defined via assignments and hidden using sethidden. Note however that logical constructs are not shown by exportprob although the mmxprs routine writeprob will report them.

If logical expressions are employed in a model, the loading of the problem into the optimizer requires the use of the helper package "advmod":

```plaintext
uses 'advmod'
```

This package is not necessary when a model uses only indicator constraints directly.

19.2 Control parameters

This module extends the getparam function and the setparam procedure in order to access all the control and problem parameters of Optimizer (for example the problem attribute LPSTATUS is mapped to the mmxprs control parameter XPRS_lpstatus). In addition to these, the following control parameters are also defined:

- **XPRS_colorder**: Reorder matrix columns before loading the problem. p. 623
- **XPRS_enumduplpol**: Handling of duplicate solutions during an enumeration. p. 623
- **XPRS_enummaxsol**: Maximum number of solutions to be saved during an enumeration. p. 623
- **XPRS_enumsols**: Number of solutions found during the last enumeration. p. 623
- **XPRS_fullversion**: Optimizer version number. p. 624
- **XPRS_loadnames**: Enable/disable loading of MPS names into the Optimizer. p. 624
- **XPRS_problem**: Optimizer problem pointers. p. 624
- **XPRS_probname**: Read/set the problem name used by the Optimizer. p. 625
- **XPRS_verbose**: Enable/disable message printing by the Optimizer. p. 625

Example:

```plaintext
setparam("XPRS_verbose", true) ! Turn on message printing
pstat:= getparam("XPRS_lpstatus") ! Get the problem LP optimization status
writeln("Best bound=", getparam("XPRS_bestbound")) ! Display the best bound value
```
XPRS_colorder

Description: Reorder matrix columns before loading the problem.
Type: Integer, read/write
Values:
0  Mosel implicit ordering
1,3  Reorder using a numeric criterion
2  Alphabetical order of the variable names (this requires the names to be available)
4  Random ordering
Default value: 0
Module: mmxprs

XPRS_enumsols

Description: Number of solutions found during the last enumeration. The value of this parameter is -1 if no enumeration has been run.
Type: Integer, read only
Affects routines: maximize, minimize
Module: mmxprs

XPRS_enummaxsol

Description: Maximum number of solutions to be saved during an enumeration.
Type: Integer, read/write
Default value: 10
Affects routines: maximize, minimize
Module: mmxprs

XPRS_enumduplpol

Description: Handling of duplicate solutions during an enumeration. Refer to the MSP control parameter MSP_DUPLICATESOLUTIONSPOLICY for further information.
Type: Integer, read/write
### Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>All solutions kept</td>
</tr>
<tr>
<td>1</td>
<td>Continuous</td>
</tr>
<tr>
<td>2</td>
<td>Discrete and continuous separate</td>
</tr>
<tr>
<td>3</td>
<td>Discrete only</td>
</tr>
</tbody>
</table>

**Default value** 3

**Affects routines** `maximize, minimize`

**Module** `mmxprs`

### XPRS_fullversion

**Description**
The full Optimizer version number in the form `major.minor.build` (e.g. "20.01.03").

**Type** String, read only

**Module** `mmxprs`

### XPRS_loadnames

**Description**
Enable/disable loading of MPS names into the Optimizer.

**Type** Boolean, read/write

**Values**
- `true` Enable loading of names
- `false` Disable loading of names

**Default value** `false`

**Affects routines** `loadprob, maximize, minimize`

**Module** `mmxprs`

### XPRS_problem

**Description**
The Optimizer problem (`XPRSProb`), MIP solution pool (`XPRSMipsolpool`) and MIP solution enumerator (`XPRSMipsolenum`) pointers separated by spaces. This attribute is only required in applications using both Mosel and the Optimizer at the C level.

**Type** String, read only

**Module** `mmxprs`
XPRS_probnome

Description: Read/set the problem name used by the Optimizer to build its working files (this name may contain a full path). If set to the empty string (default value), a unique name with a path to the temporary directory of the operating system is generated.

Type: String, read/write

Module: mmxprs

XPRS_verbose

Description: Enable/disable message printing by the Optimizer.

Type: Boolean, read/write

Values:
- true: Enable message printing
- false: Disable message printing

Default value: false

Module: mmxprs

19.3 Procedures and functions

This section lists in alphabetical order the functions and procedures that are provided by the mmxprs module.

- addmipsol: Add a MIP solution to the optimizer. p. 628
- basisstability: Get basis stability information. p. 629
- calcsoinfo: Calculates a property of an mpsol solution. p. 630
- clearmipdir: Delete all defined MIP directives. p. 631
- clearmodcut: Delete all defined model cuts. p. 632
- command: Execute an Optimizer command. p. 633
- copysoItoinit: Copy solution values to initial values of an NL problem. p. 634
- defdelayedrows: Define the set of constraints to be treated as delayed rows. p. 635
- defsecurevecs: Define the variables and constraints to be preserved. p. 636
- estimatemarginals: Estimate better marginal values for variables and constraints for degenerate problems. p. 637
- fixglobal: Fix values of global entitites. p. 638
- getbstat: Get the status of a variable or constraint in a basis. p. 639
- getdualray: Get a dual ray for an infeasible problem. p. 640
getiis | Compute then get the Irreductible Infeasible Sets (IIS). | p. 641
getiissense | Decode the sense part of an IIS bound type information. | p. 642
getiistype | Decode the type part of an IIS bound type information. | p. 643
getinfcause | Returns the variable or constraint causing infeasibility. | p. 644
getinfeas | Returns sets of infeasible primal and dual variables. | p. 645
getlb | Get the lower bound of a variable. | p. 646
getloadedlinctrs | Get the linear constraints loaded into the optimiser. | p. 647
getloadedmpvars | Get the decision variables loaded into the optimiser. | p. 648
getname | Get the name of a decision variable or constraint. | p. 649
getprimalray | Get a primal ray for an unbounded problem. | p. 650
getprobstat | Get the Optimizer problem status. | p. 651
getrange | Get a range value for a variable or constraint. | p. 652
getsensrng | Get sensitivity ranges for objective or RHS function coefficients. | p. 653
getsize | Get the size of a solution. | p. 654
getsol | Get the solution value of an expression from a solution object. | p. 655
getub | Get the upper bound of a variable. | p. 656
getvars | Get the set of variables of a solution. | p. 657
hasfeature | Check if a specific feature is supported by the currently used license. | p. 658
implies | Create an implies expression. | p. 659
indicator | Create an indicator constraint. | p. 660
isiisvalid | Check whether an IIS number exists. | p. 661
isintegral | Check whether a solution value is integral. | p. 662
loadbasis | Load a previously saved basis. | p. 663
loadmipsol | Load a MIP solution into the optimizer. | p. 664
loadprob | Load a problem into the optimizer. | p. 666
maximize, minimize | Maximize/minimize the current problem. | p. 667
postsolve | Postsolve the current matrix. | p. 669
readbasis | Read a basis from a file. | p. 670
readdir | Read directives from a file. | p. 671
readsol | Read a solution from a file. | p. 672
refinemipsol | Executes the MIP solution refiner on an mpsol solution. | p. 673
rejectintsol | Reject a PREINTSOL solution. | p. 674
repairinfeas | Relating bounds to repair infeasibility. | p. 675
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>resetbasis</td>
<td>Reset a basis.</td>
<td>677</td>
</tr>
<tr>
<td>resetiis</td>
<td>Reset the search for IIS.</td>
<td>678</td>
</tr>
<tr>
<td>resetsol</td>
<td>Reset a solution.</td>
<td>679</td>
</tr>
<tr>
<td>savebasis</td>
<td>Save the current basis.</td>
<td>680</td>
</tr>
<tr>
<td>savemipsol</td>
<td>Save the current solution into the provided array or solution object.</td>
<td>681</td>
</tr>
<tr>
<td>savesol</td>
<td>Save the current solution into a solution object.</td>
<td>682</td>
</tr>
<tr>
<td>savestate</td>
<td>Save current state of the Optimizer to a file.</td>
<td>683</td>
</tr>
<tr>
<td>selectsol</td>
<td>Select one of the solutions found by solution enumerator.</td>
<td>684</td>
</tr>
<tr>
<td>setarchconsistency</td>
<td>Sets the optimizer architecture control.</td>
<td>685</td>
</tr>
<tr>
<td>setbstat</td>
<td>Set the status of a variable or constraint in a basis.</td>
<td>686</td>
</tr>
<tr>
<td>setcallback</td>
<td>Set optimizer callback functions and procedures.</td>
<td>687</td>
</tr>
<tr>
<td>setcbcutoff</td>
<td>Set cutoff for PREINTSOL callback.</td>
<td>690</td>
</tr>
<tr>
<td>setgndata</td>
<td>Update data for GAPNOTIFY callback.</td>
<td>691</td>
</tr>
<tr>
<td>setlb</td>
<td>Set the lower bound of a variable.</td>
<td>692</td>
</tr>
<tr>
<td>setmipdir</td>
<td>Set a directive on a variable or Special Ordered Set.</td>
<td>693</td>
</tr>
<tr>
<td>setmodcut</td>
<td>Mark a constraint as a model cut.</td>
<td>694</td>
</tr>
<tr>
<td>setsol</td>
<td>Define the value associated to a decision variable in a solution object.</td>
<td>695</td>
</tr>
<tr>
<td>setub</td>
<td>Set the upper bound of a variable.</td>
<td>696</td>
</tr>
<tr>
<td>setucbdata</td>
<td>Update data for CHGBRANCH callback.</td>
<td>697</td>
</tr>
<tr>
<td>stopoptimize</td>
<td>Interrupt the optimizer algorithms.</td>
<td>698</td>
</tr>
<tr>
<td>unloadprob</td>
<td>Unload the problem held in the optimizer.</td>
<td>699</td>
</tr>
<tr>
<td>writebasis</td>
<td>Write the current basis to a file.</td>
<td>700</td>
</tr>
<tr>
<td>writedirs</td>
<td>Write current directives to a file.</td>
<td>701</td>
</tr>
<tr>
<td>writeprob</td>
<td>Write the current problem to a file.</td>
<td>702</td>
</tr>
<tr>
<td>writesol</td>
<td>Write a solution to a file.</td>
<td>703</td>
</tr>
<tr>
<td>xor</td>
<td>Create an exclusive or expression.</td>
<td>704</td>
</tr>
</tbody>
</table>
addmipsol

Purpose
Add a MIP solution to the optimizer.

Synopsis
procedure addmipsol(solid:string,s|array(set of mpvar) of real)
procedure addmipsol(solid:string,ms:mpsol)

Arguments
solid   Identifier to be assigned to the solution
s       An array containing the solution
ms      A solution object

Further information
1. This function is used to provide the expectations of the modeler on the values of selected variables in possible MIP solutions. It is different to loadmipsol in that it is not necessary to provide full, feasible MIP solutions. The values provided will be used by the Optimizer to attempt to generate full MIP solutions. The addmipsol function can therefore be used to trial the feasibility of certain variable value assignments without the need to fix them in the problem formulation itself.

2. The solution value array \( s \) is created by assigning values to discrete variables in the problem, such as \( s(x) := 1 \) (where \( x \) is a decision variable of type mpvar). It is also possible to use a solution that has previously been saved using the procedure savemipsol.

3. If the provided solution is found to be infeasible, a limited local search heuristic will be run in an attempt to find a close feasible integer solution.

4. The current problem definition must be loaded into the Optimizer for addmipsol to have any effect. If this has not recently been done, e.g., by calling maximize or minimize, the problem must be explicitly loaded using loadprob.

5. The function returns immediately after passing the solution to the Optimizer. The solution is placed in a pool until the optimizer is able to analyze the solution during a MIP solve.

6. The SOLNOTIFY callback function can be used to discover the outcome of a loaded solution, based on the identifier assigned to the solution (see setcallback).

Related topics
savemipsol, loadmipsol.

Module
mmxprs
basisstability

Purpose
Get basis stability information.

Synopsis
function basisstability(type:integer,norm:integer,scaled:boolean):real

Arguments
- **type** Which information to return. Possible values:
  - 0 Condition number of the basis
  - 1 Stability measure for the solution relative to the current basis
  - 2 Stability measure for the duals relative to the current basis
  - 3 Stability measure for the right hand side relative to the current basis
  - 4 Stability measure for the basic part of the objective relative to the current basis
- **norm** Which norm to use. Possible values:
  - 0 Use the infinity norm
  - 1 Use the 1 norm
  - 2 Use the Euclidian norm for vectors, and the Frobenius norm for matrices
- **scaled** If false, work on the unscaled matrix

Return value
Basis stability information.

Module
mmxprs
calcsolinfo

Purpose
Calculates a property of an mpsol solution.

Synopsis
function calcsolinfo(solution:mpsol, option:integer):mpsol

Arguments
solution The solution to be checked
option Which information to return. Possible values:
XPRS_SOLINFO_ABSPRIMALINFEAS Calculate the maximum absolute primal infeasibility
XPRS_SOLINFO_RELPRIMALINFEAS Calculate the maximum relative primal infeasibility
XPRS_SOLINFO_MAXMIPFRACTIONAL Calculate the maximum fractionality of the integer variables

Related topics
refinemipsol.

Module
mmxprs
clearmipdir

**Purpose**
Delete all defined MIP directives.

**Synopsis**
procedure clearmipdir

**Further information**
This procedure clears the list of directives defined so far.

**Related topics**
setmipdir.

**Module**
mmxprs
clearmodcut

Purpose
Delete all defined model cuts.

Synopsis
procedure clearmodcut

Further information
This procedure clears the list of model cuts defined so far.

Related topics
setmodcut.

Module
mmxprs
command

Purpose
Execute an Optimizer command or enter interactive mode of the Optimizer.

Synopsis

procedure command(cmd:string)
procedure command

Argument

cmd Command or sequence of commands separated by "\n" character

Example

Solve a MIP problem and then enter interactive mode:

command("minim\nglobal")
command

Further information

1. When used without parameter, this procedure enters an interactive mode of the Optimizer similar to the console mode: model execution is suspended and Optimizer commands can be typed directly. Model execution resumes after command quit has been typed or the input stream has reached an end of file. Using the alternate form of the procedure with an argument, one can send a command (or sequence of commands) to the Optimizer: this may be useful to execute commands for which there is no mmxprs interface. During the execution of this procedure, callbacks set up in the model are effective and the problem solution status of mmxprs is updated upon termination. Note that, commands altering the problem must be avoided (like readprob, change of name of the problem, etc.) in order to preserve consistency between Mosel and Optimizer representations of the problem.

2. When Mosel is running in restricted mode (see Section 1.3.3), the restriction NoExec disables this routine.

Module

mmxprs
**copyso1toinit**

**Purpose**
Copy solution values to initial values of an NL problem.

**Synopsis**
procedure copyso1toinit(ms:mpsol)

**Argument**
ms A solution object

**Further information**
1. This procedure copies the solution values of decision variables from the provided solution ms to their initial values for the next run. Doing so it overrides any previously set initial values for the involved variables. However, the settings for decision variables that are not included in the solution ms remain unchanged.

2. This operation can only be performed on a non-linear problem described using the module mmnl.

**Related topics**
copyso1toinit, clearinitvals, setinitval.

**Module**
mmxprs
defdelayedrows

**Purpose**
Define the set of constraints to be treated as delayed rows.

**Synopsis**
procedure defdelayedrows(cset:set of linctr)

**Argument**
cset Set of constraints to load or {} to reset a previous setting

**Further information**
This procedure stores a reference to the provided set that is used when the problem is loaded into the optimizer. This set can be modified after the call to this procedure: the optimizer will use the current content of the set at the time of loading the problem.

**Module**
mmxprs
defsecurevecs

Purpose
Define the sets of variables and constraints that must not be removed by presolve.

Synopsis
procedure defsecurevecs(vset:set of mpvar,cset:set of linctr)

Arguments
vset Set of decision variables to preserve or {} to reset a previous setting
 cset Set of constraints to preserve or {} to reset a previous setting

Further information
This procedure stores references to the provided sets that are used when the problem is loaded into the optimizer. These sets can be modified after the call to this procedure: the optimizer will use the current content of the sets at the time of loading the problem.

Module
mmxprs
estimatemarginals

**Purpose**
Estimate better marginal values for variables and constraints for degenerate problems.

**Synopsis**
procedure estimatemarginals(sbvars:array(vars: set of mpvar) of real)
procedure estimatemarginals(dualslb:array(constriants: set of linctr) of real, dualsub:array(constriants: set of linctr) of real)
procedure estimatemarginals(sbvars:array(vars: set of mpvar) of real, efforlimit:integer, delta:real)
procedure estimatemarginals(dualslb:array(constriants: set of linctr) of real, dualsub:array(constriants: set of linctr) of real, effortlimit:integer)

**Arguments**
sbvars An array of reals that will be populated with the approximations for the marginal values. The approximation is carried out for the variables included in the variables set.
dualslb An array of reals that will be populated with the approximations for the lower bounds for the row marginal values. The approximation is carried out for the constraints in the constraints set.
dualsub An array of reals that will be populated with the approximations for the upper bounds for the row marginal values. The approximation is carried out for the constraints in the constraints set.
efforlimit Effort limit spent to approximate the effect of the move of a variable, expressed as an upper limit of simplex iterations per variable.
delta The size of the perturbation applied to force a movement in the variable.

**Further information**
1. This procedure can be used to estimate the marginal values of variables in degenerate problems. In degenerate problems, the reduced costs and row duals do not always provide a good representation of the effect on the objective when forcing a move in a variable. Also, in degenerate problems, the reduced costs and row duals may depend on the final basis found, and multiple correct alternatives might exists. This function attempts to identify better marginal values by simulating a move in the variables.

2. Prior to calling estimatemarginals, the current LP problem must have been solved to optimality and an optimal basis must be available.

3. It is important to note that the procedure provides an estimate only.

4. This procedure relies on the XPRSstrongbranch and XPRSestimaterowdualranges functions, refer to the Xpress Optimizer Reference Manual for more information.

**Module**
mmxprs
**fixglobal**

**Purpose**
Fix values of global entities according to the current solution.

**Synopsis**
procedure fixglobal
procedure fixglobal(ifrnd:boolean)

**Argument**
ifrnd  if true, integer solution values are rounded

**Example**
Solve the MIP problem, reload the problem after solving, fix global entities to their solution values, and finally solve the LP for the continuous variables in order to be able to use getrange.

```plaintext
minimize(obj)
fixglobal
minimize(XPRS_LIN, obj)
writeln(getrange(XPRS_UPACT,x))
```

**Further information**
1. This procedure fixes the non-continuous variables to their value of the current solution. A call to this function is required when performing sensitivity analysis on MIP problems using getrange.
2. The first form of the procedure corresponds to fixglobal(false).

**Related topics**
getrange.

**Module**
mmxprs
getbstat

Purpose
Get the status of a variable or constraint in a basis.

Synopsis
function getbstat(b:basis,v:mpvar):integer
function getbstat(b:basis,c:linctr):integer

Arguments
b  A basis
v  A decision variable
c  A linear constraint

Return value
Basis status. For a variable:
-1  Variable is not in the basis
0   Variable is non-basic at lower bound, or superbasic at zero if the variable has no lower bound
1   Variable is basic
2   Variable is non-basic at upper bound
3   Variable is super-basic

For a constraint:
-1  Constraint is not in the basis
0   Slack, surplus or artificial is non-basic at lower bound
1   Slack, surplus or artificial is basic
2   Slack or surplus is non-basic at upper bound
3   Slack or surplus is super-basic

Related topics
savebasis, setbstat, resetbasis.

Module
mmxprs
getdualray

Purpose
Get a dual ray for an infeasible problem.

Synopsis
function getdualray(ray:array(set of linctr) of real):boolean

Argument
ray An array of reals over all constraints in the problem (as loaded) in which the dual ray is returned.

Return value
This procedure returns the dual ray found for the problem if the problem is found to be dual unbounded (thus primal infeasible) and one is available.

Further information
1. The return value of the function is true if a dual ray is available, and false otherwise.
2. The dimension and base set of the ray argument will be set up by the function.

Example

declarations
all_constraints : set of linctr
dual_ray : array(all_constraints) of real
end-declarations
if getprobstat <> XPRS_INF then
  writeln("Problem not infeasible.")
else
  HasRay := getdualray(dual_ray)
  if HasRay then
    writeln("Dual ray:")
    forall (c in all_constraints)
      writeln(getname(c), " ", dual_ray(c))
  else
    writeln("No dual ray was found")
end-if
end-if

Related topics
getprimalray

Module
mmxprs
getiis

Purpose
Compute then get the Irreductible Infeasible Sets (IIS).

Synopsis
procedure getiis(vset:set of mpvar,cset:set of linctr)
procedure getiis(numiis:integer,vset:set of mpvar,cset:set of linctr)
procedure getiis(numiis:integer,ctrtype:array(linctr) of integer)
procedure getiis(numiis:integer,duals:array(linctr) of real)
procedure getiis(numiis:integer,isolrow:array(linctr) of boolean)
procedure getiis(numiis:integer,bndtype:array(mpvar) of integer)
procedure getiis(numiis:integer,rdcs:array(mpvar) of real)
procedure getiis(numiis:integer,isolcol:array(mpvar) of boolean)

Arguments
vset Set to return the decision variables of the IIS or {} if not required

cset Set to return the constraints of the IIS or {} if not required

numiis Ordinal number of the IIS

ctrtype Array to return the sense or type of rows in the IIS (XPRS_IIS_LEQ, XPRS_IIS_GEQ,
XPRS_IIS_EQ, XPRS_IIS_SOS1, XPRS_IIS_SOS2 or XPRS_IIS_INDIC)

duals Array to return the dual multipliers associated with the rows of the IIS

isolrow Array to return the isolation status of the the rows of the IIS

bndtype Array to return the encoded sense and type of bounds in the IIS

rdcs Array to return the dual multipliers associated with the bounds of the IIS

isolcol Array to return the isolation status of the the bounds of the IIS

Further information
1. This procedure computes the IIS and stores the result in the provided parameters. The first form
of the routine (numiis not specified) computes all IIS and returns the last set found.

2. The bndtype values have to be decoded using getiissense and getiistype. The first routine may
return XPRS_IIS_LEQ (upper bound), XPRS_IIS_GEQ (lower bound), XPRS_IIS_RNG (lower and upper
bound) or XPRS_IIS_EQ (fixed bound). The second one may give XPRS_IIS_BIN (binary),
XPRS_IIS_INT (integer), XPRS_IIS_PINT (partial integer), XPRS_IIS_SEC (semi continuous) or
XPRS_IIS_SINT (semi continuous integer).

3. The sets passed to this procedure are reset before being used.

Related topics
resetiis, isiisvalid, getinfeas.

Module
mmxprs
**getiissense**

**Purpose**
Decode the sense part of an IIS bound type information.

**Synopsis**
function getiissense(bndtype):integer

**Argument**
bndtype A bound type as returned by getiis

**Return value**
Sense part of an IIS bound type.

**Related topics**
getiis, getiistype.

**Module**
mmxprs
getiistype

**Purpose**
Decode the type part of an IIS bound type information.

**Synopsis**
function getiistype(i:bndtype):integer

**Argument**
bndtype A bound type as returned by getiis

**Return value**
Type part of an IIS bound type.

**Related topics**
getiis, getiissense.

**Module**
mmxprs
getinfcause

**Purpose**

Returns the variable or constraint causing infeasibility.

**Synopsis**

procedure getinfcause(vars:set of mpvar,ctrs:set of linctr)

**Arguments**

vars Set to return the infeasible variable or {} if not required
ctrs Set to return infeasible constraint or {} if not required

**Further information**

1. This function can be used to get the variable or constraint responsible for an infeasibility detected either during matrix generation (invalid bound) or when presolving the problem.
2. The sets passed to this procedure are reset before being used.

**Related topics**

getinfeas.

**Module**

mmxprs
getinfeas

**Purpose**
Returns sets of infeasible primal and dual variables.

**Synopsis**
procedure getinfeas(mx:set of mpvar,mslack:set of linctr,mdual:set of linctr,mdj:set of mpvar)

**Arguments**
- **mx**
  Set to return the infeasible variables or {} if not required
- **mslack**
  Set to return infeasible constraints or {} if not required
- **mdual**
  Set to return dual infeasible constraints or {} if not required
- **mdj**
  Set to return the dual infeasible variables or {} if not required

**Related topics**
getiis.

**Module**
mmxprs
getlb

Purpose
Get the lower bound of a variable.

Synopsis
function getlb(x:mpvar):real

Argument
x A decision variable

Return value
Lower bound of the variable.

Further information
This function returns the lower bound of a variable that is currently held by the Optimizer. The bound value may be changed directly in the Optimizer using setlb. Changes to the variable in Mosel are not taken into account by this function unless the problem has been reloaded since (procedure loadprob).

Related topics
getub, setlb, setub.

Module
mmxprs
getloadedlinctrs

Purpose
Get the linear constraints loaded into the optimiser.

Synopsis
procedure getloadedlinctrs(sc:set of linctr)

Argument
sc         A set of linear constraints

Further information
The result of the operation is added to the current content of the provided set (i.e. the set is not cleared).

Related topics
getloadedmpvars

Module
mmxprs
getloadedmpvars

Purpose
Get the decision variables loaded into the optimiser.

Synopsis
procedure getloadedmpvars(sv:set of mpvar)

Argument
sv A set of decision variables

Further information
The result of the operation is added to the current content of the provided set (i.e. the set is not cleared).

Related topics
getloadedlinctrs

Module
mmxprs
getname

Purpose
Get the name of a decision variable or constraint of the problem.

Synopsis
function getname(x:mpvar):string
function getname(c:linctr):string
function getname(nl:nlctr):string

Arguments
x  A decision variable used in the problem
 c  A constraint (or SOS) of the problem
  nl A non linear constraint of the problem

Return value
Name of the given object.

Further information
1. This function returns the name of a decision variable or constraint of the problem that would be used for matrix exportation. The parameter of this function must be part of the problem — for instance a hidden constraint cannot be assigned a name.
2. This function requires that the matrix has been generated (e.g. by a call to exportprob or loadprob). When used with a non linear constraint it is further required for the problem to be loaded into the optimiser and the parameter XPRS_loadnames must be true.

Module
mmxprs
getprimalray

Purpose
Get a primal ray for an unbounded problem.

Synopsis
function getprimalray(ray:array(set of mpvar) of real):boolean

Argument
ray An array of reals over all constraints in the problem (as loaded) in which the primal ray is returned.

