## Contents

<table>
<thead>
<tr>
<th>Part I Function Reference</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arithmetic Functions</strong></td>
<td></td>
</tr>
<tr>
<td>Abs</td>
<td>3</td>
</tr>
<tr>
<td>ArcCos</td>
<td>4</td>
</tr>
<tr>
<td>ArcCosh</td>
<td>5</td>
</tr>
<tr>
<td>ArcSin</td>
<td>6</td>
</tr>
<tr>
<td>ArcSinh</td>
<td>7</td>
</tr>
<tr>
<td>ArcTan</td>
<td>8</td>
</tr>
<tr>
<td>ArcTanh</td>
<td>9</td>
</tr>
<tr>
<td>Ceil</td>
<td>10</td>
</tr>
<tr>
<td>Cos</td>
<td>11</td>
</tr>
<tr>
<td>Cosh</td>
<td>12</td>
</tr>
<tr>
<td>Degrees</td>
<td>13</td>
</tr>
<tr>
<td>ErrorF</td>
<td>14</td>
</tr>
<tr>
<td>Exp</td>
<td>15</td>
</tr>
<tr>
<td>Floor</td>
<td>16</td>
</tr>
<tr>
<td>Log</td>
<td>17</td>
</tr>
<tr>
<td>Log10</td>
<td>18</td>
</tr>
<tr>
<td>MapVal</td>
<td>19</td>
</tr>
<tr>
<td>Max</td>
<td>20</td>
</tr>
<tr>
<td>Min</td>
<td>21</td>
</tr>
<tr>
<td>Mod</td>
<td>22</td>
</tr>
<tr>
<td>Power</td>
<td>23</td>
</tr>
<tr>
<td>Radians</td>
<td>24</td>
</tr>
<tr>
<td>Round</td>
<td>25</td>
</tr>
<tr>
<td>Sign</td>
<td>26</td>
</tr>
<tr>
<td>Sin</td>
<td>27</td>
</tr>
<tr>
<td>Sinh</td>
<td>28</td>
</tr>
<tr>
<td>Sqr</td>
<td>29</td>
</tr>
<tr>
<td>Sqrt</td>
<td>30</td>
</tr>
<tr>
<td>Tan</td>
<td>31</td>
</tr>
<tr>
<td>Tanh</td>
<td>32</td>
</tr>
</tbody>
</table>
Contents

Trunc ................................... 33

Set Related Functions 34
Card ................................... 35
ConstraintVariables ..................... 36
Element .................................. 37
ElementCast ................................ 38
ElementRange ............................. 39
FindUsedElements ....................... 40
Ord .................................... 41
RestoreInactiveElements ............... 42
RetrieveCurrentVariableValues ........ 43
SetElementAdd ........................... 44
SetElementRename ....................... 45
StringToElement ......................... 46
SubRange ................................ 47
VariableConstraints ..................... 48

String Manipulation Functions 49
FindNthString ............................ 50
FindString ................................ 51
FormatString ............................ 52
StringCapitalize ......................... 53
StringLength ............................ 54
StringOccurrences ....................... 55
StringToLower ............................ 56
StringToUpper ............................ 57
SubString ................................ 58

Time Functions 59
Aggregate ................................ 60
CreateTimeTable ......................... 61
CurrentToMoment ......................... 62
CurrentToString ......................... 63
CurrentToTimeSlot ....................... 64
DisAggregate ............................ 65
MomentToString .......................... 66
MomentToTimeSlot ......................... 67
PeriodToString ........................... 68
StringToMoment ......................... 69
StringToTimeSlot ......................... 70
TimeSlotCharacteristic ................. 71
TimeSlotToMoment ....................... 72
TimeSlotToTimeSlot ..................... 73
<table>
<thead>
<tr>
<th>Database Functions</th>
<th>74</th>
</tr>
</thead>
<tbody>
<tr>
<td>CloseDataSource</td>
<td>75</td>
</tr>
<tr>
<td>DirectSQL</td>
<td>76</td>
</tr>
<tr>
<td>TestDatabaseTable</td>
<td>77</td>
</tr>
<tr>
<td>TestDataSource</td>
<td>78</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Case Functions</th>
<th>79</th>
</tr>
</thead>
<tbody>
<tr>
<td>CaseCreate</td>
<td>80</td>
</tr>
<tr>
<td>CaseDelete</td>
<td>81</td>
</tr>
<tr>
<td>CaseFind</td>
<td>82</td>
</tr>
<tr>
<td>CaseGetChangedStatus</td>
<td>83</td>
</tr>
<tr>
<td>CaseGetDatasetReference</td>
<td>84</td>
</tr>
<tr>
<td>CaseGetType</td>
<td>85</td>
</tr>
<tr>
<td>CaseLoadCurrent</td>
<td>86</td>
</tr>
<tr>
<td>CaseLoadIntoCurrent</td>
<td>87</td>
</tr>
<tr>
<td>CaseMerge</td>
<td>88</td>
</tr>
<tr>
<td>CaseNew</td>
<td>89</td>
</tr>
<tr>
<td>CaseSave</td>
<td>90</td>
</tr>
<tr>
<td>CaseSaveAll</td>
<td>91</td>
</tr>
<tr>
<td>CaseSaveAs</td>
<td>92</td>
</tr>
<tr>
<td>CaseSelect</td>
<td>93</td>
</tr>
<tr>
<td>CaseSelectMultiple</td>
<td>94</td>
</tr>
<tr>
<td>CaseSelectNew</td>
<td>95</td>
</tr>
<tr>
<td>CaseSetChangedStatus</td>
<td>96</td>
</tr>
<tr>
<td>CaseSetCurrent</td>
<td>97</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dataset Functions</th>
<th>98</th>
</tr>
</thead>
<tbody>
<tr>
<td>DatasetCreate</td>
<td>99</td>
</tr>
<tr>
<td>DatasetDelete</td>
<td>100</td>
</tr>
<tr>
<td>DatasetFind</td>
<td>101</td>
</tr>
<tr>
<td>DatasetGetCategory</td>
<td>102</td>
</tr>
<tr>
<td>DatasetGetChangedStatus</td>
<td>103</td>
</tr>
<tr>
<td>DatasetLoadCurrent</td>
<td>104</td>
</tr>
<tr>
<td>DatasetLoadIntoCurrent</td>
<td>105</td>
</tr>
<tr>
<td>DatasetMerge</td>
<td>106</td>
</tr>
<tr>
<td>DatasetNew</td>
<td>107</td>
</tr>
<tr>
<td>DatasetSave</td>
<td>108</td>
</tr>
<tr>
<td>DatasetSaveAs</td>
<td>109</td>
</tr>
<tr>
<td>DatasetSelect</td>
<td>110</td>
</tr>
<tr>
<td>DatasetSelectNew</td>
<td>111</td>
</tr>
<tr>
<td>DatasetSetChangedStatus</td>
<td>112</td>
</tr>
<tr>
<td>DatasetSetCurrent</td>
<td>113</td>
</tr>
</tbody>
</table>
Data Manager Functions

CaseTypeCategories .................................................. 115
CaseTypeContents ...................................................... 116
DataCategoryContents ................................................. 117
DataFileExists .......................................................... 118
DataFileGetAcronym ................................................... 119
DataFileGetDescription .............................................. 120
DataFileGetGroup ...................................................... 121
DataFileGetName ........................................................ 122
DataFileGetOwner ...................................................... 123
DataFileGetPath ........................................................ 124
DataFileGetTime ....................................................... 125
DataFileReadPermitted ................................................ 126
DataFileWritePermitted .............................................. 127
DataImport220 ........................................................... 128
DataManagerExport .................................................. 129
DataManagerImport ................................................... 130

Dialog Functions

DialogAsk ................................................................. 132
DialogError .............................................................. 133
DialogGetDate ........................................................... 134
DialogGetElementByData ............................................. 135
DialogGetElement ...................................................... 136
DialogGetElementByText ............................................. 137
DialogGetNumber ....................................................... 138
DialogGetPassword .................................................... 139
DialogGetString ........................................................ 140
DialogMessage .......................................................... 141
DialogProgress ........................................................ 142
StatusMessage .......................................................... 143

Page Functions

PageClose ................................................................. 145
PageGetActive .......................................................... 146
PageGetChild ............................................................ 147
PageGetFocus ........................................................... 148
PageGetNext ............................................................. 149
PageGetParent .......................................................... 150
PageGetPrevious ....................................................... 151
PageOpen ................................................................. 152
PageOpenSingle ......................................................... 153
PageRefreshAll ......................................................... 154
PageSetCursor ........................................................... 155
PageSetFocus ............................................................ 156
PrintEndReport .......................................................... 157
<table>
<thead>
<tr>
<th>File and Directory Functions</th>
<th>161</th>
</tr>
</thead>
<tbody>
<tr>
<td>DirectoryCopy</td>
<td>162</td>
</tr>
<tr>
<td>DirectoryCreate</td>
<td>163</td>
</tr>
<tr>
<td>DirectoryDelete</td>
<td>164</td>
</tr>
<tr>
<td>DirectoryExists</td>
<td>165</td>
</tr>
<tr>
<td>DirectoryGetCurrent</td>
<td>166</td>
</tr>
<tr>
<td>DirectoryMove</td>
<td>167</td>
</tr>
<tr>
<td>DirectorySelect</td>
<td>168</td>
</tr>
<tr>
<td>FileAppend</td>
<td>169</td>
</tr>
<tr>
<td>FileCopy</td>
<td>170</td>
</tr>
<tr>
<td>FileDelete</td>
<td>171</td>
</tr>
<tr>
<td>FileEdit</td>
<td>172</td>
</tr>
<tr>
<td>FileExists</td>
<td>173</td>
</tr>
<tr>
<td>FileMove</td>
<td>174</td>
</tr>
<tr>
<td>FilePrint</td>
<td>175</td>
</tr>
<tr>
<td>FileSelect</td>
<td>176</td>
</tr>
<tr>
<td>FileSelectNew</td>
<td>177</td>
</tr>
<tr>
<td>FileTime</td>
<td>178</td>
</tr>
<tr>
<td>FileView</td>
<td>179</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Distribution and Combinatoric Functions</th>
<th>180</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beta</td>
<td>181</td>
</tr>
<tr>
<td>Binomial</td>
<td>182</td>
</tr>
<tr>
<td>Combination</td>
<td>183</td>
</tr>
<tr>
<td>Exponential</td>
<td>184</td>
</tr>
<tr>
<td>ExtremeValue</td>
<td>185</td>
</tr>
<tr>
<td>Factorial</td>
<td>186</td>
</tr>
<tr>
<td>Gamma</td>
<td>187</td>
</tr>
<tr>
<td>Geometric</td>
<td>188</td>
</tr>
<tr>
<td>HyperGeometric</td>
<td>189</td>
</tr>
<tr>
<td>Logistic</td>
<td>190</td>
</tr>
<tr>
<td>LogNormal</td>
<td>191</td>
</tr>
<tr>
<td>NegativeBinomial</td>
<td>192</td>
</tr>
<tr>
<td>Normal</td>
<td>193</td>
</tr>
<tr>
<td>Pareto</td>
<td>194</td>
</tr>
<tr>
<td>Permutation</td>
<td>195</td>
</tr>
<tr>
<td>Poisson</td>
<td>196</td>
</tr>
<tr>
<td>Triangular</td>
<td>197</td>
</tr>
<tr>
<td>Uniform</td>
<td>198</td>
</tr>
<tr>
<td>Weibull</td>
<td>199</td>
</tr>
<tr>
<td><strong>Histogram Functions</strong></td>
<td>200</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----</td>
</tr>
<tr>
<td>HistogramAddObservation</td>
<td>201</td>
</tr>
<tr>
<td>HistogramCreate</td>
<td>202</td>
</tr>
<tr>
<td>HistogramDelete</td>
<td>203</td>
</tr>
<tr>
<td>HistogramGetAverage</td>
<td>204</td>
</tr>
<tr>
<td>HistogramGetBounds</td>
<td>205</td>
</tr>
<tr>
<td>HistogramGetDeviation</td>
<td>206</td>
</tr>
<tr>
<td>HistogramGetFrequencies</td>
<td>207</td>
</tr>
<tr>
<td>HistogramGetKurtosis</td>
<td>208</td>
</tr>
<tr>
<td>HistogramGetObservationCount</td>
<td>209</td>
</tr>
<tr>
<td>HistogramGetSkewness</td>
<td>210</td>
</tr>
<tr>
<td>HistogramSetDomain</td>
<td>211</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Matrix Manipulation Functions</strong></th>
<th>212</th>
</tr>
</thead>
<tbody>
<tr>
<td>MatrixActivateRow</td>
<td>213</td>
</tr>
<tr>
<td>MatrixAddColumn</td>
<td>214</td>
</tr>
<tr>
<td>MatrixAddRow</td>
<td>215</td>
</tr>
<tr>
<td>MatrixDeactivateRow</td>
<td>216</td>
</tr>
<tr>
<td>MatrixFreezeColumn</td>
<td>217</td>
</tr>
<tr>
<td>MatrixModifyCoefficient</td>
<td>218</td>
</tr>
<tr>
<td>MatrixModifyColumnType</td>
<td>219</td>
</tr>
<tr>
<td>MatrixModifyDirection</td>
<td>220</td>
</tr>
<tr>
<td>MatrixModifyLeftHandSide</td>
<td>221</td>
</tr>
<tr>
<td>MatrixModifyLowerBound</td>
<td>222</td>
</tr>
<tr>
<td>MatrixModifyRightHandSide</td>
<td>223</td>
</tr>
<tr>
<td>MatrixModifyRowType</td>
<td>224</td>
</tr>
<tr>
<td>MatrixModifyType</td>
<td>225</td>
</tr>
<tr>
<td>MatrixModifyUpperBound</td>
<td>226</td>
</tr>
<tr>
<td>MatrixReSolve</td>
<td>227</td>
</tr>
<tr>
<td>MatrixRegenerateRow</td>
<td>228</td>
</tr>
<tr>
<td>MatrixRestoreState</td>
<td>229</td>
</tr>
<tr>
<td>MatrixSaveState</td>
<td>230</td>
</tr>
<tr>
<td>MatrixUnfreezeColumn</td>
<td>231</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Licensing Functions</strong></th>
<th>232</th>
</tr>
</thead>
<tbody>
<tr>
<td>AimmsRevisionString</td>
<td>233</td>
</tr>
<tr>
<td>LicenseExpirationDate</td>
<td>234</td>
</tr>
<tr>
<td>LicenseMaintenanceExpirationDate</td>
<td>235</td>
</tr>
<tr>
<td>LicenseNumber</td>
<td>236</td>
</tr>
<tr>
<td>LicenseStartDate</td>
<td>237</td>
</tr>
<tr>
<td>LicenseType</td>
<td>238</td>
</tr>
<tr>
<td>VARLicenseExpirationDate</td>
<td>239</td>
</tr>
</tbody>
</table>
### Miscellaneous Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delay</td>
<td>241</td>
</tr>
<tr>
<td>Execute</td>
<td>242</td>
</tr>
<tr>
<td>ExitAimms</td>
<td>243</td>
</tr>
<tr>
<td>OpenDocument</td>
<td>244</td>
</tr>
<tr>
<td>ProjectDeveloperMode</td>
<td>245</td>
</tr>
<tr>
<td>ScheduleAt</td>
<td>246</td>
</tr>
<tr>
<td>SecurityGetGroups</td>
<td>247</td>
</tr>
<tr>
<td>SecurityGetUsers</td>
<td>248</td>
</tr>
<tr>
<td>SessionArgument</td>
<td>249</td>
</tr>
<tr>
<td>ShowHelpTopic</td>
<td>250</td>
</tr>
</tbody>
</table>

---

**Part II Index**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>251</td>
</tr>
</tbody>
</table>

**Index**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>252</td>
</tr>
</tbody>
</table>
Part I

Function Reference
Arithmetic Functions

AIMMS supports the following arithmetic functions:

- Abs
- ArcCosh
- ArcCos
- ArcSin
- ArcSinh
- ArcTanh
- ArcTan
- Ceil
- Cos
- Cosh
- Degrees
- Erf
- Exp
- Floor
- Log
- Log10
- MapVal
- Max
- Min
- Mod
- Power
- Radians
- Round
- Sign
- Sin
- Sinh
- Sqr
- Sqrt
- Tan
- Tanh
- Trunc
Abs

Abs(
    x  ! (input) numerical expression
)

**Arguments:**

x

A scalar numerical expression.

**Return value:**

The function Abs returns the absolute value of x.

**Remarks:**

The function Abs can be used in constraints of nonlinear mathematical programs. However, nonlinear solvers may experience convergence problems if the argument assumes values around 0.

**See also:**

Arithmetic functions are discussed in full detail in Section 6.1.4 of the Language Reference.
ArcCos

ArcCos(
  x ! (input) numerical expression
)

Arguments:

x
A scalar numerical expression in the range \([-1, 1]\).

Return value:

The ArcCos function returns the arccosine of \(x\) in the range 0 to \(\pi\) radians.

Remarks:

- A run-time error results if \(x\) is outside the range \([-1, 1]\).
- The function ArcCos can be used in constraints of nonlinear mathematical programs.

See also:

The functions ArcSin, ArcTan, Cos. Arithmetic functions are discussed in full detail in Section 6.1.4 of the Language Reference.
**ArcCosh**

```
ArcCosh(
    x ! (input) numerical expression
)
```

**Arguments:**

`x`

A scalar numerical expression in the range $[1, \infty)$. 

**Return value:**

The ArcCosh function returns the inverse hyperbolic cosine of $x$ in the range from 0 to $\infty$. 

**Remarks:**

- A run-time error results if $x$ is outside the range $[1, \infty]$. 
- The function ArcCosh can be used in constraints of nonlinear mathematical programs. 

**See also:**

The functions ArcSinh, ArcTanh, Cosh. Arithmetic functions are discussed in full detail in Section 6.1.4 of the Language Reference.
**ArcSin**

ArcSin(
    x ! (input) numerical expression
)

**Arguments:**

- **x**
  
  A scalar numerical expression in the range $[-1, 1]$.

**Return value:**

The ArcSin function returns the arcsine of $x$ in the range $-\pi/2$ to $\pi/2$ radians.

**Remarks:**

- A run-time error results if $x$ is outside the range $[-1, 1]$.
- The function ArcSin can be used in constraints of nonlinear mathematical programs.

**See also:**

The functions ArcCos, ArcTan, Sin. Arithmetic functions are discussed in full detail in Section 6.1.4 of the Language Reference.
ArcSinh

\[
\text{ArcSinh}(x) \quad \text{! (input) numerical expression}
\]

**Arguments:**

\[x\]

A scalar numerical expression.

**Return value:**

The ArcSinh function returns the inverse hyperbolic sine of \(x\) in the range from \(-\infty\) to \(\infty\).

**Remarks:**

The function \(\text{ArcSinh}\) can be used in constraints of nonlinear mathematical programs.

**See also:**

The functions \(\text{ArcCosh}, \text{ArcTanh}, \text{Sinh}\). Arithmetic functions are discussed in full detail in Section 6.1.4 of the Language Reference.
ArcTan

ArcTan(
    x ! (input) numerical expression
)

Arguments:

    x
    A scalar numerical expression.

Return value:

    The ArcTan function returns the arctangent of x in the range \(-\pi/2\) to \(\pi/2\) radians.

Remarks:

    The function ArcTan can be used in constraints of nonlinear mathematical programs.

See also:

    The functions ArcSin, ArcCos, Tan. Arithmetic functions are discussed in full detail in Section 6.1.4 of the Language Reference.
**Arithmetic Functions**

ArcTanh

ArcTanh(
  x          ! (input) numerical expression
)

**Arguments:**

x

A scalar numerical expression in the range \((-1, 1)\).

**Return value:**

The ArcTanh function returns the inverse hyperbolic tangent of x.

**Remarks:**

- A run-time error results if x is outside the range \((-1, 1)\).
- The function ArcTanh can be used in constraints of nonlinear mathematical programs.

**See also:**

The functions ArcCosh, ArcSinh, Tanh. Arithmetic functions are discussed in full detail in Section 6.1.4 of the Language Reference.
Arithmetic Functions

Ceil

Ceil(  
  x ! (input) numerical expression  
)

Arguments:

x
  A scalar numerical expression.

Return value:

The function Ceil returns the smallest integer value ≥ x.

Remarks:

- The function Ceil will round to the nearest integer, if it lies within the equality tolerances equality_absolute_tolerance and equality_relative_tolerance.
- The function Ceil can be used in the constraints of nonlinear mathematical programs. However, nonlinear solvers may experience convergence problems around integer values.

See also:

The functions Floor, Round, Trunc. Arithmetic functions are discussed in full detail in Section 6.1.4 of the Language Reference. Numeric tolerances are discussed in Section 6.2.2 of the Language Reference.
Cos

\[ \cos(x) \]

Argument:
\[ x \]
A scalar numerical expression.

Return value:
The \( \cos \) function returns the cosine of \( x \) in the range \(-1\) to \(1\).

Remarks:
The function Cos can be used in constraints of nonlinear mathematical programs.

See also:
The functions Sin, Tan, ArcCos. Arithmetic functions are discussed in full detail in Section 6.1.4 of the Language Reference.
Arithmetic Functions

---

**Cosh**

Cosh(
    x ! (input) numerical expression
)

**Arguments:**

- **x**
  A scalar numerical expression.

**Return value:**

The Cosh function returns the hyperbolic cosine of x in the range 1 to \( \infty \).

**Remarks:**

The function Cosh can be used in constraints of nonlinear mathematical programs.

**See also:**

The functions Sinh, Tanh, ArcCosh. Arithmetic functions are discussed in full detail in Section 6.1.4 of the Language Reference.
Degrees

```plaintext
Degrees(
    x               ! (input) numerical expression
)
```

Arguments:

- `x`
  A scalar numerical expression.

Return value:

The function Degrees returns the value of `x` converted from radians to degrees.

Remarks:

The function Degrees can be used in constraints of linear and nonlinear mathematical programs.

See also:

The function Radians. Arithmetic functions are discussed in full detail in Section 6.1.4 of the Language Reference.
ErrorF

ErrorF(
x ! (input) numerical expression
)

Arguments:

x

A scalar numerical expression.

Return value:

The function ErrorF returns the error function value \( \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{t^2}{2}} dt \).

Remarks:

The function ErrorF can be used in constraints of nonlinear mathematical programs.

See also:

Arithmetic functions are discussed in full detail in Section 6.1.4 of the Language Reference.
Exp

\[ \text{Exp}( \quad x \quad ! (\text{input numerical expression}) \quad ) \]

**Arguments:**

- \( x \)
  
  A scalar numerical expression.

**Return value:**

The function \( \text{Exp} \) returns the exponential value \( e^x \).

**Remarks:**

The function \( \text{Exp} \) can be used in constraints of nonlinear mathematical programs.

**See also:**

The functions \( \text{Log, Log10} \). Arithmetic functions are discussed in full detail in Section 6.1.4 of the Language Reference.
Floor

```
Floor(
    x ! (input) numerical expression
)
```

**Arguments:**

- `x`  
  A scalar numerical expression.

**Return value:**

The function `Floor` returns the largest integer value $\leq x$.

**Remarks:**

- The function `Floor` will round to the nearest integer, if it lies within the equality tolerances `equality_absolute_tolerance` and `equality_relative_tolerance`.
- The function `Floor` can be used in the constraints of nonlinear mathematical programs. However, nonlinear solvers may experience convergence problems around integer values.

**See also:**

The functions `Ceil`, `Round`, `Trunc`. Arithmetic functions are discussed in full detail in Section 6.1.4 of the Language Reference. Numeric tolerances are discussed in Section 6.2.2 of the Language Reference.
Log

Log(
    x  ! (input) numerical expression
)

Arguments:

    x
    A scalar numerical expression in the range \((0, \infty)\).

Return value:

The function Log returns the natural logarithm \(\ln(x)\).

Remarks:

- A run-time error results if \(x\) is outside the range \((0, \infty)\).
- The function Log can be used in constraints of nonlinear mathematical programs.

