Reliance 4

SCRIPTS
# Table of Contents

## 1 Introduction

## 2 VBScript Language Reference

### 2.1 Syntax of procedure and function calls

### 2.2 Working with properties and methods of objects

### 2.3 Data Type Functions

#### 2.3.1 CBool Function

#### 2.3.2 CByte Function

#### 2.3.3 CCur Function

#### 2.3.4 CDate Function

#### 2.3.5 CDbi Function

#### 2.3.6 CInt Function

#### 2.3.7 CLng Function

#### 2.3.8 CSng Function

#### 2.3.9 CStr Function

#### 2.3.10 Fix Function

#### 2.3.11 Int Function

#### 2.3.12 IsArray Function

#### 2.3.13 IsDate Function

#### 2.3.14 IsEmpty Function

#### 2.3.15 IsNull Function

#### 2.3.16 IsNumeric Function

#### 2.3.17 IsObject Function

#### 2.3.18 TypeName Function

#### 2.3.19 VarType Function

### 2.4 Date and Time Functions

#### 2.4.1 Date Function

#### 2.4.2 DateAdd Function

#### 2.4.3 DateDiff Function

#### 2.4.4 DatePart Function

#### 2.4.5 DateSerial Function

#### 2.4.6 DateValue Function

#### 2.4.7 Day Function
<table>
<thead>
<tr>
<th>Section</th>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.4.8</td>
<td>Hour Function</td>
<td>36</td>
</tr>
<tr>
<td>2.4.9</td>
<td>Minute Function</td>
<td>37</td>
</tr>
<tr>
<td>2.4.10</td>
<td>Month Function</td>
<td>38</td>
</tr>
<tr>
<td>2.4.11</td>
<td>MonthName Function</td>
<td>38</td>
</tr>
<tr>
<td>2.4.12</td>
<td>Now Function</td>
<td>39</td>
</tr>
<tr>
<td>2.4.13</td>
<td>Second Function</td>
<td>39</td>
</tr>
<tr>
<td>2.4.14</td>
<td>Time Function</td>
<td>40</td>
</tr>
<tr>
<td>2.4.15</td>
<td>Timer Function</td>
<td>41</td>
</tr>
<tr>
<td>2.4.16</td>
<td>TimeSerial Function</td>
<td>41</td>
</tr>
<tr>
<td>2.4.17</td>
<td>TimeValue Function</td>
<td>42</td>
</tr>
<tr>
<td>2.4.18</td>
<td>Weekday Function</td>
<td>43</td>
</tr>
<tr>
<td>2.4.19</td>
<td>WeekdayName Function</td>
<td>45</td>
</tr>
<tr>
<td>2.4.20</td>
<td>Year Function</td>
<td>46</td>
</tr>
<tr>
<td>2.5</td>
<td>Array Functions</td>
<td>48</td>
</tr>
<tr>
<td>2.5.1</td>
<td>Array Function</td>
<td>48</td>
</tr>
<tr>
<td>2.5.2</td>
<td>Dim Statement</td>
<td>49</td>
</tr>
<tr>
<td>2.5.3</td>
<td>Erase Statement</td>
<td>50</td>
</tr>
<tr>
<td>2.5.4</td>
<td>Filter Function</td>
<td>51</td>
</tr>
<tr>
<td>2.5.5</td>
<td>IsArray Function</td>
<td>53</td>
</tr>
<tr>
<td>2.5.6</td>
<td>Join Function</td>
<td>53</td>
</tr>
<tr>
<td>2.5.7</td>
<td>LBound Function</td>
<td>54</td>
</tr>
<tr>
<td>2.5.8</td>
<td>Private Statement</td>
<td>55</td>
</tr>
<tr>
<td>2.5.9</td>
<td>Public Statement</td>
<td>56</td>
</tr>
<tr>
<td>2.5.10</td>
<td>ReDim Statement</td>
<td>57</td>
</tr>
<tr>
<td>2.5.11</td>
<td>Split Function</td>
<td>59</td>
</tr>
<tr>
<td>2.5.12</td>
<td>UBound Function</td>
<td>60</td>
</tr>
<tr>
<td>2.6</td>
<td>String Functions</td>
<td>62</td>
</tr>
<tr>
<td>2.6.1</td>
<td>Asc Function</td>
<td>63</td>
</tr>
<tr>
<td>2.6.2</td>
<td>Chr Function</td>
<td>63</td>
</tr>
<tr>
<td>2.6.3</td>
<td>Escape Function</td>
<td>64</td>
</tr>
<tr>
<td>2.6.4</td>
<td>FormatCurrency Function</td>
<td>65</td>
</tr>
<tr>
<td>2.6.5</td>
<td>FormatDateTime Function</td>
<td>67</td>
</tr>
<tr>
<td>2.6.6</td>
<td>FormatNumber Function</td>
<td>68</td>
</tr>
<tr>
<td>2.6.7</td>
<td>FormatPercent Function</td>
<td>70</td>
</tr>
<tr>
<td>2.6.8</td>
<td>InStr Function</td>
<td>72</td>
</tr>
<tr>
<td>2.6.9</td>
<td>InStrRev Function</td>
<td>74</td>
</tr>
<tr>
<td>2.6.10</td>
<td>LCase Function</td>
<td>75</td>
</tr>
<tr>
<td>2.6.11</td>
<td>Left Function</td>
<td>76</td>
</tr>
<tr>
<td>2.6.12</td>
<td>Len Function</td>
<td>77</td>
</tr>
</tbody>
</table>
### Conversion Functions

<table>
<thead>
<tr>
<th>Section</th>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7.1</td>
<td>Asc Function</td>
<td>93</td>
</tr>
<tr>
<td>2.7.2</td>
<td>CBool Function</td>
<td>93</td>
</tr>
<tr>
<td>2.7.3</td>
<td>CByte Function</td>
<td>94</td>
</tr>
<tr>
<td>2.7.4</td>
<td>CCur Function</td>
<td>95</td>
</tr>
<tr>
<td>2.7.5</td>
<td>CDate Function</td>
<td>96</td>
</tr>
<tr>
<td>2.7.6</td>
<td>CDbl Function</td>
<td>97</td>
</tr>
<tr>
<td>2.7.7</td>
<td>Chr Function</td>
<td>98</td>
</tr>
<tr>
<td>2.7.8</td>
<td>Clnt Function</td>
<td>99</td>
</tr>
<tr>
<td>2.7.9</td>
<td>Clng Function</td>
<td>100</td>
</tr>
<tr>
<td>2.7.10</td>
<td>CSng Function</td>
<td>101</td>
</tr>
<tr>
<td>2.7.11</td>
<td>CStr Function</td>
<td>102</td>
</tr>
<tr>
<td>2.7.12</td>
<td>DateSerial Function</td>
<td>103</td>
</tr>
<tr>
<td>2.7.13</td>
<td>DateValue Function</td>
<td>104</td>
</tr>
<tr>
<td>2.7.14</td>
<td>Day Function</td>
<td>105</td>
</tr>
<tr>
<td>2.7.15</td>
<td>Fix Function</td>
<td>106</td>
</tr>
<tr>
<td>2.7.16</td>
<td>Hex Function</td>
<td>107</td>
</tr>
<tr>
<td>2.7.17</td>
<td>Hour Function</td>
<td>108</td>
</tr>
<tr>
<td>2.7.18</td>
<td>Int Function</td>
<td>108</td>
</tr>
<tr>
<td>2.7.19</td>
<td>LCase Function</td>
<td>109</td>
</tr>
<tr>
<td>2.7.20</td>
<td>Minute Function</td>
<td>110</td>
</tr>
<tr>
<td>2.7.21</td>
<td>Month Function</td>
<td>110</td>
</tr>
<tr>
<td>2.7.22</td>
<td>Oct Function</td>
<td>111</td>
</tr>
<tr>
<td>2.7.23</td>
<td>Second Function</td>
<td>112</td>
</tr>
<tr>
<td>2.7.24</td>
<td>TimeSerial Function</td>
<td>112</td>
</tr>
</tbody>
</table>
2.10.3 VBScript Statements

2.8 Math Functions

2.9 Miscellaneous Functions

2.10 VBScript Statements
2.10.5 Erase Statement ................................................................. 154
2.10.6 Execute Statement ............................................................. 155
2.10.7 Exit Statement ................................................................. 156
2.10.8 For Each...Next Statement ............................................... 158
2.10.9 For...Next Statement ....................................................... 159
2.10.10 Function Statement ......................................................... 161
2.10.11 If...Then...Else Statement .............................................. 164
2.10.12 On Error Statement .......................................................... 166
2.10.13 Option Explicit Statement .............................................. 167
2.10.14 Private Statement ............................................................ 168
2.10.15 Public Statement .............................................................. 169
2.10.16 Randomize Statement ..................................................... 170
2.10.17 ReDim Statement ............................................................. 171
2.10.18 Rem Statement ............................................................... 172
2.10.19 Select Case Statement ..................................................... 173
2.10.20 Set Statement ................................................................. 175
2.10.21 Stop Statement ............................................................... 178
2.10.22 Sub Statement ............................................................... 178
2.10.23 While...WEnd Statement ................................................. 180
2.10.24 With Statement ............................................................... 182

2.11 VBScript Constants ............................................................ 184
  2.11.1 Color Constants ............................................................. 184
  2.11.2 Comparison Constants .................................................. 185
  2.11.3 Date and Time Constants ............................................... 185
  2.11.4 Date Format Constants .................................................. 186
  2.11.5 Miscellaneous Constants .............................................. 187
  2.11.6 MsgBox Constants ......................................................... 187
  2.11.7 String Constants ........................................................... 189
  2.11.8 Tristate Constants ........................................................ 190
  2.11.9 VarType Constants ......................................................... 190

2.12 VBScript Operators ........................................................... 192
  2.12.1 Addition Operator (+) .................................................... 192
  2.12.2 And Operator .............................................................. 193
  2.12.3 Assignment Operator .................................................... 194
  2.12.4 Concatenation Operator (&) .......................................... 195
  2.12.5 Division Operator (/) ...................................................... 196
  2.12.6 Eqv Operator ............................................................... 196
  2.12.7 Exponentiation Operator (^) ......................................... 198
  2.12.8 Imp Operator ............................................................... 198
## Table of Contents

2.12.9 Integer Division Operator (\) ................................................................. 200
2.12.10 Is Operator .................................................................................. 200
2.12.11 Mod Operator ............................................................................ 201
2.12.12 Multiplication Operator (*) ............................................................ 202
2.12.13 Negation Operator (-) ................................................................. 202
2.12.14 Not Operator .............................................................................. 203
2.12.15 Or Operator .............................................................................. 204
2.12.16 Subtraction Operator (-) ............................................................... 205
2.12.17 Xor Operator ............................................................................. 206

2.13 Keywords .......................................................................................... 208
2.13.1 Keywords ..................................................................................... 208

2.14 Resources ......................................................................................... 209
2.14.1 Locale ID (LCID) Chart ............................................................... 209
2.14.2 Errors .......................................................................................... 209

3 Reliance-defined Objects ........................................................................ 211
3.1 Reliance-defined Objects .................................................................... 211
3.2 Execution of Scripts in the Runtime Environment ............................... 212
3.3 Processing of Data Passed to Scripts from the Runtime Environment .......... 213
3.4 Working with Global Constants, Variables, Procedures and Functions ......... 214
3.5 Tips for Writing Scripts ...................................................................... 215
3.6 RAlm Object ..................................................................................... 216
3.6.1 RAlm.AckAlarm Procedure ............................................................ 216
3.6.2 RAlm.AckAllAlarms Procedure ...................................................... 217
3.6.3 RAlm.CreateAlarm Procedure ..................................................... 217
3.6.4 RAlm.CurrentAlarms Procedure .................................................. 219
3.6.5 RAlm.CurrentAlarmsByDevice Procedure .................................... 220
3.6.6 RAlm.CurrentAlarmsEx Procedure ............................................... 221
3.6.7 RAlm.DbAlarms Procedure ............................................................ 222
3.6.8 RAlm.DbAlarmsByFilter Procedure ............................................ 223
3.6.9 RAlm.DbAlarmsByDevice Procedure ........................................... 224
3.6.10 RAlm.DbAlarmsEx Procedure ....................................................... 225
3.6.11 RAlm.DisableDeviceAlarms Procedure ....................................... 226
3.6.12 RAlm.EnableDeviceAlarms Procedure ......................................... 226
3.6.13 Alarm Triggering Condition Constants ......................................... 227