Return value
This procedure returns the primal ray found for the problem if the problem is found to be primal unbounded (thus dual infeasible) and one is available.

Further information
1. The return value of the function is true if a primal ray is available, and false otherwise.
2. The dimension and base set of the ray argument will be set up by the function.

Example

declarations
all_variables : set of mpvar
primal_ray : array(all_variables) of real
end-declarations
if getprobstat <> XPRS_UNB then
  writeln("Problem is not unbounded.")
else
  HasRay := getprimalray(primal_ray)
  if HasRay then
    writeln("Primal ray:")
    forall (c in all_variables)
      writeln(getname(c), " ", primalray(c))
  else
    writeln("No primal ray was found")
  end-if
end-if

Related topics
getdualray

Module
mmxprs
getprobstat

**Purpose**
Get the Optimizer problem status.

**Synopsis**
```
function getprobstat:integer
```

**Return value**
Status of the problem currently held in the Optimizer:
- **XPRS_OPT**: Solved to optimality
- **XPRS_UNF**: Unfinished
- **XPRS_INF**: Infeasible
- **XPRS_UNB**: Unbounded
- **XPRS_OTH**: Unsolved or objective worse than cutoff

**Example**
The following procedure displays the current problem status:
```
procedure print_status
declarations
  status: string
end-declarations

  case getprobstat of
    XPRS_OPT: status:="Optimum found"
    XPRS_UNF: status:="Unfinished"
    XPRS_INF: status:="Infeasible"
    XPRS_UNB: status:="Unbounded"
    XPRS_OTH: status:="Failed"
    else status:="???
  end-case

  writeln("Problem status: ", status)
end-procedure
```

**Further information**
More detailed information than what is provided by this function can be obtained with function `getparam`, retrieving the problem attributes **XPRS_presolvestate**, **XPRS_lpsstatus**, and **XPRS_mipstatus** (see the *Xpress Optimizer Reference Manual*).

**Related topics**
- `getparam`

**Module**
`mmxprs`
getrange

Purpose
Get a range value for a variable or constraint.

Synopsis
function getrange(w:integer, x:mpvar):real
function getrange(w:integer, c:linctr):real

Arguments
  w    Which information to return. Possible values:
        XPRS_UPACT  Upper activity
        XPRS_LOACT  Lower activity
        XPRS_UUP    Upper unit cost
        XPRS_UDN    Lower unit cost
        XPRS_UCOST  Upper cost (variable only)
        XPRS_LCOST  Lower cost (variable only)
  x    A variable of the problem
  c    A constraint of the problem

Return value
Range information depending on the value of w.

Further information
This function returns ranging information to be used for sensitivity analysis after the problem has been optimized. On MIP problems, global entities have to be “fixed” using the procedure fixglobal before this function can be called.

Related topics
fixglobal.

Module
mmxprs
**getsensrng**

**Purpose**
Get sensitivity ranges for objective or RHS function coefficients.

**Synopsis**
```
function getsensrng(w:integer, x:mpvar):real
function getsensrng(w:integer, c:linctr):real
```

**Arguments**
- `w` Which information to return. Possible values:
  - XPRS_UP Upper sensitivity range
  - XPRS_DN Lower sensitivity range
- `x` A variable of the problem
- `c` A constraint of the problem

**Return value**
Sensitivity range information depending on the value of `w`.

**Further information**
This function returns sensitivity ranges for RHS function coefficients (if used with a constraint) and for objective function coefficients (if used with a variable). `getsensrng` can be called only if an optimal LP solution is available and the problem is not MIP presolved.

**Module**
```
mmxprs
```
**Purpose**
Get the size of a solution.

**Synopsis**
```
function getsize(ms:mpsol):integer
```

**Argument**
- `ms` A solution object

**Return value**
The number of variables stored in the solution.

**Related topics**
- `getvars`.

**Module**
`mmxprs`
**Purpose**
Get the solution value of an expression from a solution object.

**Synopsis**
function getsol(ms:mpsol,v:mpvar):real
function getsol(ms:mpsol,c:linctr):real
function getsol(ms:mpsol,nl:nlctr):real

**Arguments**
- **ms** A solution object
- **v** A decision variable
- **c** A linear constraint
- **nl** A non linear constraint

**Return value**
Solution value or 0.

**Further information**
This function returns an evaluation of an expression using the provided solution object as solution values for the decision variables.

**Related topics**
setsol, savesol, savemipsol.

**Module**
mmxprs
**getub**

**Purpose**
Get the upper bound of a variable.

**Synopsis**
```plaintext
function getub(x:mpvar):real
```

**Argument**
- `x` A decision variable

**Return value**
Upper bound of the variable.

**Further information**
The bound value may be changed directly in the optimizer using `setub`. Changes to the variable in Mosel are not taken into account by this function unless the problem has been reloaded since (procedure `loadprob`).

**Related topics**
- `getlb`, `setlb`, `setub`.

**Module**
- `mmxprs`
getvars

**Purpose**
Get the set of variables of a solution.

**Synopsis**

```plaintext
procedure getvars(ms:mpsol,s:set of mpvar)
```

**Arguments**

- `ms` A solution object
- `s` A set of decision variables

**Further information**
This procedure returns in the parameter `s` the set of variables used by a solution object. Note that this procedure replaces the content of the set.

**Related topics**

- `getsize`

**Module**

`mmxprs`
hasfeature

**Purpose**
Check if a specific feature is supported by the currently used license.

**Synopsis**

```typescript
function hasfeature(feature: string): boolean
```

**Argument**

- **feature**
  The name of the feature to check, as it would appear in the Xpress license file.

**Return value**

- **true** if the requested feature is supported, **false** otherwise.

**Module**

`mmxprs`
**implies**

**Purpose**
Create an *implies* expression.

**Synopsis**

```plaintext
function implies(c1:log_or_linctr,c2:log_or_linctr):logctr
```

**Arguments**
- `c1`  A linear constraint (linctr) or logical expression (logctr)
- `c2`  A linear constraint (linctr) or logical expression (logctr)

**Return value**
A new logctr representing the expression.

**Example**
The following example shows several ways of stating the logical relation ‘if $x_1 \geq 10$ then $x_1 + x_2 \geq 12$ and not $x_2 \leq 5$’. The implied constraint $L$ is itself a logical constraint, built up by using the operators and and not in combination with linear constraints.

```plaintext
declarations
R=1..2
C: array(range) of linctr ! Linear constraints
L: logctr ! Logical constraint
x: array(R) of mpvar ! Decision variables
end-declarations

C(1):= x(1)>=10 ! Define (temporary) linear ctrs
C(2):= x(2)<=5
C(3):= x(1)+x(2)>=12

implies(C(1), C(3) and not C(2)) ! State the implication
forall(j in 1..3) C(j):=0 ! Delete the auxiliary ctrs

! The same implication constraint can be stated by:
implies(x(1)>=10, x(1)+x(2)>=12 and not x(2)<=5)

! Or also by:
L:= x(1)+x(2)>=12 and not x(2)<=5 ! Define (temporary) logical ctr
implies(x(1)>=10, L) ! State the implication
L:= 0 ! Delete the auxiliary ctr
```

**Further information**

1. This function creates a logctr constraint representing an *implies* condition: *if c1 is valid then c2 is enforced.*
2. The helper package ‘advmod’ must be loaded if this function is used:
   ```plaintext
   uses 'advmod'
   ```

**Related topics**
- indicator, xor

**Module**
- mmxprs
indicator

**Purpose**
Create an *indicator constraint*.

**Synopsis**
function indicator(type:integer,y:mpvar,ctr:linctr):logctr

**Arguments**
- **type** The indicator type:
  - -1 for indicator \( y=0 \) -> \( ctr \)
  - 1 for indicator \( y=1 \) -> \( ctr \)
- **y** The variable associated to the constraint
- **ctr** A linear inequality constraint

**Return value**
A new logctr representing the indicator.

**Example**
This example shows how to define two indicator constraints. The second constraint labeled \( L \) is stated with the help of an auxiliary linear constraint definition. This temporary constraint \( C \) needs to be deleted from the problem after having been used in the definition of the indicator constraint. The notation \( b(1)=1 \) \( \rightarrow \ldots \) should be read as 'if \( b(1) \) takes the value 1 then \( \ldots \) must hold'.

```plaintext
declarations
R=1..2, S=1..3
C: linctr ! Linear constraint
L: logctr ! Logical (indicator) constraint
x: array(S) of mpvar ! Decision variables
b: array(R) of mpvar ! Indicator variables
end-declarations

forall(i in R)
  b(i) is_binary ! Indicator variables must be binaries

C:= x(2)+x(3)<=5 ! Constraint to transform into indicator ctr.

! Define 2 indicator constraints
indicator(1, b(1), x(1)+x(2)>=12) ! b(1)=1 -> x(1)+x(2)>=12
L:= indicator(-1, b(2), C) ! b(2)=0 -> x(2)+x(3)<=5

C:=0 ! Delete the auxiliary constraint definition
```

**Related topics**
implies, xor

**Module**
mmxprs
isiisvalid

**Purpose**
Check whether an IIS number exists.

**Synopsis**
function isiisvalid(numiis:integer):boolean

**Argument**
umiis  Ordinal number of the IIS

**Return value**
true if numiis corresponds to an existing IIS.

**Related topics**
resetiis, getiis.

**Module**
mmxprs
**isintegral**

**Purpose**
Check whether a variable (or set of variables) solution value is integral.

**Synopsis**
```
function isintegral(x:mpvar):boolean
function isintegral(s:set of mpvar):boolean
```

**Arguments**
- `x` A decision variable
- `s` A set of decision variables

**Return value**
- `true` if the variable (or all variables of the set) is integral.

**Further information**
This function checks whether the current solution value of a variable is integral with respect to the tolerance value of the optimizer (`XPRS_MIPSOL`). When used with a set, the function returns `true` if all variables of the set satisfy the condition.

**Module**
`mmxprs`
loadbasis

Purpose
Load a previously saved basis.

Synopsis
procedure loadbasis(b:basis)

Argument
b A basis

Example
The following saves a basis, changes the problem, and then loads it into the Optimizer, reloading the old basis:

declarations
MinCost:linctr
mybasis:basis
end-declarations

savebasis(mybasis)
...
loadprob(MinCost)
loadbasis(mybasis)

Further information
1. This procedure loads a basis into the optimizer that has previously been saved using procedure savebasis or constructed using setbstat.

2. The problem must be loaded in the Optimizer for loadbasis to have any effect. If this has not recently been carried out using maximize or minimize it must be explicitly loaded using loadprob.

Related topics
loadprob, savebasis, setbstat, getbstat, resetbasis.

Module
mmxprs
loadmipsol

Purpose
Load a MIP solution into the optimizer.

Synopsis
function loadmipsol(s:array(set of mpvar) of real):integer
function loadmipsol(solnum:integer):integer
function loadmipsol(ms:mpsol):integer

Arguments
   s    An array containing the solution
   solnum Solution number (between 1 and \texttt{XPRS_enumsols})
   ms   A solution object

Return value
Operation status:
-1  Solution rejected because an error occurred
  0  Solution accepted
  1  Solution rejected because it is infeasible
  2  Solution rejected because it is cut off
  3  Solution rejected because the LP reoptimization was interrupted

Example
The following saves a MIP solution, modifies the problem, and then loads it into the Optimizer, reloading the MIP solution:

declarations
   MinCost:linctr
   mysol: array(set of mpvar) of real
   result: integer
end-declarations

savemipsol(mysol)
...
   ! Make some changes
loadprob(MinCost)
result:= loadmipsol(mysol)
if result<>0 then writeln("Loading MIP solution failed"); end-if
minimize(MinCost)
Further information

1. This function loads a MIP solution into the optimizer that has previously been saved using procedure `savemipsol` or constructed by some external heuristic. In the latter case a value needs to be assigned to each discrete variable in the problem, such as `mysol(x) := 1` (where `x` is a decision variable of type `mpvar`).

2. The values for the continuous variables in the `s` array are ignored and are calculated by fixing the integer variables and reoptimizing.

3. The second form of the routine can be called after a search for n-best solutions has been performed by the optimiser: the selected solution is used as input.

4. The current problem definition must be loaded into the Optimizer for `loadmipsol` to have any effect. If this has not recently been done, e.g., by calling `maximize` or `minimize`, the problem must be explicitly loaded using `loadprob`.

5. If the MIP solution is accepted by the Optimizer it causes the `MIPABSCUTOFF` control to be set accordingly. The provided MIP solution may help guiding the MIP heuristics but the branch-and-bound search will start from the initial LP relaxation solution as usual.

Related topics
- `savemipsol`, `addmipsol`.

Module
- `mmxprs`
loadprob

**Purpose**
Load a problem into the optimizer.

**Synopsis**
```plaintext
procedure loadprob(obj:linctr)
procedure loadprob(force:boolean, obj:linctr)
procedure loadprob(obj:linctr, extravar:set of mpvar)
procedure loadprob(force:boolean, obj:linctr, extravar:set of mpvar)
procedure loadprob(qobj:qexp)
procedure loadprob(qobj:qexp, extravar:set of mpvar)
procedure loadprob(nlobj:nlct)
procedure loadprob(nlobj:nlct, extravar:set of mpvar)
procedure loadprob(rbobj:robustct)
procedure loadprob(rbobj:robustct, extravar:set of mpvar)
```

**Arguments**
- **obj**  
  Objective function constraint
- **qobj**  
  Quadratic objective function (with module **mmquad**)
- **nlobj**  
  Non linear objective function (with module **mmnl**)
- **rbobj**  
  Robust objective function (with module **mmrobust**)
- **force**  
  Load the matrix even if not required
- **extravar**  
  Extra variables to include

**Further information**
1. This procedure explicitly loads a problem into the optimizer. It gets called automatically by the optimization procedures **minimize** and **maximize** if the problem has been modified in Mosel since the last call to the optimizer. Nevertheless in some cases, namely before loading a basis, it may be necessary to reload the problem explicitly using this procedure. If the problem has not been modified since the last call to **loadprob**, the problem is not reloaded into the optimizer. The parameter **force** can be used to force a reload of the problem in such a case. The parameter **extravar** is a set of variables to be included into the problem even if they do not appear in any constraint (i.e. they become empty columns in the matrix).

2. Support for quadratic programming requires the module **mmnl**.

3. Support for general nonlinear programming requires the module **mmxnlp**.

4. Support for robust programming requires the module **mmrobust**.

**Related topics**
- **minimize**, **maximize**.

**Module**
- **mmxprs**
maximize, minimize

Purpose
Maximize/minimize the current problem.

Synopsis
procedure maximize(alg:integer, obj:linctr)
procedure maximize(obj:linctr)
procedure maximize(alg:integer, qobj:qexp)
procedure maximize(qobj:qexp)
procedure maximize(alg:integer, nlobl:nlctr)
procedure maximize(nlobl:nlctr)
procedure maximize(rbobj:robustctr)
procedure maximize(alg:integer, rbobj:robustctr)

Arguments
alg    Algorithm choice:
        XPRS_BAR     Newton-Barrier to solve LP
        XPRS_DUAL    Dual simplex
        XPRS_NET     Network solver
        XPRS_LIN     Only solve LP ignoring all global entities
        XPRS_PRI     Primal simplex
        XPRS_ENUM    Start a search for the n-best MIP solutions
        XPRS_LPSTOP  Stop the MIP solution process after solving the first LP
        XPRS_CONT    Continue a previously interrupted solution process
        XPRS_LOCAL   Solve the linearization of the problem (mmxnlp only)
        XPRS_CORELP  Solve the linear part of the problem (mmxnlp only)
        XPRS_TUNE    Enable the tuner

obj    Objective function constraint
qobj   Quadratic objective function (with module mmquad)
nlobj  Non linear objective function (with module mmnl)
rbobj  Robust objective function (with module mmrobust)

Example
The following maximizes Profit using the dual simplex algorithm and stops before the global search:

declarations
   Profit:linctr
end-declarations

maximize(XPRS_DUAL+XPRS_LPSTOP, Profit)

The following minimizes MinCost using the Newton-Barrier algorithm and ignoring all global entities

declarations
   MinCost:linctr
end-declarations

minimize(XPRS_BAR+XPRS_LIN, MinCost)
Further information

1. This procedure calls the Optimizer to maximize/minimize the current problem (excluding all
hidden constraints) using the given constraint as objective function. Optionally, the algorithm to
be used can be defined. By default, the global search is executed automatically if the problem
contains any global entities. Where appropriate, several algorithm choice parameters may be
combined (using plus signs).

2. If XPRS_LIN is specified, then the discreteness of all global entities is ignored, even during the
presolve procedure.

3. If XPRS_LPSTOP is specified, then just the LP at the top node is solved and no Branch-and-Bound
search is initiated. But the discreteness of the global entities is taken into account in presolving
the LP at the top node. Note also that getprobstat still returns information related to the MIP
problem when this option is used although only an LP solve has been executed and the solution
information returned by getsol corresponds to the current LP solution. However, if the the MIP is
solved to optimality during this call, the MIP optimal solution will be returned by getsol.

4. If XPRS_CONT is used after a solve has completed, the routine returns immediately without altering
the current problem status.

5. If XPRS_ENUM is specified, the optimiser starts a search for the n-best MIP solutions. The maximum
number of solutions to store may be specified using the XPRS_enummaxsol (default: 10). After the
execution of the enumeration, the number of solutions found during the search is returned by
the control parameter XPRS_enumsols. The procedure selectsol can then be used to select one of
these solutions.

6. If XNLP_LOCAL is specified for a non-linear problem having been loaded using mmxnlp and which
have been solved using XSLP, then the current linearization will be reoptimized.

7. If XPRS_TUNE is specified the problem will be tuned and then solved with the best control settings
identified by the tuner. For a user guide about the tuner, please refer to the documentation of
the Xpress Optimizer.

8. If XNLP_CORELP is specified for a non-linear problem having been loaded using mmxnlp, then only
the linear part of the problem will be loaded and optimized. This is useful for checking if the
linear part of the problem is well posed.

9. Support for quadratic programming requires the module mmnl.

10. Support for general nonlinear programming requires the module mmxnlp.

11. Support for robust programming requires the module mmrobust.

Related topics
   postsolve, loadprob, selectsol.

Module
   mmxprs
postsolve

**Purpose**
Postsolve the current matrix.

**Synopsis**
procedure postsolve

**Further information**
After an optimisation operation has been interrupted before its completion, the matrix held into the optimiser remains in a *presolved* state. In this state direct matrix operations (like fixing bounds) cannot be applied: this routine restores the problem in its original state that is just after it was loaded into the optimiser. As an alternative to postsolving the matrix, the problem may be entirely reloaded using `loadprob`.

**Related topics**
maximize, minimize.

**Module**
mmxprs
readbasis

Purpose
Read a basis from a file.

Synopsis
procedure readbasis(fname:string,options:string)

Arguments
fname       Extended file name
options     String of options

Further information
This procedure reads in a basis from a file by calling the function XPRSreadbasis of the Optimizer. Note that basis save/read procedures can be used only if the constraint and variable names have been loaded into the Optimizer (control parameter XPRS_loadnames set to true) and all constraints are named. For more detail on the options and behavior of this procedure refer to the Xpress Optimizer Reference Manual.

Related topics
writebasis.

Module
mmxprs
readdirs

**Purpose**
Read directives from a file.

**Synopsis**
procedure readdirs(fname:string)

**Argument**
fname     Extended file name

**Further information**
This procedure reads in directives from a file by calling the function XPRSreaddirs of the Optimizer. Note that directives save/read procedures can be used only if variable names have been loaded into the Optimizer (parameter XPRS_loadnames set to true).

**Related topics**
writedirs.

**Module**
mmxprs
readsol

Purpose
Read a solution from a file.

Synopsis
procedure readsol(fname:string,options:string)

Arguments
fname    Extended file name
options  String of options

Further information
This procedure reads in a solution from a file by calling the function XPRSreadslxsol of the Optimizer. Note that solution save/read procedures can be used only if the constraint and variable names have been loaded into the Optimizer (control parameter XPRS_loadnames set to true) and all constraints are named. For more detail on the options and behavior of this procedure refer to the Xpress Optimizer Reference Manual.

Related topics
writesol.

Module
mmxprs
refinemipsol

Purpose
Executes the MIP solution refiner on an mpsol solution.

Synopsis
function refinemipsol(solution:mpsol):mpsol
function refinemipsol(solution:mpsol, options:integer):mpsol

Arguments
solution  The solution to be refined
options   Options passed to the solution refiner. Please refer to XPRSrefinemipsol for the available options

Related topics
calcsolinfo.

Module
mmxprs
rejectintsol

**Purpose**
Reject the solution provided to the PREINTSOL callback.

**Synopsis**
procedure rejectintsol

**Further information**
This procedure cannot be called from outside of the PREINTSOL callback.

**Related topics**
setcallback.

**Module**
mmxprs
repairinfeas

**Purpose**
Relaxing bounds to repair infeasibility.

**Synopsis**

```plaintext
procedure repairinfeas(alrp:array(linctr) of real, agrp:array(linctr) of real, 
albp:array(mpvar) of real, aubp:array(mpvar) of real)
procedure repairinfeas(alrp:array(linctr) of real, agrp:array(linctr) of real, 
albp:array(mpvar) of real, aubp:array(mpvar) of real, phs2:string, 
delta:real,optfg:string)
procedure repairinfeas(flags:string, lrp:real, grp:real, lbp:real, ubp:real, 
delta:real)
procedure repairinfeas(flags:string)
procedure repairinfeas(alrp:array(linctr) of real, agrp:array(linctr) of real, 
albp:array(mpvar) of real, aubp:array(mpvar) of real,alrb:array(linctr) of 
real,agrb:array(linctr) of real,albb:array(mpvar) of real,aubb:array(mpvar) of 
real,phs2:string,delta:real,optfg:string)
```

**Arguments**

- **alrp** Array of preferences for relaxing the less or equal side of row
- **agrp** Array of preferences for relaxing the greater or equal side of row
- **albp** Array of preferences for relaxing lower bounds
- **aubp** Array of preferences for relaxing upper bounds
- **alrb** Array of upper bounds to be imposed on the amount of relaxation allowed for the less or equal side of row
- **agrb** Array of upper bounds to be imposed on the amount of relaxation allowed for the greater or equal side of row
- **albb** Array of upper bounds to be imposed on the amount of relaxation allowed for lower bounds
- **aubb** Array of upper bounds to be imposed on the amount of relaxation allowed for upper bounds
- **phs2** A 1-character string controlling the second phase optimization
- **lrp** Preference for relaxing the less or equal side of row
- **grp** Preference for relaxing the greater or equal side of row
- **lbp** Preference for relaxing lower bounds
- **ubp** Preference for relaxing upper bounds
- **delta** Relaxation multiplier for the second phase-1
- **flags** A 3-character string defining the p/o/g flags
- **optfg** Flags to be passed to the optimizer
Further information

1. This routine is an interface to the Optimizer functions XPRSSrepairweightedinfeas and XPRSSrepairinfeas. Please refer to the Xpress Optimizer Reference Manual for further details.

2. The 2 first forms call the Optimizer routine XPRSSrepairweightedinfeas. Missing preferences are treated as 0; the default value for phs2 is "d" and the default value for delta is 0.001.

3. The third and fourth forms call the Optimizer routine XPRSSrepairinfeas. If flags is not specified (empty string), a default value of "cog" is used. If preferences and delta are not given, all preferences are set to 1 and delta is 0.001.

4. The last form calls the Optimizer routine XPRSSrepairweightedinfeasbounds, allowing to bound the amount of relaxation applied on a per row or bound basis. Only positive bounds are applied; a zero or negative bound is ignored and the amount of relaxation allowed for the corresponding row or bound is not limited. The effect of a zero bound on a row or bound would be equivalent with not relaxing it, and can be achieved by setting its preference array value to zero instead, or not including it in the preference arrays. The default value for phs2 is "d".

5. Negative preferences translate to quadratic penalties applied for the corresponding rows or bounds.

Module

mmxprs
resetbasis

Purpose
Reset a basis.

Synopsis
procedure resetbasis(b:basis)

Argument
b A basis

Further information
This function clears the information stored in a basis object.

Related topics
loadbasis, savebasis, setbstat, resetbasis.

Module
mmxprs
resetiis

Purpose
Reset the search for IIS.

Synopsis
procedure resetiis

Further information
This procedure resets the search for IIS and clears all information already computed related to IIS.

Related topics
getiis.

Module
mmxprs
**Purpose**
Reset a solution.

**Synopsis**
```
procedure resetsol(ms: mpsol)
```

**Argument**
- `ms` A solution object

**Further information**
This function clears the information stored in a solution object.

**Related topics**
- `setsol`, `savesol`, `savemipsol`, `getsize`.

**Module**
- `mmxprs`
savebasis

Purpose
Save the current basis.

Synopsis
procedure savebasis(b:basis)

Argument
b A basis

Further information
This function saves the current basis into the provided basis object.

Related topics
loadbasis, setbstat, getbstat, resetbasis.

Module
mmxprs
savemipsol

Purpose
Save the current solution into the provided array or solution object.

Synopsis
procedure savemipsol(s:array(set of mpvar) of real)
procedure savemipsol(ms:mpsol)

Arguments
s An array to return the solution
ms A solution object

Further information
1. This procedure saves the current solution into the provided array. The resulting datastructure may be used as input for the loadmipsol function.

2. If the index set of the array is dynamic, the procedure may extend it in order to have all variables of the problem. Otherwise the solution is saved only for the variables included in this set.

3. Only non-continuous variables are saved when this procedure is used with an mpsol argument. Use savesol to save the values of all variables.

Related topics
loadmipsol, savesol.

Module
mmxprs
savesol

Purpose
Save the current solution into a solution object.

Synopsis
procedure savesol(ms:mpsol)

Argument
ms A solution object

Further information
This procedure saves the current solution into the provided solution object. As opposed to the savemipsol routine all variables are saved independently of their type.

Related topics
savemipsol.

Module
mmxprs
savestate

**Purpose**
Save current state of the Optimizer to a file.

**Synopsis**
procedure savestate(fname:string)

**Argument**
fname  Extended file name

**Further information**
1. The produced file can then be used as input to Optimizer console using optimizer's command RESTORE.

2. When the mmxnlp model is used and a non-linear problem is loaded, savestate creates an extra file containing the non-linear information relevant to the problem; i.e. two files are created, with the default extensions of .svf for the information stored by mmxprs, and .svx for the non-linear counterpart.