See also:

The functions Exp, Log10. Arithmetic functions are discussed in full detail in Section 6.1.4 of the Language Reference.
**Log10**

\[
\text{Log10}(x) \quad ! \quad \text{(input) numerical expression}
\]

**Arguments:**

\(x\)

A scalar numerical expression in the range \((0, \infty)\).

**Return value:**

The function \text{Log10} returns the base-10 logarithm of \(x\).

**Remarks:**

- A run-time error results if \(x\) is outside the range \((0, \infty)\).
- The function \text{Log10} can be used in constraints of nonlinear mathematical programs.

**See also:**

The functions \text{Exp}, \text{Log}. Arithmetic functions are discussed in full detail in Section 6.1.4 of the Language Reference.
MapVal

MapVal(
    x ! (input) numerical expression
)

Arguments:

x

A scalar numerical expression.

Return value:

The function \texttt{MapVal} returns the (integer) mapping value of any real or special number \( x \), according to the following table.

<table>
<thead>
<tr>
<th>Value ( x )</th>
<th>Description</th>
<th>MapVal value</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{number}</td>
<td>any valid real number</td>
<td>0</td>
</tr>
<tr>
<td>UNDF</td>
<td>undefined (result of an arithmetic error)</td>
<td>4</td>
</tr>
<tr>
<td>NA</td>
<td>not available</td>
<td>5</td>
</tr>
<tr>
<td>INF</td>
<td>(+ \infty)</td>
<td>6</td>
</tr>
<tr>
<td>-INF</td>
<td>(- \infty)</td>
<td>7</td>
</tr>
<tr>
<td>ZERO</td>
<td>numerically indistinguishable from zero, but has the logical value of one.</td>
<td>8</td>
</tr>
</tbody>
</table>

See also:

Special numbers in AIMMS and the \texttt{MapVal} function are discussed in full detail in Section 6.1.1 of the Language Reference.
Arithmetic Functions

Max

Max(
    x1, ! (input) numerical expression
    x2, ! (input) numerical expression
    ..
)  

Arguments:

x1, x2, ...

Multiple numerical expressions.

Return value:

The function Max returns the largest number among x1, x2, ... .

Remarks:

The function Max can be used in constraints of nonlinear mathematical programs. However, nonlinear solvers may experience convergence problems if the first order derivatives of two arguments between which the Max function switches are discontinuous.

See also:

The function Min. Arithmetic functions are discussed in full detail in Section 6.1.4 of the Language Reference.
Min

Min(
    x1, ! (input) numerical expression
    x2, ! (input) numerical expression
    ..
)

Arguments:

x1,x2,...
    Multiple numerical expressions.

Return value:

The function Min returns the smallest number among x1, x2,....

Remarks:

The function Min can be used in constraints of nonlinear mathematical programs. However, nonlinear solvers may experience convergence problems if the first order derivatives of two arguments between which the Min function switches are discontinuous.

See also:

The function Max. Arithmetic functions are discussed in full detail in Section 6.1.4 of the Language Reference.
**Mod**

Mod(
    x, ! (input) numerical expression
    y ! (input) numerical expression
)

**Arguments:**

- **x**
  - A scalar numerical expression.
- **y**
  - A scalar numerical expression unequal to 0.

**Return value:**

The function Mod returns the remainder of \( x \) after division by \( y \).

**Remarks:**

- A run-time error results if \( y \) equals 0.
- The function Mod can be used in constraints of mathematical programs. However, nonlinear solver may experience convergence problems if \( x \) assumes values around multiples of \( y \).

**See also:**

Arithmetic functions are discussed in full detail in Section 6.1.4 of the Language Reference.
Power

Power(
    x, ! (input) numerical expression
    y ! (input) numerical expression
)

Arguments:

x
    A scalar numerical expression.

y
    A scalar numerical expression.

Return value:

The function Power returns $x$ raised to the power $y$.

Remarks:

- The following combination of arguments is allowed:
  - $x > 0$
  - $x = 0$ and $y > 0$
  - $x < 0$ and $y$ a positive integer
    In all other cases a run-time error will result.
- The function can be used in constraints of nonlinear mathematical programs.

See also:

Arithmetic functions are discussed in full detail in Section 6.1.4 of the Language Reference.
Radians

Radians(
    x ! (input) numerical expression
)

Arguments:

x
    A scalar numerical expression.

Return value:

The function Radians returns the value of x converted from degrees to radians.

Remarks:

The function Radians can be used in constraints of linear and nonlinear mathematical programs.

See also:

The function Degrees. Arithmetic functions are discussed in full detail in Section 6.1.4 of the Language Reference.
**Round**

```plaintext
Round(
    x ! (input) numerical expression
    n ! (optional) integer expression
)
```

**Arguments:**

- `x`: A scalar numerical expression.
- `n (optional)`: An integer expression.

**Return value:**

The function `Round` returns the integer value nearest to `x`. In the presence of the optional argument `n` the function `Round` returns the value of `x` rounded to `n` decimal places left (`n < 0`) or right (`n > 0`) of the decimal point.

**Remarks:**

The function `Round` can be used in constraints of nonlinear mathematical programs. However, nonlinear solvers may experience convergence problems around the discontinuities of the `Round` function.

**See also:**

The functions `Ceil`, `Floor`, `Trunc`. Arithmetic functions are discussed in full detail in Section 6.1.4 of the Language Reference.
Sign

\[ \text{Sign}(x) \]

\[ x \quad \text{! (input) numerical expression} \]

**Arguments:**

- \( x \)
  
  A scalar numerical expression.

**Return value:**

The function \( \text{Sign} \) returns +1 if \( x > 0 \), −1 if \( x < 0 \) and 0 if \( x = 0 \).

**Remarks:**

The function \( \text{Sign} \) can be used in constraints of nonlinear mathematical programs. However, nonlinear solver may experience convergence problems round 0.

**See also:**

The function \( \text{Abs} \). Arithmetic functions are discussed in full detail in Section 6.1.4 of the Language Reference.
Sin

\[ \text{Sin}( \ x \quad \! \text{(input) numerical expression} \quad ) \]

Arguments:

\[ x \]

A scalar numerical expression.

Return value:

The \(\text{Sin}\) function returns the sine of \(x\) in the range \(-1\) to \(1\).

Remarks:

The function \(\text{Sin}\) can be used in constraints of nonlinear mathematical programs.

See also:

The functions \(\text{Cos}, \text{Tan}, \text{ArcSin}\). Arithmetic functions are discussed in full detail in Section 6.1.4 of the Language Reference.
Sinh

\begin{verbatim}
Sinh(
x     ! (input) numerical expression
)
\end{verbatim}

**Arguments:**

\begin{verbatim}
x
A scalar numerical expression.
\end{verbatim}

**Return value:**

The `Sinh` function returns the hyperbolic sine of \( x \) in the range \(-\infty \) to \( \infty \).

**Remarks:**

The function `Sinh` can be used in the constraints of nonlinear mathematical programs.

**See also:**

The functions `Cosh`, `Tanh`, `ArcSinh`. Arithmetic functions are discussed in full detail in Section 6.1.4 of the Language Reference.
**Sqr**

\[ Sqr( \quad x \quad \text{! (input) numerical expression} \quad ) \]

**Arguments:**

- **x**
  - A scalar numerical expression.

**Return value:**

The function `Sqr` returns \( x^2 \).

**Remarks:**

The function `Sqr` can be used in constraints of nonlinear mathematical programs.

**See also:**

The functions `Power`, `Sqrt`. Arithmetic functions are discussed in full detail in Section 6.1.4 of the Language Reference.
Sqrt

Sqrt(
    x                   ! (input) numerical expression
)

Arguments:

x

A scalar numerical expression in the range \([0, \infty)\).

Return value:

The function Sqrt returns the \(\sqrt{x}\).

Remarks:

- A run-time error results if \(x\) is outside the range \([0, \infty)\).
- The function Sqrt can be used in the constraints of nonlinear mathematical programs.

See also:

The functions Power, Sqr. Arithmetic functions are discussed in full detail in Section 6.1.4 of the Language Reference.
Tan

\[
\text{Tan}(x) \quad \text{! (input) numerical expression}
\]

Arguments:

- \(x\)
  - A scalar numerical expression in the range \((-\pi/2, \pi/2)\).

Return value:

The Tan function returns the tangent of \(x\) in the range \(-\infty\) to \(\infty\).

Remarks:

- A run-time error results if \(x\) is outside the range \((-\pi/2, \pi/2)\).
- The function Tan can be used in constraints of nonlinear mathematical programs.

See also:

The functions Cos, Sin, ArcTan. Arithmetic functions are discussed in full detail in Section 6.1.4 of the Language Reference.
Tanh

Tanh(  
    x                  ! (input) numerical expression  
)  

Arguments:

x

A scalar numerical expression.

Return value:

The Tanh function returns the hyperbolic tangent of x in the range −1 to 1.

Remarks:

The function Tanh can be used in constraints of nonlinear mathematical programs.

See also:

The functions Cosh, Sinh, ArcTanh. Arithmetic functions are discussed in full detail in Section 6.1.4 of the Language Reference.
Trunc

```plaintext
Trunc(
    x ! (input) numerical expression
)
```

**Arguments:**

- `x`
  - A scalar numerical expression.

**Return value:**

The function Trunc returns the truncated value of $x$: $\text{Sign}(x) \cdot \text{Floor}(\text{Abs}(x))$.

**Remarks:**

- The function Trunc will round to the nearest integer, if it lies within the equality tolerances `equality_absolute_tolerance` and `equality_relative_tolerance`.
- The function Trunc can be used in the constraints of nonlinear mathematical programs. However, nonlinear solver may experience convergence problems around integer argument values.

**See also:**

The functions Ceil, Floor, Round. Arithmetic functions are discussed in full detail in Section 6.1.4 of the Language Reference. Numeric tolerances are discussed in Section 6.2.2 of the Language Reference.
Set Related Functions

AIMMS supports the following set related functions:

- Card
- ConstraintVariables
- Element
- ElementCast
- ElementRange
- FindUsedElements
- Ord
- RestoreInactiveElements
- RetrieveCurrentVariableValues
- SetElementAdd
- SetElementRename
- StringToElement
- SubRange
- VariableConstraints
Card

The function Card returns the cardinality of its identifier argument.

\[ \text{Card}(Id) \]

**Arguments:**

- *Id*  
  A reference to a set or an indexed identifier.

**Return value:**

If *Id* is a set, the function Card returns the number of elements in *Id*. If *Id* is an indexed identifier, the function Card returns the number of nondefault values stored for *Id*.

**Remarks:**

The Card function cannot be applied to slices of indexed identifiers. In such a case, you can use the Count operator to count the number of nondefault elements.

**See also:**

The Count operator (see also Section 6.1.6 of the Language Reference).
**ConstraintVariables**

The function `ConstraintVariables` returns all the symbolic variables that are referred in a certain collection of constraints, including the variables that are referred in the definitions of these variables.

```plaintext
ConstraintVariables(
  contraint-set ! (input) a subset of AllConstraints
)
```

**Arguments:**

- `contraint-set` 
  The set of constraints for which you want to retrieve the referred variables.

**Return value:**

The function returns a subset of the set `AllVariables`, containing the variables found.

**See also:**

The function `VariableConstraints`.
**Element**

With the function `Element` you can retrieve the \( n \)-th element from a set.

```vbnet
Element(S, n)
```

**Arguments:**

- **S**
  The set from which an element is to be returned.

- **n**
  An integer expression indicating the ordinal number of the element to be returned.

**Return value:**

The function `Element` returns the \( n \)-th element of set \( S \).

**Remarks:**

If there is no \( n \)-th element in \( S \), the function returns the empty element `'` instead.
**ElementCast**

With the function `ElementCast` you can cast an element of one set to an (existing) element with the same name in a set with a different root set.

```plaintext
ElementCast(
    set, ! (input) a set expression
    element-name, ! (input) a scalar element expression
    [create] ! (optional) 0 or 1
)
```

**Arguments:**

- **set**
  A set in which you want to find a specific element name.

- **element-name**
  A scalar element expression, representing the element that you want to convert to a different root set hierarchy.

- **create (optional)**
  An indicator whether or not a nonexisting element are added to the set during the call.

**Return value:**

The function returns the existing element or, if the element cannot be converted to an existing element and the argument `create` is not set to 1, the function returns the empty element. If `create` is set to 1, nonexisting elements will be created on the fly.

**See also:**

The procedure `SetElementAdd`.
**ElementRange**

With the function `ElementRange` you can create a set with elements in which each element can be constructed using a prefix string, a postfix string, and a a sequential number.

```
ElementRange(
    first, ! (input) integer expression
    last, ! (input) integer expression
    [increment,] ! (optional) integer expression
    [prefix,] ! (optional) string expression
    [postfix,] ! (optional) string expression
    [fill] ! (optional) 0 or 1
)
```

**Arguments:**

`first`
The integer value for which the first element must be created

`last`
The integer value for which the last element must be created

`increment (optional)`
The integer-valued interval length between two consecutive elements. If omitted, then the default interval length of 1 is used.

`prefix (optional)`
The prefix string for every element. If omitted, then the elements have no prefix (and thus start with the number).

`postfix (optional)`
The postfix string for every element. If omitted, then the elements have no postfix (and thus end with the number).

`fill (optional)`
This logical indicator specifies whether the numbers must be padded with leading zeroes. If omitted, then the default value 1 is used.

**Return value:**
The function returns a set containing the created elements.
FindUsedElements

The procedure FindUsedElements finds all elements of a particular set that are in use in a given collection of indexed model identifiers.

FindUsedElements(
    source-set, ! (input) a set
    identifier-set, ! (input) a subset of AllIdentifiers
    result-set ! (output) a subset
)

Arguments:

source-set
    The set for which you want to find the used elements.

identifier-set
    A subset of AllIdentifiers, holding identifiers that are indexed over source-set.

result-set
    A subset of source-set. On return this subset will contain the elements that are currently used (i.e. have corresponding nondefault values) in the identifiers contained in identifier-set.
The function `Ord` returns the ordinal number of a set element relative to a set.

```plaintext
Ord(e, ! (input) element expression
[S] ! (optional) set reference
)
```

**Arguments:**

- `e`
  - An element expression for which you want to obtain the ordinal number.

- `S (optional)`
  - The set with respect to which you want the ordinal number to be taken. If omitted, `S` is assumed to be the range of the argument `e`.

**Return value:**

The function `Ord` returns the ordinal number of `e` in set `S`.

**Remarks:**

A compile time error occurs if the argument `S` is not present, and AIMMS is unable to determine the range of `e`. 
**RestoreInactiveElements**

The procedure RestoreInactiveElements finds and restores all elements that were previously removed from a particular set, but for which inactive data still exists in a given collection of indexed model identifiers.

\[
\text{RestoreInactiveElements}( \text{search-set}, \text{identifier-set}, \text{result-set})
\]

**Arguments:**

**search-set**

The set for which you want to find the inactive elements.

**identifier-set**

A subset of AllIdentifiers, holding identifiers that are indexed over **search-set**.

**result-set**

A subset of **search-set**. On return this subset will contain all the inactive elements that are currently used (i.e. have corresponding nondefault values) in the identifiers contained in **identifier-set**.

**Remarks:**

The inactive elements found are placed in the **result-set**, but are also automatically added to the **search-set**.
RetrieveCurrentVariableValues

With the procedure RetrieveCurrentVariableValues you can obtain the variable values for a given collection of variables during a running solution process. This procedure can only be called from within the context of a solver callback procedure.

```
RetrieveCurrentVariableValues(
    variable-set   ! (input) a subset of AllVariables
)
```

Arguments:

- `variable-set`  
  A subset of AllVariables, holding all the variables for which you want to retrieve the current values.

See also:

Solver callback procedures are discussed in full detail in Section 15.2 of the Language Reference.
SetElementAdd

With the procedure SetElementAdd you can add new elements to a set. When you apply SetElementAdd to a root set, the element will be added to that root set. When you apply it to a subset, the element will be added to the subset as well as to all its supersets, up to and including its associated root set.

SetElementAdd(
    set, ! (input) a set
    new-element, ! (output) an element parameter
    element-name ! (input) a scalar string expression
)

Arguments:

set
    The root set or subset to which you want to add the element.

new-element
    An element parameter into set, that on return will point to the newly added element.

element-name
    A string holding the name of the element to be added.

Remarks:

If the element already exists in the set, the procedure does not make any changes to the set, and on return the element parameter new-element will point to the existing element.

See also:

The function ElementCast. The procedures SetElementRename, StringToElement.
**SetElementRename**

With the procedure `SetElementRename` you can rename an element in a set.

```plaintext
SetElementRename(
    set, ! (input) a set
    element, ! (input) an element
    new-name ! (input) a scalar string expression
)
```

**Arguments:**

- **set**
  The root set or subset in which you want to rename an element.
- **element**
  The element that you want to rename.
- **new-name**
  A string holding the new name of the element.

**Remarks:**

If the new name for the element already exists in the set, the procedure will generate an execution error.

**See also:**

The procedure `SetElementAdd`. The function `StringToElement`. 

---

*Set Related Functions*
**StringToElement**

With the function StringToElement you can convert a string into an (existing) element of a set.

```
StringToElement(
    set, ! (input) a set expression
    element-name, ! (input) a scalar string
    [create] ! (optional) 0 or 1
)
```

**Arguments:**

- `set`  
  A set in which you want to find a specific element name.

- `element-name`  
  A scalar string expression, representing the string that you want to convert.

- `create` (optional)  
  An indicator whether or not a nonexisting element are added to the set during the call.

**Return value:**

The function returns the existing element or, if the string cannot be converted to an existing element and the argument `create` is not set to 1, the function return the empty element. If `create` is set to 1, nonexisting elements will be created on the fly.

**See also:**

The function `ElementCast`. The procedure `SetElementAdd`.
SubRange

The function SubRange extracts a subrange of consecutive elements from an existing set.

SubRange(set, ! (input) a simple or compound set
            first, ! (input) an element
            last, ! (input) an element)

Arguments:

set
  The set containing the subrange of elements that you want to extract.

first
  An element in set representing the first element of the subrange.

last
  An element in set representing the last element of the subrange.

Return value:

The function returns a set containing the subrange of elements extracted from set. If the element first is positioned after last, then the empty set is returned.
VariableConstraints

The function VariableConstraints returns all the symbolic constraints that refer to one or more variables in a given set of variables.

```
VariableConstraints(
    variable-set  ! (input) a subset of AllVariables
)
```

Arguments:

- `variable-set`
  - The set of variables for which you want to retrieve the constraints that refer to them.

Return value:

The function returns a subset of the set AllConstraints, containing the constraints found.

See also:

- The function ConstraintVariables.
String Manipulation Functions

AIMMS supports the following functions for manipulating strings:

- `FindNthString`
- `FindString`
- `FormatString`
- `StringCapitalze`
- `StringLength`
- `StringOccurences`
- `StringToLower`
- `StringToUpper`
- `SubString`
FindNthString

The function FindNthString searches for the \( n \)-th occurrence of a substring (a key) within a search string.

\[
\text{FindNthString(}
\begin{align*}
\text{searchstring}, & \quad ! \text{ (input) a scalar string expression} \\
\text{key}, & \quad ! \text{ (input) a scalar string expression} \\
n & \quad ! \text{ (input) an integer expression}
\end{align*}
\)

Arguments:

- \( \text{searchstring} \): The string in which you want to find the substring \( \text{key} \).
- \( \text{key} \): The substring to search for.
- \( n \): The function will search for the \( n \)-th occurrence of the substring. If this number is negative, then the function will search backwards starting from the right.

Remarks:

As with all string comparisons within AIMMS, the function FindNthString is case-insensitive.

Return value:

The function returns the start position of the \( n \)-th occurrence of the substring starting from the left (or right). If the substring does not exist within the string, or does not occur \( n \) times then the function returns 0.

See also:

The functions FindString, StringOccurrences.
FindString

The function FindString searches for the occurrence of a substring (a key) within a search string.

```aimms
FindString(
    searchstring, ! (input) a scalar string expression
    key       ! (input) a scalar string expression
)
```

Arguments:

- `searchstring`:
  The string in which you want to find the substring `key`.

- `key`:
  The substring to search for.

Remarks:

As with all string comparisons within AIMMS, the function FindString is case-insensitive.

Return value:

The function returns the start position of the first occurrence of the substring. If the substring does not exist, then the function returns 0.

See also:

The function `FindNthString`.
String Manipulation Functions

FormatString

With the FormatString function you can compose a string that is built up from combinations of numbers, strings and set elements. The FormatString function accepts a varying number of arguments, defined by the conversion specifiers in the format string.

FormatString(
  formatstring, ! (input) a scalar string expression
  arguments,   ! (input) a list of numbers, strings, and set elements
  ...
)

Arguments:

formatstring
  A format string that specifies how the returned string is composed. The string should contain the proper conversion specifier for each following argument.

arguments,...
  One or more arguments of type number, string or element. The order of these arguments must coincide with the order of the conversion specifiers in formatstring.

Return value:

The function returns the formatted string.

See also:

For a detailed description of the conversion specifiers in AIMMS see Section 5.3.2 of the Language Reference.
StringManipulationFunctions

StringCapitalize

The function StringCapitalize converts the first character of a string to upper case, and all other characters to lower case.

```haskell
StringCapitalize(  
  text ! (input) a scalar string expression  
)
```

Arguments:

- **text**
  - The string that you want to capitalize.

Return value:

- The function returns the capitalized string.

See also:

- The functions StringToLower, StringToUpper.
String Manipulation Functions

StringLength

The function `StringLength` returns the number of characters in a string.

```cpp
StringLength(
    text ! (input) a scalar string expression
)
```

**Arguments:**

- `text`
  - The string for which you want to retrieve the length.

**Return value:**

The function returns the number of characters in the string.
String Occurrences

The function **StringOccurrences** counts the number of occurrences of a particular substring in a string.

```plaintext
StringOccurrences(
    searchstring, ! (input) a string expression
    substring   ! (input) a string expression
)
```

**Arguments:**

- **searchstring**
  A string in which you want to find the substring(s).

- **substring**
  The substring.

**Return value:**

The function returns how many occurrences of the substring exist.

**See also:**

The functions **FindString**, **FindNthString**.
StringManipulationFunctions

StringToLower

The function StringToLower converts all characters of a string to lower case.

\[
\text{StringToLower(}
\text{text} \quad ! \text{(input) a scalar string expression}
\text{)}
\]

Arguments:

\text{text}

The string that you want to convert to lower case characters.

Return value:

The function returns the lower case string.

See also:

The functions \text{StringToUpper}, \text{StringCapitalize}.
StringToUpper

The function `StringToUpper` converts all characters of a string to upper case.

```
StringToUpper(
    text ! (input) a scalar string expression
)
```

Arguments:

- `text`
  - The string that you want to convert to upper case characters.

Return value:

The function returns the upper case string.

See also:

The functions `StringToLower`, `StringCapitalize`. 
SubString

The function SubString retrieves a substring from a specific string, based on the start and end position of this substring within this string.

```
SubString(
    text, ! (input) a scalar string expression
    begin-pos, ! (input) an integer value
    end-pos   ! (input) an integer value
)
```

Arguments:

- **text**
  
  The string from which you want to retrieve the substring.

- **begin-pos**
  
  The start position of the substring within `txt`.

- **end-pos**
  
  The end position of the substring within `txt`.

Return value:

The function returns the requested substring.