3.7 RConst Object .................................................................................... 229
3.7.1 RConst.ProjObjType Property ...................................................... 229
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.8</td>
<td>RDb Object</td>
<td>234</td>
</tr>
<tr>
<td>3.8.1</td>
<td>RDb.AppendRecord Procedure</td>
<td>234</td>
</tr>
<tr>
<td>3.8.2</td>
<td>RDb.CreateTableObject Function</td>
<td>235</td>
</tr>
<tr>
<td>3.8.3</td>
<td>RDb.GetTagHistValue Function</td>
<td>236</td>
</tr>
<tr>
<td>3.8.4</td>
<td>RDb.GetTagStatistics Procedure</td>
<td>238</td>
</tr>
<tr>
<td>3.9</td>
<td>RDev Object</td>
<td>240</td>
</tr>
<tr>
<td>3.9.1</td>
<td>RDev.ConnectToCommDriver Procedure</td>
<td>240</td>
</tr>
<tr>
<td>3.9.2</td>
<td>RDev.SendCustomData Procedure</td>
<td>241</td>
</tr>
<tr>
<td>3.9.3</td>
<td>RDev.RDev.ReceiveCustomDataReply Procedure</td>
<td>242</td>
</tr>
<tr>
<td>3.10</td>
<td>RError Object</td>
<td>244</td>
</tr>
<tr>
<td>3.10.1</td>
<td>RError.Code Property</td>
<td>244</td>
</tr>
<tr>
<td>3.10.2</td>
<td>RError.Description Property</td>
<td>245</td>
</tr>
<tr>
<td>3.10.3</td>
<td>The List of Reliance-defined Objects Error Codes</td>
<td>245</td>
</tr>
<tr>
<td>3.11</td>
<td>RInet Object</td>
<td>254</td>
</tr>
<tr>
<td>3.11.1</td>
<td>RInet.SendMail Function</td>
<td>254</td>
</tr>
<tr>
<td>3.12</td>
<td>RModem Object</td>
<td>256</td>
</tr>
<tr>
<td>3.12.1</td>
<td>RModem.GSMSendATCommand Function</td>
<td>256</td>
</tr>
<tr>
<td>3.12.2</td>
<td>RModem.GSMGetSMSStatus Function</td>
<td>257</td>
</tr>
<tr>
<td>3.12.3</td>
<td>RModem.GSMSendSMS Function</td>
<td>259</td>
</tr>
<tr>
<td>3.12.4</td>
<td>RModem.GSMSendSMSEx Function</td>
<td>260</td>
</tr>
<tr>
<td>3.12.5</td>
<td>The List of Error Codes (CMS) According to GSM 07.05 Standard</td>
<td>261</td>
</tr>
<tr>
<td>3.13</td>
<td>RProj Object</td>
<td>265</td>
</tr>
<tr>
<td>3.13.1</td>
<td>RProj.GetObject Function</td>
<td>265</td>
</tr>
<tr>
<td>3.13.2</td>
<td>RProj.GetObjectList Function</td>
<td>266</td>
</tr>
<tr>
<td>3.13.3</td>
<td>RProj.ObjectExists Function</td>
<td>269</td>
</tr>
<tr>
<td>3.14</td>
<td>RScr Object</td>
<td>271</td>
</tr>
<tr>
<td>3.14.1</td>
<td>RScr.DisableScript Procedure</td>
<td>272</td>
</tr>
<tr>
<td>3.14.2</td>
<td>RScr.EnableScript Procedure</td>
<td>273</td>
</tr>
<tr>
<td>3.14.3</td>
<td>RScr.ExecScript Procedure</td>
<td>274</td>
</tr>
<tr>
<td>3.14.4</td>
<td>RScr.ExecScriptForThinClient Procedure</td>
<td>275</td>
</tr>
<tr>
<td>3.14.6</td>
<td>RScr.GetCurrentScriptData Function</td>
<td>277</td>
</tr>
<tr>
<td>3.14.7</td>
<td>RScr.GetCurrentScriptDataEx Function</td>
<td>279</td>
</tr>
<tr>
<td>3.14.8</td>
<td>RScr.GetCurrentThreadName Function</td>
<td>284</td>
</tr>
<tr>
<td>3.14.9</td>
<td>RScr.GetScriptInfo Function</td>
<td>285</td>
</tr>
<tr>
<td>3.14.10</td>
<td>RScr.GetScriptText Function</td>
<td>286</td>
</tr>
<tr>
<td>3.14.11</td>
<td>RScr.ResetCurrentScriptUser Procedure</td>
<td>287</td>
</tr>
<tr>
<td>3.14.12</td>
<td>RScr.SetCurrentScriptUser Procedure</td>
<td>288</td>
</tr>
</tbody>
</table>
3.14.13 Basic Events .................................................................................................................. 289
3.14.14 Events Triggered by a Component .................................................................................. 289
3.14.15 Events Triggered by a Change in Tag Value, Quality, or Time Stamp .................. 291
3.14.16 Events Triggered by an Alarm ...................................................................................... 293
3.14.17 Events Triggered by a Text Message ............................................................................ 295
3.14.18 Events Triggered by a Thin Client Request ................................................................. 296
3.14.19 Events Triggered by the MatriX Service ....................................................................... 297
3.14.20 Events Triggered by an Embedded Web Browser ......................................................... 298
3.15 RSys Object .......................................................................................................................... 301
3.15.1 RSys.ActivateWindow Procedure ................................................................................. 303
3.15.2 RSys.CloseWindow Procedure ...................................................................................... 304
3.15.3 RSys.ConvertTimeToDST Function .............................................................................. 304
3.15.4 RSys.CopyFile Function ............................................................................................... 305
3.15.5 RSys.CreateDir Function ............................................................................................ 307
3.15.6 RSys.DateTimeToInt64Time Function ......................................................................... 308
3.15.7 RSys.DeleteFile Function ............................................................................................. 309
3.15.8 RSys.DirExists Function ............................................................................................ 310
3.15.9 RSys.EditCustomReport Procedure ........................................................................... 311
3.15.10 RSys.ExecAction Procedure ...................................................................................... 312
3.15.11 RSys.ExecActionAndWait Function ............................................................................ 312
3.15.12 RSys.ExecApp Procedure ........................................................................................... 314
3.15.13 RSys.ExecAppAndWait Function .................................................................................. 315
3.15.14 RSys.ExitRuntimeModule Procedure ....................................................................... 316
3.15.15 RSys.FileExists Function .......................................................................................... 317
3.15.16 RSys.GetBit Function ................................................................................................. 318
3.15.17 RSys.GetByte Function .............................................................................................. 319
3.15.18 RSys.GetWord Function ............................................................................................. 320
3.15.19 RSys.GetComputerName Function ............................................................................. 321
3.15.20 RSys.GetProgramLanguage Function ....................................................................... 321
3.15.21 RSys.GetProjectDir Function ..................................................................................... 323
3.15.22 RSys.GetProjectLanguage Function ........................................................................... 323
3.15.23 RSys.GetUTCDateTime Function .................................................................................. 324
3.15.24 RSys.Int64TimeToDateTime Function ........................................................................ 325
3.15.25 RSys.IsActiveServer ................................................................................................. 326
3.15.26 RSys.IsPrimaryServer ............................................................................................... 327
3.15.27 RSys.IsSecondaryServer ........................................................................................... 328
3.15.28 RSys.LocalDateTimeToUTCDateTime Function ........................................................... 329
3.15.29 RSys.LogMessage Procedure .................................................................................... 329
3.15.30 RSys.Now Function ..................................................................................................... 330
3.15.31 RSys.PlaySound Procedure ................................................................. 331
3.15.32 RSys.PrintCustomReport Procedure ............................................... 332
3.15.33 RSys.PrintDbReport Procedure ......................................................... 332
3.15.34 RSys.PrintDbTrend Procedure ............................................................ 333
3.15.35 RSys.PrintTagDbTrend Procedure ....................................................... 334
3.15.36 RSys.PathToRelativePath Function .................................................... 334
3.15.37 RSys.RelativePathToPath Function .................................................... 336
3.15.38 RSys.RemoveDir Function .................................................................. 337
3.15.39 RSys.RenameFile Function ................................................................. 338
3.15.40 RSys.ReplaceCZChars Function .......................................................... 339
3.15.41 RSys.RestartProject Procedure ........................................................ 340
3.15.42 RSys.RestartWindows Procedure ..................................................... 342
3.15.43 RSys.SaveCustomReport Procedure ................................................ 342
3.15.44 RSys.SetBit Procedure ...................................................................... 343
3.15.45 RSys.SetByte Procedure .................................................................. 344
3.15.46 RSys.SetWord Procedure .................................................................. 345
3.15.47 RSys.SetLocalTime Function ............................................................. 346
3.15.48 RSys.SetMainWindowTitle Procedure .............................................. 347
3.15.49 RSys.SetMonitorPower Procedure ................................................... 348
3.15.50 RSys.ShiftLeft Function .................................................................... 349
3.15.51 RSys.ShiftRight Function .................................................................. 350
3.15.52 RSys.ShowCustomReport Procedure .............................................. 350
3.15.53 RSys.ShowDbReport Procedure ............................................................ 351
3.15.54 RSys.ShowDbTrend Procedure ............................................................ 352
3.15.55 RSys.ShowTagDbReport Procedure .................................................. 352
3.15.56 RSys.ShowTagDbTrend Procedure ...................................................... 353
3.15.57 RSys.ShutDownWindows Procedure .............................................. 354
3.15.58 RSys.SetProgramLanguage Procedure ........................................ 354
3.15.59 RSys.SetProjectLanguage Procedure ............................................. 355
3.15.60 RSys.Sleep Procedure ....................................................................... 356
3.15.61 RSys.UTCDateTimeToLocalDateTime Function .................................... 356

3.16 TTable-type Objects .................................................................................. 358
3.16.1 TTable_archiveName Property ............................................................. 359
3.16.2 TTable.DatabaseName Property ............................................................ 360
3.16.3 TTable.DateFieldValue Property ........................................................ 360
3.16.4 TTable.IsArchive Property .................................................................. 362
3.16.5 TTable.TimeFieldValue Property ........................................................ 363
3.16.6 TTable.Append Procedure .................................................................. 364
3.16.7 TTable.Bof Function ........................................................................... 365
3.17 RTag Object .................................................................................................................... 387

3.17.1 RTag.DecTagValue Procedure ......................................................................................... 388
3.17.2 RTag.GetBit Function ....................................................................................................... 389
3.17.3 RTag.GetByte Function .................................................................................................... 390
3.17.4 RTag.GetTagElementValue Function ............................................................................... 391
3.17.5 RTag.GetTagElementValueAsStr Function ......................................................................... 392
3.17.6 RTag.GetTagValue Function ............................................................................................ 393
3.17.7 RTag.GetTagValueAsStr Function ..................................................................................... 394
3.17.8 RTag.GetWord Function ................................................................................................... 395
3.17.9 RTag.IncTagValue Procedure .......................................................................................... 396
3.17.10 RTag.MoveTagElementValues Procedure ...................................................................... 397
3.17.11 RTag.MoveTagElementValuesToSimpleTag Procedure .................................................. 398
3.17.12 RTag.MoveTagValue Procedure ..................................................................................... 399
3.17.13 RTag.MoveTagValueToArrayTag Procedure .................................................................... 400
3.17.14 RTag.SetBit Procedure .................................................................................................. 401
3.17.15 RTag.SetByte Procedure ............................................................................................... 402
3.17.16 RTag.SetTagElementValue Procedure .......................................................................... 403
3.17.17 RTag.SetTagElementValues Procedure ........................................................................ 404
3.17.18 RTag.SetTagValue Procedure ....................................................................................... 405

3.16.8 TTable.Cancel Procedure .................................................................................................. 366
3.16.9 TTable.CloseTable Procedure ........................................................................................ 367
3.16.10 TTable.CreateTable Function ......................................................................................... 368
3.16.11 TTable.Delete Procedure ............................................................................................... 369
3.16.12 TTable.DeleteTable Function ......................................................................................... 370
3.16.13 TTable.Edit Procedure ................................................................................................... 371
3.16.14 TTable.EmptyTable Function ........................................................................................ 372
3.16.15 TTable.Eof Function ...................................................................................................... 373
3.16.16 TTable.FieldExists Function ......................................................................................... 374
3.16.17 TTable.First Procedure .................................................................................................. 375
3.16.18 TTable.GetFieldValue Function ....................................................................................... 376
3.16.19 TTable.Last Procedure ................................................................................................ 377
3.16.20 TTable.MoveBy Procedure ............................................................................................ 378
3.16.21 TTable.Next Procedure .................................................................................................. 379
3.16.22 TTable.OpenTable Function .......................................................................................... 380
3.16.23 TTable.Post Procedure .................................................................................................. 381
3.16.24 TTable.Prior Procedure ................................................................................................ 382
3.16.25 TTable.SetFieldValue Procedure .................................................................................. 383
3.16.26 TTable.TableExists Function ...................................................................................... 384
3.16.27 TTable.UpdateTableStructure Procedure .................................................................... 385

3.17.1 RTag.DecTagValue Procedure ......................................................................................... 388
3.17.2 RTag.GetBit Function ....................................................................................................... 389
3.17.3 RTag.GetByte Function .................................................................................................... 390
3.17.4 RTag.GetTagElementValue Function ............................................................................... 391
3.17.5 RTag.GetTagElementValueAsStr Function ......................................................................... 392
3.17.6 RTag.GetTagValue Function ............................................................................................ 393
3.17.7 RTag.GetTagValueAsStr Function ..................................................................................... 394
3.17.8 RTag.GetWord Function ................................................................................................... 395
3.17.9 RTag.IncTagValue Procedure .......................................................................................... 396
3.17.10 RTag.MoveTagElementValues Procedure ...................................................................... 397
3.17.11 RTag.MoveTagElementValuesToSimpleTag Procedure .................................................. 398
3.17.12 RTag.MoveTagValue Procedure ..................................................................................... 399
3.17.13 RTag.MoveTagValueToArrayTag Procedure .................................................................... 400
3.17.14 RTag.SetBit Procedure .................................................................................................. 401
3.17.15 RTag.SetByte Procedure ............................................................................................... 402
3.17.16 RTag.SetTagElementValue Procedure .......................................................................... 403
3.17.17 RTag.SetTagElementValues Procedure ........................................................................ 404
3.17.18 RTag.SetTagValue Procedure ....................................................................................... 405
| 3.17.19 | RTag.SetWord Procedure | 407 |
| 3.17.20 | RTag.ShiftLeft Procedure | 408 |
| 3.17.21 | RTag.ShiftRight Procedure | 409 |
| 3.17.22 | RTag.TagExists Function | 410 |
| 3.17.23 | RTag.UpdateTagValue Procedure | 411 |
| 3.18   | RUser Object | 412 |
| 3.18.1 | RUser.CheckUserAccessRights Function | 412 |
| 3.18.2 | RUser.CheckUserPassword | 414 |
| 3.18.3 | RUser.GetLoggedOnUserName Function | 415 |
| 3.18.4 | RUser.GetUserAlarmGroups Function | 416 |
| 3.18.5 | RUser.GetUserCode Function | 417 |
| 3.18.6 | RUser.GetUserID Function | 418 |
| 3.18.7 | RUser.IsUserAdmin Function | 419 |
| 3.18.8 | RUser.LogOffUser Procedure | 420 |
| 3.18.9 | RUser.LogOnUser Procedure | 420 |
| 3.18.10 | RUser.LogOnUserWithCode Function | 421 |
| 3.18.11 | RUser.LogOnUserWithNameAndPassword Function | 422 |
| 3.18.12 | RUser.UserExists Function | 423 |
| 3.19   | RWS Object | 424 |
| 3.19.1 | RWS.GetThinClientList Procedure | 424 |
1 Introduction

About Visual Basic Script

Visual Basic Script, or VBScript for short, is a scripting language developed by Microsoft. The code written using a scripting language is interpreted when executed. This is in contrast to programming languages where the code must be compiled and linked before it can be executed. VBScript is designed for writing scripts for programs running on the Windows operating systems.

About this document

This document contains an introduction to VBScript, provides help on the basic procedures and functions and points out some basic syntax rules. For detailed information on VBScript, see the original help.

In addition to help on VBScript, this document also contains detailed information on Reliance-defined objects, which enable you to access the Reliance runtime environment from scripts.

Information on VBScript and Reliance is also available on the Internet at:

www.reliance-scada.com
2 VBScript Language Reference

2.1 Syntax of procedure and function calls

A function is a routine that returns a value when it executes. A procedure is a routine that does not return a value. In VBScript, there is a syntax difference between calling a procedure and function.

When calling a procedure with parameters, the parameters cannot be enclosed in parentheses.

When calling a function with parameters, the parameters cannot be enclosed in parentheses if the return value is not processed. Otherwise, the parameters must be enclosed in parentheses.

For detailed information on procedures and functions in VBScript, see the original help.

Example

```vbnet
Dim ArrayOfNumbers(10)
Dim Response
' A procedure, the parameters are not enclosed in parentheses.
Erase ArrayOfNumbers
' The return value will be processed,
' the parameters are enclosed in parentheses.
Response = MsgBox("Continue?", vbYesNo, "Confirm")
If Response = vbYes Then
  ' ...
Else
  ' ...
End If
' The return value will not be processed,
' the parameters are not enclosed in parentheses.
MsgBox "Finished.", vbOKOnly, "Information"
```
2.2 Working with properties and methods of objects

**VBScript** enables you to work with objects. Using a reference to an object, it is possible to access properties and call methods (procedures and functions) of the object.

**Syntax**

```
MyObject.Function1
```

When accessing properties of an object, separate the object reference and the name of the property by a period. When calling a method of an object, separate the object reference and the name of the method by a period. Moreover, a method call must comply with the syntax of procedure and function calls.

For detailed information on objects in **VBScript**, see the original help.

**Example**

```vbscript
Dim fso, MyFile
' Create an object for working with files.
Set fso = CreateObject("Scripting.FileSystemObject")
' Create the text file C:\testfile.txt
' by calling a method, which returns another object.
Set MyFile = fso.CreateTextFile("C:\testfile.txt", True)
' Write a single line to the file by calling the WriteLine method.
MyFile.WriteLine("This is a test. ")
MyFile.Close ' Close the file by calling the Close method.
Set MyFile = Nothing
Set fso = Nothing
```
2.3 Data Type Functions

- CBool Function
- CByte Function
- CCur Function
- CDate Function
- CDbI Function
- CInt Function
- CLng Function
- CSng Function
- CStr Function
- Fix Function
- Int Function
- IsArray Function
- IsDate Function
-IsEmpty Function
- IsNull Function
- IsNumeric Function
- IsObject Function
- TypeName Function
- VarType Function

2.3.1 CBool Function

Returns an expression that has been converted to a Variant of subtype Boolean.

Syntax

\[
\text{CBool}(Expression)
\]

The Expression argument is any valid expression.
Data Type Functions

Remarks

If Expression is zero, False is returned; otherwise, True is returned. If Expression can't be interpreted as a numeric value, a run-time error occurs.

Example

The following example uses the CBool function to convert an expression to a Boolean. If the expression evaluates to a nonzero value, CBool returns True; otherwise, it returns False.

```vba
Dim A, B, Check
A = 5
B = 5 ' Initialize variables.
Check = CBool(A = B) ' Check contains True.
A = 0 ' Define variable.
Check = CBool(A) ' Check contains False.
```

2.3.2 CByte Function

Returns an expression that has been converted to a Variant of subtype Byte.

Syntax

```
CByte(Expression)
```

The Expression argument is any valid expression.

Remarks

In general, you can document your code using the subtype conversion functions to show that the result of some operation should be expressed as a particular data type rather than the default data type. For example, use CByte to force byte arithmetic in cases where currency, single-precision, double-precision, or integer arithmetic normally would occur.

Use the CByte function to provide internationally aware conversions from any other data type to a Byte subtype. For example, different decimal separators are properly recognized depending on the locale setting of your system, as are different thousand separators.
If *Expression* lies outside the acceptable range for the Byte subtype, an error occurs.

### Example

The following example uses the **CByte** function to convert an expression to a byte.

```vba
Dim MyDouble, MyByte
MyDouble = 125.5678  ' MyDouble is a Double.
MyByte = CByte(MyDouble)  ' MyByte contains 126.
```

### 2.3.3 **CCur** Function

Returns an expression that has been converted to a Variant of subtype **Currency**.

**Syntax**

```
CCur(Expression)
```

The *Expression* argument is any valid expression.

**Remarks**

In general, you can document your code using the subtype conversion functions to show that the result of some operation should be expressed as a particular data type rather than the default data type. For example, use **CCur** to force currency arithmetic in cases where integer arithmetic normally would occur.