**Module**
mmxprs
selectsol

**Purpose**
Select one of the solutions found by solution enumerator.

**Synopsis**
```
procedure selectsol(solnum:integer)
```

**Argument**
- **solnum**  
  Solution number (between 1 and \text{XPRS\_enumsol})

**Further information**
1. This routine can be called after a search for \textit{n-best} solutions has been performed by the optimizer in order to select a particular solution.
2. Once a solution has been selected, the functions \textit{getsol} (applied to decision variables) and \textit{getobjval} return values related to this solution.

**Module**
- \texttt{mmxprs}
**setarchconsistency**

**Purpose**
Sets the value of the optimizer architecture control.

**Synopsis**
```
procedure setarchconsistency(controlvalue:integer)
```

**Argument**
- `controlvalue` Value of the optimizer architecture control

**Further information**
Please refer to the *Xpress Optimizer Reference Manual* for more details.

**Module**
`mmxprs`
setbstat

Purpose
Set the status of a variable or constraint in a basis.

Synopsis
procedure setbstat(b:basis,v:mpvar,s:integer)
procedure setbstat(b:basis,c:linctr,s:integer)

Arguments
b  A basis
v  A decision variable
c  A linear constraint
s  Basis status. For a variable:
   -1  Remove the variable from the basis
   0  Variable is non-basic at lower bound, or superbasic at zero if the variable has no lower bound
   1  Variable is basic
   2  Variable is non-basic at upper bound
   3  Variable is super-basic
For a constraint:
   -1  Remove the constraint from the basis
   0  Slack, surplus or artificial is non-basic at lower bound
   1  Slack, surplus or artificial is basic
   2  Slack or surplus is non-basic at upper bound
   3  Slack or surplus is super-basic

Related topics
savebasis, getbstat, resetbasis.

Module
mmxprs
setcallback

Purpose
Set optimizer callback functions and procedures.

Synopsis
procedure setcallback(cbtype:integer, cb:string)

Arguments

<table>
<thead>
<tr>
<th>cbtype</th>
<th>Type of the callback:</th>
</tr>
</thead>
<tbody>
<tr>
<td>XPRS_CB_LPLOG</td>
<td>Simplex log callback</td>
</tr>
<tr>
<td>XPRS_CB_CUTLOG</td>
<td>Cut log callback</td>
</tr>
<tr>
<td>XPRS_CB_GLOBALLOG</td>
<td>Global log callback</td>
</tr>
<tr>
<td>XPRS_CB_BARLOG</td>
<td>Barrier log callback</td>
</tr>
<tr>
<td>XPRS_CB_CHGNODE</td>
<td>User select node callback</td>
</tr>
<tr>
<td>XPRS_CB_PRENODE</td>
<td>User preprocess node callback</td>
</tr>
<tr>
<td>XPRS_CB_OPTNODE</td>
<td>User optimal node callback</td>
</tr>
<tr>
<td>XPRS_CB_INFNODE</td>
<td>User infeasible node callback</td>
</tr>
<tr>
<td>XPRS_CB_INTSOL</td>
<td>User integer solution callback</td>
</tr>
<tr>
<td>XPRS_CB_NODECUTOFF</td>
<td>User cut-off node callback</td>
</tr>
<tr>
<td>XPRS_CB_NEWNODE</td>
<td>New node callback</td>
</tr>
<tr>
<td>XPRS_CB_BARITER</td>
<td>Barrier iteration callback</td>
</tr>
<tr>
<td>XPRS_CB_CUTMGR</td>
<td>Cut manager (branch-and-bound node) callback</td>
</tr>
<tr>
<td>XPRS_CB_CHGBRANCH</td>
<td>User choose branching variable callback</td>
</tr>
<tr>
<td>XPRS_CB_PREINTSOL</td>
<td>Integer solution callback called before acceptation</td>
</tr>
<tr>
<td>XPRS_CB_GAPNOTIFY</td>
<td>Gap notify callback</td>
</tr>
<tr>
<td>XPRS_CB_SOLNOTIFY</td>
<td>Integer notify callback called each time a solution added with <code>addmipsol</code> is processed</td>
</tr>
</tbody>
</table>

cb Name of the callback function/procedure; the parameters and the type of the return
value (if any) vary depending on the type of the callback:

- function cb:boolean
- function cb:boolean
- function cb:boolean
- function cb:boolean
- function cb(node:integer):integer
- function cb:boolean
- function cb:boolean
- procedure cb
- procedure cb
- procedure cb(node:integer)
- procedure cb(parent:integer,new:integer,branch:integer)
- function cb:integer
- function cb:boolean
- procedure cb(e:integer,u:integer,d:real)
- procedure cb(isheur:boolean,cutoff:real)
- procedure cb(rt:real,at:real,aot:real,abt:real)
- procedure cb(solid:string,status:integer)

**Example**

The following example defines a procedure to handle solution printing and sets it to be called whenever an integer solution is found using the integer solution callback:

```plaintext
public procedure printsol
declarations
  objval:real
end-declarations

objval := getparam("XPRS_lpobjval")
writeln("Solution value: ", objval)
end-procedure

setcallback(XPRS_CB_INTSOL, "printsol")
```
Further information

1. This procedure sets the optimizer callback functions and procedures. For a detailed description of these callbacks the user is referred to the Xpress Optimizer Reference Manual.

2. Passing an empty string ("") as the function name disables the corresponding callback.

3. The arguments of the Mosel subroutines implementing callback functions correspond to the arguments documented in the Xpress Optimizer Reference Manual, with the exception of arguments that are used for passing back information to the solver: these are replaced by the subroutine return values. For the logging callbacks, the return value true interrupts the solving. For the PRENODE and OPTNODE callbacks the return value true declares the current node to be infeasible. The return value of the BARITER callback is the selected barrier action (see XPRSaddcbbariteraction in the Xpress Optimizer Reference Manual for details). The cut manager routine is called repeatedly at each node until it returns false.

4. Whilst the solution values can be accessed from Mosel in any callback function/procedure, all other information such as the problem status or the value of the objective function must be obtained directly from the Optimizer using function getparam.

5. The function setucbdata can be used to return information to the optimizer from the callback ‘CHGBRANCH’.

6. The functions rejectintsol and setcbcutoff can be used to return information to the optimizer from the callback ‘PREINTSOL’.

7. The function setgndata can be used to return information to the optimizer from the callback ‘GAPNOTIFY’.

8. When the mmxnlp model is used, this function can also be used to set the callbacks relevant to non-linear problems only. Please see the documentation of the mmxnlp module for the list of extra callbacks.

Module

mmxprs
setcbcutoff

**Purpose**
Set the cutoff to be returned to the Optimizer by the PREINTSOL callback.

**Synopsis**
```
procedure setcbcutoff(cutoff:real)
```

**Argument**
cutoff   New cutoff value for the current solution

**Further information**
This procedure cannot be called from outside of the PREINTSOL callback.

**Related topics**
setcallback.

**Module**
mmxprs
setgndata

Purpose
Update data to be returned to the Optimizer by the GAPNOTIFY callback.

Synopsis
procedure setgndata(what:integer, target:real)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>what</td>
<td>What target to update. Possible values:</td>
</tr>
<tr>
<td></td>
<td>XPRS_GN_RELTARGET Relative gap</td>
</tr>
<tr>
<td></td>
<td>XPRS_GN_ABSTARGET Absolute gap</td>
</tr>
<tr>
<td></td>
<td>XPRS_GN_ABSOBJTARGET Absolute gap on objective</td>
</tr>
<tr>
<td></td>
<td>XPRS_GN_ASBBOUNDTARGET Absolute gap on bound</td>
</tr>
<tr>
<td>target</td>
<td>New target value</td>
</tr>
</tbody>
</table>

Further information
This procedure stores the provided information that will be returned to the optimizer when the callback terminates. This procedure cannot be called from outside of the GAPNOTIFY callback.

Related topics
setcallback.

Module
mmxprs
setlb

**Purpose**
Set the lower bound of a variable.

**Synopsis**
procedure setlb(x:mpvar,r:real)

**Arguments**
- x A decision variable
- r Lower bound value

**Further information**
This procedure changes the lower bound of a variable directly in the Optimizer, that is, the bound change is not recorded in the problem definition held in Mosel. Since this change is immediate, there is no need to reload the problem into the Optimizer (indeed, doing so resets the variable to the lower bound value computed by Mosel).

**Related topics**
getlb, getub, loadprob, setub.

**Module**
mmxprs
setmipdir

Purpose
Set a directive on a variable or Special Ordered Set.

Synopsis
procedure setmipdir(x:mpvar,t:integer,r:real)
procedure setmipdir(x:mpvar,t:integer)
procedure setmipdir(c:linctr,t:integer,r:real)
procedure setmipdir(c:linctr,t:integer)

Arguments
x A decision variable
c A linear constraint (of type SOS)
r A real value
t Directive type, which may be one of:
   XPRS_PR r is a priority (integer value between 1 and 1000 where 1 is the highest priority, 1000 the lowest)
   XPRS_UP Force up first
   XPRS_DN Force down first
   XPRS_PU r is an up pseudo cost
   XPRS_PD r is a down pseudo cost
   XPRS_BR Force branching even if satisfied

Further information
This procedure sets a directive on a global entity. Note that the priority value is converted into an integer. The directives are loaded into the Optimizer at the same time as the problem itself.

Related topics
clearmipdir, readdirs, writedirs.

Module
mmxprs
**setmodcut**

**Purpose**
Mark a constraint as a model cut.

**Synopsis**
procedure setmodcut(c:linctr)

**Argument**
c A linear constraint

**Further information**
This procedure marks the given constraint as a model cut. The list of model cuts is sent to the Optimizer when the matrix is loaded.

**Related topics**
clearmodcut.

**Module**
mmxprs
setsol

Purpose
Define the value associated to a decision variable in a solution object.

Synopsis
procedure setsol(ms:mpsol,v:mpvar,s:real)

Arguments
ms   A solution object
v    A decision variable
s    The solution value

Further information
This procedure associates a solution value to a decision variable in a solution object. If the variable is already included in the solution, its value is replaced. Otherwise the solution is extended with the new variable.

Related topics
getsol, savesol, savemipsol.

Module
mmxprs
setub

Purpose
Set the upper bound of a variable.

Synopsis
procedure setub(x:mpvar,r:real)

Arguments
x A decision variable
r Upper bound value

Further information
This procedure changes the upper(lower) bound of a variable directly in the Optimizer, that is, the bound is modified in the problem that is currently loaded in the Optimizer but does not get recorded in the problem definition held in Mosel. If the problem has not yet been loaded into the Optimizer then the new bound value is ignored. Reloading the problem into the Optimizer after a call to setub(setlb) will reset the upper (lower) bound for the variable to the value computed by Mosel, that is, the bound value resulting from setub(setlb) is overwritten.

Related topics
gelb, getub, loadprob, setlb.

Module
mmxprs
**setucbdata**

**Purpose**
Update data to be returned to the Optimizer by the CHGBRANCH callback.

**Synopsis**

```haskell
procedure setucbdata(x:mpvar, u:integer, e:real)
procedure setucbdata(s:linctr, u:integer, e:real)
procedure setucbdata(n:integer, u:integer, e:real)
```

**Arguments**

- **x** A decision variable
- **s** An SOS
- **n** A column or SOS number as provided by the optimizer
- **u** Direction for branching. Possible values:
  - 0 Upward branch made second (branch on column)
  - 1 Upward branch made first (branch on column)
  - 2 Upward branch made second (branch on SOS)
  - 3 Upward branch made first (branch on SOS)
- **e** Estimated degradation at the node

**Further information**
This procedure stores the provided information that will be returned to the optimizer when the callback terminates. This procedure cannot be called from outside of the CHGBRANCH callback.

**Related topics**

- `setcallback`

**Module**

`mmxprs`
**stopoptimize**

**Purpose**
Interrupt the optimizer algorithms.

**Synopsis**
procedure stopoptimize(why:integer)

**Argument**

<table>
<thead>
<tr>
<th>why</th>
<th>The reason for stopping. Possible reasons:</th>
</tr>
</thead>
<tbody>
<tr>
<td>XPRS_STOP_TIMELIMIT</td>
<td>Time limit hit</td>
</tr>
<tr>
<td>XPRS_STOP_CTRLC</td>
<td>Control C hit</td>
</tr>
<tr>
<td>XPRS_STOP_NODELIMIT</td>
<td>Node limit hit</td>
</tr>
<tr>
<td>XPRS_STOP_ITERLIMIT</td>
<td>Iteration limit hit</td>
</tr>
<tr>
<td>XPRS_STOP_MIPGAP</td>
<td>MIP gap is sufficiently small</td>
</tr>
<tr>
<td>XPRS_STOP_SOLLIMIT</td>
<td>Solution limit hit</td>
</tr>
<tr>
<td>XPRS_STOP_USER</td>
<td>User interrupt</td>
</tr>
</tbody>
</table>

**Further information**
This procedure can be called from any callback. It is ignored if used from outside an optimization process.

**Module**
mmxprs
unloadprob

Purpose
Unload the problem held in the optimizer.

Synopsis
procedure unloadprob

Further information
1. This procedure "unloads" the optimizer by releasing all the resources it has allocated for its processing (internal representation, solution information, working files).

2. This procedure resets the control parameters XPRS_EXTRACOLS, XPRS_EXTRAROWS, XPRS_EXTRAELEMENTS to their default values.

Related topics
maximize, minimize, loadprob.

Module
mmxprs
writebasis

**Purpose**
Write the current basis to a file.

**Synopsis**
procedure writebasis(fname:string,options:string)

**Arguments**
- **fname**  Extended file name
- **options**String of options

**Further information**
This procedure writes the current basis to a file by calling the Optimizer function XPRS_writebasis. Note that basis save/read procedures can be used only if the constraint and variable names have been loaded into the Optimizer (parameter XPRS_loadnames set to true) and all constraints are named. For more detail on the options and behavior of this procedure, refer to the Xpress Optimizer Reference Manual.

**Related topics**
readbasis.

**Module**
mmxprs
writedirs

**Purpose**
Write current directives to a file.

**Synopsis**
procedure writedirs(fname:string)

**Argument**
fname     Extended file name

**Further information**
This procedure writes the current directives to a file using the Optimizer file format.

**Related topics**
clearmipdir, setmipdir, reaaddirs.

**Module**
mmxprs
**writeprob**

**Purpose**
Write the current problem to a file.

**Synopsis**
procedure writeprob(fname:string, options:string)
procedure writeprob(fname:string, options:string, fnamed:string)

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fname</td>
<td>Extended file name for the matrix</td>
</tr>
<tr>
<td>options</td>
<td>String of format options (default: full precision)</td>
</tr>
<tr>
<td>fnamed</td>
<td>Extended file name for the directives</td>
</tr>
</tbody>
</table>

**Example**
Load the current problem into the Optimizer and save it to an MPS file "mypad.mps" in hexadecimal format ('x') and to the file "mypad.lp" in LP format ('l') with full precision ('p') using scrambled names ('s'):

```plaintext
loadprob(myobj)
writeprob("mypad.mps","x")
writeprob("mypad.lp","lps")
```

**Further information**
This procedure writes the current problem held in the Optimizer to a file by calling the Optimizer function XPRSwriteprob and XPRSwritedirs if a file name for the directives is also specified. Note that the matrix written by this procedure may be different from the one produced by exportprob since it may include the effects of presolve or cuts generated by the Optimizer. For more detail on the options and behavior of this procedure, refer to the Xpress Optimizer Reference Manual.

**Related topics**
exportprob, writedirs.

**Module**
mmxprs
writesol

Purpose
Write a solution to a file.

Synopsis
procedure writesol(fname:string,options:string)
procedure writesol(ms:mpsol,sname:string,fname:string,options:string)

Arguments
fname     Extended file name
options   String of options
ms        A solution object
sname     Solution name

Further information
1. When using the first syntax this procedure writes the current solution to a file by calling the Optimizer function XPRSwriteslxsol. For more detail on the options and behavior of this procedure, refer to the Xpress Optimizer Reference Manual.

2. With the second syntax, the file is generated from a solution object. In this case, the solution name has to be provided (default name is "solution") and the only supported option is "x" to output the numbers in hexadecimal.

3. Solution save/read procedures can be used only if the constraint and variable names have been loaded into the Optimizer (parameter XPRS_loadnames set to true) and all constraints are named.

Related topics
readsol.

Module
mmxprs
**xor**

**Purpose**
Create an exclusive or expression.

**Synopsis**
function xor(c1: log_or_linctr, c2: log_or_linctr): logctr

**Arguments**
c1 A linear constraint (linctr) or logical expression (logctr)
c2 A linear constraint (linctr) or logical expression (logctr)

**Return value**
A new logctr representing the expression.

**Example**
This example shows how to state an exclusive ‘or’ constraint that expresses the disjunction between two tasks with start time \( s_j \) and fixed duration \( D_j \). A non-exclusive ‘or’ relation can be stated by using the or operator as shown in the last line (constraint \( L \)).

```plaintext
declarations
R=1..2
C: array(range) of linctr ! Linear constraints
L: logctr ! Logical constraint
s: array(R) of mpvar ! Decision variables (start times)
D: array(R) of real ! Data (durations)
end-declarations

C(1):= s(1)+D(1)>=s(2) ! Define (temporary) linear constraints
C(2):= s(2)+D(2)>=s(1)

xor(C(1), C(2)) ! State an exclusive 'or'
forall(j in 1..2) C(j):=0 ! Delete the auxiliary constraints

! The same 'xor' constraint can be stated by:
xor(s(1)+D(1)>=s(2), s(2)+D(2)>=s(1))

! A non-exclusive 'or' relation is stated by using the 'or' operator:
L:= s(1)+D(1)>=s(2) or s(2)+D(2)>=s(1)
```

**Further information**
1. This function creates a logctr constraint representing an exclusive or condition: either \( c1 \) or \( c2 \) is valid, not both.
2. The helper package ‘advmod’ must be loaded if this function is used:
   ```plaintext
   uses 'advmod'
   ```

**Related topics**
indicator, implies

**Module**
mmxprs
19.4 Cut Pool Manager

This section contains the functions and procedures of the Xpress Optimizer cut manager. For a detailed description of the cut manager and its functionality the user is referred to the Xpress Optimizer Reference Manual. To run the cut manager from Mosel, it may be necessary to (re)set certain control parameters of the optimizer. For example, switching off presolve and automatic cut generation, and reserving space for extra rows in the matrix may be useful:

```c
setparam("XPRS_presolve", 0); /* Switch presolve off... */
setparam("XPRS_presolveops", 2270); /* ...or use secure setting for presolve */
setparam("XPRS_cutstrategy", 0); /* Switch automatic cut generation off */
setparam("XPRS_extrarows", 5000); /* Reserve space for 5000 extra rows in the matrix*/
```

The callback functions and procedures that are relevant to the cut manager are initialized with function `setcallback`, in common with the other Optimizer callbacks.

It should be noted that cuts are not stored by Mosel but sent immediately to the Optimizer. Consequently, if a problem is reloaded into the Optimizer, any previously defined cuts will be lost. In Mosel, cuts are defined by specifying a linear expression (i.e. an unbounded constraint) and the operator sign (inequality/equality). If instead of a linear expression a constraint is given, it will also be added to the system as an additional constraint.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>addcut</code></td>
<td>Add a cut to the problem in the optimizer.</td>
<td>706</td>
</tr>
<tr>
<td><code>addcuts</code></td>
<td>Add an array of cuts to the problem in the optimizer.</td>
<td>707</td>
</tr>
<tr>
<td><code>delcuts</code></td>
<td>Delete cuts from the problem in the optimizer.</td>
<td>708</td>
</tr>
<tr>
<td><code>dropcuts</code></td>
<td>Drop a set of cuts from the cut pool.</td>
<td>709</td>
</tr>
<tr>
<td><code>getcnlist</code></td>
<td>Get the set of cuts active at the current node.</td>
<td>710</td>
</tr>
<tr>
<td><code>getcplist</code></td>
<td>Get a set of cut indices from the cut pool.</td>
<td>711</td>
</tr>
<tr>
<td><code>loadcuts</code></td>
<td>Load a set of cuts into the problem in the optimizer.</td>
<td>712</td>
</tr>
<tr>
<td><code>storecut</code></td>
<td>Store a cut into the cut pool.</td>
<td>713</td>
</tr>
<tr>
<td><code>storecuts</code></td>
<td>Store an array of cuts into the cut pool.</td>
<td>714</td>
</tr>
</tbody>
</table>
addcut

**Purpose**
Add a cut to the problem in the optimizer.

**Synopsis**
procedure addcut(cuttype:integer, type:integer, linexp:linctr)

**Arguments**
cuttype     Integer number for identification of the cut
type   Cut type (equation/inequality), which may be one of:
      CT_GEQ    Inequality (greater or equal)
      CT_LEQ    Inequality (less or equal)
      CT_EQ     Equality
linexp   Linear expression (= unbounded constraint)

**Further information**
This procedure adds a cut to the problem in the Optimizer. The cut is applied to the current node and all its descendants.

**Related topics**
addcuts, delcut.

**Module**
mmxprs
addcuts

Purpose
Add an array of cuts to the problem in the optimizer.

Synopsis
procedure addcuts(cuttype:array(range) of integer, type:array(range) of integer,
linexp:array(range) of linctr)

Arguments
cuttype Array of integer number for identification of the cuts
type Array of cut types (equation/inequality):
CT_GEQ Inequality (greater or equal)
CT_LEQ Inequality (less or equal)
CT_EQ Equality
linexp Array of linear expressions (= unbounded constraints)

Further information
This procedure adds an array of cuts to the problem in the Optimizer. The cuts are applied to the
current node and all its descendants. Note that the three arrays that are passed as parameters to
this procedure must have the same index set.

Related topics
addcut, delcut.

Module
mmxprs
delcuts

Purpose
Delete cuts from the problem in the optimizer.

Synopsis
procedure delcuts(keepbasis:boolean, cuttype:integer, interpret:integer,delta:real,
cuts:set of integer)
procedure delcuts(keepbasis:boolean, cuttype:integer, interpret:integer, delta:real)

Arguments
- **keepbasis**: false  Cuts with non-basic slacks may be deleted
  true  Ensures that the basis will be valid
- **cuttype**: Integer number for identification of the cut(s)
- **interpret**: The way in which the cut type is interpreted:
  -1  Delete all cuts
  1  Treat cut types as numbers
  2  Treat cut types as bitmaps (delete cut if any bit matches any bit set in cuttype)
  3  Treat cut types as bitmaps (delete cut if all bits match those set in cuttype)
- **delta**: Only delete cuts with an absolute slack value greater than delta. To delete all the cuts set this parameter to a very small value (e.g. -MAX_REAL)
- **cuts**: Set of cut indices, if not specified all cuts of type cuttype are deleted

Further information
This procedure deletes cuts from the problem loaded in the Optimizer. If a cut is ruled out by any of the given criteria it will not be deleted.

Related topics
addcut, addcuts.

Module
mmxprs
dropcuts

Purpose
Drop a set of cuts from the cut pool.

Synopsis
procedure dropcuts(cuttype:integer, interpret:integer, cuts:set of integer)
procedure dropcuts(cuttype:integer, interpret:integer)

Arguments
- cuttype: Integer number for identification of the cut(s)
- interpret: The way in which the cut type is interpreted:
  - 1: Treat cut types as numbers
  - 2: Treat cut types as bitmaps (delete cut if any bit matches any bit set in cuttype)
  - 3: Treat cut types as bitmaps (delete cut if all bits match those set in cuttype)
- cuts: Set of cut indices in the cut pool, if not specified all cuts of type cuttype are deleted

Further information
This procedure drops a set of cuts from the cut pool. Only those cuts which are not applied to active nodes in the branch-and-bound tree will be deleted.

Related topics
storecut, storecuts.

Module
mmxprs
getcnlist

Purpose
Get the set of cuts active at the current node.

Synopsis
procedure getcnlist(cuttype:integer, interpret:integer, cuts:set of integer)

Arguments
- cuttype: Integer number for identification of the cut(s), -1 to return all active cuts
- interpret: The way in which the cut type is interpreted:
  -1  Get all cuts
  1   Treat cut types as numbers
  2   Treat cut types as bitmaps (get cut if any bit matches any bit set in cuttype)
  3   Treat cut types as bitmaps (get cut if all bits match those set in cuttype)
- cuts: Set of cut indices

Further information
This procedure gets the set of active cut indices at the current node in the Optimizer. The set of cut indices is returned in the parameter cuts.

Related topics
getcplist.

Module
mmxprs
**getcplist**

**Purpose**
Get a set of cut indices from the cut pool.

**Synopsis**
procedure getcplist(cuttype:integer, interpret:integer, delta:real, cuts:set of integer, viol:array(range) of real)

**Arguments**
cuttype     Integer number for identification of the cut(s)
interpret   The way in which the cut type is interpreted:
            -1         Get all cuts
            1         Treat cut types as numbers
            2         Treat cut types as bitmaps (get cut if any bit matches any bit set in cuttype)
            3         Treat cut types as bitmaps (get cut if all bits match those set in cuttype)
delta       Only return cuts with an absolute slack value greater than delta
cuts        Set of cut indices in the cut pool
viol         Array where the slack variables for the cuts will be returned

**Further information**
This procedure gets a set of cut indices from the cut pool. The set of indices is returned in the parameter cuts.

**Related topics**
getcplist.

**Module**
mmxprs
loadcuts

Purpose
Load a set of cuts from the cut pool into the problem in the optimizer.

Synopsis
procedure loadcuts(cuttype:integer, interpret:integer, cuts:set of integer)
procedure loadcuts(cuttype:integer, interpret:integer)

Arguments
    cuttype  Integer number for identification of the cut(s)
    interpret  The way in which the cut type is interpreted:
                -1  Load all cuts
                 1  Treat cut types as numbers
                 2  Treat cut types as bitmaps (load cut if any bit matches any bit set in cuttype)
                 3  Treat cut types as bitmaps (load cut if all bits match those set in cuttype)
    cuts  Set of cut indices in the cut pool, if not specified all cuts of type cuttype are loaded

Further information
This procedure loads a set of cuts into the Optimizer. The cuts remain active at all descendant nodes.

Related topics
storecut, storecuts.