Remarks:

If the arguments `begin-pos` and `end-pos` are positive, then the position is calculated from the start of the string (i.e. the first character is on position 1). If the arguments `begin-pos` and `end-pos` are negative, then the position is calculated from the end of the string (i.e. the last character is on position $-1$). `begin-pos` must be less than or equal to `end-pos`, and if either of the values exceeds the length of the string, they are automatically set within the proper range.
Time Functions

AIMMS supports the following time-related functions:

- Aggregate
- CreateTimeTable
- CurrentToMoment
- CurrentToString
- CurrentToTimeSlot
- DisAggregate
- MomentToString
- MomentToTimeSlot
- PeriodToString
- StringToMoment
- StringToTimeSlot
- TimeSlotCharacteristic
- TimeSlotToMoment
- TimeSlotToString
Aggregate

With the procedure Aggregate you can aggregate time-dependent data from a calendar time scale (time slots) to a horizon time scale (periods).

```
Aggregate(
    timeslot-data, ! (input) an indexed identifier over a calendar
    period-data, ! (output) an indexed identifier over a horizon
    timetable, ! (input) an AIMMS time table
    type, ! (input) an element in the set AggregationTypes
    [locus] ! (optional) a value between 0 and 1
)
```

Arguments:

timeslot-data
An identifier (slice) containing the data to be aggregated. The domain sets in the index domain of this identifier should at least contain a calendar set, and all other sets should coincide with the domain of period-data.

period-data
An identifier (slice) that on return will contain the aggregated data. The domain sets in the index domain of this identifier should at least contain a horizon set, and all other sets should coincide with the domain of timeslot-data.

timetable
An one-dimensional element parameter over a horizon, and with range a calendar. This horizon and calendar should match with the index domains of timeslot-data and period-data.

type
An element of the pre-defined set AggregationTypes (summation, average, maximum, minimum, or interpolation).

locus (only for interpolation type)
A number between 0 and 1, that indicates at which moment in a period the quantity is to be measured.

See also:
The procedure DisAggregate. Time-dependent aggregation and disaggregation is discussed in full detail in Section 24.5 of the Language Reference.
**CreateTimeTable**

With the procedure `CreateTimeTable` you can create a timetable in AIMMS.

```plaintext
CreateTimeTable(
    timetable, ! (output) an indexed set
    current-timeslot, ! (input) an element in a calendar
    current-period, ! (input) an element in a horizon
    period-length, ! (input) one-dimensional integer parameter
    length-dominates, ! (input) one-dimensional binary parameter
    inactive-slots, ! (input) a subset of a calendar
    delimiter-slots ! (input) a subset of a calendar
)
```

**Arguments:**

- **timetable**
  An indexed set in a calendar and defined over the horizon to be linked to the calendar. This argument implicitly sets the calendar and horizon used for the creation of the timetable. The other arguments of the procedure should match with this calendar and horizon.

- **current-timeslot**
  An element of a calendar (a time slot) that should be aligned with the **current-period** in the horizon.

- **current-period**
  An element of a horizon (a period) that should be aligned with the **current-timeslot** in the calendar.

- **period-length**
  A one-dimensional integer parameter, specifying the desired length of each period in the horizon in terms of the number of time slots to be contained in it.

- **length-dominates**
  A one-dimensional binary parameter, indicating whether reaching the specified **period-length** dominates over the presence of any delimiter slot for every period in the horizon.

- **inactive-slots**
  A subset of the calendar, indicating the time slots that must be excluded from the timetable.

- **delimiter-slots**
  A subset of the calendar, indicating the time slots that will (usually) result in starting a new period in the horizon.

**See also:**

The procedures `Aggregate`, `DisAggregate`. For a more detailed description of the creation of timetables, see Section 24.4 of the Language Reference.
**CurrentToMoment**

The function `CurrentToMoment` converts the current time to the elapsed time with respect to a specific reference date.

```plaintext
CurrentToMoment(
    unit, ! (input) a time unit
    reference-date ! (input) a string expression
)
```

**Arguments:**

- `unit`
  - The time unit that is used to return the elapsed time.

- `reference-date`
  - A string that holds the begin date using the fixed format for date and time.

**Return value:**

The result of `CurrentToMoment` is the elapsed time in `unit` since `reference-date`.

**See also:**

The function `StringToMoment`.
**CurrentToString**

The function `CurrentToString` creates a string representation of the current time in the specified format.

```plaintext
CurrentToString(
    format ! (input) a string expression
)
```

**Arguments:**

- `format`
  A string that holds the date and time format used in the returned string.

**Return value:**

The result of `CurrentToString` is a description of the current time according to `format`.

**See also:**

The functions `MomentToString`, `CurrentToMoment`. 
CurrentToTimeSlot

The function CurrentToTimeSlot determines the time slot in a calendar that corresponds with the current time.

```haskell
CurrentToTimeSlot(
    calendar ! (input) a calendar
)
```

**Arguments:**

`calendar`
An identifier of type calendar.

**Return value:**

The function CurrentToTimeSlot returns the time slot in the calendar that contains the current moment.

**See also:**

The functions `StringToTimeSlot`, `MomentToTimeSlot`. 
DisAggregate

With the procedure DisAggregate you can disaggregate time-dependent data from a horizon time scale (periods) to a calendar time scale (time slots).

\[
\text{DisAggregate}(\text{period-data}, \text{timeslot-data}, \text{timetable}, \text{type}, \text{locus})
\]

Arguments:

- **period-data**
  - An identifier (slice) containing the data to be disaggregated. The domain sets in the index domain of this identifier should at least contain a horizon set, and all other sets should coincide with the domain of timeslot-data.

- **timeslot-data**
  - An identifier (slice) that on returns will contain the disaggregated data. The domain sets in the index domain of this identifier should at least contain a calendar set, and all other sets should coincide with the domain of period-data.

- **timetable**
  - An one-dimensional element parameter over a horizon, and with range a calendar. This horizon and calendar should match with the index domains of timeslot-data and period-data.

- **type**
  - An element of the pre-defined set AggregationTypes (summation, average, maximum, minimum, or interpolation).

- **locus (only for interpolation type)**
  - A number between 0 and 1, that indicates at which moment in a period the quantity is to be measured.

See also:

The procedure Aggregate. Time-dependent aggregation and disaggregation is discussed in full detail in Section 24.5 of the Language Reference.
MomentToString

The function MomentToString creates a string representation of a moment, that is calculated from a given amount of elapsed time since a specific reference date.

\[
\text{MomentToString}(\text{format-string}, \text{unit}, \text{reference-date}, \text{elapsed-time})
\]

Arguments:

- **format-string**
  A string that holds the date and time format used in the returned string.

- **unit**
  The time unit that is used in the argument \text{elapsed-time}.

- **reference-date**
  A string that holds the begin date using the fixed format for date and time.

- **elapsed-time**
  A numerical value of the time elapsed since \text{reference-date}.

Return value:

The result of MomentToString is a string describing the corresponding moment according to \text{format-string}.

See also:

The function StringToMoment.
MomentToTimeSlot

The function MomentToTimeSlot determines the time slot in a calendar that corresponds with the moment that is specified as the elapsed time since a specific reference date.

\[
\text{MomentToTimeSlot}(\text{calendar}, \text{reference-time}, \text{elapsed-time})
\]

Arguments:

- \(\text{calendar}\): An identifier of type calendar.
- \(\text{reference-time}\): A string expression holding the reference time using the fixed date and time format.
- \(\text{elapsed-time}\): The elapsed time since \(\text{reference-time}\) measured in terms of the calendar’s unit.

Return value:

The function MomentToTimeSlot returns the time slot in the calendar that contains the given moment.

See also:

The functions TimeSlotToMoment, CurrentToTimeSlot, StringToTimeSlot.
PeriodToString

With the function PeriodToString you can obtain a description of a period in a timetable that consists of multiple calendar slots.

```plaintext
PeriodToString(
    format, ! (input) a string expression
timetable, ! (input) a time table
period ! (input) an element in a horizon
)
```

Arguments:

- **format**
  A string that holds the date and time format used in the returned string. This format string can contain period specific conversion specifiers to generate a description referring to both the beginning and end of the period.

- **timetable**
  An AIMMS timetable, this must be a one-dimensional element parameter in a calendar and indexed over a horizon.

- **period**
  An element in the horizon that is defined by timetable.

Return value:

The result of MomentToString is a string describing the corresponding moment according to format.

See also:

The procedure CreateTimeTable.
**StringToMoment**

The function `StringToMoment` converts a given time string (in a free time format) to the elapsed time with respect to a specific reference date.

```plaintext
StringToMoment(
    format,  ! (input) a string expression
    unit,    ! (input) a time unit
    reference-date, ! (input) a string expression
    moment-string  ! (input) a string expression
)
```

**Arguments:**

- `format`  
  A string that holds the date and time format used in the `moment-string`.

- `unit`  
  The time unit that is used to return the elapsed time.

- `reference-date`  
  A string that holds the begin date using the fixed format for date and time.

- `moment-string`  
  A string representing a specific date and time moment using the format specified in `format`.

**Return value:**

The result of `StringToMoment` is the elapsed time in `unit` between `reference-date` and `moment-string`.

**See also:**

The functions `MomentToString`, `CurrentToMoment`. 
**StringToTimeSlot**

The function `StringToTimeSlot` determines the time slot in a calendar that corresponds with the a moment that is specified using a free format string.

```plaintext
StringToTimeSlot(
    format, ! (input) a string expression
    calendar, ! (input) a calendar
    moment ! (input) a string expression
)
```

**Arguments:**

- **format**
  A string expression holding the format used in `moment`.

- **calendar**
  An identifier of type calendar.

- **moment**
  A string expression of the moment (using the format given in `format`) that should be matched with the time slots in the calendar.

**Return value:**

The function `StringToTimeSlot` returns the time slot in the calendar that contains the given moment.

**See also:**

The functions `CurrentToTimeSlot, MomentToTimeSlot`. 
Time Functions

**TimeSlotCharacteristic**

The function `TimeSlotCharacteristic` obtains a numeric value which characterizes the time slot, in terms of its day of the week, its day in the year, etc.

```plaintext
TimeSlotCharacteristic(
    time-slot,  ! (input) an element (time-slot) in a calendar
    characteristic  ! (input) an element in TimeslotCharacteristics
)
```

**Arguments:**

- `time-slot`
  A element refering to a time-slot in a calendar.

- `characteristic`
  An element in the predefined set `TimeslotCharacteristics`, each element in this set refers to a specific value that can be retrieved for a time slot.

**Return value:**

The function `TimeSlotCharacteristic` returns a numerical value for the requested time slot characteristic.

**See also:**

The function `TimeSlotCharacteristic` is discussed in full detail in Section 24.4 of the Language Reference.
Time Functions

TimeSlotToMoment

The function TimeSlotToMoment calculates the elapsed time since a specific reference date for a given time slot in a calendar.

\[
\text{TimeSlotToMoment}(
\text{calendar, } ! (\text{input}) \text{ a calendar}
\text{reference-time, } ! (\text{input}) \text{ a string expression}
\text{time-slot } ! (\text{input}) \text{ an element (time-slot) in the calendar}
\)
\]

Arguments:

\begin{itemize}
\item \textit{calendar}  
An identifier of type calendar.
\item \textit{reference-time}  
A string expression holding the reference time using the fixed date and time format.
\item \textit{time-slot}  
A specific time slot in the calendar.
\end{itemize}

Return value:

The function TimeSlotToMoment returns the elapsed time since the reference date for the given time slot (measured in the calendar's unit).

See also:

The functions MomentToTimeSlot, CurrentToTimeSlot, StringToTimeSlot.
TimeSlotToString

The function TimeSlotToString creates a string representation of a specific time slot in a calendar.

```prolog
TimeSlotToString(
    format, ! (input) a string expression
    calendar, ! (input) a calendar
    time-slot ! (input) an element (time-slot) in the calendar
)
```

Arguments:

- `format`:
  A time format used for the returned string representation.

- `calendar`:
  An identifier of type calendar.

- `time-slot`:
  A specific time-slot in the calendar.

Return value:

The function TimeSlotToString returns a string representation of the time slot.

See also:

The functions `MomentToString`, `CurrentToTimeSlot`, `StringToTimeSlot`. 
Database Functions

AIMMS supports the following database related functions:

- CloseDataSource
- DirectSQL
- TestDatabaseTable
- TestDataSource
**CloseDataSource**

With the function CloseDataSource you can temporarily close the connection to a data source. AIMMS automatically opens the connection to a data source if needed, and closes the connection when the project is exited.

```
CloseDataSource(
    data-source ! (input) a string expression
)
```

**Arguments:**

- **data-source**
  A string containing the name of a data source.
DirectSQL

With the procedure DirectSQL you can directly execute SQL statements within a data source.

\[
\text{DirectSQL}\left(\text{data-source, SQL-statement}\right)
\]

**Arguments:**

- **data-source**
  A string containing the name of a data source.

- **SQL-statement**
  A string containing the SQL statement that must be executed within the data source.

**Return value:**

The procedure returns 1 if the SQL statement is executed successfully, or 0 if the execution failed. In case of failure, the corresponding error message can be obtained through the predefined string parameter CurrentErrorMessage.

**Remarks:**

- If the SQL statement also produces a result set, then this set is ignored by AIMMS.
- Note that the SQL dialect used by, for instance, Oracle, SQL Server and Microsoft Access may differ. If a call to DirectSQL fails because of such differences, you should inspect CurrentErrorMessage for further details.

**See also:**

Calling stored procedures and executing SQL queries through AIMMS DATABASE PROCEDURES is discussed in Section 20.5 of the Language Reference.
TestDatabaseTable

With the function TestDatabaseTable you can check whether a given table name exists in a specific data source.

TestDatabaseTable(
    data-source, ! (input) a string expression
    table-name    ! (input) a string expression
)

Arguments:

data-source
   A string containing the name of a data source.

table-name
   A string containing a the name of a table in data-source.

Return value:

   The function returns 1 if the database table is present in the given data source, or 0 otherwise.

Remarks:

   The function TestDatabaseTable will not let you check whether the table contains the columns which you expect it to contain. If you try to access columns in the database table which are not present during either a READ or WRITE statement, AIMMS will generate a run-time error.

See also:

   The function TestDataSource.
TestDataSource

With the function TestDataSource you can test for the presence of a data source on a host computer, before reading or writing to it. If you try to read or write to a non-existing data source, AIMMS will generate error messages which may be confusing for your end users.

```
TestDataSource(
    data-source ! (input) a string expression
)
```

**Arguments:**

- `data-source`
  A string containing the name of a data source.

**Return value:**

The function returns 1 if the data source is present, or 0 otherwise.

**See also:**

The function TestDatabaseTable.
Case Functions

AIMMS supports the following functions for accessing the cases in the Data Manager:

- CaseCreate
- CaseDelete
- CaseFind
- CaseGetChangedStatus
- CaseGetDatasetReference
- CaseGetType
- CaseLoadCurrent
- CaseLoadIntoCurrent
- CaseMerge
- CaseNew
- CaseSave
- CaseSaveAll
- CaseSaveAs
- CaseSelect
- CaseSelectMultiple
- CaseSelectNew
- CaseSetChangedStatus
- CaseSetCurrent
CaseCreate

The function `CaseCreate` creates a new case node in the Data Management tree. The name of the case and the folder in which it is created is given as an argument to the function.

```plaintext
CaseCreate(
    case-path, ! (input) scalar string expression
    case        ! (output) element parameter into AllCases
)
```

**Arguments:**

- **case-path**
  A string expression holding the path and name of the new case. The path is specified relative to the root of the case tree.

- **case**
  An element parameter into `AllCases`. On successful return this parameter will refer to the newly created element in `AllCases`.

**Return value:**

The function returns 1 if the case is created successfully. It returns 0 if the case could not be created or if the case already exists.

**Remarks:**

If the specified path contains folders that do not exist, then these folders are created automatically. To check whether a specific case path already exists you can use the function `CaseFind`.

**See also:**

The function `CaseFind`, `CaseDelete`.
CaseDelete

The function **CaseDelete** deletes a specific case node from the Data Management tree.

```java
CaseDelete(
    case ! (input) element parameter into AllCases
)
```

**Arguments:**

- **case**
  An element parameter into AllCases, representing the case that you want to delete.

**Return value:**

The function returns 1 if the case is deleted successfully, or 0 otherwise.

**See also:**

The function **CaseFind**.
CaseFind

The function CaseFind searches the Data Management tree for a case with a specific name.

CaseFind(
    case-path, ! (input) scalar string expression
    case ! (output) element parameter into AllCases
)

Arguments:

* case-path
  A string expression holding the path and name of a case. The path is specified relative to the root of the case tree.

* case
  An element parameter into AllCases. On successful return this parameter will refer to the case found.

Return value:

The function returns 1 if the case is found, and 0 otherwise.

See also:

The functions CaseCreate, CaseDelete.
CaseGetChangedStatus

The function CaseGetChangedStatus returns whether the data of the currently active case has changed and thus needs to be saved.

Arguments:
None

Return value:
The function returns 1 if the data has changed, 0 otherwise.

See also:
The functions CaseSetChangedStatus, CaseSave.
Case Functions

CaseGetDatasetReference

With the function CaseGetDatasetReference you can, for every data category, obtain a reference to the dataset that is included in a specific case.

\[\text{CaseGetDatasetReference}(\text{case}, \text{data-category}, \text{dataset})\]

Arguments:

- **case**: An element in the set \(\text{AllCases}\), referring to the case for which you want to retrieve the dataset reference.
- **data-category**: An element in the set \(\text{AllDataCategories}\), referring to the specific data category for which you want to obtain the dataset reference.
- **dataset**: An element parameter into \(\text{AllDataSets}\), on return this argument will contain the included dataset. It is set to the empty element if no dataset is included or if the dataset no longer exists.

Return value:

If any of the first two arguments does not refer to a valid case or data category, or if the data category is not part of the case type, then the function returns \(-1\) and \(\text{CurrentErrorMessage}\) will contain a proper error message. If a dataset is included, and this dataset still exists, then the function returns \(1\) and the argument \(\text{dataset}\) will refer to that dataset. If there is no dataset included, then the function returns \(1\) and \(\text{dataset}\) is set to the empty element. If a dataset is include, but this dataset has been deleted, then the function returns \(0\) and \(\text{dataset}\) is set to the empty element.

Remarks:

You can use the functions CaseGetType and CaseTypeCategories to check whether a specific data category is part of a case.

See also:

The functions CaseGetType, CaseTypeCategories.
**Case Functions**

---

**CaseGetType**

The function `CaseGetType` retrieves the case type for a specific case.

```plaintext
CaseGetType(
    case,  ! (input) element of the set AllCases
    case-type  ! (output) element parameter into AllCaseTypes
)
```

**Arguments:**

- `case`
  - An element of the set `AllCases`, referring to the case for which you want to retrieve its case type.

- `case-type`
  - An element parameter into `AllCaseTypes`, on successful return this argument will contain the case type for the given case.

**Return value:**

The function returns 1 on success, 0 otherwise.
Case Functions

CaseLoadCurrent

The function CaseLoadCurrent loads an existing case as the new current case. You can use it to load either a case that is passed as argument to the function, or a case that the user can select via a dialog box. If the data of the currently loaded case has changed, then the user is asked to save this data first.

```plaintext
CaseLoadCurrent(
    case, ! (input, output) An element parameter into AllCases
    [dialog] ! (optional) 0 or 1
)
```

Arguments:

- `case`
  An element parameter into the pre-defined set AllCases. If the argument `dialog` is set to 0, then no dialog box is shown and the case to which the element parameter currently refers is loaded. If the argument `dialog` is set to 1, then the value of the element parameter is used to initialize the dialog box. The case that the user has selected and is loaded successfully is returned through this argument.

- `dialog (optional)`
  An integer value indicating whether or not the user gets a dialog box in which he can select the case to load. The default value is 1, thus if this argument is omitted the dialog box will be shown.

Return value:

The function returns 1 on success. If the user cancelled the operation, then the function returns 0. If any other error occurs then the function returns −1 and CurrentErrorMessage will contain a proper error message.

Remarks:

If you want to suppress the dialog box for the unsaved data, then you may call CaseSetChangedStatus(0) prior to CaseLoadCurrent.

See also:

The functions CaseLoadIntoActive, CaseMerge, CaseSave, CaseSetChangedStatus.
Case Functions

CaseLoadIntoCurrent

The function CaseLoadIntoCurrent loads the data of an existing case into the current case. You can use it to load either a case that is passed as argument to the function, or a case that the user can select via a dialog box. The data that is stored in the case will overwrite any data of the currently active case, and thus this current case is set to have changed data.

CaseLoadIntoCurrent(  
  case, ! (input, output) An element parameter into AllCases  
  [dialog] ! (optional) 0 or 1  
)

Arguments:

  case  
  An element parameter into the pre-defined set AllCases. If the argument dialog is set to 0, then no dialog is shown and the case to which the element parameter currently refers is loaded. If the argument dialog is set to 1, then the value of the element parameter is used to initialize the dialog box. The case that the user has selected and is loaded successfully is returned through this argument.

  dialog (optional)  
  An integer value indicating whether or not the user gets a dialog box in which he can select the case to load. The default value is 1, thus if this argument is omitted the dialog box will be shown.

Return value:

The function returns 1 on success. If the user cancelled the operation, then the function returns 0. If any other error occurs then the function returns −1 and CurrentErrorMessage will contain a proper error message.

See also:

The functions CaseLoadActive, CaseMerge, CaseSave, CaseSetChangedStatus.
Case Merge

The function CaseMerge merges the data of an existing case with the current data. You can use it to merge either a case that is passed as argument to the function, or a case that the user can select via a dialog box. Only the non-default data that is stored in the case will be merged with the data of the currently active case. This current case is set to have changed data.

\[
\text{CaseMerge}(\text{case, ! (input, output) An element parameter into AllCases, \\
\text{dialog] ! (optional) 0 or 1)}
\]

**Arguments:**

- **case**
  
  An element parameter into the pre-defined set AllCases. If the argument dialog is set to 0, then no dialog box is shown and the case to which the element parameter currently refers is merged. If the argument dialog is set to 1, then the value of the element parameter is used to initialize the dialog box. The case that the user has selected and is merged successfully is returned through this argument.

- **dialog (optional)**
  
  An integer value indicating whether or not the user gets a dialog box in which he can select the case to merge. The default value is 1, thus if this argument is omitted the dialog box will be shown.

**Return value:**

The function returns 1 on success. If the user cancelled the operation, then the function returns 0. If any other error occurs then the function returns \(-1\) and CurrentErrorMessage will contain a proper error message.

**See also:**

The functions CaseLoadActive, CaseLoadIntoActive, CaseSave, CaseGetChangedStatus.
CaseNew

The function CaseNew starts a new case. The function is similar to the command Case New from the Data menu. The function does not change any of the current data, it only assures that there is no longer a current case. If you did have a current case and the data of this case has been changed, then AIMMS will ask whether or not this case should be saved first.

Arguments:

None

Return value:

The function returns 1 on success. If the user cancelled the operation, then the function returns 0. If any other error occurs then the function returns −1 and CurrentErrorMessage will contain a proper error message.

Remarks:

If you use CaseNew, then the name of this new case is not specified until you save the case. If you want to start a new named case, then you can use the following piece of code:

```plaintext
if ( CaseGetChangedStatus ) then
  if ( CaseSave = 0 ) then
    return ;
  endif ;
endif ;
if ( CaseSelectNew( a_case ) ) then
  CaseSetCurrent( a_case );
  CaseSetChangedStatus( a_case, 1 );
endif
```

See also:

The functions CaseLoadCurrent, CaseSave, CaseSelectNew, CaseSetCurrent.
CaseSave

The function CaseSave saves the data to the current case. If there is no current case, then the function behaves exactly as the CaseSaveAs function. If the case has active references to datasets that contain changed data, then these datasets are saved as well.

```plaintext
CaseSave( [confirm] ! (optional) 0, 1 or 2 )
```

Arguments:

confirm (optional)

An integer to indicate whether or not the function should ask for confirmation before overwriting the existing case. If 0, then no confirmation dialog box is shown. If 1 (default), then whether the confirmation dialog box is shown depends on the case type properties. If 2, then always a confirmation dialog box is shown.

Return value:

The function returns 1 if the case is saved successfully. It returns 0 if the user canceled the save operation. If any other error occurs, then the function returns -1 and CurrentErrorMessage will contain an error message.

See also:

The functions CaseSaveAs, CaseSaveAll, CaseLoadCurrent, CaseGetChangedStatus.
Case Save All

With the function CaseSaveAll you can save (via a single call) the current case and all active datasets that need saving.

``` AIMMS 
CaseSaveAll( [confirm] ! (optional) integer value (0, 1 or 2) )
```

**Arguments:**

- **confirm (optional)**
  - If 0, then cases and datasets are saved without confirmation. If 2, then AIMMS will display a dialog box for the cases and datasets that are about to be saved and ask for confirmation. If 1 (default), then AIMMS will use the the properties of the case type and data categories to determine whether a confirmation dialog box should be displayed.