You should use the **CCur** function to provide internationally aware conversions from any other data type to a Currency subtype. For example, different decimal separators and thousands separators are properly recognized depending on the locale setting of your system.
Example

The following example uses the **CCur** function to convert an expression to a Currency.

```vba
Dim MyDouble, MyCurr
MyDouble = 543.214588 ' MyDouble is a Double.
' Convert result of MyDouble * 2 (1086.429176) to a Currency (1086.4292).
MyCurr = CCur(MyDouble * 2)
```

### 2.3.4 CDate Function

Returns an expression that has been converted to a **Variant** of subtype **Date**.

**Syntax**

```vba
CDate(Date)
```

The **Date** argument is any valid date expression.

**Remarks**

Use the **IsDate** function to determine if **Date** can be converted to a date or time. **CDate** recognizes date literals and time literals as well as some numbers that fall within the range of acceptable dates. When converting a number to a date, the whole number portion is converted to a date. Any fractional part of the number is converted to a time of day, starting at midnight.

**CDate** recognizes date formats according to the locale setting of your system. The correct order of day, month, and year may not be determined if it is provided in a format other than one of the recognized date settings. In addition, a long date format is not recognized if it also contains the day-of-the-week string.

---

Data Type Functions

Conversion Functions

Example

The following example uses the **CDate** function to convert a string to a date. In general, hard coding dates and times as strings (as shown in this example) is not recommended. Use date and time literals (such as `#10/19/1962#`, `#4:45:23 PM#`) instead.

```vba
Dim MyShortTime, MyDate, MyTime
```
MyDate = "October 19, 1962" ' Define date.
MyShortDate = CDate(MyDate) ' Convert to Date data type.
MyTime = "4:35:47 PM" ' Define time.
MyShortTime = CDate(MyTime) ' Convert to Date data type.

### 2.3.5 CDbl Function

Returns an expression that has been converted to a **Variant** of subtype **Double**.

**Syntax**

```vbs
CDbl(Expression)
```

The `Expression` argument is any valid expression.

**Remarks**

In general, you can document your code using the subtype conversion functions to show that the result of some operation should be expressed as a particular data type rather than the default data type. For example, use `CDbl` or `CSng` to force double-precision or single-precision arithmetic in cases where currency or integer arithmetic normally would occur.

Use the `CDbl` function to provide internationally aware conversions from any other data type to a **Double** subtype. For example, different decimal separators and thousands separators are properly recognized depending on the locale setting of your system.

- Data Type Functions
- Conversion Functions

**Example**

This example uses the `CDbl` function to convert an expression to a **Double**.

```vbs
Dim MyCurr, MyDouble
MyCurr = CCur(234.456784) ' MyCurr is a Currency (234.4567).
' Convert result to a Double (19.2254576).
MyDouble = CDbl(MyCurr * 8.2 * 0.01)
```
2.3.6 CInt Function

Returns an expression that has been converted to a **Variant** of subtype **Integer**.

**Syntax**

```
CInt(Expression)
```

The *Expression* argument is any valid expression.

**Remarks**

In general, you can document your code using the subtype conversion functions to show that the result of some operation should be expressed as a particular data type rather than the default data type. For example, use **CInt** or **CLng** to force integer arithmetic in cases where currency, single-precision, or double-precision arithmetic normally would occur.

Use the **CInt** function to provide internationally aware conversions from any other data type to an **Integer** subtype. For example, different decimal separators are properly recognized depending on the locale setting of your system, as are different thousand separators.

If *Expression* lies outside the acceptable range for the **Integer** subtype, an error occurs.

**NOTE**

**CInt** differs from the **Fix** and **Int** functions, which truncate, rather than round, the fractional part of a number. When the fractional part is exactly 0.5, the **CInt** function always rounds it to the nearest even number. For example, 0.5 rounds to 0, and 1.5 rounds to 2.

**Example**

The following example uses the **CInt** function to convert a value to an Integer.

```
Dim MyDouble, MyInt
MyDouble = 2345.5678 ' MyDouble is a Double.
MyInt = CInt(MyDouble) ' MyInt contains 2346.
```
### 2.3.7 CLng Function

Returns an expression that has been converted to a **Variant** of subtype **Long**.

**Syntax**

\[ \text{CLng(Expression)} \]

The *Expression* argument is any valid expression.

**Remarks**

In general, you can document your code using the subtype conversion functions to show that the result of some operation should be expressed as a particular data type rather than the default data type. For example, use *CInt* or *CLng* to force integer arithmetic in cases where currency, single-precision, or double-precision arithmetic normally would occur.

Use the *CLng* function to provide internationally aware conversions from any other data type to a **Long** subtype. For example, different decimal separators are properly recognized depending on the locale setting of your system, as are different thousand separators.

If *Expression* lies outside the acceptable range for the **Long** subtype, an error occurs.

**NOTE**

*CLng* differs from the *Fix* and *Int* functions, which truncate, rather than round, the fractional part of a number. When the fractional part is exactly 0.5, the *CLng* function always rounds it to the nearest even number. For example, 0.5 rounds to 0, and 1.5 rounds to 2.

#### Example

The following example uses the **CLng** function to convert a value to a **Long**.

```vbscript
Dim MyVal1, MyVal2, MyLong1, MyLong2
MyVal1 = 25427.45
MyVal2 = 25427.55    ' MyVal1, MyVal2 are Doubles.
MyLong1 = CLng(MyVal1)    ' MyLong1 contains 25427.
```
2.3.8 CSng Function

Returns an expression that has been converted to a Variant of subtype Single.

Syntax

\[
\text{CSng(} \text{Expression} \text{)}
\]

The Expression argument is any valid expression.

Remarks

In general, you can document your code using the data type conversion functions to show that the result of some operation should be expressed as a particular data type rather than the default data type. For example, use CDbl or CSng to force double-precision or single-precision arithmetic in cases where currency or integer arithmetic normally would occur.

Use the CSng function to provide internationally aware conversions from any other data type to a Single subtype. For example, different decimal separators are properly recognized depending on the locale setting of your system, as are different thousand separators.

If Expression lies outside the acceptable range for the Single subtype, an error occurs.

Example

The following example uses the CSng function to convert a value to a Single.

```vbnet
Dim MyDouble1, MyDouble2 ' MyDouble1, MyDouble2 are Doubles.
Dim MySingle1, MySingle2
MyDouble1 = 75.3421115
MyDouble2 = 75.3421555
MySingle1 = CSng(MyDouble1) ' MySingle1 contains 75.34211.
MySingle2 = CSng(MyDouble2) ' MySingle2 contains 75.34216.
```
2.3.9 **CStr Function**

Returns an expression that has been converted to a **Variant** of subtype **String**.

**Syntax**

\[ \text{CStr}(\text{Expression}) \]

The \textit{Expression} argument is any valid expression.

**Remarks**

In general, you can document your code using the data type conversion functions to show that the result of some operation should be expressed as a particular data type rather than the default data type. For example, use \textbf{CStr} to force the result to be expressed as a \textbf{String}.

You should use the \textbf{CStr} function instead of \textbf{String} to provide internationally aware conversions from any other data type to a \textbf{String} subtype. For example, different decimal separators are properly recognized depending on the locale setting of your system.

The data in \textit{Expression} determines what is returned according to the following table.

<table>
<thead>
<tr>
<th>If \textit{Expression} is</th>
<th>CStr returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>A \textbf{String} containing \textit{True} or \textit{False}.</td>
</tr>
<tr>
<td>Date</td>
<td>A \textbf{String} containing a date in the short-date format of your system.</td>
</tr>
<tr>
<td>Null</td>
<td>A run-time error.</td>
</tr>
<tr>
<td>Empty</td>
<td>A zero-length \textbf{String} (&quot;&quot;).</td>
</tr>
<tr>
<td>Error</td>
<td>A \textbf{String} containing the word \textit{Error} followed by the error number.</td>
</tr>
<tr>
<td>Other numeric</td>
<td>A \textbf{String} containing the number.</td>
</tr>
</tbody>
</table>
**Example**

The following example uses the **CStr** function to convert a numeric value to a **String**.

```
Dim MyDouble, MyString
MyDouble = 437.324  ' MyDouble is a Double.
MyString = CStr(MyDouble)  ' MyString contains "437.324".
```

### 2.3.10 Fix Function

Returns the integer portion of a number.

**Syntax**

```
Fix(Number)
```

The **Number** argument can be any valid numeric expression. If **Number** contains **Null**, **Null** is returned.

**Remarks**

Both **Int** and **Fix** remove the fractional part of **Number** and return the resulting integer value.

The difference between **Int** and **Fix** is that if **Number** is negative, **Int** returns the first negative integer less than or equal to **Number**, whereas **Fix** returns the first negative integer greater than or equal to **Number**. For example, **Int** converts -8.4 to -9, and **Fix** converts -8.4 to -8.

**Fix(Number)** is equivalent to:

```
Sgn(Number) * Int(Abs(Number))
```

**Example**

The following examples illustrate how the **Int** and **Fix** functions return integer portions of numbers.
2.3.11 Int Function

Returns the integer portion of a number.

Syntax

\[ \text{Int}(\text{Number}) \]

The \text{Number} argument can be any valid numeric expression. If \text{Number} contains 
\text{Null}, \text{Null} is returned.

Remarks

Both \text{Int} and \text{Fix} remove the fractional part of \text{Number} and return the resulting integer value.

The difference between \text{Int} and \text{Fix} is that if \text{Number} is negative, \text{Int} returns the first negative integer less than or equal to \text{Number}, whereas \text{Fix} returns the first negative integer greater than or equal to \text{Number}. For example, \text{Int} converts -8.4 to -9, and \text{Fix} converts -8.4 to -8.

\text{Fix}(\text{Number}) \text{ is equivalent to.}

\[ \text{Sgn}(\text{Number}) \times \text{Int}(\text{Abs}(\text{Number})) \]

Data Type Functions

Conversion Functions

Math Functions

Example

The following examples illustrate how the \text{Int} and \text{Fix} functions return integer portions of numbers.
Dim MyNumber
MyNumber = Int(99.8) ' Returns 99.
MyNumber = Fix(99.2) ' Returns 99.
MyNumber = Int(-99.8) ' Returns -100.
MyNumber = Fix(-99.8) ' Returns -99.
MyNumber = Int(-99.2) ' Returns -100.
MyNumber = Fix(-99.2) ' Returns -99.

2.3.12 IsArray Function

Returns a Boolean value indicating whether a variable is an array.

Syntax
IsArray(VarName)

The VarName argument can be any variable.

Remarks
IsArray returns True if the variable is an array; otherwise, it returns False. IsArray is especially useful with variants containing arrays.

Example
The following example uses the IsArray function to test whether MyVariable is an array.

Dim MyVariable
Dim MyArray(3)
MyArray(0) = "Sunday"
MyArray(1) = "Monday"
MyArray(2) = "Tuesday"
MyVariable = IsArray(MyArray) ' MyVariable contains "True".

2.3.13 IsDate Function

Returns a Boolean value indicating whether an expression can be converted to a date.
Data Type Functions

Syntax

IsDate(Expression)

The Expression argument can be any date expression or string expression recognizable as a date or time.

Remarks

IsDate returns True if the Expression is a date or can be converted to a valid date; otherwise, it returns False. In Microsoft Windows, the range of valid dates is January 1, 100 A.D. through December 31, 9999 A.D.; the ranges vary among operating systems.

Example

The following example uses the IsDate function to determine whether an expression can be converted to a date.

```vbnet
Dim MyDate, YourDate, NoDate, MyCheck
MyDate = "October 19, 1962"
YourDate = #10/19/62#
NoDate = "Hello"
MyCheck = IsDate(MyDate)  ' Returns True.
MyCheck = IsDate(YourDate)  ' Returns True.
MyCheck = IsDate(NoDate)  ' Returns False.
```

2.3.14 IsEmpty Function

Returns a Boolean value indicating whether a variable has been initialized.

Syntax

IsEmpty(Expression)

The Expression argument can be any expression. However, because IsEmpty is used to determine if individual variables are initialized, the Expression argument is most often a single variable name.
Remarks

IsEmpty returns True if the variable is uninitialized, or is explicitly set to Empty; otherwise, it returns False. False is always returned if Expression contains more than one variable.

Example

The following example uses the IsEmpty function to determine whether a variable has been initialized.

```
Dim MyVar, MyCheck
MyCheck = IsEmpty(MyVar) ' Returns True.
MyVar = Null ' Assign Null.
MyCheck = IsEmpty(MyVar) ' Returns False.
MyVar = Empty ' Assign Empty.
MyCheck = IsEmpty(MyVar) ' Returns True.
```

2.3.15 IsNull Function

Returns a Boolean value that indicates whether an expression contains no valid data (Null).

Syntax

```
IsNull(Expression)
```

The Expression argument can be any expression.

Remarks

IsNull returns True if Expression is Null, that is, it contains no valid data; otherwise, IsNull returns False. If Expression consists of more than one variable, Null in any constituent variable causes True to be returned for the entire expression.

The Null value indicates that the variable contains no valid data. Null is not the same as Empty, which indicates that a variable has not yet been initialized. It is also not the same as a zero-length string (""), which is sometimes referred to as a null string.
IMPORTANT

Use the IsNull function to determine whether an expression contains a Null value. Expressions that you might expect to evaluate to True under some circumstances, such as If Var = Null and If Var <> Null, are always False. This is because any expression containing a Null is itself Null, and therefore, False.

Example

The following example uses the IsNull function to determine whether a variable contains a Null.

Dim MyVar, MyCheck
MyCheck = IsNull(MyVar) ' Returns False.
MyVar = Null ' Assign Null.
MyCheck = IsNull(MyVar) ' Returns True.
MyVar = Empty ' Assign Empty.
MyCheck = IsNull(MyVar) ' Returns False.

2.3.16 IsNumeric Function

Returns a Boolean value indicating whether an expression can be evaluated as a number.

Syntax

IsNumeric(Expression)

The Expression argument can be any expression.

Remarks

IsNumeric returns True if the entire Expression is recognized as a number; otherwise, it returns False. IsNumeric returns False if Expression is a date expression.
Example
The following example uses the **IsNumeric** function to determine whether a variable can be evaluated as a number.

```vba
Dim MyVar, MyCheck
MyVar = 53 ' Assign a value.
MyCheck = IsNumeric(MyVar) ' Returns True.
MyVar = "459.95" ' Assign a value.
MyCheck = IsNumeric(MyVar) ' Returns True.
MyVar = "45 Help" ' Assign a value.
MyCheck = IsNumeric(MyVar) ' Returns False.
```

### 2.3.17 IsObject Function

Returns a Boolean value indicating whether an expression references a valid Automation object.

**Syntax**

```
IsObject(Expression)
```

The *Expression* argument can be any expression.

**Remarks**

*IsObject* returns **True** if *Expression* is a variable of **Object** subtype or a user-defined object; otherwise, it returns **False**.

Example
The following example uses the **IsObject** function to determine if an identifier represents an object variable.

```vba
Dim MyInt, MyCheck, MyObject
Set MyObject = Me
MyCheck = IsObject(MyObject) ' Returns True.
MyCheck = IsObject(MyInt) ' Returns False.
```
### 2.3.18 TypeName Function

Returns a string that provides **Variant** subtype information about a variable.

**Syntax**

\[
\text{TypeName}(\text{VarName})
\]

The required \textit{VarName} argument can be any variable.

**Return Values**

The **TypeName** function has the following return values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte</td>
<td>Byte value.</td>
</tr>
<tr>
<td>Integer</td>
<td>Integer value.</td>
</tr>
<tr>
<td>Long</td>
<td>Long integer value.</td>
</tr>
<tr>
<td>Single</td>
<td>Single-precision floating-point value.</td>
</tr>
<tr>
<td>Double</td>
<td>Double-precision floating-point value.</td>
</tr>
<tr>
<td>Currency</td>
<td>Currency value.</td>
</tr>
<tr>
<td>Decimal</td>
<td>Decimal value.</td>
</tr>
<tr>
<td>Date</td>
<td>Date or time value.</td>
</tr>
<tr>
<td>String</td>
<td>Character string value.</td>
</tr>
<tr>
<td>Boolean</td>
<td>Boolean value; <strong>True</strong> or <strong>False</strong>.</td>
</tr>
<tr>
<td>Empty</td>
<td>Uninitialized.</td>
</tr>
<tr>
<td>Null</td>
<td>No valid data.</td>
</tr>
<tr>
<td>&lt;Object type&gt;</td>
<td>Actual type name of an object.</td>
</tr>
</tbody>
</table>
## Data Type Functions

### Object
- Generic object.

### Unknown
- Unknown object type.

### Nothing
- Object variable that doesn't yet refer to an object instance.

### Error
- Error.

### Example
The following example uses the `TypeName` function to return information about a variable.

```vbscript
Dim ArrayVar(4), MyType
NullVar = Null ' Assign Null value.
MyType = TypeName("VBScript") ' Returns "String".
MyType = TypeName(4) ' Returns "Integer".
MyType = TypeName(37.50) ' Returns "Double".
MyType = TypeName(NullVar) ' Returns "Null".
MyType = TypeName(ArrayVar) ' Returns "Variant()".
```

### 2.3.19 VarType Function

Returns a value indicating the subtype of a variable.