Module
mmxprs
storecut

**Purpose**
Store a cut into the cut pool.

**Synopsis**
```plaintext
function storecut(nodupl:integer, cuttype:integer, type:integer,
linexp:linctr):integer
```

**Arguments**
- `nodupl` Flag indicating how to deal with duplicate entries:
  - 0: No check
  - 1: Check for duplicates among cuts of the same cut type
  - 2: Check for duplicates among all cuts
- `cuttype` Integer number for identification of the cut
- `type` Cut type (equation/inequality):
  - CT_GEQ: Inequality (greater or equal)
  - CT_LEQ: Inequality (less or equal)
  - CT_EQ: Equality
- `linexp` Linear expression (= unbounded constraint)

**Return value**
Index number of the cut stored in the cut pool.

**Further information**
This function stores a cut into the cut pool without applying it to the problem at the current node. The cut has to be loaded into the problem with procedure `loadcuts` in order to become active at the current node.

**Related topics**
- `dropcut`, `loadcuts`, `storecuts`.

**Module**
`mmxprs`
storecuts

Purpose

Store an array of cuts into the cut pool.

Synopsis

procedure storecuts(nodupl:integer, cuttype:array(range) of integer,
                    type:array(range) of integer,
                    linexp:array(range) of linctr,
                    ndx_a:array(range) of integer);

procedure storecuts(nodupl:integer, cuttype:array(range) of integer,
                    type:array(range) of integer,
                    linexp:array(range) of linctr,
                    ndx_s:set of integer);

Arguments

nodupl Flag indicating how to deal with duplicate entries:
0 No check
1 Check for duplicates among cuts of the same cut type
2 Check for duplicates among all cuts

cuttype Array of integer number for identification of the cuts
type Array of cut types (equation/inequality):
  CT_GEQ Inequality (greater or equal)
  CT_LEQ Inequality (less or equal)
  CT_EQ  Equality

linexp Array of linear expressions (= unbounded constraints)
ndx_a Interval of index numbers of stored cuts
ndx_s Set of index numbers of stored cuts

Further information

This function stores an array of cuts into the cut pool without applying them to the problem at the current node. The cuts have to be loaded into the problem with procedure loadcuts in order to become active at the current node. The cut manager returns the indices of the stored cuts in the form of an array ndx_a or a set of integers ndx_s. Note that the four arrays that are passed as parameters to this procedure must have the same index set.

Related topics

dropcut, loadcuts, storecut.

Module

mmxprs
The module $r$ makes it possible to easily exchange data with R and execute R scripts or evaluate expressions in the R language.

R is a free software environment for statistical computing and graphics. R is available as Free Software under the terms of the Free Software Foundation’s GNU General Public License. To use this module, the following line must be included in the header of the Mosel model file:

```
uses 'r';
```

## 20.1 Introduction

This module implements functionality for exchanging data between a Mosel model and R and for calling R functions from a Mosel model.

The $r$ module also defines an I/O driver for exchanging data using the initializations from and initializations to Mosel constructs.

It is the Mosel run-time library that loads and runs R, not vice-versa.

The purpose of the module is to make the extensive data processing capabilities of R available within Mosel. The interactive and graphing features of R are beyond the scope of this module as it does not implement a full interactive R GUI. However, it is possible to use some of these to a limited extent.

### 20.1.1 Prerequisite

This module does not include R binaries. In order to use R you need a working installation of R, version 3.0 or newer and targeting the same platform as Mosel (you won’t be able to use, e.g., the Windows 32-bit version of R from the Windows 64-bit version of Mosel). The most recent supported R version is 3.3.x. To download R, please visit the R Project web site at www.r-project.org.

This module will try to load R from the directory specified by the $R\_HOME$ environment variable, if set, or from the default R installation location otherwise.

More specifically Mosel looks for a file named $R\.dll$ in Windows and $libR\.so$ in Linux.

For Windows platforms, the default location is retrieved from the registry (from registry key HKEY\_LOCAL\_MACHINE\Software\R-core\R\InstallationPath); it is /usr/lib64/R for 64bit Linux and /usr/lib/R for 32bit Linux.

If $R\_HOME$ and $R\_ARCH$ environment variables are defined, they are used to construct a path like $R\_HOME/lib$ in Linux and like $R\_HOME/bin/R\_ARCH$ in Windows (the default for $R\_ARCH$ is x64 or i386 respectively for Windows 64-bit and Windows 32-bit).
If you have multiple installations of R, or if R is installed in a different location or not automatically found, you will need to set the environment variable `R_HOME` to point to your R installation directory.

Note that the loading of R is not influenced by eventual Mosel statements like `setparam('workdir',...)` or `setenv('R_HOME',...)` as these don't affect the process's environment used for R loading. The environment variables or current path must eventually be set before launching Mosel in order for this to influence R loading.

As an example, if R 3.2.3 is installed in "C:\Program Files\R\R-3.2.3\bin\..." in Windows 64-bit, then the correct value for the `R_HOME` environment variable (or registry key) is `C:\Program Files\R\R-3.2.3` (thus, without the bin subdirectory) and Mosel would try and load R.dll from `C:\Program Files\R\R-3.2.3\bin\x64\R.dll`.

### 20.1.2 R initialization

The R environment is automatically initialized at the point where a Mosel model uses for the first time any function that requires it. So we can have the following small example that just prints the R version (it prints the same output as if you typed `R.version.string` on an R console):

```mosel
model "r version example"
  uses 'r';
  Rprint('R.version.string')
end-model
```

Alternatively it is possible to explicitly initialize R using the `Rinit` function. This can be useful in order to retrieve a status code or to specify non-default initialization options.

By default, R is initialized with the options "–slave –vanilla", so no site or user environment, profile, history and workspace files are processed. Please refer to the R documentation for more details on these and other options ([http://cran.r-project.org/doc/manuals/r-release/R-intro.html#Invoking-R](http://cran.r-project.org/doc/manuals/r-release/R-intro.html#Invoking-R)).

Upon startup, only the "utils", "stats", and "methods" R packages are loaded by default. Other packages can be loaded via R statements (using for example the `library` or `require` R functions) or a different initial package list can be specified by setting the `R_DEFAULT_PACKAGES` environment variable (prior to running Mosel).

As R is single-threaded, it is not possible to create more than one R session per model, nor to execute two models in parallel if both use R.

### 20.1.3 Data types

The types of data that can be exchanged with R are the four Mosel elementary types boolean, integer, real and string, plus arrays, lists and sets of these (nested compositions are not supported). Both static and dynamic Mosel arrays are supported and mapped into R atomic vectors of the corresponding type. Mosel lists and sets can also be exported into R vectors.

There is no direct mapping of more complex R types such as factors or data.frames, however these can be exchanged after conversion to basic types. For example, a factor can be loaded into a Mosel array as an array of integers with:

```mosel
Rgetarr("unclass(f)", intarray)
```

or as an array of strings with:

```mosel
Rgetarr("levels(f)[f]", strarray)
```
Note that the first is also equivalent to this simpler form:

\[
\text{Rgetarr("f", intarray)}
\]

since this module ignores the factor's "class" and "levels" attributes; similarly the second is equivalent to the simpler:

\[
\text{Rgetarr("f", strarray)}
\]

since the casting to string, performed within R, automatically takes into account the "levels" attribute.

To load a data.frame into Mosel, it should be converted to a matrix (for instance using \texttt{as.matrix} if the column types allow that) or split into individual column vectors.

For the opposite operation, that is, exporting a Mosel array to R, note that Mosel arrays are always exported as R (dense) atomic vectors. Any index that is not a 1-based integer range is created in R as a named index. Index names, in R, are always strings, so for example, when the Mosel array in the following example is converted to R, the index set \texttt{J} is kept as an unnamed integer index, while \texttt{I} (which does not start with 1) and \texttt{K} (which has holes) are created as named indices.

```
model "array to r"
  uses "r"

  declarations
    I= 2..3
    J= (1,2)
    K= (1,3)
    a: array(I,J,K) of integer
  end-declarations

  a(2,1,1):=4 ! Define some test data entry
  Rset('aR',a) ! Copy data to R
  writeln("Array in R:")
  Rprint("aR") ! Display data held in R
  writeln("dimnames(aR):")
  Rprint("dimnames(aR)") ! Display R indices
end-model
```

Executing this model generates the following output:

```
Array in R:
  , , 1
    [,1] [,2]
 2   4   0
 3   0   0
  , , 3
    [,1] [,2]
 2   0   0
 3   0   0

  dimnames(aR):
    [[1]]
    [1] "2" "3"
    [[2]]
      NULL
    [[3]]
    [1] "1" "3"
```
Note how the first and last entry of \texttt{dimnames}, which correspond to indices $I$ and $K$ respectively, are set to the list of index elements converted to strings; while the second entry is left to \texttt{NULL} since the index set $J$ is a 1-based integer index with contiguous elements.

Conversion to R data frames or other complex R data structures should be done in the R realm and is outside of the scope of this guide. A few examples are shown below, but please refer to the R documentation for further information.

Some common and useful R functions to convert vectors into data frames are \texttt{e.g. data.frame()}, \texttt{as.data.frame()}, and the functions from the \texttt{reshape} or \texttt{reshape2} packages, just to name a few. Also functions \texttt{names()} (for 1-dimensional vectors) or \texttt{dimnames()} (for any vectors) can be used to retrieve the index names of a vector.

In the following example, a Mosel single-indexed \texttt{demand} array is converted to a 2-column R data frame: the first column for the index and the second column for the value:

```r
model "dataframe"
uses "r"
declarations
Locations = {12,34,56}
demand: dynamic array(Locations) of real
demand(12):=1*100
\text{\textbackslash ! Fill array with some data}
end-declarations
forall(l in Locations) demand(l):=l*100
Rset("demand", demand)
Rprint("table <- data.frame(Loc=names(demand),Dem=demand,row.names=NULL)")
end-model
```

This is the resulting output:

<table>
<thead>
<tr>
<th>Loc</th>
<th>Dem</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1200</td>
</tr>
<tr>
<td>2</td>
<td>3400</td>
</tr>
<tr>
<td>3</td>
<td>5600</td>
</tr>
</tbody>
</table>

Alternatively, calling \texttt{data.frame} just as \texttt{data.frame(demand)} without any other parameters would create a data frame with a single column (for \texttt{demand}) and named rows, thus yielding:

<table>
<thead>
<tr>
<th>demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
</tr>
<tr>
<td>34</td>
</tr>
<tr>
<td>56</td>
</tr>
</tbody>
</table>

A \textbf{bidimensional} demand array such as:

```r
declarations
Locations = \{12,34,56\}
C=\{"A","B"\}
demand: dynamic array(Locations,C) of real
demand(12,"A"):=1234
demand(56,"B"):=6789
Rset("demand", demand)
```

could be converted to a data frame via \texttt{data.frame(Loc=dimnames(demand)[[1]],demand, row.names=NULL)} which results in the following form:

<table>
<thead>
<tr>
<th>Loc</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1234</td>
<td>NA</td>
</tr>
<tr>
<td>2</td>
<td>34</td>
<td>NA</td>
</tr>
<tr>
<td>3</td>
<td>56</td>
<td>6789</td>
</tr>
</tbody>
</table>
Finally, for instance by calling function `melt` from the `reshape2` package such as `melt(demand, varnames = c('Loc','Prod'), value.name = 'Demand')`, it is possible to obtain a data frame with a column for each index plus a column with the array values like the following:

<table>
<thead>
<tr>
<th>Loc</th>
<th>Prod</th>
<th>Demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
<td>A 1234</td>
</tr>
<tr>
<td>2</td>
<td>34</td>
<td>A NA</td>
</tr>
<tr>
<td>3</td>
<td>56</td>
<td>A NA</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>B NA</td>
</tr>
<tr>
<td>5</td>
<td>34</td>
<td>B NA</td>
</tr>
<tr>
<td>6</td>
<td>56</td>
<td>B 6789</td>
</tr>
</tbody>
</table>

### 20.2 Example

The following example shows how to execute R statements and exchange data with the R workspace.

```mosel
data model "r example"
uses "r"

declarations
  CITIES = {"LONDON", "PARIS", "ROME"}
  ZONES = 1..4
  mosarray, backarr, backarrio: array(ZONES, CITIES) of integer
  backnum: real
end-declarations

setparam("Rverbose",true) ! Enable showing R error messages

! Reval evaluates arbitrary R statements
Reval("t<-Sys.time();now<-format(t, "%H:%M")")
! Rprint also prints the result (via the R print function)
Rprint("paste('Hello from R at',now)

! Assign some Mosel scalars to R vars and show results
Rset("a_num", 1.2)
Reval("str(a_num)"
Rset("a_chr", "word")
Reval("str(a_chr)"

! The lvalue can be any R valid lvalue, e.g. the dim attribute
Rset("a_vec", 1..6) ! a_vec is an R vector
Rset("dim(a_vec)", [2,3]) ! change its dimensions
writeln("a_vec")
Rprint("a_vec") ! now it is a 2x3 matrix

! Assign a Mosel array to an R variable
forall(i in ZONES, c in CITIES) mosarray(i,c):=i*10+getsize(c)
Rset("arr", mos_array)
! The R vector keeps index names
writeln("arr")
Rprint("arr")

! Retrieve R variables
writeln("a_num is ", Rgetreal("a_num"))
writeln("a_chr is ", Rgetstr("aChr"))
Rgetarr("arr", backarr)
writeln("arr is ", backarr)

! Data can also be exchanged via the I/O driver
newnumber:=1.3
mosarray(1,"LONDON"):=1
! Send data to R
initializations to "r.rws:"
  newnumber as "a_num"
```
mosarray as "arr"
end-initializations
! Get data back from R
initializations from "r.rws:"
  backnum as "a_num"
  backarrio as "arr"
end-initializations
writeln("backnum is ", backnum)
writeln("backarrio is ", backarrio)

end-model

20.3 Control parameters

The following parameters are defined by the module \textit{r}:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{Rcleanscript}</td>
<td>R cleanup script to be run at the end of a session.</td>
<td>p. 721</td>
</tr>
<tr>
<td>\textit{Rinteractive}</td>
<td>Control the R interactive flag.</td>
<td>p. 720</td>
</tr>
<tr>
<td>\textit{Rsessionmode}</td>
<td>R session handling mode.</td>
<td>p. 721</td>
</tr>
<tr>
<td>\textit{Runloadscript}</td>
<td>R unload script to be run at the end of a session.</td>
<td>p. 721</td>
</tr>
<tr>
<td>\textit{Rusemosstreams}</td>
<td>Enable/disable R output redirection.</td>
<td>p. 721</td>
</tr>
<tr>
<td>\textit{Rverbose}</td>
<td>Enable/disable R error messages.</td>
<td>p. 720</td>
</tr>
</tbody>
</table>

\textbf{Rverbose}

\textbf{Description}  Enables or disables the printing of error messages when errors occur during the evaluation of R statements. When this parameter is set, two corresponding R options are set accordingly, namely \texttt{show.error.messages}, which is set to the same value as this parameter, and \texttt{warn} which is set to 1 or -1 when this parameter is set to \texttt{true} or \texttt{false} respectively.

\textbf{Type}  Boolean, read/write

\textbf{Default value}  false

\textbf{Module}  \textit{r}

\textbf{Rinteractive}

\textbf{Description}  This has effects on eventual user prompts and confirmation requests from R, please refer to R documentation (e.g. on \texttt{interactive()}) for more details.

\textbf{Type}  Boolean, read/write

\textbf{Default value}  false

\textbf{Module}  \textit{r}
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Default value</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rseemosstreams</td>
<td>By default R sends output to stdout and stderr. Using this parameter it is possible to redirect R console output to Mosel streams instead. Note that the R notion of stdin connection is not affected by this parameter.</td>
<td>Boolean, read/write</td>
<td>true</td>
<td>r</td>
</tr>
<tr>
<td>Rcleanscript</td>
<td>This parameter can be used to specify the R statement(s) to be executed at the end of an R session; its purpose should be to clear the R workspace. Note that this script is run only for session modes 2 and 3.</td>
<td>String, write only</td>
<td>remove all objects currently defined in the R workspace</td>
<td>r</td>
</tr>
<tr>
<td>Runloadscript</td>
<td>This parameter can be used to specify the R statement(s) to be executed at the end of an R session; its purpose should be to free and unload all resources used by R. Note that this script is run only for session mode 3 and after the Rcleanscript statement(s).</td>
<td>String, write only</td>
<td>unload all packages and dynamic libraries</td>
<td>r</td>
</tr>
<tr>
<td>Rsessionmode</td>
<td>Specifies what actions are taken at the end of an R session.</td>
<td>Integer, read/write</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Values
0  END: the R session is ended.
1  KEEP: the R session is kept alive and the current R workspace is preserved.
2  CLEAR: the R session is kept alive and the Rcleanscript is executed.
3  UNLOAD: both the Rcleanscript and Runloadscript are executed, then R is unloaded.

Default value 2

Notes
This parameter is useful mainly when multiple Mosel models that use R are executed within the same process. When an R session is ended, R does not allow the creation of further sessions, therefore R will not be usable again within the same process. Session mode 3, by unloading R, should enable the possibility to create new R sessions within the same process, however it may not completely free all resources allocated by R.

Affects routines Rfree.

Module r

20.4 Procedures and functions

All R statements are evaluated in the R Global Environment, more often known as the user’s workspace.

In general, the procedures and functions of r do not fail in case of R parsing or evaluation errors but set an internal status variable that can be read with Rerrcode. To make sure that an operation has been performed correctly, it is recommended to check the value of this variable after each call.

Rclearerr Clear the last error code and message. p. 736
Rerrcode Get the last error code. p. 734
Rerrmsg Get the last error message. p. 735
Reval Evaluate R statements. p. 723
Rfree Terminate an R session. p. 724
Rgetarr Get the resulting array of an R expression. p. 725
Rgetbool Get the boolean value of an R expression. p. 726
Rgetint Get the integer value of an R expression. p. 727
Rgetreal Get the real value of an R expression. p. 728
Rgetstr Get the string value of an R expression. p. 729
Rinit Initialize an R session. p. 730
Rprint Evaluate R statements and print the result. p. 731
Rset Assign a Mosel value to an R object. p. 732
Rsource Evaluate an R script file. p. 733
Reval

**Purpose**
Evaluate R statements.

**Synopsis**
procedure Reval(cmd:string)

**Argument**
- cmd: Statements to evaluate

**Example**
The following example loads the datasets package, calculates summary statistics of the attenu dataset and prints results:

```
Reval('library(datasets); s<-summary(attenu)')
Rprint('s')
```

**Further information**
It is possible to evaluate multiple statements separating them with semicolons.

**Related topics**
- Rprint.
Rfree

**Purpose**
Terminate an R session.

**Synopsis**
procedure Rfree

**Example**
The following example terminates the R session:

Rfree

**Further information**
The use of this procedure is optional: R is automatically terminated at the end of a model execution. However you may prefer to terminate it sooner to free resources allocated by R.

**Related topics**
Rinit.
**Rgetarr**

**Purpose**
Get the resulting array of an R expression.

**Synopsis**
procedure Rgetarr(cmd:string, arr:array)

**Arguments**
- **cmd** Statements to evaluate
- **arr** Destination Mosel array

**Example**
The following example loads the R cars example dataset into the Mosel array cars:

```mosel
declarations
cars:array(range, set of string) of integer
end-declarations
Rgetarr('as.matrix(datasets::cars)', cars)
```

**Further information**
1. If **cmd** contains more than one statement, the returned value is the result of the last one.
2. The Mosel array **arr** must have the same number of dimensions as the R array; NA entries in R are skipped and corresponding entries in **arr** are left unchanged (note that **arr** is not cleared before loading R data).
3. Supported index types for the **arr** array are string and integer. In the case of strings, the R array must have a valid `names` or `dimnames` attribute for the corresponding dimension; in the case of integers, the R integer indices (1 to n) are used. All entries of the R array are converted to the same type as the destination array within R.

**Related topics**
- Rgetarr, Rgetint, Rgetreal, Rgetstr
Rgetbool

**Purpose**
Get the boolean value of an R expression.

**Synopsis**
function Rgetbool(cmd:string):boolean

**Argument**
cmd Statements to evaluate

**Return value**
The result of the evaluation as a boolean.

**Example**
The following example checks if the R entity vec is atomic and sets the boolean variable boolvar accordingly:

```
boolvar:=Rgetbool('is.atomic(vec)')
```

**Further information**
If cmd contains more than one statement, the returned value is the result of the last one. If the results is NA, then false is returned. The returned value is first converted to logical within R.

**Related topics**
Rgetarr, Rgetint, Rgetreal, Rgetstr
Rgetint

Purpose
Get the integer value of an R expression.

Synopsis
function Rgetint(cmd:string):integer

Argument
cmd Statements to evaluate

Return value
The result of the evaluation as an integer.

Example
The following example retrieves the length of the R entity vec into variable intvar:

intvar:=Rgetint('length(vec)')

Further information
If cmd contains more than one statement, the returned value is the result of the last one. If the results is NA, then the R NA_Integer value (-2^31) is returned. The returned value is first converted to integer within R.

Related topics
Rgetarr, Rgetbool, Rgetint, Rgetstr
**Rgetreal**

**Purpose**
Get the real value of an R expression.

**Synopsis**
function Rgetreal(cmd:string):real

**Argument**
cmd Statements to evaluate

**Return value**
The result of the evaluation as a real.

**Example**
The following example retrieves the mean of the speed/dist ratios for the cars dataset into the variable realvar:

```r
realvar:=Rgetreal('mean(cars$speed/cars$dist)')
```

**Further information**
If cmd contains more than one statement, the returned value is the result of the last one. If the results is NA, then a NaN value is returned. The returned value is first converted to numeric within R.

**Related topics**
Rgetarr, Rgetbool, Rgetint, Rgetstr
Rgetstr

Purpose
Get the string value of an R expression.

Synopsis
function Rgetstr(cmd:string):string

Argument
cmd Statements to evaluate

Return value
The result of the evaluation as a string.

Example
The following example retrieves the version string of R into the variable strvar:

   strvar:=Rgetstr('R.version.string')

Further information
If cmd contains more than one statement, the returned value is the result of the last one. If the results is NA, then the string "NA" is returned. The returned value is first converted to string within R.

Related topics
Rgetarr, Rgetbool, Rgetreal, Rgetstr
Rinit

Purpose
Initialize an R session.

Synopsis
function Rinit([args:string...]): integer

Argument
args List of R startup options (optional)

Return value
0 if the initialization was successful.

Example
The following example initializes R with default options:

initok:=Rinit
if initok<>0 then writeln('Failed to initialize R')
end-if

Further information
The use of this function is optional: R is automatically initialized upon first use. Default R startup options are ”–slave”, ”–vanilla”, so no site or user environment, profile, history and workspace files are processed. Please refer to the R documentation for the exact meaning of these and other options (http://cran.r-project.org/doc/manuals/r-release/R-intro.html#Invoking-R).

Related topics
Rfree.
Rprint

Purpose
Evaluate R statements and print the result.

Synopsis
procedure Rprint(cmd:string)

Argument

cmd Statements to evaluate and print

Example
The following example prints the R version number and summary statistics of the cars dataset:

Rprint("R.version.string")
Rprint('library(datasets); summary(cars)')

Further information
It is possible to evaluate multiple statements separating them with semicolons. The return value of the last statement is printed using R’s own print function, thus with R style and formatting.

Related topics
Reval.
Rset

**Purpose**
Assign a Mosel value to an R object.

**Synopsis**
procedure Rset(dst:string, value:[set|array|list of] boolean|integer|real|string)

**Arguments**
dst An R variable name
value The Mosel value to be assigned to dst

**Example**
The following example assigns values 1.2, "hello" and an array with numbers 1 to 6 to the R objects a_num, a_string and a_vec respectively:

Rset('a_num', 1,2)
Rset('a_string', 'hello')
Rset("a_vec", 1..6) ! An R vector with real values 1 to 6
Rset("dim(a_vec)", [2,3]) ! Change its dimensions into a 2x3 matrix

**Further information**
1. A new temporary R entity is created from value and then assigned to dst
2. Argument dst can represent any assignable expression (including subsetting and attributes).
3. The type of argument value can be any elementary Mosel type, or an array, list or set of these (compositions are not supported).
4. When value is an array, for each of the array's dimensions, its index values are exported into the corresponding R array's names or dimnames attribute (after conversion to string) unless the indices are integer values from 1 to that dimension size.
5. If value is a dynamic array, only the existing values are copied to R and the remaining array entries are set to NA (Not Available).

**Related topics**
Reval.
Rsource

**Purpose**
Evaluate an R script file.

**Synopsis**
procedure Rsource(filename:string)

**Argument**
  filename Filename of the R script to evaluate

**Example**
The following example executes the myscript.R file:

  Rsource('myscript.R')

**Related topics**
  Reval.
Rerrcode

**Purpose**
Get the last error code.

**Synopsis**
function Rerrcode:integer

**Return value**
0 if the last operations were executed successfully.

**Example**
The following example prints an error message in case of errors in R evaluations:

```r
Reval('missingfunction()')
if Rerrcode<>0 then
  writeln('Something went wrong: ', Rerrmsg)
  Rclearerr
end-if
```

**Further information**
The `Rerrcode` is set to non-zero values in case of errors, but not cleared after successful operations, so it is possible to check it after several operations to verify that all executed without errors. To clear it, use function `Rclearerr`.

**Related topics**
`Rclearerr`, `Rerrmsg`. 
Rerrmsg

**Purpose**
Get the last error message.

**Synopsis**
```pascal
function Rerrmsg:string
```

**Return value**
The last error message in case of errors, or the empty string otherwise.

**Example**
The following example prints an error message in case of errors in R evaluations:
```pascal
Reval('missingfunction()')
if Rerrcode<>0 then
  writeln('Something went wrong: ', Rerrmsg)
Rclearerr
end-if
```

**Further information**
The message returned by this function is a top-level description of the error. It is possible to also retrieve R own error message for example with `Rgetstr("geterrmessage()")`.

**Related topics**
- Rclearerr, Rerrcode.
**Rclearerr**

**Purpose**
Clear the last error code and message.

**Synopsis**
procedure Rclearerr

**Example**
The following example prints an error message in case of errors in R evaluations and subsequently clears the error information:

```
Reval('missingfunction()')
if Rerrcode<>0 then
  writeln('Something went wrong: ', Rerrmsg)
  Rclearerr
end-if
```

**Related topics**
Rerrcode, Rerrmsg.
20.5 I/O drivers

In order to simplify access to R this module provides a driver that is designed to be used in initializations blocks for both reading and writing data, providing the same functionalities as the Rget and Rset functions.