**Return value:**

The function returns 1 if the data is saved successfully. It returns 0 if the user canceled the save operation. If any other error occurs, then the function returns -1 and CurrentErrorMessage will contain an error message.

**See also:**

The functions **CaseSave, DatasetSave**.
CaseFunctions

CaseSaveAs

The function CaseSaveAs shows a dialog box in which the user can specify a (new) case to which the data is saved. If the case has active references to datasets that contain changed data, then these datasets are saved as well. When saving these datasets the function will simply overwrite the current datasets, thus with CaseSaveAs you can only change the current case and not any of the current datasets.

```plaintext
CaseSaveAs(
    case ! (output) element parameter in AllCases
)
```

Arguments:

- `case` An element parameter in AllCases. On return this parameter will refer to the case that the user selected.

Return value:

The function returns 1 if the case is saved successfully. It returns 0 if the user canceled the save operation. If any other error occurs, then the function returns −1 and CurrentErrorMessage will contain an error message.

See also:

The functions `CaseSave`, `CaseSaveAll`, `CaseLoadCurrent`, `CaseGetChangedStatus`. 
CaseSelect

The function CaseSelect shows a dialog box in which the user can select an existing case.

```plaintext
CaseSelect(
    case,  ! (output) element parameter in AllCases
    [title]  ! (optional) string expression
)
```

**Arguments:**

- **case**
  An element parameter in AllCases. On return the case will refer to the selected case.

- **title (optional)**
  A string expression that is used as the title for the dialog box. If this argument is omitted, then a default title is used.

**Return value:**

The function returns 1 if the user did select a case. If the user presses **Cancel**, then the function returns 0.

**See also:**

The function CaseSelectNew.
CaseSelectMultiple

The function CaseSelectMultiple shows a dialog box in which the user can select a number of cases (and datasets). The selected subset of cases and datasets is stored in the pre-defined set CurrentCaseSelection, which is used in the page objects for which the property Multiple Cases is set.

```
CaseSelectMultiple(       
    [cases-only] ! (optional) 0 or 1
)
```

Arguments:

- **cases-only (optional)**
  - This argument controls whether the user can only select cases or can select both datasets and cases. If this argument is omitted, then the default value is 0, which means that both cases and datasets can be selected.

Return value:

- The function returns 1 if the user pressed the **OK** button, and 0 if the user pressed **Cancel**.
Case Functions

CaseSelectNew

The function CaseSelectNew shows a dialog box in which the user can select a new case.

```vba
CaseSelect(
    case, ! (output) element parameter in AllCases
    [title] ! (optional) string expression
)
```

**Arguments:**

- **case**
  An element parameter in AllCases. On return the case will refer to the selected case.

- **title (optional)**
  A string expression that is used as the title for the dialog box. If this argument is omitted, then a default title is used.

**Return value:**

The function returns 1 if the user did select a case. If the user pressed Cancel, then the function returns 0.

**Remarks:**

If via this function the user creates a new case (i.e. a new case node in the data management tree), then this case does not yet contain any data. The case will only contain data after you explicitly save data to the case.

**See also:**

The functions CaseSelect, CaseSetCurrent, CaseSave.
Case Functions

**CaseSetChangedStatus**

The function *CaseSetChangedStatus* can set the status of the current case to either changed or unchanged.

```plaintext
CaseSetChangedStatus(
    status, ! (input) 0 or 1
    [include-datasets] ! (optional input) 0 or 1
)
```

**Arguments:**

- **status**
  - An integer value holding the new case status: 0 for unchanged, 1 for changed.

- **include-datasets (optional)**
  - An integer to indicate whether or not the status of the included and active datasets should be set as well. If you omit this argument, then the default value is 0 (status of datasets is not set).

**Return value:**

The function returns 1.

**See also:**

The functions *CaseGetChangedStatus*, *DatasetSetChangedStatus*.
Case Set Current

The function Case Set Current sets the case that is regarded as the current case. It does not load or save any data or checks whether data needs to be saved. You can, for example, use it to make a newly created case the current case, so that during a Case Save the data is written to this case.

\[
\text{Case Set Current}(\text{case})
\]

Arguments:

- **case**: An element of the set All Cases, referring to the case that should become the current case.

Return value:

The function returns 1 on success, or 0 otherwise.

See also:

The functions Case New, Case Create, Case Select New, Case Save.
AIMMS supports the following functions for accessing the datasets in the Data Manager:

- DatasetCreate
- DatasetDelete
- DatasetFind
- DatasetGetCategory
- DatasetGetChangedStatus
- DatasetLoadCurrent
- DatasetLoadIntoCurrent
- DatasetMerge
- DatasetNew
- DatasetSave
- DatasetSaveAs
- DatasetSelect
- DatasetSelectNew
- DatasetSetChangedStatus
- DatasetSetCurrent
## DatasetCreate

The function `DatasetCreate` creates a new dataset node in the Data Management tree. The data category, the name of the dataset and the folder in which it is created is given as an argument to the function.

```c
DatasetCreate(
    category, ! (input) element in AllDataCategories
    dataset-path, ! (input) scalar string expression
    dataset ! (output) element parameter into AllDatasets
)
```

### Arguments:

- **category**
  - An element in `AllDataCategories`, specifying the data category for which a dataset must be created.

- **dataset-path**
  - A string expression holding the path and name of the new dataset.
  - The path is specified relative to the corresponding data category root node in the Data Management tree.

- **dataset**
  - An element parameter into `AllDatasets`. On successful return this parameter will refer to the newly created element in `AllDatasets`.

### Return value:

The function returns 1 if the dataset is created successfully. It returns 0 if the dataset could not be created or if the dataset already exists.

### Remarks:

If the specified path contains folders that do not exist, then these folders are created automatically. To check whether a specific dataset path already exists you can use the function `DatasetFind`.

### See also:

The functions `DatasetFind`, `DatasetDelete`. 
DatasetDelete

The function `DatasetDelete` deletes a specific dataset node from the Data Management tree.

```
DatasetDelete(
    category, ! (input) element in AllDataCategories
    dataset   ! (input) element parameter into AllDatasets
)
```

**Arguments:**

- `category`  
  An element in `AllDataCategories`, specifying the data category for which a dataset is to be deleted.

- `dataset`  
  An element parameter into `AllDatasets`, representing the dataset that you want to delete.

**Return value:**

The function returns 1 if the dataset is deleted successfully, or 0 otherwise.

**See also:**

The function `DatasetFind`. 
**DatasetFind**

The function `DatasetFind` searches the Data Management tree for a dataset with a specific name and belonging to a specific data category.

```
DatasetFind(
    category, ! (input) element in AllDataCategories
    dataset-path, ! (input) scalar string expression
    dataset ! (output) element parameter into AllDatasets
)
```

**Arguments:**

- **category**
  
  An element in AllDataCategories, specifying the data category for which the datasets must be searched.

- **dataset-path**
  
  A string expression holding the path and name of a dataset. The path is specified relative to the corresponding data category root node in the Data Management tree.

- **dataset**
  
  An element parameter into AllDatasets. On successful return this parameter will refer to the dataset found.

**Return value:**

The function returns 1 if the dataset is found, and 0 otherwise.

**See also:**

The functions `DatasetCreate`, `DatasetDelete`.
DatasetGetCategory

The function DatasetGetCategory retrieves the data category of a specific dataset.

\[
\text{DatasetGetCategory}(\text{dataset}, \text{category})
\]

Arguments:
- \text{case} \quad \text{An element of the set AllDatasets, referring to the dataset for which you want to retrieve its data category.}

- \text{case-type} \quad \text{An element parameter into AllDataCategories, on successfull return this argument will contain the data category of the given dataset.}

Return value:
- The function returns 1 on success, 0 otherwise.
Dataset Functions

DatasetGetChangedStatus

The function DatasetGetChangedStatus returns whether the data associated with a specific data category has changed and thus needs to be saved.

```plaintext
DatasetGetChangedStatus(
    category ! (input) element in AllDataCategories
)
```

**Arguments:**

- `category` An element in AllDataCategories, specifying the data category for which the changed status must be retrieved.

**Return value:**

The function returns 1 if the data has changed, 0 otherwise.

**See also:**

The functions `DatasetSetChangedStatus`, `DatasetSave`.
**DatasetLoadCurrent**

The function **DatasetLoadCurrent** loads an existing dataset as the new current dataset for a specific data category. You can use it to load either a dataset that is passed as argument to the function, or a dataset that the user can select via a dialog box. If the data of the corresponding data category has changed, then the user is asked to save this data first.

```plaintext
DatasetLoadCurrent(
    category,  ! (input) element in AllDataCategories
    dataset,   ! (input, output) an element parameter into AllDatasets
    [dialog]  ! (optional) 0 or 1
)
```

**Arguments:**

*category*

An element in AllDataCategories, specifying the data category for which a dataset is loaded.

*dataset*

An element parameter in the set AllDatasets. If the argument *dialog* is set to 0, then no dialog box is shown and the dataset to which the element parameter currently refers is loaded. If the argument *dialog* is set to 1, then the value of the element parameter is used to initialize the dialog box. The dataset that the user has selected and is loaded successfully is returned through this argument.

*dialog (optional)*

An integer value indicating whether or not the user gets a dialog box in which he can select the dataset to load. The default value is 1, thus if this argument is omitted the dialog box will be shown.

**Return value:**

The function returns 1 on success. If the user canceled the operation, then the function returns 0. If any other error occurs then the function returns −1 and CurrentErrorMessage will contain a proper error message.

**Remarks:**

If you want to suppress the dialog box for the unsaved data, then you may call DatasetSetChangedStatus(category,0) prior to DatasetLoadCurrent.

**See also:**

The functions **DatasetLoadIntoActive**, **DatasetMerge**, **DatasetSave**, **DatasetSetChangedStatus**.
**DatasetLoadIntoCurrent**

The function DatasetLoadIntoCurrent loads the data of an existing dataset as the new current dataset for a specific data category. You can use it to load either a dataset that is passed as argument to the function, or a dataset that the user can select via a dialog box. The data that is stored in the dataset will overwrite any data of the currently active dataset, and thus this current dataset is set to have changed data.

```plaintext
DatasetLoadIntoCurrent(
    category,  ! (input) element in AllDataCategories
    dataset,  ! (input, output) an element parameter into AllDatasets
    [dialog]  ! (optional) 0 or 1
)
```

**Arguments:**

- `category`
  - An element in AllDataCategories, specifying the data category for which a dataset is loaded.

- `dataset`
  - An element parameter in the set AllDatasets. If the argument `dialog` is set to 0, then no dialog box is shown and the dataset to which the element parameter currently refers is loaded. If the argument `dialog` is set to 1, then the value of the element parameter is used to initialize the dialog box. The dataset that the user has selected and is loaded successfully is returned through this argument.

- `dialog (optional)`
  - An integer value indicating whether or not the user gets a dialog box in which he can select the dataset to load. The default value is 1, thus if this argument is omitted the dialog box will be shown.

**Return value:**

The function returns 1 on success. If the user canceled the operation, then the function returns 0. If any other error occurs then the function returns −1 and CurrentErrorMessage will contain a proper error message.

**See also:**

The functions DatasetLoadActive, DatasetMerge, DatasetSave, DatasetSetChangedStatus.
Dataset Functions

DatasetMerge

The function DatasetMerge merges the data of an existing dataset with the current data. You can use it to merge either a dataset that is passed as argument to the function, or a dataset that the user can select via a dialog box. Only the non-default data that is stored in the dataset will be merged with the current data.

DatasetMerge(  
category,         ! (input) element in AllDataCategories  
dataset,         ! (input, output) an element parameter into AllDatasets  
[dialog]         ! (optional) 0 or 1  
)

Arguments:

*category*

*An element in AllDataCategories, specifying the data category for which a dataset is loaded.*

*dataset*

*An element parameter in the set AllDatasets. If the argument dialog is set to 0, then no dialog box is shown and the dataset to which the element parameter currently refers is loaded. If the argument dialog is set to 1, then the value of the element parameter is used to initialize the dialog box. The dataset that the user has selected and is loaded successfully is returned through this argument.*

*dialog (optional)*

*An integer value indicating whether or not the user gets a dialog box in which he can select the dataset to load. The default value is 1, thus if this argument is omitted the dialog box will be shown.*

Return value:

The function returns 1 on success. If the user cancelled the operation, then the function returns 0. If any other error occurs then the function returns −1 and CurrentErrorMessage will contain a proper error message.

See also:

The functions DatasetLoadActive, DatasetLoadIntoActive, DatasetSave, DatasetGetChangedStatus.
DatasetNew

The function DatasetNew starts a new unnamed dataset for a specific data category. The function is similar to the command Dataset New from the Data menu. The function does not change any of the current data, it only sets the current dataset to unnamed. If you did have a currently named dataset and the data of this dataset has been changed, then AIMMS will ask whether or not this dataset should be saved first.

DatasetNew(
    category ! (input) an element of AllDataCategories
)

Arguments:

category

An element in AllDataCategories, specifying the data category for which you want to start a new unnamed dataset.

Return value:

The function returns 1 on success. If the user cancelled the operation, then the function returns 0. If any other error occurs then the function returns −1 and CurrentErrorMessage will contain a proper error message.

Remarks:

If you use CaseNew, then the name of this new case is not specified until you save the case. If you want to start a new named case, then you can use the following piece of code:

if ( CaseGetChangedStatus ) then
    if ( CaseSave = 0 ) then
        return ;
    endif ;
endif ;
if ( CaseSelectNew( a_case ) ) then
    CaseSetCurrent( a_case ) ;
    CaseSetChangedStatus( a_case, 1 ) ;
endif

See also:

The functions DatasetLoadCurrent, DatasetSave, DatasetSelectNew, DatasetSetCurrent.
**DatasetSave**

The function DatasetSave saves the data of a data category to the active dataset. If there is no named active dataset, then the function behaves exactly as the DatasetSaveAs function.

```plaintext
DatasetSave(
    category, ! (input) element in AllDataCategories
    [confirm]  ! (optional) 0, 1 or 2
)
```

**Arguments:**

- **category**
  An element in AllDataCategories, specifying the data category for which you want to save the data.

- **confirm (optional)**
  An integer to indicate whether or not the function should ask for confirmation before overwriting the existing dataset. If 0, then no confirmation dialog box is shown. If 1 (default), then whether or not the confirmation dialog box is shown depends on the case type properties. If 2, then always a confirmation dialog box is shown.

**Return value:**

The function returns 1 if the dataset is saved successfully. It returns 0 if the user canceled the save operation. If any other error occurs, then the function returns −1 and CurrentErrorMessage will contain an error message.

**See also:**

The functions DatasetSaveAs, DatasetLoadCurrent, DatasetGetChangedStatus.
DatasetSaveAs

The function DatasetSaveAs shows a dialog box in which the user can specify a (new) dataset to which the data is saved.

DatasetSaveAs(
  category,  ! (input) element in AllDataCategories
  dataset  ! (output) element parameter in AllDatasets
)

Arguments:

category
  An element in AllDataCategories, specifying the data category for which you want to save the data.

dataset
  An element parameter in AllDatasets. On return this parameter will refer to the dataset that the user selected.

Return value:

The function returns 1 if the dataset is saved successfully. It returns 0 if the user cancelled the save operation. If any other error occurs, then the function returns -1 and CurrentErrorMessage will contain an error message.

See also:

The functions DatasetSave, DatasetLoadCurrent, DatasetGetChangedStatus.
Dataset Select

The function DatasetSelect shows a dialog box in which the user can select an existing dataset for a given data category.

```plaintext
DatasetSelect(
    category, ! (input) element in AllDataCategories
    dataset, ! (output) element parameter in AllDatasets
    [title] ! (optional) string expression
)
```

**Arguments:**

- `category`
  - An element in AllDataCategories, specifying the data category for which you want the user to select a dataset.

- `dataset`
  - An element parameter in AllDatasets. On return, the dataset will refer to the selected dataset.

- `title (optional)`
  - A string expression that is used as the title for the dialog box. If this argument is omitted, then a default title is used.

**Return value:**

The function returns 1 if the user did select a dataset. If the user pressed Cancel, then the function returns 0.

**See also:**

The function DatasetSelectNew.
DatasetSelectNew

The function DatasetSelectNew shows a dialog box in which the user can select a new dataset for a given data category.

DatasetSelectNew(
    category,  ! (input) element in AllDataCategories
    dataset,  ! (output) element parameter in AllDatasets
    [title]    ! (optional) string expression
)

Arguments:

category
   An element in AllDataCategories, specifying the data category for which you want to the user to select a new dataset.

dataset
   An element parameter in AllDatasets. On return the dataset will refer to the selected dataset.

title (optional)
   A string expression that is used as the title for the dialog box. If this argument is omitted, then a default title is used.

Return value:

The function returns 1 if the user did select a dataset. If the user pressed Cancel, then the function returns 0.

Remarks:

If via this function the user creates a new dataset (i.e. a new dataset node in the data management tree), then this case dataset does not yet contain any data. The dataset will only contain data after you explicitly save data to it.

See also:

The functions DatasetSelect, DatasetSetCurrent, DatasetSave.
**DatasetFunctions**

**DatasetSetChangedStatus**

The function `DatasetSetChangedStatus` can set the status of a data category to either changed or unchanged.

```plaintext
DatasetSetChangedStatus(
    category, ! (input) element in AllDataCategories
    status ! (input) 0 or 1
)
```

**Arguments:**

- `category`  
  An element in `AllDataCategories`, specifying the data category for which you want to set the changed status.

- `status`  
  An integer value holding the new dataset status: 0 for unchanged, 1 for changed.

**Return value:**

The function returns 1.

**See also:**

The function `DatasetGetChangedStatus`. 
Dataset Functions

**DatasetSetCurrent**

The function `DatasetSetCurrent` sets the dataset that is regarded as the current dataset for a given data category. It does not load or save any data or checks whether data needs to be saved. You can, for example, use it to make a newly created dataset the current dataset, so that during a `DatasetSave` the data is written to this dataset.

```plaintext
DatasetSetCurrent(category, ! (input) element in AllDataCategories
dataset ! (input) element of the set AllDatasets)
```

**Arguments:**

- **category**
  - An element in `AllDataCategories`, specifying the data category for which you want to set the current dataset.

- **data**
  - An element of the set `AllDatasets`, referring to the dataset that should become the current dataset.

**Return value:**

The function returns 1 on success, or 0 otherwise.

**See also:**

The functions `DatasetNew`, `DatasetCreate`, `DatasetSelectNew`, `DatasetSave`. 
AIMMS supports the following Data Manager functions, that are not specific for cases or datasets only:

- CaseTypeCategories
- CaseTypeContents
- DataCategoryContents
- DataFileExists
- DataFileGetAcronym
- DataFileGetDescription
- DataFileGetGroup
- DataFileGetName
- DataFileGetOwner
- DataFileGetPath
- DataFileGetTime
- DataFileReadPermitted
- DataFileWritePermitted
- DataImport220
- DataManagerExport
- DataManagerImport
CaseTypeCategories

The function CaseTypeCategories retrieves the sub-collection of data categories that is included in a specific case type.

```haskell
CaseTypeCategories(
    case-type, ! (input) element of the set AllCaseTypes
    category-set ! (output) subset of AllDataCategories
)
```

**Arguments:**

- **case-type**
  
  An element of the set AllCaseTypes, referring to the case type for which you want to retrieve the included data categories.

- **category-set**
  
  A subset of the set AllDataCategories, on successful return this subset is filled with the data categories included in the case type.

**Return value:**

The function returns 1 on success, 0 otherwise.

**See also:**

The functions `CaseGetType`, `CaseTypeContents`, `DataCategoriesContents`. 
CaseTypeContents

The function CaseTypeContents retrieves the sub-collection of identifiers that is contained in a specific case type.

```plaintext
CaseTypeContents(
    case-type, ! (input) element of the set AllCaseTypes
    identifier-set ! (output) subset of AllIdentifiers
)
```

**Arguments:**

- **case-type**
  An element of the set AllCaseTypes, referring to the case type for which you want to retrieve the contents.

- **identifier-set**
  A subset of the set AllIdentifiers, on successful return this subset is filled with all identifiers contained in the case type.

**Return value:**

The function returns 1 on success, 0 otherwise.

**Remarks:**

The function returns the contents of the case type itself, as well as the contents of all data categories that are included in the case type.

**See also:**

The functions CaseGetType, CaseTypeCategories, DataCategoriesContents.
DataCategoryContents

The function DataCategoryContents retrieves the sub-collection of identifiers that is contained in a specific data category.

\[
\text{DataCategoryContents}(\text{data-category, identifier-set})
\]

Arguments:

- **data-category**
  An element of the set AllDataCategories, referring to the data category for which you want to retrieve the contents.

- **identifier-set**
  A subset of the set AllIdentifiers, on successful return this subset is filled with all identifiers contained in the data category.

Return value:

The function returns 1 on success, 0 otherwise.

See also:

The functions CaseTypeCategories, CaseTypeContents.
DataFileExists

With the function DataFileExists you can check whether a specific element from the set AllDataFiles still refers to a valid case or dataset. Especially when multiple users have access to the same data file, an element may become invalid.

```
DataFileExists(
    datafile ! (input) element in the set AllDataFiles
)
```

**Arguments:**
- `datafile` An element in the set AllDataFiles, AllCases or AllDatasets.

**Return value:**
- The function returns 1 if the given datafile still exists, and 0 otherwise.

**Remarks:**
- Note that AllCases and AllDatasets are subsets of AllDataFiles.

**See also:**
- The function DataFileGetName.
**DataFileGetAcronym**

The predefined set `AllDataFiles` (and its subsets `AllCases` and `AllDatasets`), is an integer set. The mapping of these integers onto the cases and datasets in the project is maintained by the data manager, and is not editable. With the function `DataFileGetAcronym` you can obtain the acronym that is specified in the data manager for any element of the set `AllDataFiles` (cases or datasets).

```plaintext
DataFileGetAcronym(
    datafile,  ! (input) element in the set AllDataFiles
    acronym   ! (output) scalar string parameter
)
```

**Arguments:**

- `datafile`  
  An element in the set `AllDataFiles`.

- `acronym`  
  A scalar string valued parameter. On return this parameter will contain the acronym of the datafile. If no acronym is specified, then an empty string is returned.

**Return value:**

The function returns 1 if the given datafile still exists, and 0 otherwise.

**See also:**

The functions `DataFileExists`, `DataFileGetName`.
DataFileGetDescription

The predefined set AllDataFiles (and its subsets AllCases and AllDatasets), is an integer set. The mapping of these integers onto the cases and datasets in the project is maintained by the data manager, and is not editable. With the function DataFileGetDescription you can obtain the description that the user entered via the properties of a case or dataset.

DataFileGetDescription(
    datafile, ! (input) element in the set AllDataFiles
    description  ! (output) scalar string parameter
)

Arguments:

datafile
    An element in the set AllDataFiles.

name
    A scalar string valued parameter. On return this parameter will contain the description of the datafile. If no description has been specified, then this string is empty.

Return value:

The function returns 1 if the given datafile still exists, and 0 otherwise.

See also:

The functions DataFileExists, DataFileGetName, DataFileGetAcronym.
**DataFileGetGroup**

With the function `DataFileGetGroup` you can obtain the group name associated with the user that currently owns a specific case or dataset.

```
DataFileGetGroup(
    datafile, ! (input) element in the set AllDataFiles
    owner     ! (output) scalar string parameter
)
```

**Arguments:**

- `datafile`
  - An element in the set `AllDataFiles`.

- `owner`
  - A scalar string valued parameter. On return this parameter will contain the group name associated with the user that owns the datafile. If there is no current owner, or if the project does not have a user database associated with it, then an empty string is returned.

**Return value:**

The function returns 1 if the given datafile exists, and 0 otherwise.

**See also:**

The functions `DataFileExists`, `DataFileGetOwner`. 
Data File Get Name

The predefined set All Data Files (and its subsets All Cases and All Datasets), is an integer set. The mapping of these integers onto the cases and datasets in the project is maintained by the data manager, and is not editable. With the function Data File Get Name you can obtain the name in the data manager for any element of the set All Data Files (cases or datasets).