**Syntax**

```
VarType(VarName)
```

The `VarName` argument can be any variable.

**Return Values**

The `VarType` function returns the following values:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
</table>

---

Reliance 4 – Scripts
<table>
<thead>
<tr>
<th>Data Type Functions</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>vbEmpty</code></td>
<td>0</td>
<td><em>Empty</em> (uninitialized).</td>
</tr>
<tr>
<td><code>vbNull</code></td>
<td>1</td>
<td><em>Null</em> (no valid data).</td>
</tr>
<tr>
<td><code>vbInteger</code></td>
<td>2</td>
<td>Integer.</td>
</tr>
<tr>
<td><code>vbLong</code></td>
<td>3</td>
<td>Long integer.</td>
</tr>
<tr>
<td><code>vbSingle</code></td>
<td>4</td>
<td>Single-precision floating-point number.</td>
</tr>
<tr>
<td><code>vbDouble</code></td>
<td>5</td>
<td>Double-precision floating-point number.</td>
</tr>
<tr>
<td><code>vbCurrency</code></td>
<td>6</td>
<td>Currency.</td>
</tr>
<tr>
<td><code>vbDate</code></td>
<td>7</td>
<td>Date.</td>
</tr>
<tr>
<td><code>vbString</code></td>
<td>8</td>
<td>String.</td>
</tr>
<tr>
<td><code>vbObject</code></td>
<td>9</td>
<td>Automation object.</td>
</tr>
<tr>
<td><code>vbError</code></td>
<td>10</td>
<td>Error.</td>
</tr>
<tr>
<td><code>vbBoolean</code></td>
<td>11</td>
<td>Boolean.</td>
</tr>
<tr>
<td><code>vbVariant</code></td>
<td>12</td>
<td>Variant (used only with arrays of Variants).</td>
</tr>
<tr>
<td><code>vbDataObject</code></td>
<td>13</td>
<td>A data-access object.</td>
</tr>
<tr>
<td><code>vbByte</code></td>
<td>17</td>
<td>Byte.</td>
</tr>
<tr>
<td><code>vbArray</code></td>
<td>8192</td>
<td>Array.</td>
</tr>
</tbody>
</table>
Remarks

The `VarType` function never returns the value for `Array` by itself. It is always added to some other value to indicate an array of a particular type. The value for `Variant` is only returned when it has been added to the value for `Array` to indicate that the argument to the `VarType` function is an array. For example, the value returned for an array of integers is calculated as $2 + 8192$, or $8194$. If an object has a default property, `VarType (Object)` returns the type of its default property.

**NOTE**

These constants are specified by VBScript. As a result, the names can be used anywhere in your code in place of the actual values.

### Data Type Functions

#### Example

The following example uses the `VarType` function to determine the subtype of a variable.

```vbnet
Dim MyCheck
MyCheck = VarType(300) ' Returns 2.
MyCheck = VarType(#10/19/62#) ' Returns 7.
MyCheck = VarType("VBScript") ' Returns 8.
```
2.4 Date and Time Functions

- **Date** Function
- **DateAdd** Function
- **DateDiff** Function
- **DatePart** Function
- **DateSerial** Function
- **DateValue** Function
- **Day** Function
- **Hour** Function
- **Minute** Function
- **Month** Function
- **MonthName** Function
- **Now** Function
- **Second** Function
- **Time** Function
- **Timer** Function
- **TimeSerial** Function
- **TimeValue** Function
- **WeekDay** Function
- **WeekDayName** Function
- **Year** Function

### 2.4.1 Date Function

Returns the current system date. The returned value is expressed in local time.

The local time is dependent on the operating system settings (the time zone, automatically adjusting to daylight saving time).
Syntax

```
Date
```

Example

The following example uses the `Date` function to return the current system date.

```
Dim MyDate
MyDate = Date ' MyDate contains the current system date.
```

2.4.2 DateAdd Function

Returns a date to which a specified time interval has been added.

Syntax

```
DateAdd(Interval, Number, Date)
```

The `DateAdd` function syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Interval</code></td>
<td>Required. String expression that is the interval you want to add. See the Settings section for values.</td>
</tr>
<tr>
<td><code>Number</code></td>
<td>Required. Numeric expression that is the number of interval you want to add. The numeric expression can either be positive, for dates in the future, or negative, for dates in the past.</td>
</tr>
<tr>
<td><code>Date</code></td>
<td>Required. <strong>Variant</strong> or literal representing the date to which <code>Interval</code> is added.</td>
</tr>
</tbody>
</table>

Settings

The `Interval` argument can have the following values:
**Date and Time Functions**

### Setting and Description

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yyyy</td>
<td>Year</td>
</tr>
<tr>
<td>q</td>
<td>Quarter</td>
</tr>
<tr>
<td>m</td>
<td>Month</td>
</tr>
<tr>
<td>y</td>
<td>Day of year</td>
</tr>
<tr>
<td>d</td>
<td>Day</td>
</tr>
<tr>
<td>w</td>
<td>Weekday</td>
</tr>
<tr>
<td>ww</td>
<td>Week of year</td>
</tr>
<tr>
<td>h</td>
<td>Hour</td>
</tr>
<tr>
<td>n</td>
<td>Minute</td>
</tr>
<tr>
<td>s</td>
<td>Second</td>
</tr>
</tbody>
</table>

**Remarks**

You can use the **DateAdd** function to add or subtract a specified time interval from a date. For example, you can use **DateAdd** to calculate a date 30 days from today or a time 45 minutes from now. To add days to **Date**, you can use Day of Year ("y"), Day ("d"), or Weekday ("w").

The **DateAdd** function won’t return an invalid date. The following example adds one month to January 31.

```
NewDate = DateAdd("m", 1, "31-Jan-95")
```

In this case, **DateAdd** returns 28-Feb-95, not 31-Feb-95. If **Date** is 31-Jan-96, it returns 29-Feb-96 because 1996 is a leap year.

If the calculated date would precede the year 100, an error occurs.

If number isn’t a **Long** value, it is rounded to the nearest whole number before being evaluated.
2.4.3 DateDiff Function

Returns the number of intervals between two dates.

Syntax

\[
\text{DateDiff}(\text{Interval}, \text{Date1}, \text{Date2}[, \text{FirstDayOfWeek}[, \text{FirstWeekOfYear}]])
\]

The DateDiff function syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interval</td>
<td>Required. String expression that is the interval you want to use to calculate the differences between Date1 and Date2. See the Settings section for values.</td>
</tr>
<tr>
<td>Date1, Date2</td>
<td>Required. Date expressions. Two dates you want to use in the calculation.</td>
</tr>
<tr>
<td>FirstDayOfWeek</td>
<td>Optional. Constant that specifies the day of the week. If not specified, Sunday is assumed. See the Settings section for values.</td>
</tr>
<tr>
<td>FirstWeekOfYear</td>
<td>Optional. Constant that specifies the first week of the year. If not specified, the first week is assumed to be the week in which January 1 occurs. See the Settings section for values.</td>
</tr>
</tbody>
</table>

Settings

The Interval argument can have the following values:

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yyyy</td>
<td>Year</td>
</tr>
<tr>
<td>q</td>
<td>Quarter</td>
</tr>
</tbody>
</table>
The `FirstDayOfWeek` argument can have the following values:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>vbUseSystem</code></td>
<td>0</td>
<td>Use National Language Support (NLS) API setting.</td>
</tr>
<tr>
<td><code>vbSunday</code></td>
<td>1</td>
<td>Sunday (default)</td>
</tr>
<tr>
<td><code>vbMonday</code></td>
<td>2</td>
<td>Monday</td>
</tr>
<tr>
<td><code>vbTuesday</code></td>
<td>3</td>
<td>Tuesday</td>
</tr>
<tr>
<td><code>vbWednesday</code></td>
<td>4</td>
<td>Wednesday</td>
</tr>
<tr>
<td><code>vbThursday</code></td>
<td>5</td>
<td>Thursday</td>
</tr>
<tr>
<td><code>vbFriday</code></td>
<td>6</td>
<td>Friday</td>
</tr>
<tr>
<td><code>vbSaturday</code></td>
<td>7</td>
<td>Saturday</td>
</tr>
</tbody>
</table>

The `FirstWeekOfYear` argument can have the following values:
### Date and Time Functions

#### Constant Value Description

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbUseSystem</td>
<td>0</td>
<td>Use National Language Support (NLS) API setting.</td>
</tr>
<tr>
<td>vbFirstJan1</td>
<td>1</td>
<td>Start with the week in which January 1 occurs (default).</td>
</tr>
<tr>
<td>vbFirstFourDays</td>
<td>2</td>
<td>Start with the week that has at least four days in the new year.</td>
</tr>
<tr>
<td>vbFirstFullWeek</td>
<td>3</td>
<td>Start with the first full week of the new year.</td>
</tr>
</tbody>
</table>

#### Remarks

You can use the `DateDiff` function to determine how many specified time intervals exist between two dates. For example, you might use `DateDiff` to calculate the number of days between two dates, or the number of weeks between today and the end of the year.

To calculate the number of days between `Date1` and `Date2`, you can use either Day of year ("y") or Day ("d"). When `Interval` is Weekday ("w"), `DateDiff` returns the number of weeks between the two dates. If `Date1` falls on a Monday, `DateDiff` counts the number of Mondays until `Date2`. It counts `Date2` but not `Date1`. If `Interval` is Week ("ww"), however, the `DateDiff` function returns the number of calendar weeks between the two dates. It counts the number of Sundays between `Date1` and `Date2`. `DateDiff` counts `Date2` if it falls on a Sunday; but it doesn't count `Date1`, even if it does fall on a Sunday.

If `Date1` refers to a later point in time than `Date2`, the `DateDiff` function returns a negative number.

The `FirstDayOfWeek` argument affects calculations that use the "w" and "ww" interval symbols.

If `Date1` or `Date2` is a date literal, the specified year becomes a permanent part of that date. However, if `Date1` or `Date2` is enclosed in quotation marks (" ") and you omit the year, the current year is inserted in your code each time the `Date1` or `Date2` expression is evaluated. This makes it possible to write code that can be used in different years.

When comparing December 31 to January 1 of the immediately succeeding year, `DateDiff` for Year ("yyyy") returns 1 even though only a day has elapsed.
Example

The following example uses the **DateDiff** function to display the number of days between a given date and today.

```vbscript
Function DiffADate(TheDate)
    DiffADate = "Days from today: " & DateDiff("d", Now, TheDate)
End Function
```

### 2.4.4 DatePart Function

Returns the specified part of a given date.

**Syntax**

```vbscript
DatePart(Interval, Date[, FirstDayOfWeek[, FirstWeekOfYear]])
```

The **DatePart** function syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Interval</strong></td>
<td>Required. String expression that is the interval of time you want to return. See the Settings section for values.</td>
</tr>
<tr>
<td><strong>Date</strong></td>
<td>Required. Date expression you want to evaluate.</td>
</tr>
<tr>
<td><strong>FirstDayOfWeek</strong></td>
<td>Optional. Constant that specifies the day of the week. If not specified, Sunday is assumed. See the Settings section for values.</td>
</tr>
<tr>
<td><strong>FirstWeekOfYear</strong></td>
<td>Optional. Constant that specifies the first week of the year. If not specified, the first week is assumed to be the week in which January 1 occurs. See the Settings section for values.</td>
</tr>
</tbody>
</table>

**Settings**

The **Interval** argument can have the following values:
### Date and Time Functions

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yyyy</td>
<td>Year</td>
</tr>
<tr>
<td>q</td>
<td>Quarter</td>
</tr>
<tr>
<td>m</td>
<td>Month</td>
</tr>
<tr>
<td>y</td>
<td>Day of year</td>
</tr>
<tr>
<td>d</td>
<td>Day</td>
</tr>
<tr>
<td>w</td>
<td>Weekday</td>
</tr>
<tr>
<td>ww</td>
<td>Week of year</td>
</tr>
<tr>
<td>h</td>
<td>Hour</td>
</tr>
<tr>
<td>n</td>
<td>Minute</td>
</tr>
<tr>
<td>s</td>
<td>Second</td>
</tr>
</tbody>
</table>

The *FirstDayOfWeek* argument can have the following values:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbUseSystem</td>
<td>0</td>
<td>Use National Language Support (NLS) API setting.</td>
</tr>
<tr>
<td>vbSunday</td>
<td>1</td>
<td>Sunday (default)</td>
</tr>
<tr>
<td>vbMonday</td>
<td>2</td>
<td>Monday</td>
</tr>
<tr>
<td>vbTuesday</td>
<td>3</td>
<td>Tuesday</td>
</tr>
<tr>
<td>vbWednesday</td>
<td>4</td>
<td>Wednesday</td>
</tr>
<tr>
<td>vbThursday</td>
<td>5</td>
<td>Thursday</td>
</tr>
<tr>
<td>vbFriday</td>
<td>6</td>
<td>Friday</td>
</tr>
<tr>
<td>vbSaturday</td>
<td>7</td>
<td>Saturday</td>
</tr>
</tbody>
</table>

The *FirstWeekOfYear* argument can have the following values:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
</table>
### Remarks

You can use the `DatePart` function to evaluate a date and return a specific interval of time. For example, you might use `DatePart` to calculate the day of the week or the current hour.

The `FirstDayOfWeek` argument affects calculations that use the "w" and "ww" interval symbols.

If `Date` is a date literal, the specified year becomes a permanent part of that date. However, if `Date` is enclosed in quotation marks (" "), and you omit the year, the current year is inserted in your code each time the `Date` expression is evaluated. This makes it possible to write code that can be used in different years.

### Example

This example takes a date and, using the `DatePart` function, displays the quarter of the year in which it occurs.

```vba
Function GetQuarter(TheDate)
    GetQuarter = DatePart("q", TheDate)
End Function
```

#### 2.4.5 DateSerial Function

Returns a Variant of subtype Date for a specified year, month, and day.
Syntax

DateSerial(Year, Month, Day)

The DateSerial function syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Year</td>
<td>Number between 100 and 9999, inclusive, or a numeric expression.</td>
</tr>
<tr>
<td>Month</td>
<td>Any numeric expression.</td>
</tr>
<tr>
<td>Day</td>
<td>Any numeric expression.</td>
</tr>
</tbody>
</table>

Remarks

To specify a date, such as December 31, 1991, the range of numbers for each DateSerial argument should be in the accepted range for the unit; that is, 1–31 for days and 1–12 for months. However, you can also specify relative dates for each argument using any numeric expression that represents some number of days, months, or years before or after a certain date.

For the Year argument, values between 0 and 99, inclusive, are interpreted as the years 1900–1999. For all other Year arguments, use a complete four-digit year (for example, 1800).

When any argument exceeds the accepted range for that argument, it increments to the next larger unit as appropriate. For example, if you specify 35 days, it is evaluated as one month and some number of days, depending on where in the year it is applied. However, if any single argument is outside the range -32,768 to 32,767, or if the date specified by the three arguments, either directly or by expression, falls outside the acceptable range of dates, an error occurs.
Example

The following example uses numeric expressions instead of absolute date numbers. Here the `DateSerial` function returns a date that is the day before the first day (1 - 1) of two months before August (8 - 2) of 10 years before 1990 (1990 - 10); in other words, May 31, 1980.

```vbscript
Dim MyDate1, MyDate2
MyDate1 = DateSerial(1970, 1, 1) ' Returns January 1, 1970.
MyDate2 = DateSerial(1990 - 10, 8 - 2, 1 - 1) ' Returns May 31, 1980.
```

### 2.4.6 DateValue Function

Returns a **Variant** of subtype **Date**.

**Syntax**

```
DateValue(Date)
```

The `Date` argument is normally a string expression representing a date from January 1, 100 through December 31, 9999. However, `Date` can also be any expression that can represent a date, a time, or both a date and time, in that range.

**Remarks**

If the `Date` argument includes time information, **DateValue** doesn't return it. However, if `Date` includes invalid time information (such as "89:98"), an error occurs.