20.5.1 Driver rws

The driver can only be used in ‘initializations’ blocks. It does not take any argument and provides access to the R workspace.

In the block, each label entry is understood as one or more R statements. For ‘from’ blocks, if the label contains more than one statement, the value from the last one is returned. For ‘to’ blocks, the label must contain only one expression.

This driver requires an existing R session, therefore it is necessary to initialize R (either by calling function Rinit or any of the other module functions that create an R session) before using it.

Example:

```mosel
initok:=Rinit ! Initialize R
initializations to "r.rws:" ! Send data to R
scalarvar as "val"
arrayvar as "arr"
end-initializations
initializations from "r.rws:" ! Get data from R
backscalar as "val"
backarr as "arr"
end-initializations
```

20.6 Troubleshooting

This section describes some known issues and possible solutions.

- When running a model in Windows, a dialog is shown with title ‘Unable to locate component’ and content ‘The application has failed to start because Rlapack.dll was not found...’.
  This may occur with Windows 2003. Please add your R binary directory (usually ‘C:\Program Files\R\R-3.x.x\bin\i386’ or ‘C:\Program Files\R\R-3.x.x\bin\x64’) to the system PATH environment variable.

- When an R session is initialized in Windows, R installs a console handler to detect Ctrl-C events which may prevent Mosel from properly detecting these same events itself.

- In Linux, R may fail to load if the dynamic libraries in $R_HOME/lib cannot be found by the dynamic linker. In this case, please add $R_HOME/lib to the LD_LIBRARY_PATH environment variable.

- This module is not compatible with Mosel security restrictions, therefore it would fail to load if Mosel is run in restricted mode.
The zlib Mosel module is an interface to the zlib compression library by Jean-Loup Gailly and Mark Adler (http://zlib.net). Thanks to two IO drivers it makes possible the creation and use of compressed files from Mosel models. As an additional feature this module also integrates the MiniZip library by Gilles Vollant (http://www.winimage.com/zLibDll/minizip.html) for supporting the ZIP archive format.

21.1 I/O drivers

The following two drivers behave the same: a stream open for reading is decompressed and a stream open for writing is created compressed. Both drivers are also based on the same compression algorithm (deflate) but use different container formats. The last published driver (zip) can only be open for reading: it will be used to access a file stored in a zip archive. For more advanced use of ZIP archives please refer to the dedicated routines proposed by the mmsystem module.

21.1.1 Driver gzip
gzip:filename

This driver handles files compressed using the gzip compression format: this corresponds to files created using the gzip compression tool.

For instance the following statement decompresses the file "myfile.gz":

```plaintext
fcopy("zlib.gzip:myfile.gz","myfile")
```

21.1.2 Driver deflate
deflate:filename

This driver handles files compressed using the zlib compression format. This driver can read documents compressed by gzip but compressed files it generates are not compatible with this tool.

21.1.3 Driver zip
zip:zipfile, filename

This driver handles archives using the ZIP format. It can be used only for reading files: the filename of this driver consists in two parts separated by a coma. The first part is the name of the archive to open and the second one is the archive member name.
For instance the following statement compiles the file "main.mos" stored in the archive "myproject.zip":

    compile("G","zlib.zip:myproject.zip,main.mos","tmp:main.bim")
Appendix
APPENDIX A

Syntax diagrams for the Mosel language

A.1 Main structures and statements

(Model) ::= - 'model' String - - Identifier - - ... - - (Directives) - (Parameters) - (Body) - 'end-model'

(Package) ::= - 'package' - Identifier - ... - - (Directives) - (Parameters) - (Body) - 'end-package'

(Directives) ::= - 'uses' String - ',' - 'imports' String - ',' - 'options' Identifier - ',' - 'version' Integer - '.' - 'Integer - '.' - 'Integer

(Parameters) ::= ... - 'parameters' ... - Identifier '=' - (Expression) ... - 'end-parameters'

(Body) ::= - (Declarations) - (Requirements) - (SubProgram_decl) - (SubProgram_def) - 'include' - String - Identifier - (Statement)
Syntax diagrams for the Mosel language

(Elif_body) ::= 'elif' - ⟨Bool_expr⟩ 'then' - ⟨Stat_list⟩

(Else_body) ::= 'else' - ⟨Stat_list⟩

(Case_body) ::= ⟨Expression⟩ ':' ⟨Statement⟩ ⟨Do_block⟩

(With_block) ::= 'with' ⟨Name_ref⟩ 'and' Identifier 'do' · · · 'end-do'

(Init_block) ::= 'initializations' 'from' ⟨String_expr⟩ · · · 'to'

··· ⟨Init_item⟩ · · · 'as' ⟨String_expr⟩

··· 'end-initializations'

(Init_item) ::= Identifier (' (' ⟨Init_fieldsel⟩ ')') 'evaluation of' ⟨Expression⟩

(Init_fieldsel) ::= Identifier (' (' ⟨Init_fieldsel⟩ ')')

(Do_block) ::= 'do' - ⟨Stat_list⟩ 'end-do'

(Stat_list) ::= ⟨Statement⟩
A.2 Expressions

\[
\begin{align*}
\langle Expression \rangle & ::= (\langle Bool_expr \rangle) \quad (\langle Set_expr \rangle) \quad (\langle List_expr \rangle) \quad (\langle Arith_expr \rangle) \quad (\langle String_expr \rangle) \quad (\langle Array_expr \rangle) \quad (\langle Ctr_expr \rangle) \\
(\langle Comparator \rangle) & ::= '<' \quad '<=\' \quad '=' \quad '<>' \quad '>=' \quad '>' \\
(\langle Bool_expr \rangle) & ::= (\langle Bool_expr \rangle) 'and' (\langle Bool_expr \rangle) \quad 'or' (\langle Bool_expr \rangle) \\
(\langle Set_expr \rangle) & ::= (\langle Set_expr \rangle) '+' (\langle Set_expr \rangle) \quad '-' (\langle Set_expr \rangle) \quad '*' (\langle Set_expr \rangle) \\
(\langle List_expr \rangle) & ::= (\langle List_expr \rangle) '+' (\langle List_expr \rangle) \quad '-' (\langle List_expr \rangle) \quad 'sum' (\langle List_expr \rangle) \\
\end{align*}
\]
Syntax diagrams for the Mosel language

\[
\begin{align*}
\langle \text{Arith\_expr} \rangle & ::= \langle \text{Arith\_expr} \rangle + \langle \text{Arith\_expr} \rangle \\
& \quad \ldots \\
& \quad \langle \text{Arith\_expr} \rangle - \langle \text{Arith\_expr} \rangle \\
& \quad \langle \text{Arith\_expr} \rangle - \langle \text{Arith\_expr} \rangle \\
& \quad \langle \text{Arith\_expr} \rangle \langle \text{Arith\_expr} \rangle \\
& \quad \langle \text{Arith\_expr} \rangle \langle \text{Arith\_expr} \rangle \\
& \quad \langle \text{Arith\_expr} \rangle \langle \text{Arith\_expr} \rangle \\
& \quad \langle \text{Arith\_expr} \rangle \langle \text{Arith\_expr} \rangle \\
\langle \text{String\_expr} \rangle & ::= \langle \text{String\_expr} \rangle \langle \text{String\_expr} \rangle \\
& \quad \langle \text{Expression} \rangle \\
& \quad \langle \text{Expression} \rangle \\
& \quad \langle \text{Expression} \rangle \\
& \quad \langle \text{Expression} \rangle \\
& \quad \langle \text{Expression} \rangle \\
\langle \text{Array\_expr} \rangle & ::= \langle \text{Array\_expr} \rangle \langle \text{Iterator\_list} \rangle - \langle \text{Arith\_expr} \rangle \\
\langle \text{Iterator\_list} \rangle & ::= \langle \text{Iterator\_list} \rangle , \langle \text{Iterator\_list} \rangle - \langle \text{Arith\_expr} \rangle \\
\langle \text{Iterator\_list} \rangle & ::= \langle \text{Iterator\_list} \rangle , \langle \text{Iterator\_list} \rangle - \langle \text{Arith\_expr} \rangle \\
\langle \text{Iterator} \rangle & ::= \langle \text{Iterator} \rangle \langle \text{List\_expr} \rangle - \langle \text{Set\_expr} \rangle \\
& \quad \langle \text{Expression} \rangle \\
& \quad \langle \text{Expression} \rangle \\
& \quad \langle \text{Expression} \rangle \\
& \quad \langle \text{Expression} \rangle \\
& \quad \langle \text{Expression} \rangle \\
\end{align*}
\]
A.3  Initializations data file format
A Mosel instance may be started remotely using either the `connect` procedure of the module `mmjobs` or its equivalent routine of the XPRD library. From any of these environments one can compile, load and run models as well as access files on the remote system. The remote invocation protocol also makes it possible to control more precisely the execution of models (e.g. suspending execution or profiling) and query information of a model (such as the value of entities) or the entire instance (e.g. to retrieve the list of available modules).

This protocol relies on two mechanisms:

1. the procedure `setcontrol` (available in `mmjobs` and XPRD) to (re)define Mosel instance control parameters
2. the special remote file `mcmd` that only supports read access: the file name is interpreted as a query and the retrieved data is the answer to this request

### B.1 Instance control parameters

Mosel instance control parameters are either defined at the instance level or they apply to a specific model (see `setcontrol`). Some of the parameters serve for changing the behaviour of other commands, others provide a means to execute some specific command.

The supported instance parameters are:

- `zerotol` (real,instance): set the zero tolerance used for comparison and displaying real numbers (i.e. a real number smaller than the tolerance is treated as 0)
- `realfmt` (string,instance): set the C-format used to display reals
- `flushdso` (none,instance): unloads unused modules (i.e. calls XPRMflushdso)
- `lang` (string,instance): set the language of the instance
- `defaultnode` (integer,instance): set the default node number used by the "rmt:" IO driver when it is used without node reference (see Section 7.4.6)
- `runmode` (int,model): set the execution mode of a model (cannot be changed during the execution of the model):
  - 0: default
  - 1: debug
  - 2: profile
Remote Invocation Protocol

- **dbgctrl** (string,model): send a command to the debugger (model must be running in debug mode). See Section B.4
- **dbgbrksub** (int,model): toggle `breaksub` mode during a debugging session (default is 0)
- **sdmax** (int,instance): set the maximum depth of a call stack dump (default is 0)

### B.2 mcmd pseudo file

The special remote file `mcmd` takes the form of an I/O driver where the file name is interpreted as a query and the retrieved data is the answer to this request. Except for the `dsostream` command that can also be open in write mode, `mcmd` only supports read access. A file name for this driver has the following form:

```
mcmd: cmd[-opts][@mod[.submod]] cmdargs
```

- **cmd**: the operation to execute (e.g. `eval`, `profres`...).
- **opts**: options to change the format of the result. By default all data are sent using the Mosel binary format `bin:`. Adding option `t` switches to text format (still compatible with initialisations blocks) and `j` will cause results to be sent as a JSON object (not compatible with initialisations blocks). If option `z` is used the resulting file is compressed with gzip. A given command may also support additional options (see `lslib`). Except for the `eval` command, the result set publishes always the same records that are either scalars of basic types or lists of basic types. When a collection of values is returned a specific label indicating the dimension of the list precedes the list.
- **mod**: master model on which the operation will be performed.
- **submod**: submodel. Only some operations can be applied to a submodel. Note that submodel `0` is the master model itself (the first submodel has ID `1`).

**Supported commands:**

#### covres (model)

Retrieve profiling results for test coverage. This command can only be called after the model has been run in profiling mode (see Section B.3).

```
tottime: real
Rlines: range
lines: array(Rlines) of integer
iters: array(Rlines) of integer
Rfiles: range
files: array(Rfiles) of string
Rstarts: range
starts: array(Rstarts) of integer
```

#### dbgbrk [ lndx [ cond* ] ] (model or submodel)

Breakpoint management. The model must be suspended. Without any parameters this command returns the current list of breakpoints; the first parameter is interpreted as a line index and the second parameter is a logical expression (i.e. the breakpoint condition). With only one parameter, the breakpoint on the corresponding line index is removed (no operation is performed if there was no breakpoint). Use `*` to remove all breakpoints. With two parameters the corresponding breakpoint is created (or modified). To set an unconditional breakpoint use `*` as the condition. Note that breakpoints are attached to a model: even if several models running concurrently are resulting from the same source file, setting a breakpoint for one model (instance) will have no effect on the others.
Remote Invocation Protocol

\[
\text{RIndex: range} \\
\text{indx: array(RIndex) of integer} \\
\text{cond: array(RIndex) of string}
\]

\text{DBGFLndx [ fctname|* ] (model or submodel)}

Line indices corresponding to a function name. This command returns the line indices corresponding to the beginning of the requested function (several values are returned when the routine is overloaded). Without any arguments or with argument ‘*’, the command returns all functions of the model. If option ‘N’ is used, the arrays are sorted according to the function names. With option ‘L’ arrays are sorted following the line indices order.

\[
\text{RSign: range} \\
\text{sign: array(RSign) of string} \\
\text{indx: array(RSign) of integer} \\
\text{name: array(RSign) of string}
\]

\text{DBGINDEX (model or submodel)}

Retrieve the mapping of the line indices of a model/submodel. The model must be either suspended or not running. The debugger interface works on line indices: each line index corresponds to a file name and a line number in this file.

\[
\text{RFiles: range} \\
\text{files: array(RLines:range) of integer} \\
\text{RLines: range} \\
\text{lines: array(RFiles:range) of string}
\]

\text{DBGSTAT (model or submodel)}

Current execution status of a model. The model must be suspended. If no submodel is specified, statuses of all submodels are returned in addition to the master model (to get the status of the master model only use submodel ‘0’).

\[
\text{Rid: range} \\
\text{id: array(Rid) of integer} \\
\text{stat: array(Rid) of integer} \\
\text{stlev: array(Rid) of integer} \\
\text{indx: array(Rid) of integer}
\]

\text{DBGSTLEV [ stlev|* [ maxlev ] ] (model or submodel)}

Stack management. The model must be suspended. Without any arguments or with argument ‘*’, the command returns a stack dump (i.e. a list of line indices). If the argument is $\geq 0$, it becomes the current stack level. The optional argument ‘maxlev’ defines the maximum number of levels to return (default:10). The stack level defines the context in which expression evaluations are performed in the ‘eval’ command.

\[
\text{Rid: range} \\
\text{id: array(Rid) of integer} \\
\text{stat: array(Rid) of integer} \\
\text{stlev: array(Rid) of integer} \\
\text{indx: array(Rid) of integer}
\]

\text{DSOSTREAM dsoname [specific parameters] (model or submodel)}

This command opens a stream to the specified module (this command supports both read and write mode). The module must implement the SRV_DSOSTRE service. The behaviour of the stream and the expected parameters depend on the implementation.

\text{Eval label:expression [ range ] (model or submodel)}

Evaluate an expression in the context of the provided model/submodel. Execution of the model must be completed or suspended. If option ‘i’ is used, array indices are reported as order numbers instead of values. With option ‘n’ array values are replaced by empty
strings. The label "label" is used to identify the expression in the result file: if it is '.' no label is generated (the data result is directly sent to the result file), if it is omitted then the expression itself is used as the label, otherwise the provided string is the label. Several expressions may be evaluated in a single request (in this case they must all be labelled). It is possible to grab only a part of a collection (array, set or list) by specifying range information. Ranges definitions take one of these two forms:

- [ maxelt ]: get at most 'maxelt' elements
- [ skip maxelt ]: get at most 'maxelt' entries after skipping 'skip' elements

Several range definitions may be specified (separated by blanks): they are used when exploring complex structures (e.g. a list of list). The structure and type of the result set depends on the expression.

info (instance, model or submodel)
This command reports all symbols defined by Mosel (if used without specifying any argument), a module (the argument is the module name) or a model (the argument is a model or submodel ID). In the case of a module, the command loads the module if it is not yet in memory. For a model (or package), it must have been loaded prior to this command since it is referenced by its ID and be either not running or suspended. Information returned by the ‘info’ command:

```plaintext
fmt: integer
Rhdr: range
hdr: array(Rhdr) of string
Rdeps: range
deps: array(Rdeps) of integer
depsvers: array(Rdeps) of integer
depstyp: array(Rdeps) of integer
Rtype: range
type: array(Rtype) of string
typscod: array(Rtype) of integer
Rparms: range
parms: array(Rparms) of string
parmsval: array(Rparms) of integer
parmsdesc: array(Rparms) of string
Rconsts: range
consts: array(Rconsts) of string
conststyp: array(Rconsts) of integer
Rcstint: range
constint: array(Rcstint) of integer
Rcststr: range
conststr: array(Rcststr) of string
Rcstdbl: range
cstdbl: array(Rcstdbl) of real
Rvars: range
vars: array(Rvars) of string
varstyp: array(Rvars) of integer
Rarrndx: range
arrndx: array(Rarrndx) of string
Rfct: range
fct: array(Rfct) of string
fctsign: array(Rfct) of string
fcttyp: array(Rfct) of integer
Rdtyp: range
dtyp: array(Rdtyp:range) of string
dtypetyp: array(Rdtyp) of integer
Rrecsstart: range
recsstart: array(Rrecsstart) of integer
Rrecfield: range
recfield: array(Rrecfield) of string
recftyp: array(Rrecfield) of integer
Riodrv: range
iodrv: array(Riodrv) of string
iodrvinfo: array(Riodrv) of string
```
lsattr \textit{(model or submodel)}
Return the list of available types attributes. The array ‘attrsntyp’ gives the type supporting
the attribute and ‘attrsatyp’ is the type of the attribute.

\begin{verbatim}
Rattrs: range
atts: array(Rattrs) of string
attrsntyp: array(Rattrs) of integer
attrsatyp: array(Rattrs) of integer
\end{verbatim}

lslib \textit{(instance)}
Return the list of available packages and modules. If option ‘p’ is used, packages are
reported with their full path.

\begin{verbatim}
Rpkgs: range
pkgs: array(Rpkgs) of string
Rdsos: range
daos: array(Rdsos) of string
\end{verbatim}

lsloc \textit{(model or submodel)}
This command is similar to ‘info’ but it can only be applied to a suspended model: it reports
all local variables.

\begin{verbatim}
Rtyps: range
typs: array(Rtype) of string
typscod: array(Rtype) of integer
Rvars: range
vars: array(Rvars) of string
varstyp: array(Rvars) of integer
Rarrndx: range
arrndx: array(Rarrndx) of string
\end{verbatim}

profres \textit{[ path ] (model)}
Retrieve profiling results. This command can only be called after the model has been run in
profiling mode (see Section B.3). The path argument indicates which execution is
requested: several "executions" may be available when the model starts other models with
‘mmjobs’ (the returned data set includes the number of additional executions available via
the nbsub field). For instance the path 1.3 corresponds to the third "execution" started by
the first "execution".

\begin{verbatim}
tottime:real
nbsub: integer
nbnoprf:integer
Rlines: range
lines: array(Rlines) of integer
iters: array(Rlines) of integer
times: array(Rlines) of real
elaps: array(Rlines) of real
Rfiles: range
files: array(Rfiles) of string
Rstarts:range
starts: array(Rstarts) of integer
\end{verbatim}

B.3 Profiler interface

Profiling a model requires a bim file compiled with option ‘-G’. The runmode has to be set to ‘2’
before starting the execution. After the end of the profiler run, calls to the command
'mcmd:profres' or 'mcmd: covres' can be used for retrieving the results.

B.4 Debugger interface

The debugger can be started even if the flag '-G' has not been used for compilation but in this
case most commands will fail to return useful information. To run a debugging session the
runmode of the model must be set to '1' before starting its execution. If the model was
compiled with '-G', the execution is immediately suspended before the first statement of the
model and a notification event is sent.

During a debugging session changes of the model execution status are notified by specific events
of class 'EVENT_DBG' (32770). The value of these events is a 32bit integer (cast to a real): the first
16 bits are a parameter (meaning depending on the reason) and the following 16 bits indicate
the reason for the notification:

■ DBG_NOTIF_START (1«16): Submodel starting (the parameter is the submodel ID)
■ DBG_NOTIF_END (2«16): Submodel ending (the parameter is the submodel ID)
■ DBG_NOTIF_STOP (3«16): Execution suspended (the parameter is the VM status)

When an event 'DBG_NOTIF_STOP' is received, the model (and its submodels) is in suspended state
and can be sent commands (see Section B.2). To continue execution a control parameter
'dbgctrl' has to be set. The possible values are (the operation applies to the master model unless
a submodel number 'num' is given):

■ C: continue
■ E: end of execution, abort debugging session
■ N [num]: continue to next statement
■ S [num]: step into subroutine
■ F [num]: continue up to end of subroutine
■ T num lndx: continue up to the specified line number on submodel 'num' (0 for master
model)

Additionally, during the execution of a model running in debugging mode (but that is not
suspended), the following 'dbgctrl' commands can be used:

■ B: suspend execution (e.g. consequence of ctrl-C)
■ E: end of execution, abort debugging session

When the execution of the model is about to end (including after an error), it is suspended just
before exiting such that the user can look at the current status.
Appendix C

Error messages

The Mosel error messages listed in the following are grouped according to the following categories:

- **General errors**: may occur either during compilation or when running a model.
- **Parser/compiler errors**: raised during the model compilation.
- **Runtime errors**: when running a model.

All messages are identified by their code number, preceded either by the letter E for error or W for warning. Errors cause the compilation or execution of a model to fail, warnings simply indicate that there may be something to look into without causing a failure or interruption.

This chapter documents the error messages directly generated by Mosel, not the messages stemming from Mosel modules or from other libraries used by modules.

### C.1 General errors

These errors may occur either during compilation or when running a model.

- **E-1 Internal error in ‘location’ (errortype)**
  An unrecoverable error has been detected, Mosel exits. Please contact Xpress Support.

- **E-2 General error in ‘location’ (errortype)**
  An internal error has been detected but Mosel can recover. Please contact Xpress Support.

- **E-4 Not enough memory**
  Your system has not enough memory available to compile or execute a Mosel model.

- **E-21 I cannot open file ‘file’ for writing (driver_error)**
  Likely causes are an incorrect access path or write-protected files.

- **E-22 I cannot open file ‘file’ for reading (driver_error)**
  Likely causes are an incorrect access path or filename or not read-enabled files.

- **E-23 Error when writing to the file ‘file’ (driver_error)**
  The file could be opened for writing but an error occurred during writing (e.g. disk full).

- **E-24 Error when reading from the file ‘file’ (driver_error)**
  The file could be opened for reading but an error occurred while reading it.
**E-25** *Unfinished string*
A string is not terminated, or different types of quotes are used to indicate start and end of a string.

*Examples:*

```plaintext
writeln("mytext")
```

**E-26** *Identifier expected*
May occur when reading data files: a label is missing or a numerical value has been found where a string is expected.

*Examples:*

```plaintext
declarations
D: range
end-declarations

initializations from "test.dat"
D
end-initializations

Contents of test.dat:

```
[1 2 3]
```

The label D: is missing.

**E-27** *Number expected*
May occur when reading data files: another data type has been found where a numerical value is expected.

*Examples:*

```plaintext
declarations
C: set of real
end-declarations

initializations from "test.dat"
C
end-initializations

Contents of test.dat:

```
C: [1 2 c]
```

c is not a number.

**E-28** *Digit expected for constant exponent*
May occur when using scientific notation for real values.

*Examples:*

```plaintext
b := 2E -10
```

E must be immediately followed by a signed integer (i.e. no spaces).

**E-29** *Wrong file descriptor number for selection (num)*
fselect is used with an incorrect parameter value.

**E-34** *I cannot find IO driver ‘driver’*
The system cannot locate the IO driver `driver` for opening a file. This may happen if the driver is provided by a module not already loaded in memory. To avoid this problem the module name should be given with the driver name. For instance use "mmdbc.odbc" instead of "odbc" alone.

**E-35** *Error when closing file ‘file’ (driver_error)*
An error occurred while closing a file. Typically the last write operation for clearing buffers failed.
E-36  **Read error (file)**  
I/O error during file reading.

E-37  **Invalid character**  
Invalid character sequence found while reading a text file, non-conforming to the current encoding. Possibly an incorrect encoding (Mosel default is UTF-8) has been specified for accessing this file.

E-38  **Unknown compiler flag(s) ‘flag’ ignored**  
Some of the flag(s) used with `compile` have not been recognized, please refer to the list documented for `compile`.

E-39  **Unknown BIM reader flag(s) ‘flag’ ignored**  
Some of the flag(s) used with `load` have not been recognized, please refer to the list documented for `load`.

E-40  **Unsupported encoding ‘encoding’ (ignored)**  
The encoding name specified after the marker `!@encoding` is unknown.

### C.2 Parser/compiler errors

Whenever possible Mosel displays the location where an error has been detected during compilation in the format `(line_number/character_position_in_line).

E-100  **Syntax error before token**  
The parser cannot continue to analyze the source file because it has encountered an unexpected token. When the error is not an obvious syntax error, make sure you are not using an identifier that has not been defined before.

*Examples:*

```
token: )
    writeln(3 mod)
```

*mod* must be followed by an integer (or a numerical expression evaluating to an integer).

```
token: write
    if i > 0
        write("greater")
    end-if
```

*then* has been omitted.

```
token: end
    if i > 0 then write("greater") end-if
```

A semicolon must be added to indicate termination of the statement preceeding the `end-if`.

E-101  **Incompatible types (type_of_problem)**  
We try to apply an operation to incompatible types. Check the types of the operands.

*Examples:*

```
type_of_problem: assignment
    i:=0
    i:=1.5
```
The first assignment defines \( i \) as an integer, the second tries to re-assign it a real value: \( i \) needs to be explicitly declared as a real.

**type_of_problem**: cmp

\[ 12=1=2 \]

A truth value (the result of \( 12=1 \)) is compared to a numerical value.

**E-102 Incompatible types for parameters of 'routine'**

A subroutine is called with the wrong parameter type. This message may also be displayed instead of E-104 if a subroutine is called with the wrong number of parameters. (This is due to the possibility to overload the definition of subroutines).

**Examples**:

```plaintext
procedure myprint(a:integer)
  writeln("a: ", a)
endprocedure

myprint(1.5)
```

The subroutine `myprint` is called with a real-valued argument instead of an integer.