Data File Get Name (
  datafile, ! (input) element in the set All Data Files
  name ! (output) scalar string parameter
)

Arguments:

datafile
  An element in the set All Data Files.

name
  A scalar string valued parameter. On return this parameter will contain the name of the datafile. This name does not include the name of the folder(s) in which it is located.

Return value:

The function returns 1 if the given datafile still exists, and 0 otherwise.

See also:

The functions Data File Exists, Data File Get Path, Data File Get Acronym.
DataFileGetOwner

With the function DataFileGetOwner you can obtain the name of the user that currently owns a specific case or dataset.

\[
\text{DataFileGetOwner}(\text{datafile}, \text{owner})
\]

Arguments:

- **datafile**
  An element in the set AllDataFiles.

- **owner**
  A scalar string valued parameter. On return this parameter will contain the name of the user that owns the datafile. If there is no current owner, or if the project does not have a user database associated with it, then an empty string is returned.

Return value:

The function returns 1 if the given datafile exists, and 0 otherwise.

See also:

The functions DataFileExists, DataFileGetGroup.
DataManager Functions

DataFileGetPath

The predefined set AllDataFiles (and its subsets AllCases and AllDatasets), is an integer set. The mapping of these integers onto the cases and datasets in the project is maintained by the data manager, and is not editable. With the function DataFileGetPath you can obtain the path in the data manager for any element of the set AllDataFiles (cases or datasets). The path of a datafile consists of a sequence folder names and the name of the datafile itself, separated by backslash characters.

DataFileGetPath(  
    datafile, ! (input) element in the set AllDataFiles  
    path ! (output) scalar string parameter  
)

Arguments:

    datafile
    - An element in the set AllDataFiles.

    name
    - A scalar string valued parameter. On return this parameter will contain the path of the datafile.

Return value:

The function returns 1 if the given datafile still exists, and 0 otherwise.

See also:

The functions DataFileExists, DataFileGetName, DataFileGetAcronym.
DataFileGetTime

With the function DataFileGetTime you can obtain the time on which the data of a specific case or dataset was last modified (saved).

\[
\text{DataFileGetTime}(
    \text{datafile}, \quad ! \text{(input) element in the set AllDataFiles}
    \text{file-time} \quad ! \text{(output) scalar string parameter}
)
\]

Arguments:

- **datafile**
  
  An element in the set AllDataFiles.

- **file-time**
  
  A scalar string valued parameter. On return this parameter will contain a string representation of the modification time, using AIMMS’ standard date and time format: "YYYY-MM-DD hh:mm:ss".

Return value:

The function returns 1 if the given datafile exists and contains saved data. If the datafile does not exist, or if no data has yet been saved in the datafile, then the function returns 0.

See also:

- The functions DataFileExists, FileTime.
**DataFileReadPermitted**

With the function `DataFileReadPermitted` you can check whether the current user has read permission for the specified case or dataset. For example, you can use this function to issue your own error message if the permission is not granted. If the current user does not have read permission, then any call to a data manager function that involves a read operation will result in an error message, and fails.

```plaintext
DataFileReadPermitted(
    datafile ! (input) element in the set AllDataFiles
)
```

**Arguments:**

`datafile`

An element in the set `AllDataFiles`.

**Return value:**

The function returns 1 if the current user does have read permission, and 0 otherwise.

**See also:**

The function `DataFileWritePermitted`.
DataFileWritePermitted

With the function `DataFileWritePermitted` you can check whether the current user has write permission for the specified case or dataset. For example, you can use this function to issue your own error message if the permission is not granted. If the current user does not have write permission, then any call to a data manager function that involves a write (save) operation will result in an error message, and fails.

```plaintext
DataFileWritePermitted(
    datafile ! (input) element in the set AllDataFiles
)
```

**Arguments:**

- `datafile`
  - An element in the set `AllDataFiles`.

**Return value:**

The function returns 1 if the current user does have write permission, and 0 otherwise.

**See also:**

The function `DataFileReadPermitted`. 
Data Manager Functions

DataImport220

With the function DataImport220 you can load a separate AIMMS case file, such as the case files that were created with AIMMS 2.20. After importing a case file using this function you can save the data as a new case node in the Data Management tree.

\[
\text{DataImport220}(\text{filename } ! \text{ (output) a string parameter})
\]

Arguments:

\textit{filename}

A string parameter, that on return will contain the name of the file that the user selected for importing.

Return value:

The function returns 1 on success. If the user canceled the operation, then the function returns 0. If any other error occurs then the function returns −1 and CurrentErrorMessage will contain a proper error message.

Remarks:

This function is especially useful for converting old cases to the new AIMMS.

See also:

The function \texttt{CaseSaveAs}. 
Data Manager Functions

**DataManagerExport**

With the function `DataManagerExport` you can export a collection of cases and datasets from the data management tree to a new data file.

```plaintext
DataManagerExport(
    filename, ! (input) a scalar string expression
    datafiles ! (output) a subset of AllDataFiles
)
```

**Arguments:**

- **filename**  
  A string containing the name of the data file to which the cases and datasets must be exported.

- **datafiles**  
  A subset of `AllDataFiles`, containing the cases and datasets that you want to export. Any dataset that is referred to by a case in this set is automatically added to the set.

**Return value:**

The function returns 1 on success, or 0 otherwise.

**See also:**

The function `DataManagerImport`.
 DataManagerImport

With the function `DataManagerImport` you can import the entire data management tree that is stored in another data file into your current data management tree. If the imported tree contains cases (or datasets) that already exist in the current tree, then you can choose whether these cases (or datasets) should overwrite the current nodes or should be imported as new nodes.

```fortran
DataManagerImport(
    filename, ! (input) a scalar string expression
    [overwrite] ! (optional) 0, 1 or 2
)
```

**Arguments:**

- `filename`
  A string containing the name of the data file that must be imported.

- `overwrite` (optional)
  This integer indicates whether or not existing cases (or datasets) are overwritten by cases (or datasets) from the imported file. If 0 (the default), then a dialog box is displayed in which the user can decide to overwrite the existing node or to create a new node. If 1, then existing nodes are always overwritten. If 2, then all imported cases and datasets will result in new nodes in the tree.

**Return value:**

The function returns 1 on success. If the user canceled the operation, then the function returns 0. If any other error occurs then the function returns -1 and `CurrentErrorMessage` will contain a proper error message.

**See also:**

The function `DataManagerExport`. 
Dialog Functions

AIMMS supports the following functions for simple interaction with the end user:

- DialogAsk
- DialogError
- DialogGetDate
- DialogGetElement
- DialogGetElementByData
- DialogGetElementByText
- DialogGetNumber
- DialogGetPassword
- DialogGetString
- DialogMessage
- DialogProgress
- StatusMessage
Dialog Functions

DialogAsk

The function DialogAsk displays a small dialog box containing a message and two or three buttons. Usually these buttons are an OK and Cancel, or Yes, No and Cancel, but they can contain any text you want. The function returns the number of the button that is pressed by the user.

```plaintext
DialogAsk(
    message, ! (input) string expression
    button1, ! (input) string expression
    button2, ! (input) string expression
    [button3] ! (optional) string expression
)
```

Arguments:

message
A scalar string expression containing the text you want to display in the dialog box.

button1
A scalar string expression containing the text of the first button.

button2
A scalar string expression containing the text of the second button.

button3 (optional)
A scalar string expression containing the text of the third button. If this argument is omitted then the dialog box will only show two buttons.

Return value:

The function returns the number of the button that is pressed: 1 for the first button, 2 for the second button or 3 for the third button.

Remarks:

If the user presses the Esc key, or closes the dialog box via the [x] in the top right corner, then this is interpreted as pressing the last button in the dialog box (which is usually the Cancel button).

See also:

The functions DialogMessage, DialogError.
**DialogError**

The function `DialogError` displays a small dialog box containing a specified error message and an OK button. The execution will be halted until the user presses the OK button.

```plaintext
DialogError(
    message       ! (input) string expression
)
```

**Arguments:**

- `message` A scalar string expression containing the text you want to display in the dialog box.

**Remarks:**

The procedures `DialogMessage` and `DialogError` only differ in the icon that is displayed at the left side of the dialog box.

**See also:**

The functions `DialogMessage`, `DialogAsk`, `DialogProgress`. 
Dialog Functions

DialogGetDate

The function DialogGetDate displays a standard Windows date selection dialog box. The function returns the date (in the specified format) selected by the user.

```
DialogGetDate(
    message, ! (input) string expression
    date-format, ! (input) string expression
    date, ! (inout) scalar string parameter
    [nr-rows,] ! (optional) integer expression
    [nr-columns] ! (optional) integer expression
)
```

Arguments:

- **message**
  A scalar string expression containing the text you want to display in the dialog box.

- **date-format**
  A scalar string expression containing the date format of the date argument.

- **date**
  A scalar string parameter in which the selected date is returned according to the date format specified in date-format.

- **nr-rows (optional)**
  A scalar integer expression in the range 1,...,3 containing the number of rows to be displayed in the date selection dialog box.

- **nr-columns (optional)**
  A scalar integer expression in the range 1,...,4 containing the number of rows to be displayed in the date selection dialog box.

Return value:

The function returns 1 if the user completed the date selection dialog box successfully, or 0 otherwise.

Remarks:

If the date argument contains a valid date according to the format specified in date-format, AIMMS will set the initial date in the date selection dialog box equal to the specified date.

See also:

The date format specification components are discussed in full detail in Section 24.7.1 of the Language Reference.
Dialog Functions

DialogGetElementByData

The function DialogGetElementByData is an extension of the function DialogGetElementByText. Instead of only showing a list box with only a single string per element, this function allows you to show a list box with multiple columns of text per element. The text that is displayed in each column is specified via a 2-dimensional string parameter. The first dimension of this parameter corresponds to the rows of the list box, the second dimension corresponds to the column in the list box.

```plaintext
DialogGetElementByData(
    message, ! (input) string expression
    reference, ! (input/output) scalar element parameter
    element-text ! (input) 2-dimensional string parameter
)
```

Arguments:

- **message**
  A scalar string expression containing the text you want to display as title of the dialog box.

- **reference**
  A scalar element parameter. When creating the dialog box, the range set of this parameter is used to fill the list with elements, and the current value of the element parameter will be initially selected. On return, this parameter will refer to the selected element.

- **element-text**
  A 2-dimensional string parameter. The first index in its domain should matches the range set of the element parameter `reference`, the second index defines the number of columns that are shown. Instead of the element names, the dialog box will display multiple columns of text derived from this parameter.

Return value:

The function returns 1 if the user has pressed the **OK** button, and 0 if he has pressed the **Cancel** button.

See also:

The functions `DialogGetElement`, `DialogGetElementByText`. 
**DialogGetElement**

The function DialogGetElement displays a dialog box in which the user can select an element from a list of set elements.

```plaintext
DialogGetElement(  
    message, ! (input) string expression  
    reference ! (input/output) scalar element parameter  
)
```

**Arguments:**

- **message**
  A scalar string expression containing the text you want to display as title of the dialog box.

- **reference**
  A scalar element parameter. When creating the dialog box, the range set of this parameter is used to fill the list with elements, and the current value of the element parameter will be initially selected. On return, this parameter will refer to the selected element.

**Return value:**

The function returns 1 if the user has pressed the **OK** button, and 0 if he has pressed the **Cancel** button.

**See also:**

The functions DialogGetElementByText, DialogGetElementByData, DialogGetNumber.
Dialog Functions

DialogGetElementByText

The function DialogGetElementByText displays a dialog box in which the user can select an element from a set. However, other than DialogGetElement, this function does not show a list of element names but a list of strings, which are given as a separate argument to the function.

DialogGetElementText(
    message,  ! (input) string expression
    reference,  ! (input/output) scalar element parameter
    element-text  ! (input) 1-dimensional string parameter
)

Arguments:

message
A scalar string expression containing the text you want to display as title of the dialog box.

reference
A scalar element parameter. When creating the dialog box, the range set of this parameter is used to fill the list with elements, and the current value of the element parameter will be initially selected. On return, this parameter will refer to the selected element.

element-text
A 1-dimensional string parameter, with a domain that matches the range set of the element parameter reference. Instead of the element names, the dialog box will display the corresponding strings of this parameter.

Return value:

The function returns 1 if the user has pressed the OK button, and 0 if he has pressed the Cancel button.

See also:

The functions DialogGetElement, DialogGetElementByData.
DialogFunctions

The function DialogGetNumber displays a small dialog box in which the user can enter a single numerical value. The dialog box remains on the screen (and thus halts the execution) until the user presses either the OK or the Cancel button.

```dialog
dialogonenumber(message, ! (input) string expression
reference, ! (input/output) scalar numerical identifier
[decimals] ! (optional) integer)
```

**Arguments:**

- `message` A scalar string expression containing the text you want to display in front of the edit field.
- `reference` A scalar identifier. When creating the dialog box, its value is used to fill the edit field. After the user presses the OK button, the edited value is returned through this argument.
- `decimals` A integer expression to indicate the number of decimals that is displayed initially.

**Return value:**

The function returns 1 if the user has pressed the OK button, and 0 if he has pressed the Cancel button.

**See also:**

The functions DialogGetString, DialogGetElement.
Dialog Functions

DialogGetPassword

The function DialogGetPassword displays a small dialog box in which the user can enter a password string. In the dialog box the string is presented by a sequence of asterisks. The dialog box remains on the screen (and thus halts the execution) until the user presses either the OK or the Cancel button.

```
DialogGetPassword(
    message, ! (input) string expression
    reference ! (input/output) scalar string parameter
)
```

**Arguments:**

- **message**
  A scalar string expression containing the text you want to display in front of the edit field.

- **reference**
  A scalar string valued identifier containing the password. When creating the dialog box, its value is used to fill the edit field. After the user presses the OK button, the edited password string is returned through this argument.

**Return value:**

The function returns 1 if the user has pressed the OK button, and 0 if he has pressed the Cancel button.

**See also:**

The function DialogGetString.
Dialog Functions

DialogGetString

The function DialogGetString displays a small dialog in which the user can enter a text string. The dialog remains on the screen (and thus halts the execution) until the user presses either the OK or the Cancel button.

```
DialogGetString(
    message,    ! (input) string expression
    reference   ! (input/output) scalar string parameter
)
```

Arguments:

message
A scalar string expression containing the text you want to display in front of the edit field.

reference
A scalar string valued identifier. When creating the dialog, its value is used to fill the edit field. After the user presses the OK button, the edited string is returned through this argument.

Return value:

The function returns 1 if the user has pressed the OK button, and 0 if he has pressed the Cancel button.

See also:

The functions DialogGetNumber, DialogGetPassword, DialogGetElement.
**DialogMessage**

The function `DialogMessage` displays a small dialog box containing a specified informational message and an **OK** button. The execution will be halted until the user presses the **OK** button.

```plaintext
DialogMessage(
    message ! (input) string expression
)
```

**Arguments:**

- `message`
  A scalar string expression containing the text you want to display in the dialog box.

**Remarks:**

The procedures `DialogMessage` and `DialogError` only differ in the icon that is displayed at the left side of the dialog box.

**See also:**

The functions `DialogError`, `DialogAsk`. 
Dialog Progress

The function Dialog Progress displays a small dialog box containing a specified error message and a progress bar that can indicate how much of a specific task has already been processed. This dialog box will not halt the execution, and you can call the function sequentially during a timely task to change either the displayed message or the length of the progress bar.

```
DialogProgress(
    message, ! (input) string expression
    [percentage] ! (optional) integer expression
)
```

**Arguments:**

- **message**
  A scalar string expression containing the text you want to display in the dialog box.

- **percentage (optional)**
  A scalar value between 0 and 100. It is used to set the length of the progress bar at the bottom of the dialog box. If this argument is omitted then the progress bar is not displayed.

**Remarks:**

The progress dialog box does not adjust the length of the progress bar itself, so you must do it yourself by sequentially calling the function with an increasing percentage. The progress dialog box is automatically removed from the screen if the execution terminates. If you want to remove the dialog box yourself, then you should call Dialog Progress with an empty message string: Dialog Progress("").

**See also:**

The functions Dialog Message, Dialog Error, Dialog Ask.
**StatusMessage**

With the function StatusMessage you can display a short message in the status bar at the bottom of the AIMMS window.

```plaintext
StatusMessage(
    message ! (input) string expression
)
```

**Arguments:**

- `message`  
  A scalar string expression containing the text you want to display in the status bar.

**Remarks:**

If you have set the status bar to be hidden (via the project options), then the message will not be visible to the user.

**See also:**

The functions [DialogMessage](#), [DialogProgress](#).
AIMMS supports the following functions for opening, closing, and manipulating the pages in the interface:

- PageClose
- PageGetActive
- PageGetChild
- PageGetFocus
- PageGetNext
- PageGetParent
- PageGetPrevious
- PageOpen
- PageOpenSingle
- PageRefreshAll
- PageSetCursor
- PageSetFocus
- PrintEndReport
- PrintPage
- PrintPageCount
- PrintStartReport
Page Functions

PageClose

With the function PageClose you can close a page that is currently open.

\[
\text{PageClose}(\text{page} \quad ! \text{(optional,input) string expression})
\]

Arguments:

\textit{page (optional)}

A string expression representing the name of the page that you want to close. This name is the unique name as it appears in the Page Manager tree. If you omit this argument, then PageClose closes the currently active page.

Return value:

The function returns 1 if the page is closed successfully, or a 0 otherwise.

See also:

The functions PageOpen, PageOpenSingle.
PageGetActive

With the function PageGetActive you can retrieve the name of the currently active page.

```c
PageGetActive(  page ! (output) scalar string identifier  )
```

**Arguments:**

- `page`
  - A string identifier to hold the name of the page that is currently active.

**Return value:**

The function returns 1 on success, or 0 if there is no currently active page.

**See also:**

The function PageGetFocus.
The function `PageGetChild` retrieves the name of the first child page for a specific page in the Page Manager tree.

```dylan
PageGetChild(
    page, ! (input) scalar string expression
    child-page ! (output) scalar string identifier
)
```

**Arguments:**
- `page`  
  A string expression containing the name of a (parent) page in the Page Manager tree.
- `child-page`  
  A scalar string identifier to hold the name of the first child page beneath the given parent page (if any).

**Return value:**

The function returns 1 on success, or 0 if the given page name does not exist or if the page does not have any child pages.

**See also:**

The functions `PageGetParent`, `PageNext`, `PagePrevious`. 
PageGetFocus

With the function PageGetFocus you can retrieve the name of the currently active page.

\[
\text{PageGetFocus}(\ \ \text{page,} \quad \text{tag})
\]

Arguments:

- **page**
  A string identifier to hold the name of the currently active page.

- **tag**
  A string identifier to hold the tag name of the object that currently has the keyboard input focus.

Return value:

The function returns 1 on success, or 0 if there is no currently active page or if no object has the input focus.

Remarks:

You can specify a unique tag name for each page object via the object properties. If no tag name has been given explicitly, then the type of object is returned ("Table", "Bar Chart", etc.)

See also:

The functions PageSetFocus, PageGetActive.
PageGetNext

The function `PageGetNext` retrieves the name of the next page for a specific page in the Page Manager tree. The next page is the page that has the same parent page, and is positioned directly below the given page.

```
PageGetNext(
    page,   ! (input) scalar string expression
    next-page ! (output) scalar string identifier
)
```

**Arguments:**

- `page`
  A string expression containing the name of a (child) page in the Page Manager tree.

- `next-page`
  A scalar string identifier to hold the name of the next page of the given page (if it exists).

**Return value:**

The function returns 1 on success, or 0 if the given page name does not exist or if the page does not have a next page.

**See also:**

PageGetParent

The function PageGetParent retrieves the name of the parent page for a specific page in the Page Manager tree.

```
PageGetParent(
    page,       ! (input) scalar string expression
    parent-page ! (output) scalar string identifier
)
```

**Arguments:**

- **page**
  - A string expression containing the name of a (child) page in the Page Manager tree.

- **parent-page**
  - A scalar string identifier to hold the name of the parent page of the given page (if it exists).

**Return value:**

The function returns 1 on success, or 0 if the given page name does not exist or if the page does not have a parent page.

**See also:**

The functions PageGetChild, PageGetNext, PageGetPrevious.
PageGetPrevious

The function PageGetPrevious retrieves the name of the previous page for a specific page in the Page Manager tree. The previous page is the page that has the same parent page, and is positioned directly above the given page.

PageGetPrevious(
  page, ! (input) scalar string expression
  previous-page ! (output) scalar string identifier
)

Arguments:

  page
    A string expression containing the name of a (child) page in the Page Manager tree.

  previous-page
    A scalar string identifier to hold the name of the previous page of the given page (if it exists).

Return value:

The function returns 1 on success, or 0 if the given page name does not exist or if the page does not have a previous page.

See also:

PageOpen

With the function PageOpen you can open any page that is defined in the Page Manager. If the page is already open, then the function will make this page the active page. The PageOpen function does not halt the execution, unless the page to open is defined as a dialog page. In the latter case, the execution is halted until the user closes the page.

```
PageOpen(
    page ! (input) string expression
)
```

**Arguments:**

- `page`
  
  A string expression representing the name of the page that you want to open. This name is the unique name as it appears in the Page Manager tree.

**Return value:**

The function returns 1 if the page is opened successfully. If the function fails to open the page it returns 0, and the pre-defined parameter `CurrentErrorMessage` will contain a proper error message.

**See also:**

The functions `PageOpenSingle`, `PageClose`. 
PageOpenSingle

The function PageOpenSingle is similar to PageOpen, except that after successful opening the page PageOpenSingle makes sure that all other currently opened pages are closed.

PageOpenSingle(
    page ! (input) string expression
)

**Arguments:**

*page*

A string expression representing the name of the page that you want to open. This name is the unique name as it appears in the Page Manager tree.

**Return value:**

The function returns 1 if the page is opened successfully. If the function fails to open the page it returns 0, and the pre-defined parameter CurrentErrorMessage will contain a proper error message.

**See also:**

The functions PageOpen, PageClose.
PageRefreshAll

Normally, the data on all open pages is refreshed automatically each time AIMMS has finished executing a function. Via a call to PageRefreshAll you can refresh the data on all pages at any time during a function run (for example to show intermediate results).

PageRefreshAll

**Arguments:**

*None*

**Remarks:**

Pages that you open from within a function will always show the data that is available at that moment, so it is not necessary to call PageRefreshAll for a newly opened page.

**See also:**

The function PageOpen.
Page Functions

PageSetCursor

With the function PageSetCursor you have maximum control over where you want to set the current keyboard input focus. Similar to PageSetFocus you can specify which page object should get the focus, but additionally you can specify the data element that should be highlighted within the focus object.

PageSetCursor(
    page ! (input) scalar string expression
    tag, ! (input) scalar string expression
    scalar-reference, ! (input) scalar identifier
)

Arguments:

page
A string expression representing the name of the page in which you want to set the input focus.

tag
A string expression representing the tag name of the object that should get the keyboard input focus.

scalar-reference
A scalar data element that matches the element that you want to highlight within the object.

Return value:

The function returns 1 on success. If it fails, then it returns 0 and the pre-defined identifier CurrentErrorMessage will contain a proper error message.

Remarks:

You can specify a unique tag name for each page object via the object properties.

See also:

The function PageSetFocus.
**PageSetFocus**

With the function `PageSetFocus` you can set the keyboard input focus to a specific object within a specific page. If the page is not open, then the function will first try to open the page.

```plaintext
PageSetFocus(
    page, ! (input) scalar string expression
    tag    ! (input) scalar string expression
)
```

**Arguments:**

- **page**  
  A string expression representing the name of the page in which you want to set the input focus.

- **tag**  
  A string expression representing the tag name of the object that should get the keyboard input focus.

**Return value:**

The function returns 1 on success. If it fails to set the focus to the specified object, then the return value is 0 and `CurrentErrorMessage` will contain a proper error message.

**Remarks:**

You can specify a unique tag name for each page object via the object properties.