If `Date` is a string that includes only numbers separated by valid date separators, **DateValue** recognizes the order for month, day, and year according to the short date format you specified for your system. **DateValue** also recognizes unambiguous dates that contain month names, either in long or abbreviated form. For example, in addition to recognizing 12/30/1991 and 12/30/91, **DateValue** also recognizes December 30, 1991 and Dec 30, 1991.

If the year part of `Date` is omitted, **DateValue** uses the current year from your computer's system date.
Example

The following example uses the `DateValue` function to convert a string to a date. You can also use date literals to directly assign a date to a `Variant` variable, for example, `MyDate = #9/11/63#`.

```vba
Dim MyDate
MyDate = DateValue("September 11, 1963") ' Return a date.
```

2.4.7 Day Function

Returns a whole number between 1 and 31, inclusive, representing the day of the month.

Syntax

```vba
Day(Date)
```

The `Date` argument is any expression that can represent a date. If `Date` contains `Null`, `Null` is returned.

Example

The following example uses the `Day` function to obtain the day of the month from a specified date.

```vba
Dim MyDay
```

2.4.8 Hour Function

Returns a whole number between 0 and 23, inclusive, representing the hour of the day.
Syntax

\[ \text{Hour}(\text{Time}) \]

The \text{Time} argument is any expression that can represent a time. If \text{Time} contains \text{Null}, \text{Null} is returned.

Example

The following example uses the \text{Hour} function to obtain the hour from the current time.

\begin{verbatim}
Dim MyTime, MyHour
MyTime = Now
' MyHour contains the number representing the current hour.
MyHour = Hour(MyTime)
\end{verbatim}

2.4.9 Minute Function

Returns a whole number between 0 and 59, inclusive, representing the minute of the hour.

Syntax

\[ \text{Minute}(\text{Time}) \]

The \text{Time} argument is any expression that can represent a time. If \text{Time} contains \text{Null}, \text{Null} is returned.

Example

The following example uses the \text{Minute} function to return the minute of the hour.

\begin{verbatim}
Dim MyVar
MyVar = Minute(Now)
\end{verbatim}
2.4.10  Month Function

Returns a whole number between 1 and 12, inclusive, representing the month of the year.

Syntax

Month(Date)

The Date argument is any expression that can represent a date. If Date contains Null, Null is returned.

Example

The following example uses the Month function to return the current month.

Dim MyVar
' MyVar contains the number corresponding to the current month.
MyVar = Month(Now)

2.4.11  MonthName Function

Returns a string indicating the specified month.

Syntax

MonthName(Month[, Abbreviate])

The MonthName function syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Month</td>
<td>Required. The numeric designation of the month. For example, January is 1, February is 2, and so on.</td>
</tr>
</tbody>
</table>
Abbreviate | Optional. Boolean value that indicates if the month name is to be abbreviated. If omitted, the default is False, which means that the month name is not abbreviated.

---

### Example

The following example uses the MonthName function to return an abbreviated month name for a date expression.

```vbnet
Dim MyVar
MyVar = MonthName(10, True) ' MyVar contains "Oct".
```

#### 2.4.12 Now Function

Returns the current system date and time. The returned value is expressed in local time.

The local time is dependent on the operating system settings (the time zone, automatically adjusting to daylight saving time).

**Syntax**

```vbnet
Now
```

### Example

The following example uses the Now function to return the current date and time.

```vbnet
Dim MyVar
MyVar = Now ' MyVar contains the current date and time.
```

#### 2.4.13 Second Function

Returns a whole number between 0 and 59, inclusive, representing the second of the minute.
Syntax

\textbf{Second}(\textit{Time})

The \textit{Time} argument is any expression that can represent a time. If \textit{Time} contains \textbf{Null}, \textbf{Null} is returned.

Example

The following example uses the \textbf{Second} function to return the current second.

\begin{verbatim}
Dim MySec
' MySec contains the number representing the current second.
MySec = Second(Now)
\end{verbatim}

\section*{2.4.14 Time Function}

Returns a \textbf{Variant} of subtype \textbf{Date} indicating the current system time. The returned value is expressed in local time.

The local time is dependent on the operating system settings (the time zone, automatically adjusting to daylight saving time).

Syntax

\textbf{Time}

Example

The following example uses the \textbf{Time} function to return the current system time.

\begin{verbatim}
Dim MyTime
MyTime = Time ' Returns the current system time.
\end{verbatim}
2.4.15 Timer Function

Returns the number of seconds that have elapsed since 12:00 AM (midnight).

Syntax

```
Timer
```

Example

The following example uses the `Timer` function to determine the time it takes to iterate a `For...Next` loop `N` times.

```
Function TimeIt(N)
    Dim StartTime, EndTime
    StartTime = Timer
    For I = 1 To N
        Next
    EndTime = Timer
    TimeIt = EndTime - StartTime
End Function
```

2.4.16 TimeSerial Function

Returns a `Variant` of subtype `Date` containing the time for a specific hour, minute, and second.

Syntax

```
TimeSerial(Hour, Minute, Second)
```

The `TimeSerial` function syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hour</td>
<td>Number between 0 (12:00 A.M.) and 23 (11:00 P.M.), inclusive, or a numeric expression.</td>
</tr>
<tr>
<td>Minute</td>
<td>Any numeric expression.</td>
</tr>
</tbody>
</table>
**Remarks**

To specify a time, such as 11:59:59, the range of numbers for each `TimeSerial` argument should be in the accepted range for the unit; that is, 0–23 for hours and 0–59 for minutes and seconds. However, you can also specify relative times for each argument using any numeric expression that represents some number of hours, minutes, or seconds before or after a certain time.

When any argument exceeds the accepted range for that argument, it increments to the next larger unit as appropriate. For example, if you specify 75 minutes, it is evaluated as one hour and 15 minutes. However, if any single argument is outside the range -32,768 to 32,767, or if the time specified by the three arguments, either directly or by expression, causes the date to fall outside the acceptable range of dates, an error occurs.

**Example**

The following example uses expressions instead of absolute time numbers. The `TimeSerial` function returns a time for 15 minutes before (-15) six hours before noon (12 - 6), or 5:45:00 A.M.

```vba
Dim MyTime
MyTime = TimeSerial(12 - 6, -15, 0) ' Returns 5:45:00 AM
```

**2.4.17 TimeValue Function**

Returns a `Variant` of subtype `Date` containing the time.

**Syntax**

```
TimeValue(Time)
```

The `Time` argument is usually a string expression representing a time from 0:00:00 (12:00:00 A.M.) to 23:59:59 (11:59:59 P.M.), inclusive. However, `Time` can also be any expression that represents a time in that range. If `Time` contains `Null`, `Null` is returned.
Remarks
You can enter valid times using a 12-hour or 24-hour clock. For example, "2:24PM" and "14:24" are both valid Time arguments. If the Time argument contains date information, TimeValue doesn't return the date information. However, if Time includes invalid date information, an error occurs.

Example
The following example uses the TimeValue function to convert a string to a time. You can also use date literals to directly assign a time to a Variant (for example, MyTime = #4:35:17 PM#).

Dim MyTime
MyTime = TimeValue("4:35:17 PM") ' MyTime contains 4:35:17 PM

2.4.18 Weekday Function

Returns a whole number representing the day of the week.

Syntax

Weekday(Date[, FirstDayOfWeek])

The Weekday function syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>Any expression that can represent a date. If Date contains Null, Null is returned.</td>
</tr>
<tr>
<td>FirstDayOfWeek</td>
<td>A constant that specifies the first day of the week. If omitted, vbSunday is assumed.</td>
</tr>
</tbody>
</table>

Settings
The FirstDayOfWeek argument has these settings:
### Date and Time Functions

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbUseSystem</td>
<td>0</td>
<td>Use National Language Support (NLS) API setting.</td>
</tr>
<tr>
<td>vbSunday</td>
<td>1</td>
<td>Sunday</td>
</tr>
<tr>
<td>vbMonday</td>
<td>2</td>
<td>Monday</td>
</tr>
<tr>
<td>vbTuesday</td>
<td>3</td>
<td>Tuesday</td>
</tr>
<tr>
<td>vbWednesday</td>
<td>4</td>
<td>Wednesday</td>
</tr>
<tr>
<td>vbThursday</td>
<td>5</td>
<td>Thursday</td>
</tr>
<tr>
<td>vbFriday</td>
<td>6</td>
<td>Friday</td>
</tr>
<tr>
<td>vbSaturday</td>
<td>7</td>
<td>Saturday</td>
</tr>
</tbody>
</table>

#### Return Values

The **Weekday** function can return any of these values:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbSunday</td>
<td>1</td>
<td>Sunday</td>
</tr>
<tr>
<td>vbMonday</td>
<td>2</td>
<td>Monday</td>
</tr>
<tr>
<td>vbTuesday</td>
<td>3</td>
<td>Tuesday</td>
</tr>
<tr>
<td>vbWednesday</td>
<td>4</td>
<td>Wednesday</td>
</tr>
<tr>
<td>vbThursday</td>
<td>5</td>
<td>Thursday</td>
</tr>
<tr>
<td>vbFriday</td>
<td>6</td>
<td>Friday</td>
</tr>
<tr>
<td>vbSaturday</td>
<td>7</td>
<td>Saturday</td>
</tr>
</tbody>
</table>
Example

The following example uses the **Weekday** function to obtain the day of the week from a specified date.

```plaintext
Dim MyDate, MyWeekDay
MyDate = #October 19, 1962#  ' Assign a date.
' MyWeekDay contains 6 because MyDate represents a Friday.
MyWeekDay = Weekday(MyDate)
```

**2.4.19 WeekdayName Function**

Returns a string indicating the specified day of the week.

**Syntax**

```plaintext
WeekdayName(WeekDay[, Abbreviate[, FirstDayOfWeek]])
```

The **WeekdayName** function syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WeekDay</strong></td>
<td>Required. The numeric designation for the day of the week. Numeric value of each day depends on setting of the <strong>FirstDayOfWeek</strong> setting.</td>
</tr>
<tr>
<td><strong>Abbreviate</strong></td>
<td>Optional. Boolean value that indicates if the weekday name is to be abbreviated. If omitted, the default is <strong>False</strong>, which means that the weekday name is not abbreviated.</td>
</tr>
<tr>
<td><strong>FirstDayOfWeek</strong></td>
<td>Optional. Numeric value indicating the first day of the week. See the Settings section for values.</td>
</tr>
</tbody>
</table>

**Settings**

The **FirstDayOfWeek** argument can have the following values:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
</table>

Date and Time Functions

<table>
<thead>
<tr>
<th>vbUseSystem</th>
<th>0</th>
<th>Use National Language Support (NLS) API setting.</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbSunday</td>
<td>1</td>
<td>Sunday</td>
</tr>
<tr>
<td>vbMonday</td>
<td>2</td>
<td>Monday</td>
</tr>
<tr>
<td>vbTuesday</td>
<td>3</td>
<td>Tuesday</td>
</tr>
<tr>
<td>vbWednesday</td>
<td>4</td>
<td>Wednesday</td>
</tr>
<tr>
<td>vbThursday</td>
<td>5</td>
<td>Thursday</td>
</tr>
<tr>
<td>vbFriday</td>
<td>6</td>
<td>Friday</td>
</tr>
<tr>
<td>vbSaturday</td>
<td>7</td>
<td>Saturday</td>
</tr>
</tbody>
</table>

Example

The following example uses the `WeekDayName` function to return the specified day.

```vbnet
Dim MyDate
MyDate = WeekDayName(6, True) ' MyDate contains Fri.
```

2.4.20 Year Function

Returns a whole number representing the year.

Syntax

```vbnet
Year(Date)
```

The `Date` argument is any expression that can represent a date. If `Date` contains `Null`, `Null` is returned.
Example

The following example uses the **Year** function to obtain the year from a specified date.

```vbscript
Dim MyDate, MyYear
MyDate = #October 19, 1962# ' Assign a date.
MyYear = Year(MyDate) ' MyYear contains 1962.
```
2.5 Array Functions

- **Array Function**
- **Dim Statement**
- **Erase Statement**
- **Filter Function**
- **IsArray Function**
- **Join Function**
- **LBound Function**
- **Private Statement**
- **Public Statement**
- **ReDim Statement**
- **Split Function**
- **UBound Function**

### 2.5.1 Array Function

Returns a **Variant** containing an array.

**Syntax**

```
Array(ArgList)
```

The required `ArgList` argument is a comma-delimited list of values that are assigned to the elements of an array contained with the **Variant**. If no arguments are specified, an array of zero length is created.

**Remarks**

The notation used to refer to an element of an array consists of the variable name followed by parentheses containing an index number indicating the desired element.

**NOTE**

A variable that is not declared as an array can still contain an array. Although a **Variant** variable containing an array is conceptually different from an array variable containing **Variant** elements, the array elements are accessed in the same way.
Array Functions

Example

In the following example, the first statement creates a variable named A. The second statement assigns an array to variable A. The last statement assigns the value contained in the second array element to another variable.

```
Dim A
A = Array(10, 20, 30)
B = A(2) ' B is now 30.
```

2.5.2 Dim Statement

Declares variables and allocates storage space.

Syntax

```
Dim VarName[(Subscripts)], VarName[(Subscripts)]...
```

The `Dim` statement syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VarName</td>
<td>Name of the variable; follows standard variable naming conventions.</td>
</tr>
<tr>
<td>Subscripts</td>
<td>Dimensions of an array variable; up to 60 multiple dimensions may be declared. The <code>Subscripts</code> argument uses the following syntax:</td>
</tr>
<tr>
<td></td>
<td><code>upperbound[, upperbound]...</code></td>
</tr>
<tr>
<td></td>
<td>The lower bound of an array is always zero.</td>
</tr>
</tbody>
</table>

Remarks

Variables declared with `Dim` at the script level (globally) are available to all procedures in all scripts within the same thread (similar to `Public` statement). At the procedure level, variables are available only within the procedure.
You can also use the **Dim** statement with empty parentheses to declare a dynamic array. After declaring a dynamic array, use the **ReDim** statement within a procedure to define the number of dimensions and elements in the array. If you try to redeclare a dimension for an array variable whose size was explicitly specified in a **Dim** statement, an error occurs.

**TIP**

When you use the **Dim** statement in a procedure, you generally put the **Dim** statement at the beginning of the procedure.

---

**Example**

The following examples illustrate the use of the **Dim** statement.

```
Dim Names(9)   ' Declare an array with 10 elements.
Dim Names()    ' Declare a dynamic array.
Dim MyVar, MyNum ' Declare two variables.
```

---

**2.5.3 Erase Statement**

Reinitializes the elements of fixed-size arrays and deallocates dynamic-array storage space.

**Syntax**

```
Erase Array
```

The **Array** argument is the name of the array variable to be erased.

**Remarks**

It is important to know whether an array is fixed-size (ordinary) or dynamic because **Erase** behaves differently depending on the type of array. **Erase** recovers no memory for fixed-size arrays. **Erase** sets the elements of a fixed array as follows:

<table>
<thead>
<tr>
<th>Type of array</th>
<th>Effect of Erase on fixed-array elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed numeric array</td>
<td>Sets each element to zero.</td>
</tr>
</tbody>
</table>
Array Functions

Fixed string array
Sets each element to zero-length ("").

Array of objects
Sets each element to the special value Nothing.

**Erase** frees the memory used by dynamic arrays. Before your program can refer to the dynamic array again, it must redefine the array variable's dimensions using a ReDim statement.

---

**Array Functions**

**VBScript Statements**

**Example**

The following example illustrates the use of the **Erase** statement.

```vbnet
Dim NumArray(9)
Dim DynamicArray()
ReDim DynamicArray(9) ' Allocate storage space.
Erase NumArray ' Each element is reinitialized.
Erase DynamicArray ' Free memory used by array.
```

### 2.5.4 Filter Function

Returns a zero-based array containing a subset of a string array based on a specified filter criteria.

**Syntax**

```
Filter(InputStrings, Value[, Include[, Compare]])
```

The **Filter** function syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>InputStrings</strong></td>
<td>Required. One-dimensional array of strings to be searched.</td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td>Required. String to search for.</td>
</tr>
</tbody>
</table>
Array Functions

**Include**
Optional. Boolean value indicating whether to return substrings that include or exclude `Value`. If `Include` is **True**, `Filter` returns the subset of the array that contains `Value` as a substring. If `Include` is **False**, `Filter` returns the subset of the array that does not contain `Value` as a substring.

**Compare**
Optional. Numeric value indicating the kind of string comparison to use. See the **Settings** section for values.