**E-103 Incorrect number of subscripts for 'array'(num1/num2)**

An array is used with \( num2 \) subscripts instead of the number of subscripts \( num1 \) indicated at its declaration.

**Examples**:

```plaintext
'array'(num1/num2): 'A'(2/1)
```

**E-104 Incorrect number of parameters for 'routine'(num1/num2)**

Typically displayed if `write` or `read` are used without argument(s).

**E-106 Division by zero detected**

Explicit division by 0 (otherwise error only detected at runtime).

**E-107 Math error detected on function 'fct'**

For example, a negative number is used with a fractional exponent.

**E-108 Logical expression expected here**

Something else than a logical condition is used in an if statement.

**E-109 Trying to redefine 'name'**

Objects can only be defined once, changing their type is not possible.

**Examples**:

```plaintext
i:=0
```

**E-111 Logical expression expected for operator 'op'**

**Examples**:

```plaintext
op: and
```
2+3 and true

E-112  **Numeric expression expected for operator ‘op’**

*Examples:*

```plaintext
op: +

12+(13)
```

```plaintext
op: *

uses "mmxprs"

declarations
x:mpvar
end-declarations

minimize(x*x)
```

Multiplication of decision variables of type mpvar is only possible if a suitable module (like mmnl) supporting non-linear expressions is loaded.

E-113  **Wrong type for conversion**

Mosel performs automatic conversions when required (for instance from an integer to a real) or when explicitly requested by using the type name, e.g. integer(12.5). This error is raised when an unsupported conversion is requested or when no implicit conversion can be applied.

E-114  **Unknown type for constant ‘const’**

A constant is defined but there is not enough information to deduce its type or the type implied cannot be used for a constant (for instance a linear constraint).

E-115  **Expression cannot be passed by reference**

We try to use a constant where an identifier is expected. For instance, only non-constants can be used in an initializations block.

E-118  **Wrong logical operator**

A logical operator is used with a type for which it is not defined.

*Examples:*

```plaintext
if("abc" in "acd") then writeln("?"}; end-if
```

The operator in is not defined for strings.

W-121  **Statement with no effect**

A statement is used that has no effect, for example `r += 0`.

E-122  **Control parameter ‘param’ unknown**

The control parameters of Mosel are documented in the Mosel Reference manual under function `getparam`. All control parameters provided by a module, e.g. `mmxprs`, can be display with the command `EXAM`, e.g. `exam -p mmxprs`. In IVE this information is displayed by the module browser.

E-123  **‘identifier’ is not defined**

`identifier` is used without or before declaring it. Check the spelling of the name. If `identifier` is defined by a module, make sure that the corresponding module is loaded. If `identifier` is a subroutine that is defined later in the program, add a `forward` declaration at the beginning of the model.

E-124  **An expression cannot be used as a statement**

An expression stands where a statement is expected. In this case, the expression is ignored — typically, a constraint has been stated and the constraint type is missing (i.e. `>=` or `<=`...) or an equality constraint occurs without decision variables, e.g. `2=1`. This error also appears when the return value of a function call is not retrieved.
Error messages

E-125  **Set expression expected**
For instance computing the union between an integer constant and a set of integers:
union(12+{13})

E-126  **String expression expected**
A string is expected here: for instance a file name for an initializations block.

E-127  **A function cannot be of type ‘type’**
Some types cannot be the return value of a function. Typically no function can return a
decision variable (type mpvar).

E-128  **Type ‘type’ has no field named ‘field’**
Trying to access an unknown field in a record type.

*Examples:*

```plaintext
declarations
myrec=record
  i,j:integer
end-record
r:myrec
end-declarations
r.k:=0
```

k is not a field of r.

E-129  **Type ‘type’ is not a record**
Trying to use a record dereference on an object that is not a record. For instance using
i.j with i defined as an integer.

E-130  **A type definition cannot be local**
It is not possible to declare a type in a procedure or function.

W-131  **Array ‘identifier’ is not indexed by ranges: assignment may be incorrect**
When performing an inline initialization (operator ::) on an array, it is recommended to
list indices if the indexing sets are not ranges. Indeed, since order of set elements is not
guaranteed the values provided may not be assigned to the expected cells in the array.

*Examples:*

```plaintext
declarations
a:array({3,2,1}) of integer
end-declarations
! a::[3,2,1]  => a(1)=3 a(2)=2 a(3)=1
a::([3,2,1])[3,2,1]  => a(1)=1 a(2)=2 a(3)=3
```

E-132  **Set or list expression expected**
Aggregate operators (like sum or forall) require sets or lists to describe the domains for
their loops.

*Examples:*

```plaintext
declarations
i:integer
end-declarations
forall(i = 2) writeln(i)
```

Since i is declared as an integer before the loop, the expression i=2 is a logical
expression (it checks whether i is equal to 2) instead of an index definition.

W-144  **Symbol ‘identifier’ implicitly declared**
When a model is compiled with option -wi this message gets displayed for every symbol
that is not explicitly declared by the model.

E-147  **Trying to interrupt a non existing loop**
break or next is used outside of a loop.
E-148  **Procedure/function ‘identifier’ declared but not defined**  
A procedure or functions is declared with forward, but no definition of the subroutine body has been found or the subroutine body does not contain any statement.

E-149  **Some requirements are not met**  
A package may declare requirements: these are symbols that must be declared by models using this package. This error occurs when a model uses a package without providing the definitions for all the requirements.

E-150  **End of file inside a commentary**  
A commentary (usually started with (!)) is not terminated. This error may occur, for instance, with several nested commentaries.

E-151  **Incompatible type for subscript num of ‘identifier’**  
The subscript counter num may be wrong if an incorrect number of subscripts is used.  
*Examples:*

```plaintext
declarations
 A:array(1..2,3..4) of integer
end-declarations

writeln(A(1.3))
```

This prints the value 2 for num, although the second subscript is actually missing.

W-152  **Empty set for a loop detected**  
This warning will be printed in a few cases where it is possible to detect an empty set during compilation.

E-153  **Trying to assign the index ‘idx’**  
Loop indices cannot be re-assigned.  
*Examples:*

```plaintext
declarations
 C: set of string
 D: range
end-declarations

forall(d in D) d+=1
forall(c in C) if (c='a') then c:='A'; end-if
```

Both of these assignments will raise the error. To replace an element of the set C, the element needs to be removed and the new element added to the set.

E-154  **Unexpected end of file**  
May occur, for instance, if an expression at the end of the model file is incomplete and in addition end-model is missing.

E-155  **Empty ‘case’**  
A case statement is used without defining any choices.

E-156  **‘identifier’ has no type**  
The type of identifier cannot be deduced. Typically, an undeclared object is assigned an empty set.

E-157  **Scalar expression expected**  
*Examples:*

```plaintext
declarations
 B={'a','b','c'}
end-declarations

case B of
  1: writeln("stop")
```
end-case
The \texttt{case} statement can only be used with the basic types (integer, real, boolean, string).

D:: [1,2]

Declaration of arrays by assignment is only possible if the index set can be deduced (e.g. definition of an array of linear constraints in a loop).

**E-159** \textbf{Compiler option ‘option’ unknown}
Valid compiler options include explterm and noimplicit. See section 2.3.3 for more details.

**E-160** \textbf{Definition of functions and procedures cannot be nested}
May occur, for instance, if \texttt{end-procedure} or \texttt{end-function} is missing and the definition of a second subroutine follows.

**E-161** \textbf{Expressions not allowed as procedure/function parameter}
Occurs typically if the index set(s) of an array are defined directly in the procedure/function prototype.

\textit{Examples:}

\begin{verbatim}
procedure myproc(F:array(1..5) of real)
  writeln("something")
end-procedure
\end{verbatim}

Replace either by \texttt{array(range)} or \texttt{array(set of integer)} or define \texttt{A:=1..5} outside of the subroutine definition and use \texttt{array(A)}

**E-162** \textbf{Non empty string expected here}
This error is raised, for example, by \texttt{uses "}.

**E-163** \textbf{Array declarations in the form of a list are not allowed as procedure/function parameter}
Basic types may be given in the form of a list, but not arrays.

\textit{Examples:}

\begin{verbatim}
procedure myproc(F,G,H:array(range) of real, a,b,c:real)
  writeln("something")
end-procedure
\end{verbatim}

Separate declaration of every array is required:

\begin{verbatim}
procedure myproc(F:array(range) of real, G:array(range) of real, H:array(range) of real, a,b,c:real)
\end{verbatim}

**W-164** \textbf{A local symbol cannot be made public}

\textit{Examples:}

\begin{verbatim}
procedure myproc
  declarations
  public i:integer
end-declarations
  i:=1
end-procedure
\end{verbatim}

Any symbol declared in a subroutine is local and cannot be made public.

**E-165** \textbf{Declaration of ‘identifier’ hides a parameter}
The name of a function/procedure parameter is re-used in a local declaration.

\textit{Examples:}
procedure myproc(D:array(range) of real)
declarations
  D: integer
end-declarations
writeln(D)
end-procedure

Rename either the subroutine argument or the name used in the declaration.

W-166 ';
missing at end of statement
If the option explterm is employed, then all statements must be terminated by a
semicolon.

E-167 Operator 'op' not defined
A constructor for a type is used in a form that is not defined.

Examples:

  uses "complex"
  c:=complex(1,2,3)

  The module complex defines constructors for complex numbers from one or two
  reals, but not from three.

E-168 'something' expected here
Special case of "syntax error" (E-100) where the parser is able to provide a guess of what
is missing.

Examples:

  something: :=
    a: 3

  The assignment is indicated by :=.

  something: of

  declarations
    S: set integer
  end-declarations

  of has been omitted.

  something: ...

  declarations
    A: array(1:2) of integer
  end-declarations

  Ranges are specified by ...

E-169 'identifier' cannot be used as an index name (the identifier is already in use or declared)

Examples:

  i:=0
  sum(i in 1..10)

  The identifier i has to be replaced by a different name in one of these lines.

E-170 '=' expects a scalar here (use 'in' for a set)
Special case of syntax error (E-100).

Examples:

  sum(i = 1..10)

  Replace = by in.

E-171 The [upper/lower] bound of a range is not an integer expression

Examples:
declarations
A: array(1..2.5) of integer
end-declarations

Ranges are intervals of integers, so the upper bound of the index range must be changed to either 2 or 3.

E-172 Only a reference to a public set is allowed here
All index sets of a public array must also be public.

E-173 Statement allowed in packages only
The block requirements can only be used in packages.

E-175 Index sets of array types must be named
User types defined as arrays must be indexed by named sets (i.e. declared separately).
For instance it is not allowed to use range or set of string as an index of such an array.

E-176 Only a public type is allowed here
If a user type depending on another user type is declared declared public, the secondary type must also be public.
For instance, assuming type T1 is private, it is not possible to declare T2 as a public T2=set of T1.

E-177 Incorrect number of initializers (n1/n2)
In an inline initialization (operator ::) the number of provided values to assign does not match the list of indices.

E-202 Integer constant expected
Versions numbers (stated by means of the version compiler directive) must consist in 1 to 3 numbers separated by dots (e.g. 1.2.3). This error is displayed if a version number does not conform to this syntax.

E-207 Problem reference/type expected here
The operator with is used with something that is not a problem.

E-208 There can be only one counter
The as counter declaration can appear only once in an iterator list.

E-209 Missing loop indices
Typically an iterator list contains only a counter declaration: it is necessary to provide at least one index.

E-210 String starting at line line is unfinished
A multiline string is not correctly terminated with the matching end marker.

E-211 Invalid annotation syntax (ignored)
Malformed annotation that cannot be identified (e.g. containing .. or invalid characters—only alphanumeric and underscore are allowed in annotation names).

E-212 Annotations: invalid path ‘name’
Some portion of the path forming an annotation identifier, e.g. cat1.cat2 is the path for the annotation !@cat1.cat2.name, cannot be accessed.

E-213 Annotations: name ‘name’ not found
Some portion of the path forming an annotation identifier, e.g. cat1.cat2 is the path for the annotation !@cat1.cat2.name, is not defined.

E-214 Annotations: trying to redefine ‘name’ (ignored)
An annotation can only be defined once.

E-215 Annotations: invalid definition string for ‘name’ (value)
Incorrect or incomplete definition declaration in an @mc.def statement, such as duplicate or missing property or value, use of an unknown keyword. Please refer to the list of permissible declaration statements in Section 2.13.3.
E-217  **Annotations: wrong value ‘value’ for ’name’ (expecting: value2)**
An annotation is assigned a value that does not correspond to the value type or set of values that have been specified in its declaration (via @mc.def).

E-218  **Annotations: missing chapter for ‘name’**
moseldoc is trying to add a documentation entry under a chapter or section that has not (yet) been defined.

### C.2.1 Errors related to modules

**E-302**  **The symbol ‘identifier’ from ‘module’ cannot be defined (redefinition)**
Two different modules used by a model define the same symbol (incompatible definitions).

**E-303**  **Wrong type for symbol ‘identifier’ from ‘module’**
Internal error in the definition of a user module (an unknown type is used): refer to the list of type codes in the Native Interface reference manual.

**W-306**  **Unknown operator ‘op’ (code num) in module ‘module’**
Internal error in the definition of a user module: refer to the list of operator codes in the Native Interface reference manual.

**E-307**  **Operator ‘op’ (code num) from module ‘module’ rejected**
Internal error in the definition of a user module: an operator is not defined correctly.

**E-308**  **Parameter string of a native routine corrupted**
Internal error in the definition of a user module: refer to the list of parameter type codes in the Native Interface reference manual.

**W-309**  **Problem type ‘typ’ unknown: extension ‘ext’ ignored**
A module declares a native type as a problem extension but the compiler cannot find the base type. For instance the new type is named "myprob.pb" but "myprob" does not exist.

### C.2.2 Errors related to packages

**E-320**  **Package ‘package’ not found**
A package has not been found in the module path (see section 2.3.1 for the search rules).

**E-321**  **‘file’ is not a package**
Typically displayed if a model is used as a package (the source for the bim file starts with the model keyword instead of package).

A model is compiled with package A depending on a package B. The bim file Mosel has loaded for B is not compatible with the one used for compiling A (found version num1.num2.num3, required version is num4.num5.num6).

**E-323**  **Package ‘package’ imported several times**
A package cannot be imported several times in a model. This error occurs usually when a model uses packages A and B, and package B already includes A.

### C.3 Runtime errors

Runtime errors are usually displayed without any information about where they have occurred. To obtain the location of the error, use the flag g with the compile, cload, or execute command.
C.3.1 Initializations

E-30  **Duplicate label ‘label’ at line num of file ‘file’ (ignored)**
The same label is used repeatedly in a data file.

*Examples:*

```
D: [1 2 3]
D: [1 2 4]
```

E-31  **Error when reading label ‘label’ at (num1,num2) of file ‘file’**
The data entry labeled *label* has not been read correctly. Usually this message is preceded by a more detailed one, e.g. E-24, E-27 or E-28.

E-32  **Error when writing label ‘label’ at (num1,num2) of file ‘file’**
The data entry labeled *label* has not been written correctly. Usually this message is preceded by a more detailed one, e.g. E-23.

E-33  **Initialization with file ‘file’ failed for: list_of_identifier**
Summary report at the end of an initializations section. Usually this message is preceded by more detailed ones, e.g. E-27, E-28, E-30, E-31.

C.3.2 General runtime errors

E-6  **Number of running concurrent models authorized by license is exceeded**
A program or model is trying to use more (sub)models than what is authorized by the licence.

E-51  **Division by zero**
Division by 0 resulting from the evaluation of an expression.

E-52  **Math error performing function ‘identifier’**
For example *ln* used with inadmissible argument, such as 0 or negative values.

E-72  **Not a runnable model (main procedure not found)**
Most likely, you are trying to execute a ‘package’ as if it were a ‘model’.

E-1000  **Inconsistent range**
Typically displayed if the lower bound specified for a range is greater than its upper bound.

*Examples:*

```
D:=3..-1
```

E-1001  **Conflicting types in set operation (op)**
A set operation can only be carried out between sets of the same type.

*Examples:*

```
declarations
C: set of integer
D: range
end-declarations

C:={5,7}
D:=C
```

The inverse, C:=D, is correct because ranges are a special case of sets of integers.

E-1002  **An index is out of range**
An attempt is being made to access an array entry that lies outside of the index sets of the array.
E-1003 **Trying to modify a finalized or fixed set**
Occurs, for instance, when it is attempted to re-assign a constant set or to add elements to a fixed set.

E-1004 **Trying to access an uninitialized object** (type_of_object)
Occurs typically in models that define subroutines.

*Examples:*

```fortran
type_of_object: array
forward procedure myprint
myprint
declarations
A::array(1..2,3..4) of integer
end-declarations

procedure myprint
writeln(A(1,2))
end-procedure
```

Move the declaration of A before the call of the subroutine

E-1005 **Wrong type for “procedure”**
Occurs when procedures `settype` or `getvars` are used with incorrect types.

E-1006 **Null reference** (internal_function)
This error is a special case of E-1004 when the problem is detected on an external type or scalar (e.g. accessing a record field on an object that has not been initialized).

E-1009 **Too many initializers**
The number of data elements exceeds the maximum size of an array.

*Examples:*

```fortran
declarations
A::array(1..3) of integer
end-declarations
A::[1,2,3,4]
```

E-1010 **Trying to extend a unary constraint**
Most types of unary constraints cannot be transformed into constraints on several variables.

*Examples:*

```fortran
declarations
x,y: mpvar
end-declarations

c:=x is_integer
c+=y
```

E-1013 **Infeasible constraint**
The simple cases of infeasible unnamed constraints that are detected at run time include:

*Examples:*

```fortran
declarations
x:mpvar
end-declarations
i:=-1
if(i>=0,x,0)>=1
! or:
x-x>=1
```
E-1014 **Conflicting types in array operation (op)**
An array operation (like assignment) can only be carried out between arrays of the same type and structure.

E-1015 **Trying to modify a constant list**
Occurs, for instance, when it is attempted to apply a destructive operation (like splittail) to a constant list.

E-1016 **Trying to get an element in an empty set**
The function getfirst or getlast is applied to an empty set.

E-1017 **Trying to get an element in an empty list**
The function getfirst or getlast is applied to an empty list.

E-1018 **Invalid identifier ‘identifier’ for publish**
The publish command has received an invalid identifier name (e.g. not a valid Mosel identifier or the name is already in use as Mosel identifier).

E-1100 **Empty problem**
We are trying to generate or load an empty problem into a solver (i.e. no constraints; bounds do not count as constraints).

E-1102 **Problem capacity of student license exceeded (num1 type_of_object > num2)**
The problem is too large to be solved with a student license. Use a smaller data set or try to reformulate the problem to reduce the number of variables, constraints, or global entities.

E-1103 **Too many matrix coefficients**
Matrix size exceeds machine capacity: for 32bit versions the limit are 2billion ($2 \cdot 10^{10}$) elements.

C.3.3 BIM reader

E-80 **‘file’ is not a BIM file**
Trying to load a file that does not have the structure of a BIM file.

E-82 **Wrong file version (current: num1 / required: num2) for file ‘file’**
A BIM file is loaded with an incompatible version of Mosel: preferably the same versions should be used for generating and running a BIM file.

E-83 **Bim file ‘file’ corrupted**
A BIM file has been corrupted, e.g. by saving it with a text editor.

E-84 **File ‘file’: model cannot be renamed**
A model file that is being executed cannot be re-loaded at the same time.

W-85 **Trailing data at end of file ‘file’ ignored**
At the end of a BIM file additional, unidentifiable data has been found (may be a sign of file corruption).

E-88 **Bim file ‘file’ corrupted**
Incomplete or otherwise damaged BIM file.

E-90 **Signature error (description)**
During the generation of a BIM file, a problem with the signature has occurred.

E-91 **Signature verification error (description)**
While reading a BIM file, a problem with the signature has occurred (e.g. trying to check signature for a file that is not signed; or the keys that have been employed don’t match).
**E-92** *Encryption error (description)*  
Problem with encryption during the generation of a BIM file (e.g. invalid or missing key).

**E-93** *Decryption error (description)*  
Problem with decryption while reading a BIM file (e.g. invalid or missing key).

### C.3.4 Module manager errors

**E-350** *Module ‘module’ not found*  
A module has not been found in the module path (see section 2.3.1 for the search rules).  
This message is also displayed, if a module depends on another library that has not been found (e.g. module *mmxprs* has been found but Xpress Optimizer has not been installed or cannot be located by the operating system).

**E-351** *File ‘file’ is not a Mosel DSO*  
Typically displayed if Mosel cannot find the module initialization function.

**E-352** *Module ‘module’ requires a more recent version of Mosel (unsupported interface)*  
A module is not compatible with the Mosel version used to load it.

**E-353** *Module ‘module’ disabled by restrictions*  
Module *module* either does not implement restriction handling at all or it requires features that are not authorized.  
*Examples:*  

```
mmxprs will fail with the restriction setting NoTmp
```

**E-354** *Error when initializing module ‘module’*  
Usually preceded by an error message generated by the module. Please refer to the documentation of the module for further detail.

**E-355** *Wrong version for module ‘module’ (using:num1.num2.num3/required:num4.num5.num6)*  
A model is run with a version of a module that is different from the version that has been used to compile the model (trying to run with version *num1.num2.num3*, required version is *num4.num5.num6*).

**E-358** *Error when resetting module ‘module’*  
A module cannot be executed (e.g. due to a lack of memory).

**E-359** *Driver ‘pkg.driver’ rejected (reason)*  
A module publishes an IO driver which name is invalid or that is missing some mandatory function.

**E-360** *Control parameter ‘module.param’ unknown (setting ignored)*  
It is possible to set module parameters when running a model (using the *RUN* command for instance): in the list of assignments, a control parameter cannot be found in the indicated module.

**E-361** *Version number truncated (‘vernum’)*  
A version number (for module, model or package) consists in three positive numbers *a.b.c*. This error is raised if one of these numbers is larger than 999.

**E-362** *The operating system failed to load file ‘file’ (‘description’)*  
The module file has been found but cannot be loaded by the system—there will typically be some system error message indicating the exact cause, such as wrong architecture (e.g. using a library compiled for Windows under Linux) or missing additional files.
APPENDIX D

Contacting FICO

FICO provides clients with support and services for all our products. Refer to the following sections for more information.

Product support

FICO offers technical support and services ranging from self-help tools to direct assistance with a FICO technical support engineer. Support is available to all clients who have purchased a FICO product and have an active support or maintenance contract. You can find support contact information on the Product Support home page (www.fico.com/support).

On the Product Support home page, you can also register for credentials to log on to FICO Online Support, our web-based support tool to access Product Support 24x7 from anywhere in the world. Using FICO Online Support, you can enter cases online, track them through resolution, find articles in the FICO Knowledge Base, and query known issues.

Please include ‘Xpress’ in the subject line of your support queries.

Product education

FICO Product Education is the principal provider of product training for our clients and partners. Product Education offers instructor-led classroom courses, web-based training, seminars, and training tools for both new user enablement and ongoing performance support. For additional information, visit the Product Education homepage at www.fico.com/en/product-training or email producteducation@fico.com.

Product documentation

FICO continually looks for new ways to improve and enhance the value of the products and services we provide. If you have comments or suggestions regarding how we can improve this documentation, let us know by sending your suggestions to techpubs@fico.com.
Contacting FICO

Sales and maintenance

USA, CANADA AND ALL AMERICAS
Email: XpressSalesUS@fico.com

WORLDWIDE
Email: XpressSalesUK@fico.com
Tel: +44 207 940 8718
Fax: +44 870 420 3601

Xpress Optimization, FICO
FICO House
International Square
Starley Way
Birmingham B37 7GN
UK

Related services

Strategy Consulting: Included in your contract with FICO may be a specified amount of consulting time to assist you in using FICO Optimization Modeler to meet your business needs. Additional consulting time can be arranged by contract.

Conferences and Seminars: FICO offers conferences and seminars on our products and services. For announcements concerning these events, go to www.fico.com or contact your FICO account representative.