**See also:**

The functions `PageSetCursor`, `PageGetFocus`.
PrintEndReport

With the function PageEndReport you finish the printing of a report that was started via a call to PrintStartReport.

PrintEndReport

Arguments:

None

Return value:

The function returns 1 on success, or 0, if there was no current report.

See also:

The functions PrintStartReport, PrintPage.
**PrintPage**

With the function PagePrint you can print a single print page. If the page contains a data object for which the available data does not fit onto a single printed sheet, AIMMS will print as many sheets as needed.

\[
\text{PrintPage}(\text{page}, \quad ! (\text{input}) \text{ scalar string expression} \\
\text{[filename,]} \quad ! (\text{optional}) \text{ scalar string expression} \\
\text{[from,]} \quad ! (\text{optional}) \text{ integer} \\
\text{[to]} \quad ! (\text{optional}) \text{ integer} 
\]

**Arguments:**

- **page**
  A string expression representing the name of the page that you want to print. This name is the unique name as it appears in the Page Manager tree.

- **filename (optional)**
  If this file name is specified, then AIMMS will print to the specific file and not directly to the printer. If this argument is omitted, then AIMMS will print according to the settings of the currently selected printer.

- **from (optional)**
  If the objects on the page result in multiple printed sheets, then with this argument you can specify the first sheet to print. If omitted, then printing will start at the first sheet (from = 1).

- **to (optional)**
  If the objects on the page result in multiple printed sheets, then with this argument you can specify the last sheet to print. If omitted, then printing continues until the last sheet.

**Return value:**

The function returns the actual number of pages printed if the print page is printed successfully. If the function fails to print the page it returns 0, and the pre-defined parameter CurrentErrorMessage will contain a proper error message.

**See also:**

The functions PrintPageCount, PrintStartReport.
**PrintPageCount**

The function `PagePrintCount` will return how many sheets of paper are needed to print a single print page in the interface.

```
PrintPageCount(
    page ! (input) scalar string expression
)
```

**Arguments:**

`page`
A string expression representing the name of the page that you want to print. This name is the unique name as it appears in the Page Manager tree.

**Return value:**

The function returns the number of sheets needed, or 0 if the page cannot be printed.

**See also:**

The function `PrintPage`. 
PrintStartReport

With the function PageStartReport you start printing a report that consists of the printing of multiple pages (using PagePrint). The advantage of printing via a report is that all print request until PageEndReport arrive at the printer as a single print job, and that the pages are numbered correctly.

```aimms
PrintStartReport(
   title, ! (input) scalar string expression
   [filename] ! (optional) scalar string expression
);
```

**Arguments:**

- **title**
  A string expression representing the title of the report. This title is used in the communication to the printer as the name of the print job.

- **filename (optional)**
  If this file name is specified, then AIMMS will print to the specific file and not directly to the printer. If this argument is omitted, then AIMMS will print according to the settings of the currently selected printer.

**Return value:**

The function returns 1 on success. If the function fails, then the pre-defined parameter CurrentErrorMessage will contain a proper error message.

**Remarks:**

A successful call to PrintStartReport must be followed by a call to PrintEndReport, otherwise nothing is printed, and your printer may hang.

**See also:**

The functions PrintEndReport, PrintPage.
File and Directory Functions

AIMMS supports the following functions for accessing disk files and directories:

- DirectoryCopy
- DirectoryCreate
- DirectoryDelete
- DirectoryExists
- DirectoryGetCurrent
- DirectoryMove
- DirectorySelect
- FileAppend
- FileCopy
- FileDelete
- FileEdit
- FileExists
- FileMove
- FilePrint
- FileSelect
- FileSelectNew
- FileTime
- FileView
**DirectoryCopy**

The function `DirectoryCopy` copies one or more directories to a new or other directory.

```fortran
DirectoryCopy(
    srcname, ! (input) scalar string expression
destname, ! (input) scalar string expression
confirm ! (optional) 0 or 1
)
```

**Arguments:**

- **srcname**
  A scalar string expression representing the directories(s) you want to copy. The string may contain wild-card characters such as ‘*’ and ‘?’, allowing you to copy a whole group of directories at once.

- **destname**
  A scalar string expression representing the destination directory.

- **confirm (optional)**
  An integer value that indicates whether you want to let the user confirm any copy operation that would overwrite existing files. If this argument is omitted, then the default behavior is that files are overwritten without any notice.

**Return value:**

The function returns 1 on success. If it fails, then it returns 0, and the pre-defined identifier `CurrentErrorMessage` will contain a proper error message.

**See also:**

**DirectoryCreate**

The function `DirectoryCreate` creates a new directory on your disk.

```plaintext
DirectoryCreate(  
    directory ! (input) scalar string expression  
)
```

**Arguments:**

- `directory`
  A scalar string expression representing the new directory name. If the name does not contain a full path, then the it is assumed to be relative to the current project directory.

**Return value:**

The function returns 1 if the directory is created successfully. If it fails, then it returns 0, and the pre-defined identifier `CurrentErrorMessage` will contain a proper error message.

**Remarks:**

If the new directory path contains references to non-existing directories, then the function tries to create each of these directories.

**See also:**

The functions `DirectoryExists, DirectoryDelete`. 
**DirectoryDelete**

The function `DirectoryDelete` deletes one or more directories from your disk. If any of these directories contain files, then these files are deleted as well.

```plaintext
DirectoryDelete(
    directory ! (input) scalar string expression
)
```

**Arguments:**

- `directory`
  A scalar string expression representing the directories you want to delete. The string may contain wild-card characters such as '*' and '?', allowing you to delete a whole group of directories at once.

**Return value:**

The function returns 1 on success. If it fails, then it returns 0, and the pre-defined identifier `CurrentErrorMessage` will contain a proper error message.

**See also:**

The functions `DirectoryExists`, `FileDelete`. 
**DirectoryExists**

With the function `DirectoryExists` you can check whether a specific directory name currently exists.

\[
\text{DirectoryExists}(\text{filename}) ! (\text{input}) \text{ scalar string expression}
\]

**Arguments:**

- `filename`  
  A scalar string expression representing a valid directory name. The file name may contain a partial path relative to the project directory, or a full path.

**Return value:**

The function returns 1 if the given directory name exists, or 0 otherwise.

**Remarks:**

Note that if you want use some static directory name in your model, then you have to specify two slashes behind each directory, as in "c:\windows\temp".

**See also:**

The function `DirectoryDelete`.
**DirectoryGetCurrent**

The function `DirectoryGetCurrent` retrieves the full path of the current project directory.

```plaintext
DirectoryGetCurrent(  
    directory ! (output) scalar string parameter  
)
```

**Arguments:**

- `directory`  
  A scalar string parameter, that on return will contain the path of the current project directory. The string is always terminated by a directory slash `\`.

**Return value:**

The function returns 1.

**See also:**

The function `DirectorySelect`.
DirectoryMove

The function DirectoryMove moves one or more directories to either a new name (a rename) or to another directory.

```
DirectoryMove(
    srcname, ! (input) scalar string expression
    destname, ! (input) scalar string expression
    confirm ! (optional) 0 or 1
)
```

**Arguments:**

`srcname`
A scalar string expression representing the file(s) you want to move. The string may contain wild-card characters such as ‘*’ and ‘?’, allowing you to move a whole group of directories at once.

`destname`
A scalar string expression representing the destination directory.

`confirm (optional)`
An integer value that indicates whether or not you want to let the user confirm any move operation that would overwrite existing files. If this argument is omitted, then the default behavior is that files are overwritten without any notice.

**Return value:**

The function returns 1 on success. If it fails, then it returns 0, and the pre-defined identifier `CurrentErrorMessage` will contain a proper error message.

**See also:**

DirectorySelect

With the function DirectorySelect you can let the user select an existing directory using Windows’ standard directory selection dialog box.

DirectorySelect(
    directory-name, ! (output) scalar string parameter
    [directory,] ! (optional input) scalar string expression
    [title] ! (optional input) scalar string expression
)

Arguments:

directory-name
A scalar string parameter. On return this parameter will represent the selected directory name. If the selected directory is a sub directory below the current project directory, then the directory name will be presented using a relative path. In other cases the directory name is presented using a full path specification. In both cases, the returned directory string is terminated by a \ character.

directory (optional)
A scalar string representing an existing directory. The dialog box will initially select this directory. If omitted, then the current project directory will be used.

title (optional)
A scalar string that is used as the title of the selection dialog box. If this argument is omitted, then a default title is used.

Return value:

The function returns 1 if the user did select a directory. If some error occurs or if the user presses the Cancel button, then the function returns 0.

Remarks:

If DirectorySelect returns 0, then the first argument may not contain a valid directory path. So you must always check the return value, and, if it is 0, either abort the current function or continue with some default directory name.

See also:

The functions FileSelect, DirectoryGetCurrent.
**FileAppend**

The function `FileAppend` appends the contents of one file to the end of another file. Both files must be ASCII text files.

\[
\text{FileAppend}(
    \text{filename}, \quad ! \text{(input) scalar string expression}
    \text{appendname} \quad ! \text{(input) scalar string expression}
)\]

**Arguments:**

- **filename**
  A scalar string expression representing the file name to which you want to append the contents of the second file.

- **appendname**
  A scalar string expression representing the file name that you want to append.

**Return value:**

The function returns 1 on success. If it fails, then it returns 0, and the pre-defined identifier `CurrentErrorMessage` will contain a proper error message.

**Remarks:**

If the first file (the file to which you append) does not exist, then this file will be created.

**See also:**

The functions `FileCopy`, `FileExists`. 
FileCopy

The function `FileCopy` copies one or more files to a new name or to another directory.

```plaintext
FileCopy(
    srcname, ! (input) scalar string expression
    destname, ! (input) scalar string expression
    [confirm] ! (optional) 0 or 1
)
```

**Arguments:**

- `srcname`
  A scalar string expression representing the file(s) you want to copy. The string may contain wild-card characters such as ‘*’ and ‘?’, allowing you to copy a whole group of files at once.

- `destname`
  A scalar string expression representing the destination file name or destination directory.

- `confirm (optional)`
  An integer value that indicates whether or not you want to let the user confirm any copy operation that would overwrite existing files. If this argument is omitted, then the default behavior is that files are overwritten without any notice.

**Return value:**

The function returns 1 on success. If it fails, then it returns 0, and the pre-defined identifier `CurrentErrorMessage` will contain a proper error message.

**See also:**

The functions `FileMove`, `DirectoryCopy`, `FileExists`. 
FileDelete

The function FileDelete deletes one or more files from your disk.

\[
\text{FileDelete}(\text{filename} ! (\text{input}) \text{ scalar string expression})
\]

**Arguments:**

\* filename

A scalar string expression representing the file(s) you want to delete.
The string may contain wild-card characters such as ‘*’ and ‘?’, allowing you to delete a whole group of files at once.

**Return value:**

The function returns 1 on success. If it fails, then it returns 0, and the pre-defined identifier `CurrentErrorMessage` will contain a proper error message.

**See also:**

The functions `FileExists`, `DirectoryDelete`. 
**FileEdit**

The function `FileEdit` opens a specific file in the internal AIMMS text file editor. Optionally, you can set the cursor on a specific piece of text within the file.

```aimms
FileEdit(  
    filename,  ! (input) scalar string expression  
    [find]    ! (optional) scalar string expression  
)
```

**Arguments:**

- **filename**
  A scalar string expression representing the file name that you want to edit.

- **find (optional)**
  A scalar string expression that is used to position the cursor over a specific piece of text in the file. If this argument is omitted (or if the specified text cannot be found), then the cursor will be positioned at the top of the file.

**Return value:**

The function returns 1 on success, and 0 if it could not open the file in the editor.

**Remarks:**

If you want to use another external text editor to edit a specific file, then you can use the function `Execute`.

**See also:**

The functions `FileView`, `Execute`. 
FileExists

With the function FileExists you can check whether a specific file name currently exists.

```plaintext
FileExists( 
    filename ! (input) scalar string expression
)
```

**Arguments:**

`filename`

A scalar string expression representing a valid file name. The file name may contain a partial path relative to the project directory, or a full path.

**Return value:**

The function returns 1 if the given file name exists, and 0 otherwise.

**Remarks:**

Note that if you want use some static file name in your model, then you have to specify two slashes behind each directory, as in "c:\windows\temp\filename.dat"

**See also:**

The function FileDelete
FileMove

The function FileMove moves one or more files to either a new name (a rename) or to another directory.

FileMove(
    srcname,   ! (input) scalar string expression
    destname, ! (input) scalar string expression
    [confirm] ! (optional) 0 or 1
)

Arguments:

srcname
A scalar string expression representing the file(s) you want to move. The string may contain wild-card characters such as ‘*’ and ‘?’, allowing you to move a whole group of files at once.

destname
A scalar string expression representing the destination file name or destination directory.

confirm (optional)
An integer value that indicates whether or not you want to let the user confirm any move operation that would overwrite existing files. If this argument is omitted, then the default behavior is that files are overwritten without any notice.

Return value:

The function returns 1 on success. If it fails, then it returns 0, and the pre-defined identifier `CurrentErrorMessage` will contain a proper error message.

See also:

The functions FileCopy, DirectoryMove, FileExists.
**FilePrint**

The function `FilePrint` prints a specific text file using the currently selected printer.

```plaintext
FilePrint(
    filename   ! (input) scalar string expression
)
```

**Arguments:**

- `filename`
  - A scalar string expression representing the text file that you want to print.

**Return value:**

The function returns 1 on success, and 0 if it could not print the file.

**Remarks:**

The file is printed using the paper and font settings that are specified in the Text Printing dialog box, which is accessible from the Settings menu.

**See also:**

The function `FileEdit`. 
FileSelect

With the function FileSelect you can let the user select an existing file name using Windows' standard file selection dialog box. Usually you use this function to select some input file (i.e. a file for reading), because other than FileSelectNew, this function only allows the user to select existing files.

\[
\text{FileSelect}(
    \begin{array}{l}
    \text{filename}, \quad \text{(input/output) scalar string identifier} \\
    [\text{directory,}] \quad \text{(optional) scalar string expression} \\
    [\text{extension,}] \quad \text{(optional) scalar string expression} \\
    [\text{title}] \quad \text{(optional) scalar string expression}
    \end{array}
\)
\]

**Arguments:**

- **filename**
  A scalar string identifier holding the file name that the user selected. If on entry this string represents a valid file name, then this file name is used to initialize the dialog box.

- **directory (optional)**
  A scalar string representing an existing directory. The dialog box will initially only show the files that are located in this directory. If this argument is omitted, then the current project directory will be used.

- **extension (optional)**
  A scalar string representing a file extension. The dialog box will initially only show those files that match this extension. If this argument is omitted, then all files are shown.

- **title (optional)**
  A scalar string that is used as the title of the selection dialog box. If this argument is omitted, then a default title is used.

**Return value:**

The function returns 1 if the user actually has selected a file. If some error occurs or if the user presses the **Cancel** button, the function returns 0.

**Remarks:**

If FileSelect returns 0, then the first argument may not contain a valid file name. So you must always check the return value, and, if it is 0, either abort the current function or continue with some default file name.

**See also:**

The function FileSelectNew.
**FileSelectNew**

With the function `FileSelectNew` the user can select a new (or existing) file using Windows' file selection dialog box. Usually it is used to select an output file (i.e. for writing), because other than `FileSelect`, this function allows you to specify new file names. If an existing file name is selected, a warning will be displayed. The function does not create any files on disk or make any changes to existing files. It only returns the file name selected by the user.

```plaintext
FileSelectNew(
    filename,  ! (input/output) scalar string identifier
    [directory,]  ! (optional) scalar string expression
    [extension,]  ! (optional) scalar string expression
    [title]  ! (optional) scalar string expression
)
```

**Arguments:**

- `filename`
  A scalar string identifier holding the file name that the user specified. If on entry this string represents a valid file name, then this file name is used to initialize the dialog box.

- `directory (optional)`
  A scalar string representing an existing directory. The dialog box will initially only show the files that are located in this directory. If this argument is omitted, then the current project directory will be used.

- `extension (optional)`
  A scalar string representing a file extension. The dialog box will initially only show those files that match this extension. If this argument is omitted, then all files are shown.

- `title (optional)`
  A scalar string that is used as the title of the selection dialog box. If this argument is omitted, then a default title is used.

**Return value:**

The function returns 1 if the user actually has selected a file. If some error occurs or if the user presses the **Cancel** button, the function returns 0.

**Remarks:**

If `FileSelectNew` returns 0, then the first argument may not contain a valid file name. So you must always check the return value, and, if it is 0, either abort the current function or continue with some default file name.

**See also:**

The function `FileSelect`.
**FileTime**

The function `FileTime` retrieves the last modification time of an existing file.

```lisp
FileTime(
    filename, ! (input) scalar string expression
    file-time ! (output) scalar string identifier
)
```

**Arguments:**

- **filename**
  A scalar string expression representing an existing file name.

- **file-time**
  A scalar string identifier to hold the file modification time of the specified file. This time is represented using AIMMS' standard date and time format: "YYYY-MM-DD hh:mm:ss"

**Return value:**

The function returns 1 on success. If it failed to retrieve the file time, then it returns 0 and the pre-defined identifier `CurrentErrorMessage` will contain a proper error message.

**See also:**

The function `FileExists`.
FileView

The function FileView opens a specific file in the internal AIMMS text file viewer. Optionally, you can highlight a specific piece of text within the file.

FileView(
    filename, ! (input) scalar string expression
    find       ! (optional) scalar string expression
)

Arguments:

filename
  A scalar string expression representing the file name that you want to edit.

find (optional)
  A scalar string expression that is used to position the cursor over a specific piece of text in the file. If this argument is omitted (or if the specified text cannot be found), then the cursor will be positioned at the top of the file.

Return value:

The function returns 1 on success, and 0 if it could not open the file in the viewer.

Remarks:

If you want to use another external text editor to view a specific file, then you can use the function Execute.

See also:

The functions FileEdit, Execute.
Distribution and Combinatoric Functions

AIMMS supports several functions to obtain random numbers from discrete or continuous distribution, and additionally some combinatoric functions. The functions for discrete distributions are:

- Binomial
- Geometric
- HyperGeometric
- NegativeBinomial
- Poisson

The functions for continuous distributions are:

- Beta
- ExtremeValue
- Gamma
- Logistic
- LogNormal
- Normal
- Pareto
- Permutation
- Triangular
- Uniform
- Weibull

The combinatoric functions are:

- Combination
- Exponential
- Factorial
**Beta**

The function `Beta` draws a random value from a beta distribution.

```
Beta(
    alpha, ! (input) numerical expression
    beta, ! (input) numerical expression
    s   ! (input) numerical expression
)
```

**Arguments:**

- `alpha`  
  A scalar numerical expression > 0.

- `beta`  
  A scalar numerical expression > 0.

- `s`  
  A scalar numerical expression > 0.

**Return value:**

The function `Beta` returns a random value drawn from a beta distribution\((alpha, beta)\) with scale \(s\).

**See also:**

The `Beta` distribution is discussed in full detail in Appendix A of the Language Reference.
Binomial

The function `Binomial` draws a random value from a binomial distribution.

```plaintext
Binomial(p, n)
```

**Arguments:**

- `p`  
  A scalar numerical expression in range `(0, 1)`.  
- `n`  
  An integer numerical expression > 0.

**Return value:**

The function `Binomial` returns a random value drawn from a binomial distribution with a probability of success `p` and number of tries `n`.

**See also:**

The `Binomial` distribution is discussed in full detail in Appendix A of the Language Reference.
Combination

The function Combination computes the number of combinations of length \( m \) in \( n \) items.

\[
\text{Combination}(n, \ m) \quad ! \text{ (input) integer expression}
\]

Arguments:

\( n \)

An integer numerical expression.

\( m \)

An integer numerical expression.

Return value:

The function Combination returns \( \binom{n}{m} \), the number of combinations of length \( m \) in a given number of items \( n \).

See also:

Combinatoric functions are discussed in full detail in Section 6.1.7.
**Exponential**

The function Exponential draws a random value from an exponential distribution.

```plaintext
Exponential(lambda ! (input) numerical expression)
```

**Arguments:**

- `lambda`
  - A scalar numerical expression $> 0$.

**Return value:**

The function Exponential returns a random value drawn from a exponential distribution with rate $\lambda$.

**See also:**

The Exponential distribution is discussed in full detail in Appendix A of the Language Reference.
The function \texttt{ExtremeValue} draws a random value from an extreme value distribution.

\begin{verbatim}
ExtremeValue(m, ! (input) numerical expression
    )
\end{verbatim}

\textbf{Arguments:}

\begin{itemize}
    \item \textit{m} \hfill A scalar numerical expression.
    \item \textit{s} \hfill A scalar numerical expression $> 0$.
\end{itemize}

\textbf{Return value:}

The function \texttt{ExtremeValue} returns a random value drawn from an extreme value distribution with mode \textit{m} and scale \textit{s}.

\textbf{See also:}

The \texttt{ExtremeValue} distribution is discussed in full detail in Appendix A of the Language Reference.
Factorial

The function Factorial returns the factorial of an integer number.

```
Factorial(
    n  ! (input) integer expression
)
```

**Arguments:**

- `n` An integer numerical expression > 0.

**Return value:**

The function Factorial returns the factorial value $n!$.

**See also:**

Combinatoric functions are discussed in full detail in Section 6.1.7.
The function Gamma draws a random value from a gamma distribution.

\[
\text{Gamma}(\alpha, \beta)
\]

**Arguments:**

- \(\alpha\): A scalar numerical expression > 0.
- \(\beta\): A scalar numerical expression > 0.

**Return value:**

The function Gamma returns a random value drawn from a gamma distribution with location \(\alpha\) and shape \(\beta\).

**See also:**

The Gamma distribution is discussed in full detail in Appendix A of the Language Reference.
**Geometric**

The function `Geometric` draws a random value from a geometric distribution.

```plaintext
Geometric(p) ! (input) numerical expression
```

**Arguments:**

- `p`
  A scalar numerical expression in the range (0, 1).

**Return value:**

The function `Geometric` returns a random value drawn from a geometric distribution with a probability of success `p`.

**See also:**

The Geometric distribution is discussed in full detail in Appendix A of the Language Reference.
HyperGeometric

The function HyperGeometric draws a random value from a hypergeometric distribution.

```csharp
HyperGeometric(
    p,    ! (input) numerical expression
    n,    ! (input) integer expression
    size ! (input) integer expression
)
```

**Arguments:**

- `p`  
  A scalar numerical expression in the range $(0, 1)$.

- `n`  
  A integer numerical expression in the range $1, \ldots, size$.

- `size`  
  A integer numerical expression $> 0$.

**Return value:**

The function HyperGeometric returns a random value drawn from a hypergeometric distribution with a probability of success $p$, number of tries $n$ and population size $size$.

**Remarks:**

The probability of success $p$ must assume one of the values $i/size$, where $i$ is in the range $1, \ldots, size - 1$.

**See also:**

The HyperGeometric distribution is discussed in full detail in Appendix A of the Language Reference.
Logistic

The function Logistic draws a random value from a logistic distribution.

\[ \text{Logistic}(m, s) \]

**Arguments:**

- \( m \)
  - A scalar numerical expression.
- \( s \)
  - A scalar numerical expression > 0.

**Return value:**

The function Logistic returns a random value drawn from a logistic distribution with mean \( m \) and scale \( s \).

**See also:**

The Logistic distribution is discussed in full detail in Appendix A of the Language Reference.
LogNormal

The function LogNormal draws a random value from a lognormal distribution.

\[
\text{LogNormal}(\ m, \ s)
\]

**Arguments:**

- \( m \)
  - A scalar numerical expression > 0.
- \( s \)
  - A scalar numerical expression > 0.

**Return value:**

The function LogNormal returns a random value drawn from a lognormal distribution with mean \( m \) and standard deviation \( s \).

**See also:**

The LogNormal distribution is discussed in full detail in Appendix A of the Language Reference.
NegativeBinomial

The function NegativeBinomial draws a random value from a negative binomial distribution.

```prolog
NegativeBinomial(  
    p,  ! (input) numerical expression  
    r  ! (input) integer expression  
)
```

**Arguments:**

- $m$
  A scalar numerical expression in the range $(0, 1)$.
- $s$
  A integer numerical expression > 0.