**Settings**
The `Compare` argument can have the following values:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>vbBinaryCompare</code></td>
<td>0</td>
<td>Perform a binary comparison.</td>
</tr>
<tr>
<td><code>vbTextCompare</code></td>
<td>1</td>
<td>Perform a textual comparison.</td>
</tr>
</tbody>
</table>

**Remarks**
If no matches of `Value` are found within `InputStrings`, `Filter` returns an empty array. An error occurs if `InputStrings` is **Null** or is not a one-dimensional array.

The array returned by the `Filter` function contains only enough elements to contain the number of matched items.

**Example**
The following example uses the `Filter` function to return the array containing the search criteria "Mon".

```vba
Dim MyIndex
Dim MyArray (3)
MyArray(0) = "Sunday"
```
MyArray(1) = "Monday"
MyArray(2) = "Tuesday"
MyIndex = Filter(MyArray, "Mon") ' MyIndex(0) contains "Monday".

2.5.5 IsArray Function

Returns a Boolean value indicating whether a variable is an array.

Syntax

\[
\text{IsArray(VarName)}
\]

The \text{VarName} argument can be any variable.

Remarks

\text{IsArray} returns \textbf{True} if the variable is an array; otherwise, it returns \textbf{False}. IsArray is especially useful with variants containing arrays.

Data Type Functions

Array Functions

Example

The following example uses the IsArray function to test whether MyVariable is an array.

\[
\begin{align*}
\text{Dim MyVariable} \\
\text{Dim MyArray(3)} \\
\text{MyArray(0)} = "Sunday" \\
\text{MyArray(1)} = "Monday" \\
\text{MyArray(2)} = "Tuesday" \\
\text{MyVariable} = \text{IsArray(MyArray)} ' \text{MyVariable contains "True".}
\end{align*}
\]

2.5.6 Join Function

Returns a string created by joining a number of substrings contained in an array.

Syntax

\[
\text{Join(List[, Delimiter])}
\]

The \text{Join} function syntax has these parts:
### Array Functions

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>List</td>
<td>Required. One-dimensional array containing substrings to be joined.</td>
</tr>
<tr>
<td>Delimiter</td>
<td>Optional. String character used to separate the substrings in the returned string. If omitted, the space character (&quot; &quot;) is used. If <strong>Delimiter</strong> is a zero-length string, all items in the list are concatenated with no delimiters.</td>
</tr>
</tbody>
</table>

#### Example

The following example uses the **Join** function to join the substrings of **MyArray**.

```vbnet
Dim MyString
Dim MyArray(4)
MyArray(0) = "Mr."
MyArray(1) = "John"
MyArray(2) = "Doe"
MyArray(3) = "III"
MyString = Join(MyArray) ' MyString contains "Mr. John Doe III".
```

### 2.5.7 LBound Function

Returns the smallest available subscript for the indicated dimension of an array.

#### Syntax

```
LBound(ArrayName[, Dimension])
```

The **LBound** function syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArrayName</td>
<td>Name of the array variable; follows standard variable naming conventions.</td>
</tr>
</tbody>
</table>
### Dimension

| whole number indicating which dimension's lower bound is returned. Use 1 for the first dimension, 2 for the second, and so on. If Dimension is omitted, 1 is assumed.

### Remarks

The **LBound** function is used with the **UBound** function to determine the size of an array. Use the **UBound** function to find the upper limit of an array dimension.

The lower bound for any dimension is always 0.

---

### 2.5.8 Private Statement

Declares private variables and allocates storage space. Declares, in a **Class** block, a private variable.

**Syntax**

```
Private VarName[subscripts][, VarName[subscripts]]...
```

The **Private** statement syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>VarName</strong></td>
<td>Name of the variable; follows standard variable naming conventions.</td>
</tr>
</tbody>
</table>
| **Subscripts** | Dimensions of an array variable; up to 60 multiple dimensions may be declared. The Subscripts argument uses the following syntax:  
  upper[ , upper]...  
  The lower bound of an array is always zero. |
Remarks

A variable declared with the **Private** statement at a script level (globally) is available in all scripts within the same thread (similar to the **Public** statement). The **Private** statement can still be meaningfully used in a declaration of objects (with the **Class** statement).

A variable that refers to an object must be assigned an existing object using the **Set** statement before it can be used. Until it is assigned an object, the declared object variable is initialized as **Empty**.

You can also use the **Private** statement with empty parentheses to declare a dynamic array. After declaring a dynamic array, use the **ReDim** statement within a procedure to define the number of dimensions and elements in the array. If you try to redeclare a dimension for an array variable whose size was explicitly specified in a **Private**, **Public**, or **Dim** statement, an error occurs.

### Array Functions

### VBScript Statements

#### Example

The following example illustrates use of the **Private** statement.

```vbnet
Private MyNumber  ' Private Variant variable.
Private MyArray(9)  ' Private array variable.
' Multiple Private declarations of Variant variables.
Private MyNumber, MyVar, YourNumber
```

#### 2.5.9 Public Statement

Declares public variables and allocates storage space. Declares, in a **Class** block, a private variable.

**Syntax**

```
Public VarName([[Subscripts]]), VarName([[Subscripts]])...
```

The **Public** statement syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
</table>

---
<table>
<thead>
<tr>
<th>VarName</th>
<th>Name of the variable; follows standard variable naming conventions.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subscripts</td>
<td>Dimensions of an array variable; up to 60 multiple dimensions may be declared. The Subscripts argument uses the following syntax: upper[, upper]... The lower bound of an array is always zero.</td>
</tr>
</tbody>
</table>

**Remarks**

**Public** statement variables are available to all procedures in all scripts.

A variable that refers to an object must be assigned an existing object using the **Set** statement before it can be used. Until it is assigned an object, the declared object variable is initialized as **Empty**.

You can also use the **Public** statement with empty parentheses to declare a dynamic array. After declaring a dynamic array, use the **ReDim** statement within a procedure to define the number of dimensions and elements in the array. If you try to redeclare a dimension for an array variable whose size was explicitly specified in a **Private**, **Public**, or **Dim** statement, an error occurs.

- [ Array Functions ](#)
- [ VBScript Statements ](#)

**Example**

The following example illustrates the use of the **Public** statement.

```vb
Public MyNumber   ' Public Variant variable.
Public MyArray(9) ' Public array variable.
' Multiple Public declarations of Variant variables.
Public MyNumber, MyVar, YourNumber
```

### 2.5.10 ReDim Statement

Declares dynamic-array variables, and allocates or reallocates storage space at procedure level.
Syntax

\[ \text{ReDim /Preserve} \ VarName(\text{Subscripts})[, \ VarName(\text{Subscripts})]... \]

The \text{ReDim} statement syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{Preserve}</td>
<td>Preserves the data in an existing array when you change the size of the last dimension.</td>
</tr>
<tr>
<td>\text{VarName}</td>
<td>Name of the variable; follows standard variable naming conventions.</td>
</tr>
<tr>
<td>\text{Subscripts}</td>
<td>Dimensions of an array variable; up to 60 multiple dimensions may be declared. The \text{Subscripts} argument uses the following syntax:</td>
</tr>
<tr>
<td></td>
<td>\text{upper[, upper]}...</td>
</tr>
<tr>
<td></td>
<td>The lower bound of an array is always zero.</td>
</tr>
</tbody>
</table>

Remarks

The \text{ReDim} statement is used to size or resize a dynamic array that has already been formally declared using a \text{Private}, \text{Public}, or \text{Dim} statement with empty parentheses (without dimension subscripts). You can use the \text{ReDim} statement repeatedly to change the number of elements and dimensions in an array.

If you use the \text{Preserve} keyword, you can resize only the last array dimension, and you can’t change the number of dimensions at all. For example, if your array has only one dimension, you can resize that dimension because it is the last and only dimension. However, if your array has two or more dimensions, you can change the size of only the last dimension and still preserve the contents of the array.

The following example shows how you can increase the size of the last dimension of a dynamic array without erasing any existing data contained in the array.

\text{ReDim} \ x(10, 10, 10)

'...

\text{ReDim Preserve} \ x(10, 10, 15)
CAUTION

If you make an array smaller than it was originally, data in the eliminated elements is lost.

When variables are initialized, a numeric variable is initialized to 0 and a string variable is initialized to a zero-length string (""). A variable that refers to an object must be assigned an existing object using the Set statement before it can be used. Until it is assigned an object, the declared object variable has the special value **Nothing**.

2.5.11 Split Function

Returns a zero-based, one-dimensional array containing a specified number of substrings.

**Syntax**

\[ \text{Split}(Expression[, Delimiter[, Count[, Compare]]]) \]

The **Split** function syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expression</td>
<td>Required. String expression containing substrings and delimiters. If <em>Expression</em> is a zero-length string, <strong>Split</strong> returns an empty array, that is, an array with no elements and no data.</td>
</tr>
<tr>
<td>Delimiter</td>
<td>Optional. String character used to identify substring limits. If omitted, the space character (&quot; &quot;) is assumed to be the delimiter. If <em>Delimiter</em> is a zero-length string, a single-element array containing the entire <em>Expression</em> string is returned.</td>
</tr>
<tr>
<td>Count</td>
<td>Optional. Number of substrings to be returned; -1 indicates that all substrings are returned.</td>
</tr>
</tbody>
</table>
### Compare Argument

Optional. Numeric value indicating the kind of comparison to use when evaluating substrings. See the Settings section for values.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbBinaryCompare</td>
<td>0</td>
<td>Perform a binary comparison.</td>
</tr>
<tr>
<td>vbTextCompare</td>
<td>1</td>
<td>Perform a textual comparison.</td>
</tr>
</tbody>
</table>

### Settings

The Compare argument can have the following values:

### Example

The following example uses the `Split` function to return an array from a string. The function performs a textual comparison of the delimiter, and returns all of the substrings.

```vbscript
Dim MyString, MyArray
MyString = Split("VBScriptXisXfun!", "x", -1, 1) ' MyString(0) contains "VBScript".
       ' MyString(1) contains "is".
       ' MyString(2) contains "fun!".
```

### 2.5.12 UBound Function

Returns the largest available subscript for the indicated dimension of an array.

#### Syntax

```vbscript
UBound(ArrayName[, Dimension])
```

The `UBound` function syntax has these parts:
<table>
<thead>
<tr>
<th>ArrayName</th>
<th>Required. Name of the array variable; follows standard variable naming conventions.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension</td>
<td>Optional. Whole number indicating which dimension's upper bound is returned. Use 1 for the first dimension, 2 for the second, and so on. If Dimension is omitted, 1 is assumed.</td>
</tr>
</tbody>
</table>

**Remarks**

The **UBound** function is used with the **LBound** function to determine the size of an array. Use the **LBound** function to find the lower limit of an array dimension.

The lower bound for any dimension is always 0. As a result, **UBound** returns the following values for an array with these dimensions:

```
Dim A(100, 3, 4)
```

<table>
<thead>
<tr>
<th>Statement</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>UBound(A, 1)</td>
<td>100</td>
</tr>
<tr>
<td>UBound(A, 2)</td>
<td>3</td>
</tr>
<tr>
<td>UBound(A, 3)</td>
<td>4</td>
</tr>
</tbody>
</table>
2.6 String Functions

- Asc Function
- Chr Function
- Escape Function
- FormatCurrency Function
- FormatDateTime Function
- FormatNumber Function
- FormatPercent Function
- InStr Function
- InStrRev Function
- LCase Function
- Left Function
- Len Function
- LTrim Function
- MonthName Function
- Mid Function
- Replace Function
- Right Function
- RTrim Function
- Space Function
- StrComp Function
- String Function
- StrReverse Function
- Trim Function
- UCase Function
- Unescape Function
- WeekDayName Function
2.6.1 Asc Function

Returns the ANSI character code corresponding to the first letter in a string.

Syntax

\[ \text{Asc} \left( \text{String} \right) \]

The \textit{String} argument is any valid string expression. If the \textit{String} contains no characters, a run-time error occurs.

\textbf{NOTE}

The AscB function is used with byte data contained in a string. Instead of returning the character code for the first character, AscB returns the first byte. AscW is provided for 32-bit platforms that use Unicode characters. It returns the Unicode (wide) character code, thereby avoiding the conversion from Unicode to ANSI.

Example

In the following example, Asc returns the ANSI character code of the first letter of each string.

\begin{verbatim}
Dim MyNumber
MyNumber = Asc("A")  ' Returns 65.
MyNumber = Asc("a")  ' Returns 97.
MyNumber = Asc("Apple")  ' Returns 65.
\end{verbatim}

2.6.2 Chr Function

Returns the character associated with the specified ANSI character code.

Syntax

\[ \text{Chr} \left( \text{CharCode} \right) \]

The \textit{CharCode} argument is a number that identifies a character.
Remarks

Numbers from 0 to 31 are the same as standard, nonprintable ASCII codes. For example, \texttt{Chr(10)} returns a linefeed character.

\textbf{NOTE}

The \texttt{ChrB} function is used with byte data contained in a string. Instead of returning a character, which may be one or two bytes, \texttt{ChrB} always returns a single byte. \texttt{ChrW} is provided for 32-bit platforms that use Unicode characters. Its argument is a Unicode (wide) character code, thereby avoiding the conversion from ANSI to Unicode.

\textbf{Example}

The following example uses the \texttt{Chr} function to return the character associated with the specified character code.

\begin{verbatim}
Dim MyChar
MyChar = Chr(65) ' Returns A
MyChar = Chr(97) ' Returns a.
MyChar = Chr(62) ' Returns >.
MyChar = Chr(37) ' Returns %.
\end{verbatim}

\textbf{2.6.3 Escape Function}

Encodes a string so it contains only ASCII characters.

\textbf{Syntax}

\begin{verbatim}
Escape(String)
\end{verbatim}

The \textit{String} argument is any valid string expression. If \textit{String} contains \texttt{Null}, \texttt{Null} is returned.
Remarks

The **Escape** function returns a string (in Unicode format) that contains the contents of String. All spaces, punctuation, accented characters, and other non-ASCII characters are replaced with \%xx encoding, where xx is equivalent to the hexadecimal number representing the character. Unicode characters that have a value greater than 255 are stored using the \%uxxxx format. The string that the Escape function returns is suitable for transmission with many protocols, such as the HTTP protocol.

Example

The following example uses the **Escape** function to encode a string.

```vbnet
Dim Src, Dest
Src = "Ahoj Světe 1234" ' Text to encode.
Dest = Escape(Src)   ' Returns "Ahoj%20Sv%u011Bte%201234".
```

### 2.6.4 FormatCurrency Function

Returns an expression formatted as a currency value using the currency symbol defined in the system control panel.

**Syntax**

```
FormatCurrency(Expression[, NumDigitsAfterDecimal[, IncludeLeadingDigit[, UseParensForNegativeNumbers[, GroupDigits]]]])
```

The **FormatCurrency** function syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Expression</code></td>
<td>Required. Expression to be formatted.</td>
</tr>
<tr>
<td><code>NumDigitsAfterDecimal</code></td>
<td>Optional. Numeric value indicating how many places to the right of the decimal are displayed. Default value is -1, which indicates that the computer's regional settings are used.</td>
</tr>
</tbody>
</table>
### String Functions

#### IncludeLeadingDigit

Optional. Tristate constant that indicates whether or not a leading zero is displayed for fractional values. See the Settings section for values.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbTrue</td>
<td>-1</td>
<td>True</td>
</tr>
<tr>
<td>vbFalse</td>
<td>0</td>
<td>False</td>
</tr>
<tr>
<td>vbUseDefault</td>
<td>-2</td>
<td>Use the setting from the computer's regional settings.</td>
</tr>
</tbody>
</table>

#### UseParensForNegativeNumbers

Optional. Tristate constant that indicates whether or not to place negative values within parentheses. See the Settings section for values.

#### GroupDigits

Optional. Tristate constant that indicates whether or not numbers are grouped using the group delimiter specified in the computer's regional settings. See the Settings section for values.

### Settings

The `IncludeLeadingDigit`, `UseParensForNegativeNumbers`, and `GroupDigits` arguments have the following settings:

#### Remarks

When one or more optional arguments are omitted, values for omitted arguments are provided by the computer's regional settings. The position of the currency symbol relative to the currency value is determined by the system's regional settings.

### Example

The following example uses the `FormatCurrency` function to format the expression as a currency and assign it to MyCurrency.
Dim MyCurrency
MyCurrency = FormatCurrency(1000) ' MyCurrency contains $1000.00.

2.6.5 FormatDateTime Function

Returns an expression formatted as a date or time.