About FICO

FICO (NYSE:FICO) delivers superior predictive analytics solutions that drive smarter decisions. The company’s groundbreaking use of mathematics to predict consumer behavior has transformed entire industries and revolutionized the way risk is managed and products are marketed. FICO’s innovative solutions include the FICO® Score—the standard measure of consumer credit risk in the United States—along with industry-leading solutions for managing credit accounts, identifying and minimizing the impact of fraud, and customizing consumer offers with pinpoint accuracy. Most of the world’s top banks, as well as leading insurers, retailers, pharmaceutical companies, and government agencies, rely on FICO solutions to accelerate growth, control risk, boost profits, and meet regulatory and competitive demands. FICO also helps millions of individuals manage their personal credit health through www.myfico.com. Learn more at www.fico.com. FICO: Make every decision count™.
Index

Symbols

!  14
!, 14
!0, 47
(!, 14
(!0, 47
*, 28, 29
+, 28, 29
+, 15, 28, 29
+=, 32
., 15
., 28, 29
., 15, 28, 29
-=, 32
., 537
., 29, 537
/  28
//, 537
::, 33
:=, 31
;, 15
;, 15, 31
=, 29, 30
=, 29, 30
@, 47
_., 14, 59
_<c, 59

Numbers
1  619
2  619
3  619
4  619
5  619
6  619
7  619
8  619
9  619
10  619
11  619
12  619
13  619
14  619
15  619
16  619
17  619
18  620
19  620
20  620
21  620
22  620
23  620
24  620
25  620
26  620
27  620
28  620
29  620
30  622

A
abs, 63, 264, 538
absolute value, 63, 264
access mode, 455
activity, 96
robust constraint, 338
add
array of cuts, 707
cut, 706
image, 388, 389
addcut, 706
addcuts, 707
adddmipsol, 628
addmonths, 431
addmultistart, 599
addnode, 542
aggregate operator, 27
alias, 203
define, 205
delete, 201
list, 204
and, 15, 30, 622
annotation, 46
category, 46
current category, 47
documentation, 54, 57
global, 48
multi-line, 47
name, 46
predefined categories, 46
property, 48
annual discount, 155
anonymous constraint, 33
append
file, 92
arccos, 264
arcsin, 264
arctan, 64, 264
arguments, 40
arithmetic expression, 28
array, 22
automatic, 31
compare, 29
create a cell, 75
declaration, 22
delete a cell, 80edereference, 25
<table>
<thead>
<tr>
<th>Term</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>break</td>
<td>15, 39</td>
</tr>
<tr>
<td>BREAKPOINTS</td>
<td>7</td>
</tr>
<tr>
<td>BREAKSUB</td>
<td>7</td>
</tr>
<tr>
<td>buffer size</td>
<td>279, 298</td>
</tr>
<tr>
<td>ByteBuffer</td>
<td>192</td>
</tr>
<tr>
<td>C</td>
<td>calcosalinfo, 630</td>
</tr>
<tr>
<td>callback</td>
<td>687</td>
</tr>
<tr>
<td>canceltimer</td>
<td>233</td>
</tr>
<tr>
<td>cardinality</td>
<td>336</td>
</tr>
<tr>
<td>cascading</td>
<td>608</td>
</tr>
<tr>
<td>case</td>
<td>15, 37</td>
</tr>
<tr>
<td>case-insensitive</td>
<td>6</td>
</tr>
<tr>
<td>case-sensitive</td>
<td>15</td>
</tr>
<tr>
<td>casting</td>
<td>26</td>
</tr>
<tr>
<td>CDATA</td>
<td>535</td>
</tr>
<tr>
<td>cdata</td>
<td>537</td>
</tr>
<tr>
<td>ceil</td>
<td>73, 264</td>
</tr>
<tr>
<td>ceiling</td>
<td>538</td>
</tr>
<tr>
<td>certificate</td>
<td></td>
</tr>
<tr>
<td>client</td>
<td>352</td>
</tr>
<tr>
<td>server</td>
<td>352</td>
</tr>
<tr>
<td>certificate file</td>
<td>351</td>
</tr>
<tr>
<td>character encoding</td>
<td>51, 549, 576</td>
</tr>
<tr>
<td>chgdeltatype</td>
<td>600</td>
</tr>
<tr>
<td>child</td>
<td>537</td>
</tr>
<tr>
<td>child model</td>
<td>234</td>
</tr>
<tr>
<td>cipher algorithm</td>
<td>349</td>
</tr>
<tr>
<td>class</td>
<td></td>
</tr>
<tr>
<td>event</td>
<td>232</td>
</tr>
<tr>
<td>clear error</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>736</td>
</tr>
<tr>
<td>clearaliases</td>
<td>201</td>
</tr>
<tr>
<td>clearinitvals</td>
<td>265</td>
</tr>
<tr>
<td>clearmipdir</td>
<td>632</td>
</tr>
<tr>
<td>clearqexpstat</td>
<td>330</td>
</tr>
<tr>
<td>client certificate</td>
<td>352</td>
</tr>
<tr>
<td>client private key</td>
<td>352</td>
</tr>
<tr>
<td>close</td>
<td></td>
</tr>
<tr>
<td>file</td>
<td>52, 86</td>
</tr>
<tr>
<td>stream</td>
<td>86</td>
</tr>
<tr>
<td>coefficient</td>
<td>97</td>
</tr>
<tr>
<td>set</td>
<td>136</td>
</tr>
<tr>
<td>color</td>
<td>398</td>
</tr>
<tr>
<td>column order</td>
<td>623</td>
</tr>
<tr>
<td>command</td>
<td></td>
</tr>
<tr>
<td>Optimizer</td>
<td>633</td>
</tr>
<tr>
<td>system</td>
<td>513</td>
</tr>
<tr>
<td>command</td>
<td>633</td>
</tr>
<tr>
<td>command line interpreter</td>
<td>2</td>
</tr>
<tr>
<td>commands</td>
<td></td>
</tr>
<tr>
<td>shortening</td>
<td>10</td>
</tr>
<tr>
<td>comment</td>
<td>8, 14, 35, 535</td>
</tr>
<tr>
<td>sign</td>
<td>94</td>
</tr>
<tr>
<td>skip</td>
<td>94</td>
</tr>
<tr>
<td>commentary</td>
<td>537</td>
</tr>
<tr>
<td>multi-line</td>
<td>14</td>
</tr>
<tr>
<td>body</td>
<td></td>
</tr>
<tr>
<td>model</td>
<td>16</td>
</tr>
<tr>
<td>boolean</td>
<td>15, 21, 26, 538</td>
</tr>
<tr>
<td>Boolean expression</td>
<td>29</td>
</tr>
<tr>
<td>BREAK</td>
<td>6</td>
</tr>
<tr>
<td>dynamic</td>
<td>22</td>
</tr>
<tr>
<td>in/output</td>
<td>156</td>
</tr>
<tr>
<td>array</td>
<td>15</td>
</tr>
<tr>
<td>as</td>
<td>15</td>
</tr>
<tr>
<td>asproc</td>
<td>65</td>
</tr>
<tr>
<td>assert</td>
<td>66</td>
</tr>
<tr>
<td>assign R object</td>
<td>732</td>
</tr>
<tr>
<td>assignment</td>
<td>31</td>
</tr>
<tr>
<td>additive</td>
<td>32</td>
</tr>
<tr>
<td>constraints</td>
<td>32</td>
</tr>
<tr>
<td>subtractive</td>
<td>32</td>
</tr>
<tr>
<td>asymmetric ciphers</td>
<td>349</td>
</tr>
<tr>
<td>asynchronous HTTP request</td>
<td>165</td>
</tr>
<tr>
<td>attribute</td>
<td></td>
</tr>
<tr>
<td>delete</td>
<td>545</td>
</tr>
<tr>
<td>get first</td>
<td>552</td>
</tr>
<tr>
<td>get value</td>
<td>547</td>
</tr>
<tr>
<td>set value</td>
<td>575</td>
</tr>
<tr>
<td>test existence</td>
<td>548</td>
</tr>
<tr>
<td>attribute</td>
<td>537</td>
</tr>
<tr>
<td>autofinal</td>
<td>107, 141</td>
</tr>
<tr>
<td>axis specifer</td>
<td>536, 537</td>
</tr>
<tr>
<td>B</td>
<td></td>
</tr>
<tr>
<td>base64</td>
<td>376</td>
</tr>
<tr>
<td>basic type</td>
<td>21</td>
</tr>
<tr>
<td>basis</td>
<td></td>
</tr>
<tr>
<td>load</td>
<td>663</td>
</tr>
<tr>
<td>read</td>
<td>670</td>
</tr>
<tr>
<td>reset</td>
<td>677</td>
</tr>
<tr>
<td>save</td>
<td>680</td>
</tr>
<tr>
<td>status</td>
<td>639, 686</td>
</tr>
<tr>
<td>write</td>
<td>700</td>
</tr>
<tr>
<td>basis</td>
<td>621</td>
</tr>
<tr>
<td>basis stability</td>
<td>629</td>
</tr>
<tr>
<td>basisstability</td>
<td>629</td>
</tr>
<tr>
<td>BCONDITION</td>
<td>6</td>
</tr>
<tr>
<td>best bound</td>
<td>622</td>
</tr>
<tr>
<td>BIM</td>
<td>1, 3</td>
</tr>
<tr>
<td>documentation enabled</td>
<td>54</td>
</tr>
<tr>
<td>encryption</td>
<td>3</td>
</tr>
<tr>
<td>signed</td>
<td>3</td>
</tr>
<tr>
<td>bitflip</td>
<td>67</td>
</tr>
<tr>
<td>bitneg</td>
<td>68</td>
</tr>
<tr>
<td>bitset</td>
<td>69</td>
</tr>
<tr>
<td>bitshift</td>
<td>70</td>
</tr>
<tr>
<td>bittest</td>
<td>71</td>
</tr>
<tr>
<td>bitval</td>
<td>72</td>
</tr>
<tr>
<td>bitwise</td>
<td></td>
</tr>
<tr>
<td>and</td>
<td>71</td>
</tr>
<tr>
<td>not</td>
<td>68</td>
</tr>
<tr>
<td>or</td>
<td>69</td>
</tr>
<tr>
<td>shift</td>
<td>70</td>
</tr>
<tr>
<td>value</td>
<td>72</td>
</tr>
<tr>
<td>xor</td>
<td>67</td>
</tr>
<tr>
<td>body</td>
<td></td>
</tr>
<tr>
<td>model</td>
<td>16</td>
</tr>
<tr>
<td>boolean</td>
<td>15, 21, 26, 538</td>
</tr>
<tr>
<td>Boolean expression</td>
<td>29</td>
</tr>
<tr>
<td>BREAK</td>
<td>6</td>
</tr>
<tr>
<td>Term</td>
<td>Page Numbers</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>communication interface</td>
<td>325, 524</td>
</tr>
<tr>
<td>comparator</td>
<td>29</td>
</tr>
<tr>
<td>compile</td>
<td></td>
</tr>
<tr>
<td>model</td>
<td>3</td>
</tr>
<tr>
<td>compile</td>
<td>208</td>
</tr>
<tr>
<td>compiled</td>
<td>16</td>
</tr>
<tr>
<td>compiler annotations</td>
<td>46</td>
</tr>
<tr>
<td>compiler directives</td>
<td>16</td>
</tr>
<tr>
<td>compiler library</td>
<td>2</td>
</tr>
<tr>
<td>compiler options</td>
<td>18</td>
</tr>
<tr>
<td>concatenation</td>
<td>28</td>
</tr>
<tr>
<td>list</td>
<td>29</td>
</tr>
<tr>
<td>condition</td>
<td>27</td>
</tr>
<tr>
<td>configuration directory</td>
<td>350</td>
</tr>
<tr>
<td>path</td>
<td>354</td>
</tr>
<tr>
<td>configuration file</td>
<td>258</td>
</tr>
<tr>
<td>connect</td>
<td>283, 305</td>
</tr>
<tr>
<td>connect</td>
<td>199</td>
</tr>
<tr>
<td>connection</td>
<td></td>
</tr>
<tr>
<td>number</td>
<td>280, 299</td>
</tr>
<tr>
<td>connection template</td>
<td>195</td>
</tr>
<tr>
<td>connector</td>
<td>25</td>
</tr>
<tr>
<td>constmpl</td>
<td>195</td>
</tr>
<tr>
<td>constant</td>
<td>24</td>
</tr>
<tr>
<td>compile time</td>
<td>24</td>
</tr>
<tr>
<td>definition</td>
<td>24</td>
</tr>
<tr>
<td>run time</td>
<td>24</td>
</tr>
<tr>
<td>constants</td>
<td>62</td>
</tr>
<tr>
<td>constraint</td>
<td>21</td>
</tr>
<tr>
<td>activity</td>
<td>96</td>
</tr>
<tr>
<td>anonymous</td>
<td>33</td>
</tr>
<tr>
<td>coefficient</td>
<td>97, 98</td>
</tr>
<tr>
<td>dual</td>
<td>99</td>
</tr>
<tr>
<td>hide</td>
<td>137</td>
</tr>
<tr>
<td>name</td>
<td>33, 140</td>
</tr>
<tr>
<td>ranging information</td>
<td>652</td>
</tr>
<tr>
<td>right hand side</td>
<td>31</td>
</tr>
<tr>
<td>sensivity ranges</td>
<td>653</td>
</tr>
<tr>
<td>set coefficient</td>
<td>136</td>
</tr>
<tr>
<td>set of variables</td>
<td>116</td>
</tr>
<tr>
<td>set type</td>
<td>143</td>
</tr>
<tr>
<td>slack</td>
<td>112</td>
</tr>
<tr>
<td>test hidden</td>
<td>120</td>
</tr>
<tr>
<td>type</td>
<td>30, 115, 343</td>
</tr>
<tr>
<td>contains</td>
<td>538</td>
</tr>
<tr>
<td>context</td>
<td>325, 524</td>
</tr>
<tr>
<td>problem</td>
<td>43</td>
</tr>
<tr>
<td>context node</td>
<td>536</td>
</tr>
<tr>
<td>CONTINUE</td>
<td>7</td>
</tr>
<tr>
<td>control parameter</td>
<td>19</td>
</tr>
<tr>
<td>get</td>
<td>106</td>
</tr>
<tr>
<td>Mosel instance</td>
<td>748</td>
</tr>
<tr>
<td>set</td>
<td>141</td>
</tr>
<tr>
<td>convergence tolerance</td>
<td>594</td>
</tr>
<tr>
<td>conversion</td>
<td></td>
</tr>
<tr>
<td>basic type</td>
<td>26</td>
</tr>
<tr>
<td>copy</td>
<td></td>
</tr>
<tr>
<td>file</td>
<td>437</td>
</tr>
<tr>
<td>copynode</td>
<td>544</td>
</tr>
<tr>
<td>copysoltoinit</td>
<td>266, 634</td>
</tr>
<tr>
<td>copytext</td>
<td>432</td>
</tr>
<tr>
<td>cos</td>
<td>74, 264</td>
</tr>
<tr>
<td>count</td>
<td>15, 27</td>
</tr>
<tr>
<td>counter</td>
<td>15</td>
</tr>
<tr>
<td>create</td>
<td></td>
</tr>
<tr>
<td>directory</td>
<td>477, 478</td>
</tr>
<tr>
<td>create</td>
<td>23, 75</td>
</tr>
<tr>
<td>cross recursion</td>
<td>42</td>
</tr>
<tr>
<td>crypt</td>
<td>376</td>
</tr>
<tr>
<td>csv</td>
<td>348</td>
</tr>
<tr>
<td>CT_BIN</td>
<td>115, 143</td>
</tr>
<tr>
<td>CT_CARD</td>
<td>343</td>
</tr>
<tr>
<td>CT_CONT</td>
<td>115, 143</td>
</tr>
<tr>
<td>CT_EQ</td>
<td>115, 143, 271, 273, 343, 345</td>
</tr>
<tr>
<td>CT_FREE</td>
<td>115, 143</td>
</tr>
<tr>
<td>CT_GEQ</td>
<td>115, 143, 271, 273, 343, 345</td>
</tr>
<tr>
<td>CT_INT</td>
<td>115, 143</td>
</tr>
<tr>
<td>CT_LEQ</td>
<td>115, 143, 271, 273, 343, 345</td>
</tr>
<tr>
<td>CT_PINT</td>
<td>115, 143</td>
</tr>
<tr>
<td>CT_SCEN</td>
<td>343</td>
</tr>
<tr>
<td>CT_SEC</td>
<td>115, 143</td>
</tr>
<tr>
<td>CT_SINT</td>
<td>115, 143</td>
</tr>
<tr>
<td>CT_SOS1</td>
<td>115, 143</td>
</tr>
<tr>
<td>CT_SOS2</td>
<td>115, 143</td>
</tr>
<tr>
<td>CT_UNB</td>
<td>115, 143, 271, 273, 343, 345</td>
</tr>
<tr>
<td>currentdate</td>
<td>76</td>
</tr>
<tr>
<td>currenttime</td>
<td>77</td>
</tr>
<tr>
<td>cut</td>
<td></td>
</tr>
<tr>
<td>add</td>
<td>706</td>
</tr>
<tr>
<td>add array</td>
<td>707</td>
</tr>
<tr>
<td>delete</td>
<td>708</td>
</tr>
<tr>
<td>drop</td>
<td>709</td>
</tr>
<tr>
<td>get active</td>
<td>710</td>
</tr>
<tr>
<td>list from cut pool</td>
<td>711</td>
</tr>
<tr>
<td>load</td>
<td>712</td>
</tr>
<tr>
<td>store</td>
<td>713</td>
</tr>
<tr>
<td>store array</td>
<td>714</td>
</tr>
<tr>
<td>cuthead</td>
<td>78</td>
</tr>
<tr>
<td>cuttext</td>
<td>433</td>
</tr>
</tbody>
</table>

**D**

<table>
<thead>
<tr>
<th>Term</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA</td>
<td>535</td>
</tr>
<tr>
<td>data</td>
<td></td>
</tr>
<tr>
<td>initialization</td>
<td>34</td>
</tr>
<tr>
<td>input</td>
<td>156</td>
</tr>
<tr>
<td>local</td>
<td>40</td>
</tr>
<tr>
<td>output</td>
<td>156</td>
</tr>
<tr>
<td>read</td>
<td>132</td>
</tr>
<tr>
<td>save</td>
<td>35</td>
</tr>
<tr>
<td>data</td>
<td>537</td>
</tr>
<tr>
<td>database</td>
<td></td>
</tr>
<tr>
<td>connect</td>
<td>283, 305</td>
</tr>
<tr>
<td>disconnect</td>
<td>307</td>
</tr>
<tr>
<td>logoff</td>
<td>284</td>
</tr>
<tr>
<td>transaction</td>
<td>304, 317</td>
</tr>
<tr>
<td>date</td>
<td>76</td>
</tr>
<tr>
<td>date</td>
<td>421</td>
</tr>
<tr>
<td>datefmt</td>
<td>423</td>
</tr>
<tr>
<td>datetime</td>
<td>422</td>
</tr>
</tbody>
</table>
Index

datetimefmt, 424
debug mode
  OCI, 280
  ODBC, 299
debugger, 6
declaration, 16, 20
  forward, 42
  implicit, 32
  list, 22
  record, 23
  set, 21
declarations, 15, 20
declarative, 16
decrypt
  private key, 362
defaultnode, 196
defdelayedrows, 635
deflate, 738
defsecurevecs, 636
delattr, 545
delayed rows, 635
  nonlinear, 607
delcell, 80
delcuts, 708
DELETE, 7
delete
  cut, 708
  directives, 631
  directory, 498
  file, 438
  model cuts, 632
delnodes, 546
delta variable, 600
deltext, 434
dense, 280, 300
dependency
  module, 325, 524
  service, 325, 524
deploy executable, 53
descendant, 537
descendant-or-self, 537
detach, 210
determining row, 608
difference, 29, 30
  list, 29
  set, 29
  string, 28
digital signature, see electronic signature
directive, 693
  delete, 631
directives
  read, 671
  write, 701
directory, 446
  access mode, 455
  create, 477, 478
  delete, 498
  new, 477, 478
  remove, 498
  status, 455
disc, 155
disconnect, 284, 307
disconnect, 200
diskdata, 156, 157
DISPLAY, 7
display
  info, 7
  model, 7
  models, 8
div, 15, 28
division
  integral, 28
  remainder, 28
do, 15
doc, 47, 54
doc.annot, 57
documentation
  enable, 54
  target location, 56
documentation annotations, 54
DOWN, 7
draw
  arrow, 385
  ellipse, 386, 387
  line, 390, 393
  pie, 391
  point, 392
  polyline, 390, 393
  rectangle, 394
  text, 395, 396
drop
  cut, 709
dropcuts, 709
dropnextevent, 243
DSO, 8
dual value, 99
dynamic, 15, 23
dynamic array
  of variables, 23
dynamic shared object
    version, 8
Dynamic Shared Objects manager, 1

E
  E-1, 754
  E-100, 756
  E-1000, 765
  E-1001, 765
  E-1002, 765
  E-1003, 766
  E-1004, 766
  E-1005, 766
  E-1006, 766
  E-1009, 766
  E-101, 756
  E-1010, 766
  E-1013, 766
  E-1014, 767
  E-1015, 767
  E-1016, 767

disc, 155
disconnect, 284, 307
disconnect, 200
diskdata, 156, 157
DISPLAY, 7
display
  info, 7
  model, 7
  models, 8
div, 15, 28
division
  integral, 28
  remainder, 28
do, 15
doc, 47, 54
doc.annot, 57
documentation
  enable, 54
  target location, 56
documentation annotations, 54
DOWN, 7
draw
  arrow, 385
  ellipse, 386, 387
  line, 390, 393
  pie, 391
  point, 392
  polyline, 390, 393
  rectangle, 394
  text, 395, 396
drop
  cut, 709
dropcuts, 709
dropnextevent, 243
DSO, 8
dual value, 99
dynamic, 15, 23
dynamic array
  of variables, 23
dynamic shared object
    version, 8
Dynamic Shared Objects manager, 1

E
  E-1, 754
  E-100, 756
  E-1000, 765
  E-1001, 765
  E-1002, 765
  E-1003, 766
  E-1004, 766
  E-1005, 766
  E-1006, 766
  E-1009, 766
  E-101, 756
  E-1010, 766
  E-1013, 766
  E-1014, 767
  E-1015, 767
  E-1016, 767
<table>
<thead>
<tr>
<th>Word</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>F_ERROR</td>
<td>86, 92, 100, 103</td>
</tr>
<tr>
<td>F_INPUT</td>
<td>86, 92, 100, 103</td>
</tr>
<tr>
<td>F_LINBUF</td>
<td>92</td>
</tr>
<tr>
<td>F_OUTPUT</td>
<td>86, 92, 100, 103</td>
</tr>
<tr>
<td>F_SILENT</td>
<td>92</td>
</tr>
<tr>
<td>F_TEXT</td>
<td>92</td>
</tr>
<tr>
<td>failure</td>
<td>281, 301</td>
</tr>
<tr>
<td>false</td>
<td>15, 21, 538</td>
</tr>
<tr>
<td>fclose</td>
<td>52, 86</td>
</tr>
<tr>
<td>fcopy</td>
<td>437</td>
</tr>
<tr>
<td>fctasproc</td>
<td>18</td>
</tr>
<tr>
<td>fdelete</td>
<td>438</td>
</tr>
<tr>
<td>fflush</td>
<td>52, 87</td>
</tr>
<tr>
<td>access mode</td>
<td>455</td>
</tr>
<tr>
<td>append</td>
<td>92</td>
</tr>
<tr>
<td>close</td>
<td>86</td>
</tr>
<tr>
<td>compressed</td>
<td>738</td>
</tr>
<tr>
<td>copy</td>
<td>437</td>
</tr>
<tr>
<td>delete</td>
<td>438</td>
</tr>
<tr>
<td>ID</td>
<td>100</td>
</tr>
<tr>
<td>in/output</td>
<td>156</td>
</tr>
<tr>
<td>inclusion</td>
<td>19</td>
</tr>
<tr>
<td>initialization</td>
<td>34</td>
</tr>
<tr>
<td>IO</td>
<td>52</td>
</tr>
<tr>
<td>move</td>
<td>441</td>
</tr>
<tr>
<td>name</td>
<td>103</td>
</tr>
<tr>
<td>open</td>
<td>92</td>
</tr>
<tr>
<td>read</td>
<td>52, 132</td>
</tr>
<tr>
<td>rename</td>
<td>441</td>
</tr>
<tr>
<td>select</td>
<td>93</td>
</tr>
<tr>
<td>size</td>
<td>454</td>
</tr>
<tr>
<td>status</td>
<td>455</td>
</tr>
<tr>
<td>time</td>
<td>456</td>
</tr>
<tr>
<td>write</td>
<td>95, 152</td>
</tr>
<tr>
<td>file extension</td>
<td>3</td>
</tr>
<tr>
<td>file name prefix</td>
<td>3</td>
</tr>
<tr>
<td>file inclusion</td>
<td>3</td>
</tr>
<tr>
<td>package loading</td>
<td>3</td>
</tr>
<tr>
<td>files</td>
<td>BIM, 16</td>
</tr>
<tr>
<td>finalize</td>
<td>88</td>
</tr>
<tr>
<td>finalize</td>
<td>23, 88</td>
</tr>
<tr>
<td>findfiles</td>
<td>439</td>
</tr>
<tr>
<td>findfirst</td>
<td>89</td>
</tr>
<tr>
<td>findlast</td>
<td>90</td>
</tr>
<tr>
<td>findtext</td>
<td>440</td>
</tr>
<tr>
<td>findxsvrs</td>
<td>206</td>
</tr>
<tr>
<td>FINISH</td>
<td>7</td>
</tr>
<tr>
<td>fix</td>
<td>variable, 638</td>
</tr>
<tr>
<td>fixglobal</td>
<td>638</td>
</tr>
<tr>
<td>floor</td>
<td>91, 264, 538</td>
</tr>
<tr>
<td>flush buffer</td>
<td>87</td>
</tr>
<tr>
<td>fmove</td>
<td>441</td>
</tr>
<tr>
<td>following</td>
<td>537</td>
</tr>
<tr>
<td>fopen</td>
<td>52, 92</td>
</tr>
<tr>
<td>forall</td>
<td>15, 38</td>
</tr>
<tr>
<td>format string</td>
<td>148</td>
</tr>
<tr>
<td>formattext</td>
<td>442</td>
</tr>
<tr>
<td>forward</td>
<td>15, 42</td>
</tr>
<tr>
<td>free</td>
<td>info table, 330</td>
</tr>
<tr>
<td>from</td>
<td>15</td>
</tr>
<tr>
<td>fselect</td>
<td>52, 93</td>
</tr>
<tr>
<td>fskipline</td>
<td>52, 92, 94</td>
</tr>
<tr>
<td>fsrvdelay</td>
<td>197</td>
</tr>
<tr>
<td>fsrvnbiter</td>
<td>197</td>
</tr>
<tr>
<td>fsrvport</td>
<td>197</td>
</tr>
<tr>
<td>function</td>
<td>return value, 40</td>
</tr>
<tr>
<td>get</td>
<td>type, 40</td>
</tr>
<tr>
<td>get</td>
<td>function call, 26</td>
</tr>
<tr>
<td>fwrite</td>
<td>95</td>
</tr>
<tr>
<td>fwrite_</td>
<td>59</td>
</tr>
<tr>
<td>fwriteln</td>
<td>95</td>
</tr>
<tr>
<td>fwriteln_</td>
<td>59</td>
</tr>
<tr>
<td>G</td>
<td>generateUFparallel, 603</td>
</tr>
<tr>
<td>get</td>
<td>active cuts, 710</td>
</tr>
<tr>
<td>get</td>
<td>from cut pool, 711</td>
</tr>
<tr>
<td>get R array</td>
<td>725</td>
</tr>
<tr>
<td>get R boolean</td>
<td>726</td>
</tr>
<tr>
<td>get R integer</td>
<td>727</td>
</tr>
<tr>
<td>get R real</td>
<td>728</td>
</tr>
<tr>
<td>get R string</td>
<td>729</td>
</tr>
<tr>
<td>getact</td>
<td>96, 338</td>
</tr>
<tr>
<td>getaliases</td>
<td>204</td>
</tr>
<tr>
<td>getannidents</td>
<td>230</td>
</tr>
<tr>
<td>getannotations</td>
<td>231</td>
</tr>
<tr>
<td>getasnumber</td>
<td>444</td>
</tr>
<tr>
<td>getattr</td>
<td>547</td>
</tr>
<tr>
<td>getbanner</td>
<td>202</td>
</tr>
<tr>
<td>getboolattr</td>
<td>547</td>
</tr>
<tr>
<td>getboolvalue</td>
<td>551</td>
</tr>
<tr>
<td>getbstat</td>
<td>639</td>
</tr>
<tr>
<td>getchar</td>
<td>445</td>
</tr>
<tr>
<td>getclass</td>
<td>249</td>
</tr>
<tr>
<td>getcnlist</td>
<td>710</td>
</tr>
<tr>
<td>getcoeff</td>
<td>97</td>
</tr>
<tr>
<td>getcoeffs</td>
<td>98</td>
</tr>
<tr>
<td>getclplist</td>
<td>711</td>
</tr>
<tr>
<td>getcwd</td>
<td>446</td>
</tr>
<tr>
<td>getdate</td>
<td>447, 528</td>
</tr>
<tr>
<td>getdatetime</td>
<td>530</td>
</tr>
<tr>
<td>getday</td>
<td>448</td>
</tr>
<tr>
<td>getdaynum</td>
<td>449</td>
</tr>
<tr>
<td>getdays</td>
<td>450</td>
</tr>
<tr>
<td>getdirsep</td>
<td>451</td>
</tr>
<tr>
<td>getdsoprop</td>
<td>218</td>
</tr>
<tr>
<td>getdual</td>
<td>99</td>
</tr>
<tr>
<td>getdualray</td>
<td>640</td>
</tr>
<tr>
<td>getencoding</td>
<td>549</td>
</tr>
<tr>
<td>getendparse</td>
<td>452</td>
</tr>
<tr>
<td>getenv</td>
<td>453</td>
</tr>
<tr>
<td>getexitcode</td>
<td>226</td>
</tr>
</tbody>
</table>
Index

getfid, 52, 100
getfirst, 101
getfirstattr, 552
getfirstchild, 554
getname, 103
getfromgid, 247
getfromid, 246
getfromuid, 248
getsize, 454
getstat, 455
gftime, 456
getgid, 219
gethead, 102
gethostalias, 203
gethour, 457
getspace, 562
getid, 220
getiis, 641
getiissense, 642
getiistype, 643
getindentmode, 564
getindentskip, 565
getinfcause, 644
getfeas, 645
getintattr, 547
getintvalue, 551
gelast, 104
gelastchild, 555
geltb, 646
getlinelen, 566
getloadedlinctrs, 647
getloadedmpvars, 648
getmaxnodes, 567
getminute, 458
getmodprop, 221
getchange, 459
getmsec, 460
getname, 550, 649
getnext, 553
getnextevent, 242
getnode, 222, 556
getnodes, 557
getnominal, 342
getobjval, 105
getparam, 92, 106, 279, 297, 350, 422, 622, 651
getparent, 558
getpathsep, 461
getprimalray, 650
getprobstat, 651
getqexpnextterm, 331
getqexpso, 328
getqexpstat, 329
getqtype, 463
getrange, 652
getrcost, 108
getreadcnt, 109
getrealattr, 547
getrealvalue, 551
getreverse, 110
getrtmid, 223
getsecond, 464
getsensrng, 653
getselchar, 465
getsize, 111, 466, 538, 568, 654
getslack, 112
getsol, 113, 268, 324, 328, 337, 655
getstandalone, 560
getstart, 467
getstatus, 224
getstrattr, 547
getstrvalue, 551
getsucc, 462
getsysinfo, 468
getsysstat, 469
gettail, 114
getime, 470, 526
gettimer, 250
gettmpdir, 471
getrtrim, 472
gettxtbuf, 533
gettxtsize, 532
gettype, 115, 271, 343, 559
getub, 656
getuid, 225
getvalue, 251, 551
getvars, 116, 657
getvspace, 563
getweekday, 473
getxmlversion, 561
getyear, 474