**Return value:**

The function NegativeBinomial returns a random value drawn from a negative binomial distribution with probability $p$ and number of successes $r$.

**See also:**

The NegativeBinomial distribution is discussed in full detail in Appendix A of the Language Reference.
Normal

The function \texttt{Normal} draws a random value from a normal distribution.

\begin{verbatim}
Normal(
    m, ! (input) numerical expression
    s ! (input) numerical expression
)
\end{verbatim}

Arguments:

\begin{itemize}
    \item \textit{m}
        A scalar numerical expression.
    \item \textit{s}
        A scalar numerical expression > 0.
\end{itemize}

Return value:

The function \texttt{Normal} returns a random value drawn from a normal distribution with mean \textit{m} and standard deviation \textit{s}.

See also:

The \texttt{Normal} distribution is discussed in full detail in Appendix A of the Language Reference.
Pareto

The function Pareto draws a random value from a Pareto distribution.

\[
\text{Pareto}(\text{l}, \beta)
\]

**Arguments:**

- \(l\): A scalar numerical expression > 0.
- \(\beta\): A scalar numerical expression > 0.

**Return value:**

The function Pareto returns a random value drawn from a Pareto distribution with location \(l\) and shape \(\beta\).

**See also:**

The Pareto distribution is discussed in full detail in Appendix A of the Language Reference.
Permutation

The function Permutation computes the number of permutations of length \( m \) in \( n \) items.

\[
\text{Permutation}(n, m) \quad ! \text{(input) integer expression} \quad \text{! (input) integer expression}
\]

Arguments:

\( n \)

An integer numerical expression > 0.

\( m \)

An integer numerical expression in the range 0,\ldots, n.

Return value:

The function Permutation returns \( m! \cdot \binom{n}{m} \), the number of permutations of length \( m \) in a given number of items \( n \).

See also:

Combinatoric functions are discussed in full detail in Section 6.1.7.
Poisson

The function Poisson draws a random value from a Poisson distribution.

\[
\text{Poisson}(\lambda \text{ ! (input) numerical expression})
\]

Arguments:

\(\lambda\)

A scalar numerical expression > 0.

Return value:

The function Poisson returns a random value drawn from a Poisson distribution with average number of occurrences \(\lambda\).

See also:

The Poisson distribution is discussed in full detail in Appendix A of the Language Reference.
Triangular

The function Triangular draws a random value from a triangular distribution.

\[
\text{Triangular}(a, b, c)
\]

\textbf{Arguments:}

\begin{itemize}
\item \textit{a} \\
A scalar numerical expression.
\item \textit{b} \\
A scalar numerical expression.
\item \textit{c} \\
A scalar numerical expression.
\end{itemize}

\textbf{Return value:}

The function \texttt{Triangular} returns a random value drawn from a triangular distribution with a lower bound \textit{a}, likeliest value \textit{b} and upper bound \textit{c}.

\textbf{Remarks:}

The arguments must satisfy the relation \textit{a} < \textit{b} < \textit{c}.

\textbf{See also:}

The \texttt{Triangular} distribution is discussed in full detail in Appendix A of the Language Reference.
Uniform

The function \texttt{Uniform} draws a random value from a uniform distribution.

\begin{verbatim}
Uniform(
	 a,       ! (input) numerical expression
	 b       ! (input) numerical expression
)
\end{verbatim}

**Arguments:**

\begin{itemize}
  \item \texttt{a} \\
    A scalar numerical expression.
  \item \texttt{b} \\
    A scalar numerical expression.
\end{itemize}

**Return value:**

The function \texttt{Uniform} returns a random value drawn from a uniform distribution with lower bound \texttt{a} and upper bound \texttt{b}.

**Remarks:**

The arguments must satisfy the relation \texttt{a < b}.

**See also:**

The \texttt{Uniform} distribution is discussed in full detail in Appendix A of the Language Reference.
Weibull

The function Weibull draws a random value from a Weibull distribution.

```plaintext
Weibull(  
    l,       ! (input) numerical expression  
    beta,    ! (input) numerical expression  
    s        ! (input) numerical expression  
)
```

**Arguments:**

- `l`
  A scalar numerical expression.
- `beta`
  A scalar numerical expression > 0.
- `s`
  A scalar numerical expression > 0.

**Return value:**

The function Weibull returns a random value drawn from a Weibull distribution with location `l`, shape `beta` and scale `s`.

**See also:**

The Weibull distribution is discussed in full detail in Appendix A of the Language Reference.
AIMMS supports the following functions for creating and managing histograms:

- HistogramAddObservation
- HistogramCreate
- HistogramDelete
- HistogramGetAverage
- HistogramGetBounds
- HistogramGetDeviation
- HistogramGetFrequencies
- HistogramGetKurtosis
- HistogramGetObservationCount
- HistogramGetSkewness
- HistogramSetDomain
Histogram Add Observation

The function `HistogramAddObservation` adds a new observation to a histogram that was previously created through the function `HistogramCreate`.

```plaintext
HistogramAddObservation(
    histogram-id, ! (input) a scalar parameter
    value      ! (input) a scalar value
)
```

**Arguments:**

`histogram-id`

A scalar value representing a histogram that was previously created using the `HistogramCreate` function.

`value`

The value of a new observation that should be added to the histogram.

**Return value:**

The function returns 1 if the new observation is added successfully, or 0 otherwise.

**See also:**

The function `HistogramCreate`. Histogram support in AIMMS is discussed in full detail in Section 15.3 of the User’s Guide.
**HistogramCreate**

The function `HistogramCreate` sets up a new histogram. The created histogram does not yet contain any observations. These observations must be added later using the function `HistogramAddObservation`.

```plaintext
HistogramCreate(
    histogram-id, ! (output) a scalar parameter
    [integer-histogram,] ! (optional) 0 or 1
    [sample-buffer-size] ! (optional) a positive integer value
)
```

**Arguments:**

- **histogram-id**
  - On return, this argument will contain a unique identification number, that is used to refer to the created histogram in other functions.

- **integer-histogram (optional)**
  - A logical indicator that specifies whether the observations will be integer-valued. Default is 0 (not integer).

- **sample-buffer-size (optional)**
  - The sample buffer size used in the histogram. If omitted, a default buffer size of 512 is used.

**Return value:**

The function returns 1 if the histogram is created successfully, or 0 otherwise.

**See also:**

The functions `HistogramDelete`, `HistogramAddObservation`. Histogram support in AIMMS is discussed in full detail in Section 15.3 of the User’s Guide.
HistogramDelete

The function HistogramDelete deletes a histogram that was created using the HistogramCreate function. After the histogram has been deleted, the histogram id is no longer valid.

```
HistogramDelete(
    histogram-id ! (input) a scalar parameter
)
```

**Arguments:**

- `histogram-id`
  A scalar value representing a histogram that was previously created using the HistogramCreate function. When the function returns, this `histogram-id` no longer refers to a valid histogram.

**Return value:**

The function returns 1 if the histogram is deleted successfully, or 0 otherwise.

**See also:**

The function HistogramCreate. Histogram support in AIMMS is discussed in full detail in Section 15.3 of the User's Guide.
Histogram Functions

**HistogramGetAverage**

The function `HistogramGetAverage` returns the arithmetic mean of all observations in a histogram.

```plaintext
HistogramGetAverage(
    histogram-id ! (input) a scalar number
)
```

**Arguments:**

`histogram-id`
A scalar value representing a histogram that was previously created using the `HistogramCreate` function.

**Return value:**

The function returns the arithmetic mean of all observations added to the histogram.

**See also:**

The functions `HistogramCreate, HistogramGetObservationCount, HistogramGetDeviation, HistogramGetSkewness, HistogramGetKurtosis`. Histogram support in AIMMS is discussed in full detail in Section 15.3 of the User's Guide.
**HistogramGetBounds**

Through the function `HistogramGetBounds` you can obtain the lower and upper bounds of frequency interval in a histogram.

```plaintext
HistogramGetBounds(
    histogram-id, ! (input) a scalar number
    left-bound,   ! (output) a one-dimensional parameter
    right-bound   ! (output) a one-dimensional parameter
)
```

**Arguments:**

- `histogram-id`  
  A scalar value representing a histogram that was previously created using the `HistogramCreate` function.

- `left-bound`  
  A one-dimensional identifier that will be filled with the left bound of each interval in the histogram. The cardinality of the domain set should be at least the number of intervals.

- `right-bound`  
  A one-dimensional identifier that will be filled with the right bound of each interval in the histogram. The cardinality of the domain set should be at least the number of intervals.

**Return value:**

The function returns 1 if the bounds are retrieved successfully, or 0 otherwise.

**See also:**

The functions `HistogramCreate`, `HistogramSetDomain`. Histogram support in AIMMS is discussed in full detail in Section 15.3 of the User's Guide.
HistogramGetDeviation

The function HistogramGetDeviation returns the standard deviation of all observations in a histogram.

```
HistogramGetDeviation(
    histogram-id ! (input) a scalar number
)
```

Arguments:

- `histogram-id` A scalar value representing a histogram that was previously created using the HistogramCreate function.

Return value:

The function returns the standard deviation of all observations in the histogram.

See also:

The functions HistogramCreate, HistogramGetObservationCount, HistogramGetAverage, HistogramGetSkewness, HistogramGetKurtosis. Histogram support in AIMMS is discussed in full detail in Section 15.3 of the User's Guide.
Histogram Get Frequencies

Through the function Histogram Get Frequencies you can obtain the observed frequencies for each interval in a histogram.

```
HistogramGetFrequencies(
    histogram-id,  ! (input) a scalar number
    frequencies    ! (output) a one-dimensional parameter
)
```

**Arguments:**

- **histogram-id**
  A scalar value representing a histogram that was previously created using the Histogram Create function.

- **frequencies**
  A one-dimensional identifier that will be filled with the frequencies of each interval in the histogram. The cardinality of the domain set should be at least the number of intervals.

**Return value:**

The function returns 1 if the frequencies are retrieved successfully, or 0 otherwise.

**See also:**

The functions Histogram Create, Histogram Add Observation. Histogram support in AIMMS is discussed in full detail in Section 15.3 of the User's Guide.
HistogramGetKurtosis

The function HistogramGetKurtosis returns the kurtosis coefficient of all observations in a histogram.

```plaintext
HistogramGetKurtosis(  
    histogram-id ! (input) a scalar number  
)
```

**Arguments:**

- `histogram-id`  
  A scalar value representing a histogram that was previously created using the HistogramCreate function.

**Return value:**

The function returns the kurtosis coefficient of all observations in the histogram.

**See also:**

The functions HistogramCreate, HistogramGetObservationCount, HistogramGetAverage, HistogramGetDeviation, HistogramGetSkewness. Histogram support in AIMMS is discussed in full detail in Section 15.3 of the User's Guide.
**Histogram Functions**

---

**HistogramGetObservationCount**

The function `HistogramGetObservationCount` returns the total number of observations in a histogram.

```plaintext
HistogramGetObservationCount(
    histogram-id ! (input) a scalar number
)
```

**Arguments:**

- `histogram-id`
  
  A scalar value representing a histogram that was previously created using the `HistogramCreate` function.

**Return value:**

The function returns the total number of observations in a histogram.

**See also:**

The functions `HistogramCreate`, `HistogramGetAverage`, `HistogramGetDeviation`, `HistogramGetSkewness`, `HistogramGetKurtosis`. Histogram support in AIMMS is discussed in full detail in Section 15.3 of the User’s Guide.
**HistogramGetSkewness**

The function `HistogramGetSkewness` returns the skewness of all observations in a histogram.

```plaintext
HistogramGetSkewness(
    histogram-id ! (input) a scalar number
)
```

**Arguments:**

- `histogram-id`
  A scalar value representing a histogram that was previously created using the `HistogramCreate` function.

**Return value:**

The function returns the skewness of all observations in the histogram.

**See also:**

The functions `HistogramCreate`, `HistogramGetObservationCount`, `HistogramGetAverage`, `HistogramGetDeviation`, `HistogramGetKurtosis`. Histogram support in AIMMS is discussed in full detail in Section 15.3 of the User's Guide.
HistogramSetDomain

With the function HistogramSetDomain you can override the default layout of frequency intervals of a histogram.

```
HistogramSetDomain(
    histogram-id, ! (input) a scalar number
    intervals, ! (input) a positive integer number
    [left,] ! (optional) a scalar expression
    [width,] ! (optional) a positive scalar number
    [left-tail,] ! (optional) 0 or 1
    [right-tail] ! (optional) 0 or 1
)
```

**Arguments:**

- **histogram-id**
  A scalar value representing a histogram that was previously created using the HistogramCreate function.

- **intervals**
  The number of fixed-width intervals (not including the left- or right-tail interval).

- **left (optional)**
  The lower bound of the left-most interval (not including the left-tail interval). If omitted, then the histogram will use the observations to determine this bound.

- **width (optional)**
  The (fixed) width of each interval. If omitted, then the histogram will use the observations to determine the width.

- **left-tail (optional)**
  An indicator whether or not a left-tail interval should be created. If this argument is omitted, then the default value of 1 is used (creating a left-tail interval).

- **right-tail (optional)**
  An indicator whether or not a right-tail interval should be created. If this argument is omitted, then the default value of 1 is used (creating a right-tail interval).

**Return value:**

The function returns 1 if the domain is changed successfully, or 0 otherwise.

**See also:**

The functions HistogramCreate, HistogramGetBounds. Histogram support in AIMMS is discussed in full detail in Section 15.3 of the User's Guide.
Matrix Manipulation Functions

AIMMS supports the following matrix manipulation functions:

- MatrixActivateRow
- MatrixAddColumn
- MatrixAddRow
- MatrixDeactivateRow
- MatrixFreezeColumn
- MatrixModifyCoefficient
- MatrixModifyColumnType
- MatrixModifyDirection
- MatrixModifyLeftHandSide
- MatrixModifyLowerBound
- MatrixModifyRightHandSide
- MatrixModifyRowType
- MatrixModifyType
- MatrixModifyUpperBound
- MatrixReSolve
- MatrixRegenerateRow
- MatrixRestoreState
- MatrixSaveState
- MatrixUnfreezeColumn
Matrix Activate Row

The procedure MatrixActivateRow activates a row in the matrix that was previously deactivated.

```
MatrixActivateRow(
    MP, ! (input) a mathematical program
    row  ! (input) a scalar value
)
```

**Arguments:**

- **MP**
  A mathematical program that was previously solved. The mathematical program should be a linear or mixed-integer linear programming model.

- **row**
  A scalar reference to an existing row in the matrix; this can not be the objective row.

**Return value:**

The function returns 1 on success, or 0 otherwise.

**See also:**

The function MatrixDeactivateRow. Matrix manipulation routines are discussed in more detail in Chapter 17 of the Language Reference.
Matrix Manipulation Functions

MatrixAddColumn

The procedure MatrixAddColumn adds a column to the matrix.

MatrixAddColumn(
    MP, ! (input) a mathematical program
    column ! (input) a scalar value
)

Arguments:

    MP
        A mathematical program that was previously solved. The
        mathematical program should be a linear or mixed-integer linear
        programming model.

    column
        A scalar reference to an existing column name in the model.

Return value:

    The function returns 1 on success, or 0 otherwise.

Remarks:

    Coefficients for this column can be added to the matrix by using the
    procedure MatrixModifyCoefficient. After calling MatrixAddColumn the type
    and the lower and upper bound of the column are set according to the
    definition of the corresponding symbolic variable. By using the
    procedures MatrixModifyColumnType, MatrixModifyLowerBound and
    MatrixModifyUpperBound the column type and the lower and upper bound
    can be changed.

See also:

    The functions MatrixModifyCoefficient, MatrixModifyColumnType,
    MatrixModifyLowerBound, MatrixModifyUpperBound. Matrix manipulation
    routines are discussed in more detail in Chapter 17 of the Language
    Reference.
MatrixAddRow

The procedure MatrixAddRow adds a row to the matrix.

\[
\text{MatrixAddRow}(\text{MP}, \text{row})
\]

Arguments:

\( \text{MP} \)
- A mathematical program that was previously solved. The mathematical program should be a linear or mixed-integer linear programming model.

\( \text{row} \)
- A scalar reference to an existing row name in the model.

Return value:

The function returns 1 on success, or 0 otherwise.

Remarks:

- Initially, the row will be added with zero coefficients, regardless of whether the symbolic AIMMS constraint has a definition or not. Regeneration of all of the coefficients of the row according to its definition can be achieved through the function MatrixRegenerateRow. Individual coefficients of the row can be added by calling the procedure MatrixModifyCoefficient.
- After calling MatrixAddRow the type of the row is set to '\( \leq \)' and the right-hand-side value to INF (the left-hand-side value is set to -INF). By using the procedures MatrixModifyRowType and MatrixModifyRightHandSide the row type and right-hand-side value can be changed.
- After a call to MatrixAddRow or MatrixRegenerateRow for a certain row it is not allowed to do another call to MatrixAddRow for that row.

See also:

The functions MatrixModifyCoefficient, MatrixModifyLeftHandSide, MatrixModifyRightHandSide, MatrixModifyRowType, MatrixRegenerateRow.

Matrix manipulation routines are discussed in more detail in Chapter 17 of the Language Reference.
Matrix Deactivate Row

The procedure MatrixDeactivateRow deactivates a row in the matrix. The row will be ignored by the solver until it is activated again.

MatrixDeactivateRow(
    MP, ! (input) a mathematical program
    row ! (input) a scalar value
)

Arguments:

MP
A mathematical program that was previously solved. The mathematical program should be a linear or mixed-integer linear programming model.

row
A scalar reference to an existing row in the matrix; this can not be the objective row.

Return value:

The function returns 1 on success, or 0 otherwise.

Remarks:

Deactivating a row results in changing the type of that row into '<' and the right hand side value into INF (the row coefficients do not change).

See also:

The function MatrixActivateRow. Matrix manipulation routines are discussed in more detail in Chapter 17 of the Language Reference.
Matrix Manipulation Functions

MatrixFreezeColumn

The procedure MatrixFreezeColumn fixes the value of a column in the model. The column can be freed by using MatrixUnfreezeColumn.

MatrixFreezeColumn(
  MP,    ! (input) a mathematical program
  column, ! (input) a scalar value
  value, ! (input) a numerical expression
)

Arguments:

  MP
  A mathematical program that was previously solved. The mathematical program should be a linear or mixed-integer linear programming model.

  row
  A scalar reference to an existing column in the matrix.

  value
  The value to which the column should be fixed.

Return value:

  The function returns 1 on success, or 0 otherwise.

Remarks:

  Fixing a column to a certain value has the same effect as changing the lower and upper bound into that value.

See also:

  The functions MatrixModifyLowerBound, MatrixModifyUpperBound, MatrixUnfreezeColumn. Matrix manipulation routines are discussed in more detail in Chapter 17 of the Language Reference.
MatrixManipulateCoefficient

The procedure MatrixManipulateCoefficient changes a coefficient in the matrix. This procedure can also be used to modify a coefficient in the objective row. The value for the coefficient can be equal to 0.0 prior to calling this procedure.

MatrixManipulateCoefficient(
    MP, ! (input) a mathematical program
    row, ! (input) a scalar value
    column, ! (input) a scalar value
    value ! (input) a numerical expression
)

Arguments:

MP
    A mathematical program that was previously solved. The mathematical program should be a linear or mixed-integer linear programming model.

row
    A scalar reference to an existing row in the matrix; this might be the objective row.

column
    A scalar reference to an existing column in the matrix.

value
    The new value that should be assigned to the coefficient corresponding to row and column in the matrix. This value should be unequal to NA, INF, -INF and UNDF.

Return value:

The function returns 1 on success, and 0 otherwise.

See also:

Matrix manipulation routines are discussed in more detail in Chapter 17 of the Language Reference.
MatrixManipulation Functions

MatrixModifyColumnType

The procedure MatrixModifyColumnType changes the type of a column in the matrix into either 'continuous' or 'integer'.

MatrixModifyColumnType(
    MP,  ! (input) a mathematical program
    column,  ! (input) a scalar value
    type     ! (input) a column type
)

Arguments:

    MP
        A mathematical program that was previously solved. The mathematical program should be a linear or mixed-integer linear programming model.

    column
        A scalar reference to an existing column in the matrix.

    type
        One of the column types 'continuous' or 'integer' that should be assigned to the column.

Return value:

    The function returns 1 on success, or 0 otherwise.

See also:

    Matrix manipulation routines are discussed in more detail in Chapter 17 of the Language Reference.
Matrix Modify Direction

The procedure MatrixModifyDirection changes the direction of a mathematical program to 'maximize', 'minimize' or 'none'. The direction 'none' is the instruction to the solver to find a feasible solution. If the type of the mathematical program is 'MIP' then the solver will try to find an integer feasible solution.

MatrixModifyDirection(
    MP, ! (input) a mathematical program
    direction ! (input) a direction
)

Arguments:

MP
A mathematical program that was previously solved. The mathematical program should be a linear or mixed-integer linear programming model.

type
One of the directions 'maximize', 'minimize' or 'none'.

Return value:

The function returns 1 on success, or 0 otherwise.

See also:

Matrix manipulation routines are discussed in more detail in Chapter 17 of the Language Reference.
MatrixManipulation Functions

**MatrixModifyLeftHandSide**

The procedure MatrixModifyLeftHandSide changes the left-hand-side of a row in the matrix.

```
MatrixModifyLeftHandSide(
    MP,               ! (input) a mathematical program
    row,             ! (input) a scalar value
    value            ! (input) a numerical expression
)
```

**Arguments:**

- **MP**
  - A mathematical program that was previously solved. The mathematical program should be a linear or mixed-integer linear programming model.

- **row**
  - A scalar reference to an existing ranged row in the matrix.

- **value**
  - The new value that should be assigned to the left-hand-side of the row. This value should be unequal to NA, UNDF and INF (but might be -INF).

**Remarks:**

After a call to MatrixReSolve AIMMS checks for each modified ranged row whether or not the left-hand-side value is valid, that is, the left-hand-side value should be unequal to INF.

**See also:**

The functions MatrixModifyRightHandSide, MatrixReSolve. Matrix manipulation routines are discussed in more detail in Chapter 17 of the Language Reference.
MatrixManipulationFunctions

MatrixModifyLowerBound

The procedure MatrixModifyLowerBound changes the lower bound of a column in the matrix.

MatrixModifyLowerBound(
  MP, ! (input) a mathematical program
  column, ! (input) a scalar value
  value ! (input) a numerical expression
)

Arguments:

MP
  A mathematical program that was previously solved. The mathematical program should be a linear or mixed-integer linear programming model.

row
  A scalar reference to an existing column in the matrix.

value
  The new value that should be assigned to the lower bound of the column.

Return value:

The function returns 1 on success, and 0 otherwise.

See also:

The function MatrixModifyUpperBound. Matrix manipulation routines are discussed in more detail in Chapter 17 of the Language Reference.
Matrix Modify Right Hand Side

The procedure MatrixModifyRightHandSide changes the right-hand-side of a row in the matrix.

MatrixModifyRightHandSide(
    MP, ! (input) a mathematical program
    row, ! (input) a scalar value
    value ! (input) a numerical expression
)

Arguments:

MP
A mathematical program that was previously solved. The mathematical program should be a linear or mixed-integer linear programming model.

row
A scalar reference to an existing row in the matrix; this can not be the objective row.

value
The new value that should be assigned to the right-hand-side of the row. This value should be unequal to NA and UNDF (but might be INF or -INF).

Remarks:

■ If you assign INF to the right-hand-side value of a row with type 'ranged' or 'ranged', MatrixModifyRightHandSide will not produce an error, since you might want to change the type of this row into '<=' (using MatrixModifyRowType) immediately thereafter.

■ After a call to MatrixReSolve AIMMS checks for each modified row whether or not the right-hand-side value is valid for the current row type. If the row type is '=' then the right-hand-side value should be unequal to INF and -INF; if the row type is '<=' or 'ranged' then it should be unequal to -INF.