Syntax

FormatDateTime(Date[, NamedFormat])

The FormatDateTime function syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>Required. Date expression to be formatted.</td>
</tr>
<tr>
<td>NamedFormat</td>
<td>Optional. Numeric value that indicates the date/time format used. If omitted, vbGeneralDate is used.</td>
</tr>
</tbody>
</table>

Settings

The NamedFormat argument has the following settings:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbGeneralDate</td>
<td>0</td>
<td>Display a date and/or time. If there is a date part, display it as a short date. If there is a time part, display it as a long time. If present, both parts are displayed.</td>
</tr>
<tr>
<td>vbLongDate</td>
<td>1</td>
<td>Display a date using the long date format specified in your computer's regional settings.</td>
</tr>
<tr>
<td>vbShortDate</td>
<td>2</td>
<td>Display a date using the short date format specified in your computer's regional settings.</td>
</tr>
<tr>
<td>Function</td>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>vbLongTime</td>
<td>3</td>
<td>Display a time using the time format specified in your computer's regional settings.</td>
</tr>
<tr>
<td>vbShortTime</td>
<td>4</td>
<td>Display a time using the 24-hour format (hh:mm).</td>
</tr>
</tbody>
</table>

**Example**

The following example uses the **FormatDateTime** function to format the expression as a long date and assign it to MyDateTime.

```vbscript
Function GetCurrentDate
    ' FormatDateTime formats Date in long date.
    GetCurrentDate = FormatDateTime(Date, 1)
End Function
```

### 2.6.6 FormatNumber Function

Returns an expression formatted as a number.

**Syntax**

```vbscript
FormatNumber(Expression[, NumDigitsAfterDecimal[, IncludeLeadingDigit[, UseParensForNegativeNumbers[, GroupDigits]]]])
```

The **FormatNumber** function syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expression</td>
<td>Required. Expression to be formatted.</td>
</tr>
<tr>
<td>NumDigitsAfterDecimal</td>
<td>Optional. Numeric value indicating how many places to the right of the decimal are displayed. Default value is -1, which indicates that the computer's regional settings are used.</td>
</tr>
</tbody>
</table>
**IncludeLeadingDigit**
Optional. Tristate constant that indicates whether or not a leading zero is displayed for fractional values. See the Settings section for values.

**UseParensForNegativeNumbers**
Optional. Tristate constant that indicates whether or not to place negative values within parentheses. See the Settings section for values.

**GroupDigits**
Optional. Tristate constant that indicates whether or not numbers are grouped using the group delimiter specified in the control panel. See the Settings section for values.

### Settings
The **IncludeLeadingDigit**, **UseParensForNegativeNumbers**, and **GroupDigits** arguments have the following settings:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbTrue</td>
<td>-1</td>
<td>True</td>
</tr>
<tr>
<td>vbFalse</td>
<td>0</td>
<td>False</td>
</tr>
<tr>
<td>vbUseDefault</td>
<td>-2</td>
<td>Use the setting from the computer's regional settings.</td>
</tr>
</tbody>
</table>

### Remarks
When one or more of the optional arguments are omitted, the values for omitted arguments are provided by the computer's regional settings.
Example

The following example uses the `FormatNumber` function to format a number to have four decimal places.

```vba
Function FormatNumberDemo
    Dim MyAngle, MySecant, MyNumber
    MyAngle = 1.3 ' Define angle in radians.
    MySecant = 1 / Cos(MyAngle) ' Calculate secant.
    ' Format MySecant to four decimal places.
    FormatNumberDemo = FormatNumber(MySecant, 4)
End Function
```

2.6.7 FormatPercent Function

Returns an expression formatted as a percentage (multiplied by 100) with a trailing % character.

Syntax

```vba
FormatPercent(Expression[, NumDigitsAfterDecimal[, IncludeLeadingDigit[, UseParensForNegativeNumbers[, GroupDigits]]]])
```

The `FormatPercent` function syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Expression</code></td>
<td>Required. Expression to be formatted.</td>
</tr>
<tr>
<td><code>NumDigitsAfterDecimal</code></td>
<td>Optional. Numeric value indicating how many places to the right of the decimal are displayed. Default value is -1, which indicates that the computer's regional settings are used.</td>
</tr>
<tr>
<td><code>IncludeLeadingDigit</code></td>
<td>Optional. Tristate constant that indicates whether or not a leading zero is displayed for fractional values. See the Settings section for values.</td>
</tr>
</tbody>
</table>
**String Functions**

<table>
<thead>
<tr>
<th>UseParensForNegativeNumbers</th>
<th>Optional. Tristate constant that indicates whether or not to place negative values within parentheses. See the Settings section for values.</th>
</tr>
</thead>
<tbody>
<tr>
<td>GroupDigits</td>
<td>Optional. Tristate constant that indicates whether or not numbers are grouped using the group delimiter specified in the control panel. See the Settings section for values.</td>
</tr>
</tbody>
</table>

**Settings**

The *IncludeLeadingDigit*, *UseParensForNegativeNumbers*, and *GroupDigits* arguments have the following settings:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbTrue</td>
<td>-1</td>
<td>True</td>
</tr>
<tr>
<td>vbFalse</td>
<td>0</td>
<td>False</td>
</tr>
<tr>
<td>vbUseDefault</td>
<td>-2</td>
<td>Use the setting from the computer's regional settings.</td>
</tr>
</tbody>
</table>

**Remarks**

When one or more optional arguments are omitted, the values for the omitted arguments are provided by the computer's regional settings.

**Example**

The following example uses the `FormatPercent` function to format an expression as a percent.

```vba
Dim MyPercent
MyPercent = FormatPercent(2 / 32) ' MyPercent contains 6.25%.
```
2.6.8 InStr Function

Returns the position of the first occurrence of one string within another.

Syntax

\[
\text{InStr}([\text{Start},] \text{String1}, \text{String2}[\text{, Compare}])
\]

The `InStr` function syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>Optional. Numeric expression that sets the starting position for each search. If omitted, search begins at the first character position. If <code>Start</code> contains <code>Null</code>, an error occurs. The <code>Start</code> argument is required if <code>Compare</code> is specified.</td>
</tr>
<tr>
<td>String1</td>
<td>Required. String expression being searched.</td>
</tr>
<tr>
<td>String2</td>
<td>Required. String expression searched for.</td>
</tr>
<tr>
<td>Compare</td>
<td>Optional. Numeric value indicating the kind of comparison to use when evaluating substrings. See the Settings section for values. If omitted, a binary comparison is performed.</td>
</tr>
</tbody>
</table>

**Settings**

The `Compare` argument can have the following values:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>vbBinaryCompare</code></td>
<td>0</td>
<td>Perform a binary comparison.</td>
</tr>
<tr>
<td><code>vbTextCompare</code></td>
<td>1</td>
<td>Perform a textual comparison.</td>
</tr>
</tbody>
</table>

**Return Values**

The `InStr` function returns the following values:

<table>
<thead>
<tr>
<th>If</th>
<th>InStr returns</th>
</tr>
</thead>
</table>
### String Functions

<table>
<thead>
<tr>
<th>String1 is zero-length</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>String1 is <strong>Null</strong></td>
<td>Null</td>
</tr>
<tr>
<td>String2 is zero-length</td>
<td>Start</td>
</tr>
<tr>
<td>String2 is <strong>Null</strong></td>
<td>Null</td>
</tr>
<tr>
<td>String2 is not found</td>
<td>0</td>
</tr>
<tr>
<td>String2 is found within String1</td>
<td>Position at which match is found.</td>
</tr>
<tr>
<td>Start &gt; <strong>Len</strong>(String2)</td>
<td>0</td>
</tr>
</tbody>
</table>

#### NOTE

The **InStrB** function is used with byte data contained in a string. Instead of returning the character position of the first occurrence of one string within another, **InStrB** returns the byte position.

#### Example

The following examples use **InStr** to search a string.

```plaintext
Dim SearchString, SearchChar, MyPos
SearchString = "XXpXXpXXpXXP" ' String to search in.
SearchChar = "p" ' Search for "p".
' A textual comparison starting at position 4.
' Returns 6.
MyPos = Instr(4, SearchString, SearchChar, 1)
' A binary comparison starting at position 1.
' Returns 9.
MyPos = Instr(1, SearchString, SearchChar, 0)
' Comparison is binary by default (last argument is omitted).
' Returns 9.
MyPos = Instr(SearchString, SearchChar)
' A binary comparison starting at position 1.
' Returns 0 ("W" is not found).
MyPos = Instr(1, SearchString, "W")
```
2.6.9 InStrRev Function

Returns the position of an occurrence of one string within another, from the end of string.

Syntax

\[ \text{InStrRev} \left( \text{String1, String2[, Start[, Compare]]} \right) \]

The InStrRev function syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String1</td>
<td>Required. String expression being searched.</td>
</tr>
<tr>
<td>String2</td>
<td>Required. String expression being searched for.</td>
</tr>
<tr>
<td>Start</td>
<td>Optional. Numeric expression that sets the starting position for each search. If omitted, -1 is used, which means that the search begins at the last character position. If Start contains Null, an error occurs.</td>
</tr>
<tr>
<td>Compare</td>
<td>Optional. Numeric value indicating the kind of comparison to use when evaluating substrings. If omitted, a binary comparison is performed. See the Settings section for values.</td>
</tr>
</tbody>
</table>

Settings

The Compare argument can have the following values:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbBinaryCompare</td>
<td>0</td>
<td>Perform a binary comparison.</td>
</tr>
<tr>
<td>vbTextCompare</td>
<td>1</td>
<td>Perform a textual comparison.</td>
</tr>
</tbody>
</table>

Return Values

InStrRev returns the following values:

<table>
<thead>
<tr>
<th>If</th>
<th>InStrRev returns</th>
</tr>
</thead>
</table>
String Functions

<table>
<thead>
<tr>
<th>String1 is zero-length</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>String1 is Null</td>
<td>Null</td>
</tr>
<tr>
<td>String2 is zero-length</td>
<td>Start</td>
</tr>
<tr>
<td>String2 is Null</td>
<td>Null</td>
</tr>
<tr>
<td>String2 is not found</td>
<td>0</td>
</tr>
<tr>
<td>String2 is found within String1</td>
<td>Position at which match is found.</td>
</tr>
<tr>
<td>Start &gt; Len(String2)</td>
<td>0</td>
</tr>
</tbody>
</table>

**NOTE**

The syntax for the `InStrRev` function is not the same as the syntax for the `InStr` function.

**Example**

The following examples use the `InStrRev` function to search a string.

```vbs
Dim SearchString, SearchChar, MyPos
SearchString = "XXpXXpXXPXXP" ' String to search in.
SearchChar = "P" ' Search for "P".
' A binary comparison starting at position 10.
' Returns 9.
MyPos = InstrRev(SearchString, SearchChar, 10, 0)
' A textual comparison starting at the last position.
' Returns 12.
MyPos = InstrRev(SearchString, SearchChar, -1, 1)
' Comparison is binary by default (last argument is omitted).
' Returns 0.
MyPos = InstrRev(SearchString, SearchChar, 8)
```

**2.6.10 LCase Function**

Returns a string that has been converted to lowercase.
String Functions

Syntax

LCase(String)

The String argument is any valid string expression. If String contains Null, Null is returned.

Remarks

Only uppercase letters are converted to lowercase; all lowercase letters and nonletter characters remain unchanged.

Example

The following example uses the LCase function to convert uppercase letters to lowercase.

```
Dim MyString
Dim LCaseString
MyString = "VBScript"
LCaseString = LCase(MyString) ' LCaseString contains "vbscript".
```

2.6.11 Left Function

Returns a specified number of characters from the left side of a string.

Syntax

Left(String, Length)

The Left function syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>String expression from which the leftmost characters are returned. If String contains Null, Null is returned.</td>
</tr>
</tbody>
</table>
## String Functions

| **Length** | Numeric expression indicating how many characters to return. If 0, a zero-length string("") is returned. If greater than or equal to the number of characters in **String**, the entire string is returned. |

### Remarks

To determine the number of characters in **String**, use the **Len** function.

### NOTE

The **LeftB** function is used with byte data contained in a string. Instead of specifying the number of characters to return, **Length** specifies the number of bytes.

### Example

The following example uses the **Left** function to return the first three characters of **MyString**.

```vbs
Dim MyString, LeftString
MyString = "VBScript"
LeftString = Left(MyString, 3) ' LeftString contains "VBS".
```

### 2.6.12 Len Function

Returns the number of characters in a string or the number of bytes required to store a variable.

#### Syntax

```vbs
Len(String | VarName)
```

The **Len** function syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>String</strong></td>
<td>Any valid string expression. If <strong>String</strong> contains <strong>Null</strong>, <strong>Null</strong> is returned.</td>
</tr>
</tbody>
</table>
**VarName**  | Any valid variable name. If VarName contains **Null**, **Null** is returned.

**NOTE**
The **LenB** function is used with byte data contained in a string. Instead of returning the number of characters in a string, **LenB** returns the number of bytes used to represent that string.

---

### String Functions

**Example**
The following example uses the **Len** function to return the number of characters in a string.

```vb
Dim MyString
MyString = Len("VBSCRIPT") ' MyString contains 8.
```

#### 2.6.13 LTrim Function

Returns a copy of a string without leading spaces (**RTrim** without trailing spaces, **Trim** without both leading and trailing spaces).

**Syntax**

```
LTrim(String)
```

The **String** argument is any valid string expression. If **String** contains **Null**, **Null** is returned.

---

**Example**
The following example uses the **LTrim**, **RTrim**, and **Trim** functions to trim leading spaces, trailing spaces, and both leading and trailing spaces, respectively.

```vb
Dim MyVar
```
MyVar = LTrim("	 vbscript ")  ' MyVar contains "vbscript ".
MyVar = RTrim("  vbscript")  ' MyVar contains "  vbscript".
MyVar = Trim(" vbscript ")  ' MyVar contains "vbscript".

### 2.6.14 Mid Function

Returns a specified number of characters from a string.

**Syntax**

```
Mid(String, Start[, Length])
```

The `Mid` function syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>String expression from which characters are returned. If <code>String</code> contains <code>Null</code>, <code>Null</code> is returned.</td>
</tr>
<tr>
<td>Start</td>
<td>Character position in <code>String</code> at which the part to be taken begins. If <code>Start</code> is greater than the number of characters in <code>String</code>, <code>Mid</code> returns a zero-length string (&quot;&quot;&quot;).</td>
</tr>
<tr>
<td>Length</td>
<td>Number of characters to return. If omitted or if there are fewer than <code>Length</code> characters in the text (including the character at <code>Start</code>), all characters from the <code>Start</code> position to the end of the string are returned.</td>
</tr>
</tbody>
</table>

**Remarks**

To determine the number of characters in `String`, use the `Len` function.

**NOTE**

The `MidB` function is used with byte data contained in a string. Instead of specifying the number of characters, the arguments specify numbers of bytes.
Example

The following example uses the `Mid` function to return six characters, beginning with the fourth character, in a string.

```vbnet
Dim MyVar
MyVar = Mid("VB Script is fun!", 4, 6) ' MyVar contains "Script".
```

2.6.15 MonthName Function

Returns a string indicating the specified month.

Syntax

```vbnet
MonthName(Month[, Abbreviate])
```

The `MonthName` function syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Month</code></td>
<td>Required. The numeric designation of the month. For example, January is 1, February is 2, and so on.</td>
</tr>
<tr>
<td><code>Abbreviate</code></td>
<td>Optional. Boolean value that indicates if the month name is to be abbreviated. If omitted, the default is <code>False</code>, which means that the month name is not abbreviated.</td>
</tr>
</tbody>
</table>

Example

The following example uses the `MonthName` function to return an abbreviated month name for a date expression.

```vbnet
Dim MyVar
MyVar = MonthName(10, True) ' MyVar contains "Oct".
```
2.6.16 Replace Function

Returns a string in which a specified substring has been replaced with another substring a specified number of times.