GID
event sender, 247
model, 219, 237
wait for, 240
graph, 378
graphical interface, 2
gzip, 738

H
hasfeature, 658
hex, 376
hidden constraint, 120, 137, 269, 270
robust constraint, 339, 341
horizontal spacing, 562, 580
host alias, see alias
http driver, 191
HTTP request
asynchronous mode, 165
delete, 168
get, 169
head, 170
post, 171
put, 172
status code, 173
synchronous mode, 165
HTTP server, 158
start, 188
stop, 189
http_async, 159
http_browser, 159
http_defpage, 159
http_defport, 160
http_expire, 160
http_keeper, 160
http_listen, 161
http_maxasync, 161
http_maxconn, 161
http_maxcontime, 161
http_maxreq, 162
http_maxreqtime, 162
http_port, 163
http_proxy, 163
http_proxyport, 163
http_srvconfig, 163
http_startwb, 164
httpcancel, 167
httpdel, 168
httpget, 169
httpgetheader, 175
httphead, 170
httppending, 178
httppost, 171
httpput, 172
httpreason, 173
httpreply, 179
httpreplycode, 180
httpreplyjson, 181
httpreqfile, 182
httpreqfrom, 183
httpreqheader, 184
httpreqlabel, 185
httpreqstat, 186
httpreqtype, 187
https_cacerts, 351
https_ciphers, 351
https_clt.crt, 352
https_clt.key, 352
https_defport, 164
https_listen, 164
https_port, 165
https_srv.crt, 352
https_srv.key, 353
https_trustsrv, 353
httpstartsrv, 188
httpstopsrv, 189

I
I/O driver, 50
bool, 51
default, 50
deploy.csrc, 53
deploy.exe, 53
deploy.js, 53
deploy.jsr, 53
deploy.jsr, 53
mmjobs.shmem, 253
mmjobs.xservlet, 254
mmjobs.xssh, 254
mmoci.oci, 291
mmmodbc.odbc, 320
mmsheet.csv, 348
mmsheet.xls, 348
mmsheet.xlsx, 348
mmsl.base64, 376
mmsl.crypt, 376
mmsl.hex, 376
null, 50
r.rws, 737
rte, 50
tmp, 50
zlib deflate, 738
zlib.gzip, 738
zlib.zip, 738

ID
event sender, 246
file, 100
model, 220
stream, 100
identifier, 14
if, 15, 26, 36
IIS, see irreducible infeasible set, see irreducible
infeasible set
reset search, 678
IMCI, 325, 524
implies, 659
imports, 15, 16
in, 15, 29
include, 15, 19
indent mode, 564, 583
indent skip, 565, 584
indexing set, 22
indicator, 660
indicator constraint, 622, 660
INFINITY, 119, 121, 139
INFINITY, 62
INFO, 7
info
quadtratic expression, 329
initial step bound, 610
initial value, 262

copy solution, 266, 634
delete, 265
set, 267
initialisation vector, 377
size, 368
initialisations, 15
initializations, 15
initialize, 730
inline initialization, 33
input file, 156
input stream, 86, 92, 93, 192, 212
read, 132
test eof, 118
InputStream, 192
inserttext, 475
instance
  banner, 202
  connect, 199
  disconnect, 200
instance control parameter, 748
integer
  read, 286, 314
integer, 15, 21, 26
integrity check, 662
inter, 15, 29
inter-module communication, 325, 524
interface
  inter-module communication, 325, 524
interpreted, 16
intersection, 29
IO
  error, 92, 107, 141
  status, 92, 107
  switching between streams, 52
IO driver, see I/O driver
ioctl, 92, 107, 141
iostatus, 92, 107
irreducible infeasible set, 641, 661
is_binary, 15, 30
is_continuous, 15, 30
is_free, 15, 30
is_integer, 15, 30
is_partint, 15, 30
is_semicon, 15, 30
is_semint, 15, 30
is_sos1, 15, 30
is_sos2, 15, 30
isdynamic, 117
iseof, 118
isfinite, 119
ishidden, 120, 269, 339
isisvalid, 661
isinf, 121
isintegral, 662
isnan, 122
isodd, 123
isqueueempty, 244
isvalid, 476
item
  number read, 107
iterator, 27
J
Java
  IO drivers, 192
java, 192
jobid, 196
jraw, 192
JSON_FCT_BOOL, 570
JSON_FCT_CLOSE_ARR, 570
JSON_FCT_CLOSE_OBJ, 570
JSON_FCT_NULL, 570
JSON_FCT_NUM, 570
JSON_FCT_OPEN_ARR, 570
JSON_FCT_OPEN_OBJ, 570
JSON_FCT_TEXT, 570
jsonload, 569
jsonparsing, 570
jsonsave, 572
jsonwrite, 190
K
keepalive, 196
keepassert, 18, 66
key, 377
key derivation, 377
key size, 369
keyword
  SQL, 297
keywords, 10, 15
L
lang, 107
language, 14
largest value, 127
legend, 378
length
  string, 279, 298
library
  Run Time, 1
linctr, 15, 21, 262, 264, 321, 322, 325, 333
line breaking, 15
line control directive, 20
line length, 566, 585
lines
  number affected, 281, 300
  number transferred, 281, 301
LIST, 7
list, 22
  compare, 29
  constant, 29
  finalize, 88
  find element, 89, 90
  first element, 101
  head, 102
  last element, 104
  remove elements, 78, 79
  reverse, 110, 134
  size, 111
  split, 145, 146
  tail, 114
list, 15
ln, 124, 264
load
  basis, 663
  cut, 712
  module, 16
  package, 16, 17
  problem, 666
load, 211, 573
load document, 569, 573
loadbasis, 663
loadcuts, 712
loadmipsol, 664
loadprob, 321, 666
localedir, 107, 141
location step, 536
log, 125, 264
logarithm
  base 10, 125, 264
  natural, 124, 264
logctr, 622, 660
logical and, 30
logical expression, 622
  exclusive or, 704
  implication, 659
logical negation, 30
logical or, 30
loop, 38
loop statement, 31
lower bound, 646
  set, 692
LP format, 7
  maximization, 84
  minimization, 84
LP solution
  save, 682
LP status, 622
LSATTR, 8
LSLIBS, 8
LSMODS, 8

M
makedir, 477
makepath, 478
makesos1, 30, 126
makesos2, 30, 126
marginal values, 637
mathctrl, 141
Maths
  error, 141
matrix
  column order, 623
matrix output, 7
max, 15, 28
maximize, 321, 667
maximum value, 28, 127
MAX_INT, 62
maxlist, 127
MAX_REAL, 62
mc, 46
mc.def, 48
mc.flush, 48
mc.set, 49
M_E, 62
memory pipe, 253
mempipe, 253
message catalog, 59, 60
message digest, 350, 354, 365
  size, 370
message domain, 59
message printing, 281, 301
  Optimizer, 625
message translation, 58
min, 15, 28
minimize, 321, 667
minimum value, 28, 128
minlist, 128
MIP solution
  add, 628
  load, 664
  save, 681
mmssl
  configuration directory, 350
MSVGDISPLAY, 381
MSVGTGZ, 382
mod, 15, 28
MODEL, 8
Model, 207
model
  active, 8
  body, 16
  compile, 208
  exit code, 226
  GID, 219, 237
  handling multiple, 194
  ID, 220
  load, 211
  name, 8
  pause execution, 407
  reset, 228
  run, 4, 217
  sequence number, 8
  size, 8
  source, 1
  status, 224
  stop, 227
  structure, 16
  UID, 225, 236
  unload, 229
  version, 8
  web service, 158
model, 8, 15, 16
model cut, 694
delete, 632
model management, 207
model manager, 1
model parameter, 18
model_version, 107
modelname, 107
module, 2
  dependency, 325, 524
module structure
  advantages, 2
modules, 14
monthnames, 424
Mosel, 198
mosel, 2
debugger, 6
invocation, 2
restricted mode, 10
Mosel compiler, 1
Mosel Console, 2
Mosel instance, see instance, 468
Mosel Remote Launcher, 255
MOSEL_DSO, 17, 258
MOSEL_EXCPATH, 258, 513
MOSEL_RESTR, 10, 258
MOSEL_ROPATH, 10, 258
MOSEL_RWPATH, 10, 258
MOSEL_SDMAX, 3
MOSEL_SSL, 354
moseldoc, 58
move
  file, 441
MP type, 21
M_PI, 62
mpproblem, 43
MPS format, 7, 84
mpsol
  reset, 679
mpsol, 621
mpvar, 15, 21, 262, 321, 325
msgdigest, 365
msgsign, 366
msgverify, 367
multiple models, 194
multiple problems, 43
multistart job, 599
N
name
  scramble, 84
  variable, 649
name, 538
names
  loading, 624
NAN, 119, 122, 139
NAN, 62
nbread, 107, 132
new line, 28
newmuid, 129
newtar, 479
newzip, 480
NEXT, 8
next, 15, 39
nextfield, 481
nlctr, 262, 264
noautofinal, 18
NoDB, 10
node, 556, 557
  add, 542
  copy, 544
  delete, 546
  get first child, 554
  get last child, 555
  get name, 550
  get next, 553
  get parent, 558
  get type, 559
  get value, 551
  set name, 578
  set value, 579
node, 537
node test, 536, 537
nodenumber, 537
NoExec, 10
noimplicit, 18, 33
nominal value, 342, 344
non-relational, 280, 300
nonlinear
  complementary variables, 606
  feasibility, 618
  memory usage, 604
  scaling, 605
  tolerance, 611
nonlinear constraint
  enforced, 609
  hide, 270
  name, 272
  set type, 273
  solution, 268
  test hidden, 269
  type, 271
NoRead, 10
not, 15, 30, 538, 622
not in, 29
NoTmp, 10
NoWrite, 10
nullevent, 245
number
  connection, 280, 299
  lines, 281, 300, 301
  number, 538
O
object group, 378
objective value, 105
OCI
  debug mode, 280
  IO driver, 291
oci, 291
OCIautocommit, 279
OCIbufsize, 279
OCIcolsize, 279
OCIcommit, 289
OCIconnection, 280, 283
OCIdebug, 280
OCIexecute, 285
OCIlogoff, 284
OCIlogon, 283
OCIlogxcol, 280
OCIreadinteger, 286
OCIreadreal, 287
OCIreadstring, 288
OCIrollback, 290
OCIrowcnt, 281
OCIrowxfr, 281
OCIsuccess, 281
OCIverbose, 281
ODBC
  debug mode, 299
odd number, 123
of, 15
open
    file, 92
    stream, 92
openpipe, 483
operation
    elementary, 31
operator, 25
    arithmetic, 28
optimization
    direction, 7
Optimizer
    loading names, 624
    message printing, 625
    problem name, 625
    problem pointer, 624
    version number, 624
optimizer problem status, 651
OPTION, 8
options, 15, 16
or, 15, 30, 622
output file, 156
output stream, 86, 92, 93, 192, 212
    flush, 87
    write, 95, 152
OutputStream, 192
P
package, 45
    annotation declaration, 49
    structure, 16
package, 15, 16
parameter, 18, 106, 141
parameters, 16
parameters, 15
parent, 537
parent model, 234
parentnumber, 196
parseextn, 484
parsesint, 485
parser context, 422
parser parameter, 107
parser_date, 107
parser_file, 107
parser_line, 107
parser_time, 107
parser.UTCDate, 107
parser.UTCtime, 107
parser.version, 107
parsereal, 487
parsetext, 488
passphrase, 349
pastetext, 490
pathmatch, 491
pathsplit, 492
pause
    model execution, 407
peeknextevent, 252
PEM format, 349
plot
    add, 384
PO, see Portable Object
Portable Object, 60
Portable Object Template, 60
position, 538
postsolve, 669
POT, see Portable Object Template
predicate, 536, 537
primal solution, 113
PRINT, 9
print, 95, 152
    problem, 84
    quadratic problem, 323
    R, 731
printing, 281, 301
printing format, 107, 141
printmodelmemory, 604
printmodelscale, 605
private key, 349
private key file, 355
private symbol, 45
problem, 7
    export, 84
    handling multiple, 43
    load, 666
    main, 43
    maximize, 667
    postsolve, 669
    print, 84
    status, 651
    unload, 699
    write, 702
problem context, 43
problem name
    Optimizer, 625
problem pointer
    Optimizer, 624
problem type, 43
    extensions, 43
procedural, 16
procedure, 33
    body, 40
procedure, 15
procedures
    passing of formal parameters, 41
processing instruction, 535
processing-instruction, 537
prod, 15, 28
product, 28
public, 15, 45
public key, 349, 350
publish, 130
Q
QCQP, see Quadratically Constrained Quadratic Programming
qexp, 321, 322
qsort, 493
quadratic expression
    enumerate terms, 331
    get info, 329
size, 359
RSA key pair, 350, 358
RSAfingerprint, 357
RSAgenkey, 358
RSAgetkeysizesize, 359
RSAisprivate, 360
RSAloadkey, 361
RSAprivdecrypt, 362
RSApubencrypt, 363
RSAsavekey, 364
Rsessionmode, 721
Rset, 732
Rsource, 733
run, 217
Runloadscript, 721
running time, 470
Rusemosstreams, 721
Rverbose, 720
S
salt, 377
save
  basis, 680
  Optimizer status, 683
save, 574
save document, 572, 574
savebasis, 680
savemipsol, 681
savesol, 682
savestate, 683
scenario, 340
secure vectors, 636
select
  file, 93
  stream, 93
selection statement, 31, 36
selectsol, 684
self, 537
send, 234
sensivity ranges, 653
server
  trust, 353
server certificate, 352
server private key, 353
service
  inter-module communication, 325, 524
  module dependency, 325, 524
set, 21
  callback, 687
  compare, 29
  constant, 23
  finalize, 88
  fixed, 23
  in/output, 156
  size, 111
set, 15
  setarchconsistency, 685
setattr, 575
setbstat, 686
setcallback, 687
setcbcutoff, 690
setchar, 500
setcoeff, 136
setcomplementary, 606
setcontrol, 214
setdate, 501, 529
setdatetime, 531
setday, 502
setdefstream, 212
setdelayeddec, 607
setdetrow, 608
setencoding, 576
setendparse, 452
setenforcedctr, 609
setenv, 503
setgid, 237
setgndata, 691
sethidden, 137, 270, 341
sethostalias, 205
sethour, 504
sethspace, 580
setindentmode, 583
setindentskip, 584
setinit, 691
setinitval, 267
setioerr, 138
setlb, 692
setlinelem, 585
setmatherr, 139
setmaxnodes, 577
setminute, 505
setmipdir, 693
setmodcut, 694
setmodpar, 215
setmonth, 506
setmsec, 507
setname, 140, 272, 578
setnominal, 344
setparam, 141, 279, 297, 350, 422, 622
setqtype, 463
setrandseed, 142
setsecond, 508
setsephchar, 465
setsol, 695
setstandalone, 586
setstart, 467
setsucc, 462
settime, 509, 527
setter, 235
settol, 611
settolset, 612
settrim, 472
settype, 143, 273, 345
setub, 696
setucdata, 697
setubbase, 697
setuid, 236
setvalue, 579
setvspace, 582
setworkdir, 216
setxmlversion, 587
Index

SYS_VER, 468
SYS_WRITE, 455
SYS_MOD, 455
SYS_NOW, 421, 422
system, 513
system command, 513
system information, 468
system status, 469
SYS_TYP, 455

T
table of symbols, 45
tan, 264
tarlist, 514
temporary directory, 52
term
enumerate, 331
terminate R, 724
termination, 82
test
eof, 118
hidden constraint, 120
hidden nonlinear constraint, 269
testattr, 548
text, 421, 537
text section, 535
textfmt, 515
then, 15
time, 77
file, 456
time, 421
time measure, 470
timefmt, 423
timestamp, 150
timestamp, 150
tmpdir, 107
to, 15
tolerance
zero, 107, 141
tolerance set, 612
tolover, 517
tolset, 594
toupper, 518
transaction
commit, 289
rollback, 290
trigonometric functions, 64, 74, 144, 264
trim, 519
true, 15, 21, 26, 538
txtresize, 534
type
constraint, 115, 143
nonlinear constraint, 271, 273
problem, 43
SOS, 115, 143
variable, 30, 115, 143

U
UID

event sender, 248

model, 225, 236
wait for, 240
uncertain, 332
uncertainctr, 332
unconstrained, 30
UNDISPLAY, 9
union, 29
union, 15, 29
unique identifier, 129
unload, 229
unloadprob, 699
unpublish, 151
untar, 520
until, 15, 38
unzip, 521
UP, 9
upper bound, 656
set, 696
url, 191
URL encoding, 174
urlencode, 174
user comment, 8
user function, 593, 601
DLL, 613
Excel, 614
Excel macro, 615
info, 616
Mosel, 617
parallel, 603
user graph, 378
add file, 388, 389
add plot, 384
axes labels, 410
closing, 397
color, 398
delete object, 399
draw arrow, 385
draw circle, 386
draw ellipse, 387
draw line, 390
draw pie, 391
draw point, 392
draw polygon, 393
draw rectangle, 394
draw text, 395, 396
erase, 400
get object, 404
get style, 402, 406
get style property, 401, 405
get viewbox, 403
model termination, 420
point size, 411
refresh, 408
refresh frequency, 416
save, 409
scaling, 412
set style, 414, 418
set style property, 413, 417
view box, 415
user type, 24
definition, 24
userfunc, 593
userfuncDLL, 613
userfuncExcel, 614
userfuncExcelMacro, 615
userfuncinfo, 616
userfuncMosel, 617
uses, 15, 16
UTC, 107, 141

V
validate, 618
value
  event, 232
  variable, 21
    check intregrality, 662
    environment, 453, 503
    fix, 638
    initial value, 262, 267
    lower bound, 646
    name, 649
    ranging information, 652
    reduced cost, 108
    sensivity ranges, 653
    set coefficient, 136
    set lower bound, 692
    set type, 143
    set upper bound, 696
    solution, 113
    type, 115
    upper bound, 656
version, 8, 15, 16
vertical spacing, 563, 582
viewbox, 403
VIMA, 1
visual environment, 2

W
W-121, 758
W-131, 759
W-144, 759
W-152, 760
W-164, 761
W-166, 762
W-306, 764
W-309, 764
W-85, 767
wait, 238
waitexpired, 239
waitfor, 240
WDOnly, 10
WHERE, 9
while, 15, 38
with, 15, 39, 43
workdir, 107, 141
working directory, 52, 216, 446
write
  directives, 701
  problem, 702
write, 52, 92, 152, 325
write_, 59
writebasis, 700
writedirs, 701
writeln, 52, 92, 152, 325
writeln_, 59
writeprob, 702
writesol, 703

X
X509 certificate, 350
  compatibility, 373
  create, 375
  information, 374
x509check, 373
x509getinfo, 374
x509newcrt, 375
xbim, 18
xls, 348
xlsx, 348
XML
  decode, 590
  encode, 589
XML document, 540
XML document structure, 535
XML node number, 540
XML node type, 535
XML path, 536
xml version, 561, 587
XML_ATTR, 550, 559
XML_AUTO, 564, 583
XMLCDATA, 542, 550, 559
XML_COM, 542, 550, 559
XML_DATA, 542, 550, 559
XML_ELT, 542, 550, 559
XML_FCTCDATA, 591
XML_FCT_CLOSEELT, 591
XML_FCT_COM, 591
XML_FCT_DATA, 591
XML_FCTDECL, 591
XML_FCTOPENELT, 591
XML_FCTPINST, 591
XML_FCTTXT, 591
XML_FIRST, 542, 544
XML_FIRSTCHILD, 542, 544
XML_LAST, 542, 544
XML_LASTCHILD, 542, 544
XML MANUAL, 564, 583
XML_NEXT, 542, 544
XML_NONE, 564, 583
XML_PINST, 542, 550, 559
XML_TXT, 542, 550, 559
xmlattr, 588
xmldecode, 590
xmldecodexml, 540
xmlencode, 589
xmlparse, 591
XNLP_LOADASNL, 596
XNLP_LOADNAMES, 596
XNLP_NLPSTATUS, 596
XNLP_SOLVER, 597
XNL_P_VERBOSE, 597
XNL_P_TOLER_RA, 594
XNL_P_TOLER_RA, 594
XNL_P_TOLER_RM, 594
XNL_P_TOLER_RS, 594
XNL_P_TOLER_TA, 594
XNL_P_TOLER_TC, 594
XNL_P_TOLER_TI, 594
XNL_P_TOLER_TM, 594
XNL_P_TOLER_TS, 594
xor, 704
Xpress Optimizer, 2
Xpress Workbench, 2
xprmsrv, 255
XPRS_colorder, 623
XPRS_enumduplpol, 623
XPRS_enummaxsol, 623
XPRS_enumsols, 623
XPRS_fullversion, 624
XPRS_loadnames, 624
XPRS_problem, 624
XPRS_probnname, 625
XPRS_verbose, 625
XPRS_BAR, 667
XPRS_BR, 693
XPRS_CB_BARITER, 687
XPRS_CB_BARLOG, 687
XPRS_CB_CHGBRANCH, 687
XPRS_CB_CHGNODE, 687
XPRS_CB_CUTLOG, 687
XPRS_CB_CUTMGR, 687
XPRS_CB_GAPNOTIFY, 687
XPRS_CB_GLOBALLOG, 687
XPRS_CB_INFNODE, 687
XPRS_CB_INTSOL, 687
XPRS_CB_LPLOG, 687
XPRS_CB_NEWNODE, 687
XPRS_CB_NODECUTOFF, 687
XPRS_CB_OPTNODE, 687
XPRS_CB_PREINTSOL, 687
XPRS_CB_PRENODE, 687
XPRS_CB_SOLNOTIFY, 687
XPRS_CONT, 667
XPRS_CORELP, 667
XPRS_DN, 653, 693
XPRS_DUAL, 667
XPRS_ENUM, 667
XPRS_INF, 651
XPRS_LCOST, 652
XPRS_LIN, 667
XPRS_LOACT, 652
XPRS_LOCAL, 667
XPRS_LPSTOP, 667
XPRS_NET, 667
XPRS_OPT, 651
XPRS_OTH, 651
XPRS_PD, 693
XPRS_PR, 693
XPRS_PRI, 667
XPRS_PU, 693
XPRS_STOP_CTRL, 698
XPRS_STOP_ITERLIMIT, 698
XPRS_STOP_MIPGAP, 698
XPRS_STOP_NODELIMIT, 698
XPRS_STOP_SOLLIMIT, 698
XPRS_STOP_TIMELIMIT, 698
XPRS_STOP_USER, 698
XPRS_TUNE, 667
XPRS_UCOST, 652
XPRS_UDN, 652
XPRS_UNB, 651
XPRS_UNF, 651
XPRS_UP, 653, 693
XPRS_UPACT, 652
XPRS_UUUP, 652
xsrv, 254
xssh, 254
Z
zero tolerance, 107, 141
zerotol, 107, 141
zip, 738
ziplist, 522