See also:

The functions MatrixModifyLeftHandSide, MatrixModifyRowType, MatrixReSolve. Matrix manipulation routines are discussed in more detail in Chapter 17 of the Language Reference.
Matrix Manipulation Functions

MatrixModifyRowType

The procedure MatrixModifyRowType changes the type of a row in the matrix.

\[
\text{MatrixModifyRowType}(
\begin{array}{l}
\text{MP,} \quad \text{(input) a mathematical program} \\
\text{row,} \quad \text{(input) a scalar value} \\
\text{type} \quad \text{(input) a row type}
\end{array}
\)
\]

**Arguments:**

- **MP**
  A mathematical program that was previously solved. The mathematical program should be a linear or mixed-integer linear programming model.

- **row**
  A scalar reference to an existing row in the matrix; this cannot be the objective row.

- **type**
  One of the row types '<=' , '=' , '>=' or 'ranged' that should be assigned to the row.

**Remarks:**

The following examples show what happens if we change the row type into 'ranged':

- \(a(x) \leq 3\) modified into 'ranged' results in \(-\infty \leq a(x) \leq 3\)
- \(a(x) \geq 3\) modified into 'ranged' results in \(3 \leq a(x) \leq \infty\)
- \(a(x) = 3\) modified into 'ranged' results in \(3 \leq a(x) \leq 3\)

The next examples show what happens if we change the row type of a 'ranged' row:

- \(2 \leq a(x) \leq 4\) modified into '<=' results in \(a(x) \leq 4\)
- \(2 \leq a(x) \leq 4\) modified into '>=' results in \(a(x) \geq 2\)
- \(2 \leq a(x) \leq 4\) modified into '=' results in \(a(x) = 4\)

**See also:**

Matrix manipulation routines are discussed in more detail in Chapter 17 of the Language Reference.
Matrix Manipulation Functions

MatrixModifyType

The procedure MatrixModifyType changes the type of a mathematical program from 'MIP' into 'RMIP', or vice versa.

\[\text{MatrixModifyType}(\text{MP}, \text{type})\]

Arguments:

\textit{MP}  
A mathematical program that was previously solved. The mathematical program should be a linear or mixed-integer linear programming model.

\textit{type}  
One of the types 'MIP' or 'RMIP'.

Return value:

The function returns 1 on success, or 0 otherwise.

See also:

Matrix manipulation routines are discussed in more detail in Chapter 17 of the Language Reference.
**MatrixManipulation Functions**

MatrixModifyUpperBound

The procedure `MatrixModifyUpperBound` changes the upper bound of a column in the matrix.

```plaintext
Procedure MatrixModifyUpperBound(MP, ! (input) a mathematical program
column, ! (input) a scalar value
value ! (input) a numerical expression)
```

**Arguments:**

- `MP` A mathematical program that was previously solved. The mathematical program should be a linear or mixed-integer linear programming model.
- `row` A scalar reference to an existing column in the matrix.
- `value` The new value that should be assigned to the upper bound of the column.

**Return value:**

The function returns 1 on success, or 0 otherwise.

**See also:**

The function `MatrixModifyLowerBound`. Matrix manipulation routines are discussed in more detail in Chapter 17 of the Language Reference.
**MatrixReSolve**

The procedure `MatrixReSolve` instructs the solver to solve a mathematical program in its current state after being modified by using several matrix manipulation procedures.

```plaintext
MatrixReSolve(MP ! (input) a mathematical program )
```

**Arguments:**

- `MP`  
  A mathematical program that was previously solved. The mathematical program should be a linear or mixed-integer linear programming model.

**Return value:**

The function returns 1 on success, or 0 otherwise.

**Remarks:**

After a call to `MatrixReSolve` AIMMS will first check if all modifications performed by calling matrix manipulation procedures are all valid, before actually calling the solver. Most errors, however, will be filtered out by the matrix manipulation procedures themselves.

**See also:**

Matrix manipulation routines are discussed in more detail in Chapter 17 of the Language Reference.
MatrixRegenerateRow

The procedure MatrixRegenerateRow regenerates the coefficients of a row according to the definition of its associated symbolic constraint in the model.

```
MatrixRegenerateRow(  
    MP,   ! (input) a mathematical program  
    row  ! (input) a scalar value  
)
```

**Arguments:**

- **MP**
  
  A mathematical program that was previously solved. The mathematical program should be a linear or mixed-integer linear programming model.

- **row**
  
  A scalar reference to an existing row name in the model.

**Return value:**

The function returns 1 on success, or 0 otherwise.

**Remarks:**

- If the row does not exist yet, it will be automatically added to the matrix before generating its coefficients.
- Before regenerating the row, the function first removes all existing matrix coefficients.
- The row type and the right-hand-side value (and, if the row type is 'ranged', the left-hand-side value) are set according to the constraint definition.

**See also:**

The functions MatrixAddRow, MatrixModifyCoefficient, MatrixModifyLeftHandSide, MatrixModifyRightHandSide, MatrixModifyRowType. Matrix manipulation routines are discussed in more detail in Chapter 17 of the Language Reference.
Matrix Manipulation Functions

**MatrixRestoreState**

With procedure `MatrixRestoreState` you can restore the state of your mathematical program as it was on the moment that you called `MatrixSaveState`.

```plaintext
MatrixSaveState(
    MP, ! (input) a mathematical program
    state  ! (input) a state
)
```

**Arguments:**

- **MP**
  - A mathematical program that was previously solved. The mathematical program should be a linear or mixed-integer linear programming model.

- **state**
  - The value corresponding to a state that you want to restore.

**Return value:**

The function returns 1 on success, or 0 otherwise.

**See also:**

The function `MatrixSaveState`. Matrix manipulation routines are discussed in more detail in Chapter 17 of the Language Reference.
MatrixSaveState

With the procedure MatrixSaveState you can save the current state of a mathematical program. Later on, after manipulating the mathematical program, you can restore this state by calling MatrixRestoreState.

MatrixSaveState(
    MP, ! (input) a mathematical program
    state ! (output) an integer scalar parameter
)

Arguments:

    MP
        A mathematical program that was previously solved. The mathematical program should be a linear or mixed-integer linear programming model.

    state
        On return, contains a positive integer value assigned to the state.

Return value:

The function returns 1 on success, or 0 otherwise.

Remarks:

States are numbered from 1 upwards by AIMMS.

See also:

The function MatrixRestoreState. Matrix manipulation routines are discussed in more detail in Chapter 17 of the Language Reference.
MatrixUnfreezeColumn

The procedure MatrixUnfreezeColumn frees a column that was fixed with MatrixFreezeColumn. After calling MatrixUnfreezeColumn the value of the column can vary again between its lower and upper bound.

MatrixUnfreezeColumn(  
    MP, ! (input) a mathematical program  
    column ! (input) a scalar value  
)

Arguments:

MP

A mathematical program that was previously solved. The mathematical program should be a linear or mixed-integer linear programming model.

row

A scalar reference to an existing fixed column in the matrix.

Return value:

The function returns 1 on success, or 0 otherwise.

See also:

The functions MatrixFreezeColumn, MatrixModifyLowerBound, MatrixModifyUpperBound. Matrix manipulation routines are discussed in more detail in Chapter 17 of the Language Reference.
License Functions

AIMMS supports the following licensing functions:

- `AimmsRevisionString`
- `LicenseExpirationDate`
- `LicenseMaintenanceExpirationDate`
- `LicenseNumber`
- `LicenseStartDate`
- `LicenseType`
- `VARLicenseExpirationDate`
**AimmsRevisionString**

The function `AimmsRevisionString` returns the revision number of the current AIMMS executable.

```aimms
AimmsRevisionString(
    revision ! (output) a scalar string parameter
)
```

**Arguments:**

`revision`

A scalar string parameter that, on return, contains the current revision number.

**Return value:**

The function returns 1 on success, and 0 on failure.

**Remarks:**

The revision string returned by the function has the format “x.y.b” where 
x represents the major AIMMS version number (e.g. 3), y represents the minor AIMMS version number (e.g. 0), and where b represents the build number (e.g. 476) of the current executable.
LicenseExpirationDate

The function LicenseExpirationDate returns the expiration date of the current AIMMS license.

LicenseExpirationDate(
    date ! (output) a scalar string parameter
)

Arguments:

date
    A scalar string parameter that, on return, contains the expiration date of the current AIMMS license.

Return value:

The function returns 1 on success, and 0 on failure.

Remarks:

The date returned by the function has the standard date format "YYYY-MM-DD", or holds the text "No expiration date" if the current AIMMS license has no expiration date.

See also:

The functions LicenseStartDate, LicenseMaintenanceExpirationDate.
LicenseMaintenanceExpirationDate

The function LicenseMaintenanceExpirationDate returns the maintenance expiration date of the current AIMMS license.

LicenseMaintenanceExpirationDate(
    date ! (output) a scalar string parameter
)

Arguments:

date
A scalar string parameter that, on return, contains the maintenance expiration date of the current AIMMS license.

Return value:

The function returns 1 on success, and 0 on failure.

Remarks:

The date returned by the function has the standard date format "YYYY-MM-DD", or holds the text "No maintenance expiration date" if the current AIMMS license has no maintenance expiration date.

See also:

The functions LicenseStartDate, LicenseExpirationDate.
LicenseNumber

The function LicenseNumber returns the license number of the current AIMMS license.

```
LicenseNumber(  
    license ! (output) a scalar string parameter  
)
```

Arguments:

- `license`  
  A scalar string parameter that, on return, contains the current license number.

Return value:

The function returns 1 on success, and 0 on failure.

Remarks:

The function will return the license number as a string of the form “015.090.010.007” if you are using an AIMMS 3 license, or as a string of the form “1234.56” if you are using an AIMMS 2 license.

See also:

The function LicenseType.
LicenseStartDate

The function LicenseStartDate returns the start date of the current AIMMS license.

LicenseStartDate(
    date ! (output) a scalar string parameter
)

Arguments:

date
    A scalar string parameter that, on return, contains the start date of
    the current AIMMS license.

Return value:

    The function returns 1 on success, and 0 on failure.

Remarks:

    The date returned by the function has the standard date format
    "YYYY-MM-DD", or holds the text "No start date" if the current AIMMS
    license has no start date.

See also:

    The functions LicenseExpirationDate, LicenseMaintenanceExpirationDate.
LicenseType

The function LicenseType returns the type and size of the current AIMMS license.

LicenseType(
    type,   ! (output) a scalar string parameter
    size   ! (output) a scalar string parameter
)

Arguments:

  type
     A scalar string parameter that, on return, contains the type of the current license.

  size
     A scalar string parameter that, on return, contains the size of the current license.

Return value:

The function returns 1 on success, and 0 on failure.

Remarks:

Upon success, the type argument contains the license type description (e.g. "Economy") and the size argument contains a description of the license size (e.g. "Large").

See also:

The function LicenseNumber.
VARLicenseExpirationDate

The function VARLicenseExpirationDate returns the expiration date of a particular VAR license file.

    VARLicenseExpirationDate(  
        var-license ! (input) a scalar string expression  
        date ! (output) a scalar string parameter  
    )

Arguments:

    var-license
        The name of the VAR license whose expiration date you want to retrieve.

    date
        A scalar string parameter that, on return, contains the expiration date of the specified VAR license.

Return value:

    The function returns 1 on success, and 0 on failure.

Remarks:

    The date returned by the function has the standard date format "YYYY-MM-DD", or holds the text "No expiration date" if the specified VAR license has no expiration date.

See also:

    The function LicenseStartDate. VAR licensing is discussed in more detail in Section 17.1 of the User's Guide.
Aimms supports the following miscellaneous functions, that cannot easily be categorized in other sections:

- Delay
- Execute
- ExitAimms
- OpenDocument
- ProjectDeveloperMode
- ScheduleAt
- SecurityGetGroups
- SecurityGetUsers
- SessionArgument
- ShowHelpTopic
Delay

With the function Delay you can block the execution of your model for the indicated delay time. You can use this function, for instance, when you want to display intermediate results on a page using the function PageRefreshAll.

\[
\text{Delay(}
\begin{array}{c}
\text{delaytime} \\
! \text{(input) scalar expression}
\end{array}
\text{)}
\]

Arguments:

delaytime

The number of seconds that the execution should be blocked.

See also:

The function PageRefreshAll.
**Execute**

With the Execute function you can start another application.

\[
\text{Execute(}
\begin{array}{l}
\text{executable,} \quad \text{! (input) scalar string expression} \\
[\text{command-line,}] \quad \text{! (optional) scalar string expression} \\
[\text{workdir,}] \quad \text{! (optional) scalar string expression} \\
[\text{wait,}] \quad \text{! (optional) 0 or 1} \\
[\text{minimized}] \quad \text{! (optional) 0 or 1} \\
\end{array}
\]

**Arguments:**

- **executable**
  A string representing the name of the program that you want to execute.

- **command-line (optional)**
  A string representing the arguments that you want to pass to the program.

- **workdir (optional)**
  A string representing the directory where the program should start in. If omitted, then the current project directory is used.

- **wait (optional)**
  This argument indicates whether or not AIMMS will wait for the program to finish. The default value is 0 (not wait).

- **minimized (optional)**
  This argument indicates whether or not the program should run in a minimized state. The default is 0 (not minimized).

**Remarks:**

As a general rule, you should not wait for interactive windowed applications. Waiting for the termination of a program is necessary when the program does some form of external data processing which is required for the execution of your model.

**See also:**

The function [OpenDocument](#).
ExitAimms

With the function ExitAimms you can exit the current AIMMS session from within a procedure.

\[\text{ExitAimms(} \text{interactive} \text{) ! (optional) 0 or 1}\]

Arguments:

*interactive (optional)*

This optional argument specifies whether or not AIMMS should display a confirmation dialog box before closing the current project.

Remarks:

The function does not immediately exit AIMMS, but it will try to exit as soon as the execution of the current procedure has finished. If existing, the logoff procedure and the procedure MainTermination will be executed as normal.
OpenDocument

The function OpenDocument uses the current association of Windows to open documents, run programs, etc. Its functionality is similar to that of the Run command in the Start Menu of Windows. You can use it, for instance, to display an HTML file using the default web browser, open a Word document, or initiate an e-mail session.

```
OpenDocument(
    document ! (input) string expression
)
```

**Arguments:**

- `document`
  A string expression representing the document or program you want to open.

**Return value:**

The function returns 1 on success, or 0 otherwise.

**Examples:**

```
OpenDocument( "http://www.aimms.com" );
OpenDocument( "mailto:info@paragon.nl" );
OpenDocument( "anyfile.doc" );
OpenDocument( "c:\windows" );
```

**See also:**

The function Execute.
**ProjectDeveloperMode**

The function `ProjectDeveloperMode` indicates whether a project is opened in developer or end-user mode.

**Arguments:**

*None*

**Return value:**

The function returns 1 if the project is opened in developer mode, or 0 if the project is opened in end-user mode.
ScheduleAt

With the function ScheduleAt you schedule a specific procedure to be run at a specified moment in time.

```plaintext
ScheduleAt(
    starttime,  ! (input) scalar string expression
    procedure  ! (input) element of the set AllProcedures
)
```

Arguments:

- **starttime**
  A string representing the time at which you want to start the execution of the specified procedure. This time must be represented using AIMMS' standard time format: "YYYY-MM-DD hh:mm:ss".

- **procedure**
  An element in the set AllProcedures. This procedure cannot have any arguments.

Return value:

The function returns 1 on success, and 0 if AIMMS could not schedule the procedure at the specified start time. On failure, the pre-defined identifier CurrentErrorMessage will contain a proper error message.

Remarks:

If at the specified start time AIMMS is busy running some other task, then the procedure will start as soon as AIMMS has finished this task. If you want to run a procedure at regular intervals, then you can re-schedule the procedure from within the scheduled procedure itself.
SecurityGetGroups

With the function SecurityGetGroups you can fill a set with group names from the user database that is linked to the project.

SecurityGetGroups(
    group-set ! (output) an (empty) root set
)

Arguments:

group-set
   A root set, that on return will contain elements that represent all group names from the user database.

Return value:

The function returns 1 on success, and 0 on failure.

See also:

The function SecurityGetUsers.
SecurityGetUsers

With the function `SecurityGetUsers` you can fill a set with user names from the user database that is linked to the project. You can filter which users are included in the set based upon their group or authorization level.

```lisp
SecurityGetUsers(
    user-set, ! (output) an (empty) root set
    [group,] ! (optional) scalar string
    [level] ! (optional) element of the set AllAuthorizationLevels
)
```

**Arguments:**

- **user-set**
  A root set, that on return will contain elements that represent the user names from the user database.

- **group (optional)**
  A string representing a group name from the user database. If specified, then only the users that belong to this group are returned.

- **level (optional)**
  An element of the set AllAuthorizationLevels. If specified, then only the users that have the specified authorization level are returned.

**Return value:**

The function returns 1 on success, and 0 on failure.

**See also:**

The function `SecurityGetGroups`.
SessionArgument

With the function SessionArgument you can retrieve the string value of any user defined command line argument, that was specified during startup of AIMMS.

```
SessionArgument(
    argno,  ! (input) integer number
    argument  ! (output) string valued parameter
)
```

Arguments:

- **argno**
  An integer greater or equal to 1, representing the argument that you want retrieve. If the argument does not exist, then the function returns 0.

- **argument**
  A string valued parameter, to hold the string of the requested command line argument.

Return value:

The function returns 1 on success, and 0 if the request argument number does not exist.

Remarks:

When you open an AIMMS project from the command line, AIMMS allows you to add an arbitrary number of additional arguments directly after the project name. The function SessionArgument gives you access to these arguments. You can use these arguments, for instance, to specify a varying data source name from which you want to read data into your model, or run your project in different modes.
ShowHelpTopic

With the function ShowHelpTopic you can jump to a specific help topic in a help file.

```
ShowHelpTopic(  
  topic, ! (input) scalar string  
  [helpfile] ! (optional) scalar string  
)
```

Arguments:

- `topic`
  A string representing the help topic to jump to.

- `helpfile (optional)`
  A string representing the help file to open. If not specified, then AIMMS will use the help file that is specified in the project options.

Remarks:

AIMMS supports the following help file formats: WinHelp or WinHelp2000 (*.hlp), compiled HTML Help (*.chm), and Acrobat Reader (*.pdf).
Part II

Index
Abs, 3
Aggregate, 60
AimmsRevisionString, 233
ArcCos, 4
ArcCosh, 5
ArcSin, 6
ArcSinh, 7
ArcTan, 8
ArcTanh, 9
Beta, 181
Binomial, 182
Card, 35
CaseCreate, 80
CaseDelete, 81
CaseFind, 82
CaseGetChangedStatus, 83
CaseGetDatasetReference, 84
CaseGetType, 85
CaseLoadCurrent, 86
CaseLoadIntoCurrent, 87
CaseMerge, 88
CaseNew, 89
CaseSave, 90
CaseSaveAll, 91
CaseSaveAs, 92
CaseSelect, 93
CaseSelectMultiple, 94
CaseSelectNew, 95
CaseSetChangedStatus, 96
CaseSetCurrent, 97
CaseTypeCategories, 115
CaseTypeContents, 116
Ceil, 10
CloseDataSource, 75
Combination, 183
ConstraintVariables, 36
Cos, 11
Cosh, 12
CreateTimeTable, 61
CurrentToMoment, 62
CurrentToString, 63
CurrentToTimeSlot, 64
DataCategoryContents, 117
DataFileExists, 118
DataFileGetAcronym, 119
DataFileGetDescription, 120
DataFileGetGroup, 121
DataFileGetOwner, 123
DataFileGetPath, 124
DataFileGetTime, 125
DataFileReadPermitted, 126
DataFileWritePermitted, 127
DataImport220, 128
DataManagerExport, 129
DataManagerImport, 130
DatasetCreate, 99
DatasetDelete, 100
DatasetFind, 101
DatasetGetCategory, 102
DatasetGetChangedStatus, 103
DatasetLoadCurrent, 104
DatasetLoadIntoCurrent, 105
DatasetMerge, 106
DatasetNew, 107
DatasetSave, 108
DatasetSaveAs, 109
DatasetSelect, 110
DatasetSelectNew, 111
DatasetSetChangedStatus, 112
DatasetSetCurrent, 113
Degrees, 13
Delay, 241
DialogAsk, 132
DialogError, 133
DialogGetDate, 134
DialogGetElement, 136
DialogGetElementByData, 135
DialogGetElementByText, 137
DialogGetNumber, 138
DialogGetPassword, 139
DialogGetString, 140
DialogMessage, 141
DialogProgress, 142
DirectoryCopy, 162
DirectoryCreate, 163
DirectoryDelete, 164
DirectoryExists, 165
DirectoryGetCurrent, 166
DirectoryMove, 167
DirectorySelect, 168
DirectSQL, 76
DisAggregate, 65
Index

Element, 37
ElementCast, 38
ElementRange, 39
ErrorF, 14
Execute, 242
ExitAimms, 243
Exp, 15
Exponential, 184
ExtremeValue, 185
Factorial, 186
FileAppend, 169
FileCopy, 170
FileDelete, 171
FileEdit, 172
FileExists, 173
FileMove, 174
FilePrint, 175
FileSelect, 176
FileSelectNew, 177
FileTime, 178
FileView, 179
FindNthString, 50
FindString, 51
FindUsedElements, 40
Floor, 16
FormatString, 52
Gamma, 187
Geometric, 188
HistogramAddObservation, 201
HistogramCreate, 202
HistogramDelete, 203
HistogramGetAverage, 204
HistogramGetBounds, 205
HistogramGetDeviation, 206
HistogramGetFrequencies, 207
HistogramGetKurtosis, 208
HistogramGetObservationCount, 209
HistogramGetSkewness, 210
HistogramSetDomain, 211
HyperGeometric, 189
LicenseExpirationDate, 234
LicenseMaintenanceExpirationDate, 235
LicenseNumber, 236
LicenseStartDate, 237
LicenseType, 238
Log, 17
Log10, 18
Logistic, 190
LogNormal, 191
MapVal, 19
MatrixDeactivateRow, 216
MatrixFreezeColumn, 217
MatrixModifyCoefficient, 218
MatrixModifyColumnType, 219
MatrixModifyDirection, 220
MatrixModifyLeftHandSide, 221
MatrixModifyLowerBound, 222
MatrixModifyRightHandSide, 223
MatrixModifyRowType, 224
MatrixModifyType, 225
MatrixReSolve, 227
MatrixRestoreState, 229
MatrixSaveState, 230
MatrixUnfreezeColumn, 231
Max, 20
Min, 21
Mod, 22
MomentToString, 66
MomentToTimeSlot, 67
NegativeBinomial, 192
Normal, 193
OpenDocument, 244
Ord, 41
PageClose, 145
PageGetActive, 146
PageGetChild, 147
PageGetFirstChild, 148
PageGetNext, 149
PageGetParent, 150
PageGetPrevious, 151
PageOpen, 152
PageOpenSingle, 153
PageRefreshAll, 154
PageSetCursor, 155
PageSetFocus, 156
Pareto, 194
PeriodToString, 68
Permutation, 195
Poisson, 196
Power, 23
PrintEndReport, 157
PrintPage, 158
PrintPageCount, 159
PrintStartReport, 160
ProjectDeveloperMode, 245
Radians, 24
RestoreInactiveElements, 42
RetrieveCurrentVariableValues, 43
Round, 25
ScheduleAt, 246
SecurityGetGroups, 247
Index

SecurityGetUsers, 248
SessionArgument, 249
SetElementAdd, 44
SetElementRename, 45
ShowHelpTopic, 250
Sign, 26
Sin, 27
Sinh, 28
Sqr, 29
Sqrt, 30
StatusMessage, 143
StringCapitalize, 53
StringLength, 54
StringOccurrences, 55
StringToElement, 46
StringToLower, 56
StringToMoment, 69
StringToTimeSlot, 70
StringToUpper, 57
SubRange, 47
SubString, 58
Tan, 31
Tanh, 32
TestDatabaseTable, 77
TestDataSource, 78
TimeSlotCharacteristic, 71
TimeSlotToMoment, 72
TimeSlotToString, 73
Triangular, 197
Trunc, 33

Uniform, 198

VariableConstraints, 48
VARIABLELicenseExpirationDate, 239

Weibull, 199