Syntax

Replace(Expression, Find, ReplaceWith[, Start[, Count[, Compare]]])

The Replace function syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expression</td>
<td>Required. String expression containing substring to replace.</td>
</tr>
<tr>
<td>Find</td>
<td>Required. Substring being searched for.</td>
</tr>
<tr>
<td>ReplaceWith</td>
<td>Required. Replacement substring.</td>
</tr>
<tr>
<td>Start</td>
<td>Optional. Position within Expression where substring search is to begin. If omitted, 1 is assumed. Must be used in conjunction with Count.</td>
</tr>
<tr>
<td>Count</td>
<td>Optional. Number of substring substitutions to perform. If omitted, the default value is -1, which means make all possible substitutions. Must be used in conjunction with Start.</td>
</tr>
<tr>
<td>Compare</td>
<td>Optional. Numeric value indicating the kind of comparison to use when evaluating substrings. See the Settings section for values. If omitted, the default value is 0, which means perform a binary comparison.</td>
</tr>
</tbody>
</table>

Settings

The Compare argument can have the following values:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbBinaryCompare</td>
<td>0</td>
<td>Perform a binary comparison.</td>
</tr>
<tr>
<td>vbCrLfTextCompare</td>
<td>1</td>
<td>Perform a textual comparison.</td>
</tr>
</tbody>
</table>
Return Values

**Replace** returns the following values:

<table>
<thead>
<tr>
<th>If</th>
<th>Replace returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expression is zero-length</td>
<td>Zero-length string (&quot;&quot;).</td>
</tr>
<tr>
<td>Expression is Null</td>
<td>An error.</td>
</tr>
<tr>
<td>Find is zero-length</td>
<td>Copy of Expression.</td>
</tr>
<tr>
<td>ReplaceWith is zero-length</td>
<td>Copy of Expression with all occurrences of Find removed.</td>
</tr>
<tr>
<td>Start &gt; Len(Expression)</td>
<td>Zero-length string.</td>
</tr>
<tr>
<td>Count is 0</td>
<td>Copy of Expression.</td>
</tr>
</tbody>
</table>

Remarks

The return value of the **Replace** function is a string, with substitutions made, that begins at the position specified by **Start** and and concludes at the end of the **Expression** string. It is not a copy of the original string from start to finish.

Example

The following example uses the **Replace** function to return a string.

```
Dim MyString
' A binary comparison starting at the beginning of the string.
' Returns "XXYXXPXYX".
MyString = Replace("XXpXXPXXp", "p", "Y")
' A textual comparison starting at position 3.
' Returns "YYYXXXY".
MyString = Replace("XXpXXPXXp", "p", "Y", 3, -1, 1)
```
2.6.17 Right Function

Returns a specified number of characters from the right side of a string.

Syntax

\[ \text{Right} (\text{String}, \text{Length}) \]

The \textbf{Right} function syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>String expression from which the rightmost characters are returned. If \textit{String} contains Null, Null is returned.</td>
</tr>
<tr>
<td>Length</td>
<td>Numeric expression indicating how many characters to return. If 0, a zero-length string is returned. If greater than or equal to the number of characters in \textit{String}, the entire string is returned.</td>
</tr>
</tbody>
</table>

Remarks

To determine the number of characters in \textit{String}, use the \textbf{Len} function.

\textbf{NOTE}

The \textbf{RightB} function is used with byte data contained in a string. Instead of specifying the number of characters to return, \textbf{Length} specifies the number of bytes.

\section*{Example}

The following example uses the \textbf{Right} function to return a specified number of characters from the right side of a string.

\begin{verbatim}
Dim AnyString, MyStr
AnyString = "Hello World"   ' Define string.
MyStr = Right(AnyString, 1) ' Returns "d".
MyStr = Right(AnyString, 6) ' Returns " World".
MyStr = Right(AnyString, 20) ' Returns "Hello World".
\end{verbatim}
2.6.18 \textbf{RTrim Function}

Returns a copy of a string without trailing spaces (\texttt{LTrim} without leading spaces, \texttt{Trim} without both leading and trailing spaces).

\textbf{Syntax}

\begin{verbatim}
RTrim(String)
\end{verbatim}

The \textit{String} argument is any valid string expression. If \textit{String} contains \texttt{Null}, \texttt{Null} is returned.

\textbf{Example}

The following example uses the \texttt{LTrim}, \texttt{RTrim}, and \texttt{Trim} functions to trim leading spaces, trailing spaces, and both leading and trailing spaces, respectively.

\begin{verbatim}
Dim MyVar
MyVar = LTrim(" vbscript ") ' MyVar contains "vbscript ".
MyVar = RTrim(" vbscript ") ' MyVar contains " vbscript".
MyVar = Trim(" vbscript ") ' MyVar contains "vbscript".
\end{verbatim}

2.6.19 \textbf{Space Function}

Returns a string consisting of the specified number of spaces.

\textbf{Syntax}

\begin{verbatim}
Space(Number)
\end{verbatim}

The \textit{Number} argument is the number of spaces you want in the string.

\textbf{Example}

The following example uses the \texttt{Space} function to return a string consisting of a specified number of spaces.
Dim MyString
' Returns a string with 10 spaces.
MyString = Space(10)
' Insert 10 spaces between two strings.
MyString = "Hello" & Space(10) & "World"

2.6.20 StrComp Function

Returns a value indicating the result of a string comparison.

Syntax

StrComp(String1, String2[, Compare])

The StrComp function syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String1</td>
<td>Required. Any valid string expression.</td>
</tr>
<tr>
<td>String2</td>
<td>Required. Any valid string expression.</td>
</tr>
<tr>
<td>Compare</td>
<td>Optional. Numeric value indicating the kind of</td>
</tr>
<tr>
<td></td>
<td>comparison to use when evaluating strings.</td>
</tr>
<tr>
<td></td>
<td>If omitted, a binary comparison is performed.</td>
</tr>
<tr>
<td></td>
<td>See the Settings section for values.</td>
</tr>
</tbody>
</table>

Settings

The Compare argument can have the following values:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbBinaryCompare</td>
<td>0</td>
<td>Perform a binary comparison.</td>
</tr>
<tr>
<td>vbTextCompare</td>
<td>1</td>
<td>Perform a textual comparison.</td>
</tr>
</tbody>
</table>

Return Values

The StrComp function has the following return values:

<table>
<thead>
<tr>
<th>If</th>
<th>StrComp returns</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
String Functions

<table>
<thead>
<tr>
<th>Condition</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>String1 is less than String2</td>
<td>-1</td>
</tr>
<tr>
<td>String1 is equal to String2</td>
<td>0</td>
</tr>
<tr>
<td>String1 is greater than String2</td>
<td>1</td>
</tr>
<tr>
<td>String1 or String2 is Null</td>
<td>Null</td>
</tr>
</tbody>
</table>

Example

The following example uses the `StrComp` function to return the results of a string comparison. If the third argument is 1, a textual comparison is performed; if the third argument is 0 or omitted, a binary comparison is performed.

```vbscript
Dim MyStr1, MyStr2, MyComp
MyStr1 = "ABCD"
MyStr2 = "abcd"  ' Define variables.
MyComp = StrComp(MyStr1, MyStr2, 1)  ' Returns 0.
MyComp = StrComp(MyStr1, MyStr2, 0)  ' Returns -1.
MyComp = StrComp(MyStr2, MyStr1)     ' Returns 1.
```

2.6.21 String Function

Returns a repeating character string of the length specified.

Syntax

`String(Number, Character)`

The `String` function syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>Length of the returned string. If <code>Number</code> contains <code>Null</code>, <code>Null</code> is returned.</td>
</tr>
<tr>
<td>Character</td>
<td>Character code specifying the character or string expression whose first character is used to build the return string. If <code>Character</code> contains <code>Null</code>, <code>Null</code> is returned.</td>
</tr>
</tbody>
</table>
**Remarks**

If you specify a number for *Character* greater than 255, *String* converts the number to a valid character code using the formula:

\[ \text{character} \mod 256 \]

**Example**

The following example uses the *String* function to return repeating character strings of the length specified.

```vbscript
Dim MyString
MyString = String(5, "*")  ' Returns "*****".
MyString = String(5, 42)   ' Returns "*****".
MyString = String(10, "ABC")  ' Returns "AAAAAAAAAA".
```

### 2.6.22 StrReverse Function

Returns a string in which the character order of a specified string is reversed.

**Syntax**

\[ \text{StrReverse(String)} \]

The *String* argument is the string whose characters are to be reversed. If *String* is a zero-length string (""), a zero-length string is returned. If *String* is *Null*, an error occurs.

**Example**

The following example uses the *StrReverse* function to return a string in reverse order.

```vbscript
Dim MyStr
MyStr = StrReverse("VBScript")  ' MyStr contains "tpircSBV".
```
2.6.23 Trim Function

Returns a copy of a string without both leading and trailing spaces (LTrim without leading spaces, RTrim without trailing spaces).

Syntax

Trim(String)

The String argument is any valid string expression. If String contains Null, Null is returned.

Example

The following example uses the LTrim, RTrim, and Trim functions to trim leading spaces, trailing spaces, and both leading and trailing spaces, respectively.

Dim MyVar
MyVar = LTrim("  vbscript ") ' MyVar contains "vbscript ".
MyVar = RTrim("  vbscript ") ' MyVar contains " vbscript".
MyVar = Trim("  vbscript ") ' MyVar contains "vbscript".

2.6.24 UCase Function

Returns a string that has been converted to uppercase.

Syntax

UCase(String)

The String argument is any valid string expression. If String contains Null, Null is returned.

Remarks

Only lowercase letters are converted to uppercase; all uppercase letters and nonletter characters remain unchanged.
Conversion Functions

Example

The following example uses the **UCase** function to return an uppercase version of a string.

```vbnet
Dim MyWord
MyWord = UCase("Hello World") ' Returns "HELLO WORLD".
```

2.6.25 Unescape Function

Decodes a string encoded with the *Escape* function.

**Syntax**

```
Unescape(String)
```

The *String* argument is any valid string expression. If *String* contains **Null**, **Null** is returned.

**Remarks**

The **Unescape** function returns a string (in Unicode format) that contains the contents of *String*. ASCII character set equivalents replace all characters encoded with the %xx hexadecimal form. Characters encoded in %uxxxxx format (Unicode characters) are replaced with the Unicode character with hexadecimal encoding xxxx.

String Functions

Example

The following example uses the **Unescape** function to decode a string.

```vbnet
Dim Src, Dest
Src = "Ahoj%20Sv%u011Bte%201234" ' Text to decode.
Dest = Unescape(Src) ' Returns "Ahoj Světe 1234".
```
2.6.26 WeekdayName Function

Returns a string indicating the specified day of the week.

**Syntax**

```vba
WeekdayName(WeekDay[, Abbreviate[, FirstDayOfWeek]])
```

The `WeekdayName` function syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>WeekDay</code></td>
<td>Required. The numeric designation for the day of the week. Numeric value of each day depends on setting of the <code>FirstDayOfWeek</code> setting.</td>
</tr>
<tr>
<td><code>Abbreviate</code></td>
<td>Optional. Boolean value that indicates if the weekday name is to be abbreviated. If omitted, the default is <code>False</code>, which means that the weekday name is not abbreviated.</td>
</tr>
<tr>
<td><code>FirstDayOfWeek</code></td>
<td>Optional. Numeric value indicating the first day of the week. See the Settings section for values.</td>
</tr>
</tbody>
</table>

**Settings**

The `FirstDayOfWeek` argument can have the following values:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>vbUseSystem</code></td>
<td>0</td>
<td>Use National Language Support (NLS) API setting.</td>
</tr>
<tr>
<td><code>vbSunday</code></td>
<td>1</td>
<td>Sunday</td>
</tr>
<tr>
<td><code>vbMonday</code></td>
<td>2</td>
<td>Monday</td>
</tr>
<tr>
<td><code>vbTuesday</code></td>
<td>3</td>
<td>Tuesday</td>
</tr>
<tr>
<td><code>vbWednesday</code></td>
<td>4</td>
<td>Wednesday</td>
</tr>
<tr>
<td><code>vbThursday</code></td>
<td>5</td>
<td>Thursday</td>
</tr>
</tbody>
</table>


Example
The following example uses the **WeekDayName** function to return the specified day.

```vb
Dim MyDate
MyDate = WeekDayName(6, True) ' MyDate contains Fri.
```
2.7 Conversion Functions

- Asc Function
- CBool Function
- CByte Function
- CCur Function
- CDate Function
- CDbI Function
- Chr Function
- CInt Function
- CLng Function
- CSng Function
- CStr Function
- DateSerial Function
- DateValue Function
- Day Function
- Escape Function
- Fix Function
- Hex Function
- Hour Function
- Int Function
- LCase Function
- Minute Function
- Month Function
- Oct Function
- Second Function
- TimeSerial Function
- TimeValue Function
- UCase Function
Unescape Function

WeekDay Function

Year Function

2.7.1 Asc Function

Returns the ANSI character code corresponding to the first letter in a string.

Syntax

```
Asc(String)
```

The String argument is any valid string expression. If the String contains no characters, a run-time error occurs.

NOTE

The AscB function is used with byte data contained in a string. Instead of returning the character code for the first character, AscB returns the first byte. AscW is provided for 32-bit platforms that use Unicode characters. It returns the Unicode (wide) character code, thereby avoiding the conversion from Unicode to ANSI.

Example

In the following example, Asc returns the ANSI character code of the first letter of each string.

```
Dim MyNumber
MyNumber = Asc("A")    ' Returns 65.
MyNumber = Asc("a")    ' Returns 97.
MyNumber = Asc("Apple") ' Returns 65.
```

2.7.2 CBool Function

Returns an expression that has been converted to a Variant of subtype Boolean.
**Syntax**

\[ \text{CBool}(\text{Expression}) \]

The \textit{Expression} argument is any valid expression.

**Remarks**

If \textit{Expression} is zero, \textbf{False} is returned; otherwise, \textbf{True} is returned. If \textit{Expression} can't be interpreted as a numeric value, a run-time error occurs.

**Example**

The following example uses the \textbf{CBool} function to convert an expression to a \texttt{Boolean}. If the expression evaluates to a nonzero value, \textbf{CBool} returns \textbf{True}; otherwise, it returns \textbf{False}.

```vbnet
Dim A, B, Check
A = 5
B = 5 ' Initialize variables.
Check = CBool(A = B) ' Check contains True.
A = 0 ' Define variable.
Check = CBool(A) ' Check contains False.
```

**2.7.3 CByte Function**

Returns an expression that has been converted to a \texttt{Variant} of subtype \texttt{Byte}.

**Syntax**

\[ \text{CByte}(\text{Expression}) \]

The \textit{Expression} argument is any valid expression.
Remarks

In general, you can document your code using the subtype conversion functions to show that the result of some operation should be expressed as a particular data type rather than the default data type. For example, use **CByte** to force byte arithmetic in cases where currency, single-precision, double-precision, or integer arithmetic normally would occur.

Use the **CByte** function to provide internationally aware conversions from any other data type to a **Byte** subtype. For example, different decimal separators are properly recognized depending on the locale setting of your system, as are different thousand separators.

If Expression lies outside the acceptable range for the **Byte** subtype, an error occurs.

### Example

The following example uses the **CByte** function to convert an expression to a byte.

```vbnet
Dim MyDouble, MyByte
MyDouble = 125.5678  ' MyDouble is a Double.
MyByte = CByte(MyDouble)  ' MyByte contains 126.
```

### 2.7.4 **CCur** Function

Returns an expression that has been converted to a **Variant** of subtype **Currency**.

#### Syntax

```
CCur(Expression)
```

The **Expression** argument is any valid expression.

#### Remarks

In general, you can document your code using the subtype conversion functions to show that the result of some operation should be expressed as a particular data type rather than the default data type. For example, use **CCur** to force currency arithmetic in cases where integer arithmetic normally would occur.
You should use the `CCur` function to provide internationally aware conversions from any other data type to a `Currency` subtype. For example, different decimal separators and thousands separators are properly recognized depending on the locale setting of your system.

**Example**

The following example uses the `CCur` function to convert an expression to a Currency.

```vbs
Dim MyDouble, MyCurr
MyDouble = 543.214588 ' MyDouble is a Double.
' Convert result of MyDouble * 2 (1086.429176) to a Currency (1086.4292).
MyCurr = CCur(MyDouble * 2)
```

### 2.7.5 `CDate` Function

Returns an expression that has been converted to a `Variant` of subtype `Date`.

**Syntax**

```
CDate(Date)
```

The `Date` argument is any valid date expression.

**Remarks**

Use the `IsDate` function to determine if `Date` can be converted to a date or time. `CDate` recognizes date literals and time literals as well as some numbers that fall within the range of acceptable dates. When converting a number to a date, the whole number portion is converted to a date. Any fractional part of the number is converted to a time of day, starting at midnight.

`CDate` recognizes date formats according to the locale setting of your system. The correct order of day, month, and year may not be determined if it is provided in a format other than one of the recognized date settings. In addition, a long date format is not recognized if it also contains the day-of-the-week string.
Conversion Functions

Example

The following example uses the `CDate` function to convert a string to a date. In general, hard coding dates and times as strings (as shown in this example) is not recommended. Use date and time literals (such as `#10/19/1962#`, `#4:45:23 PM#`) instead.

```vbs
Dim MyShortTime, MyDate, MyTime
MyDate = "October 19, 1962" ' Define date.
MyShortDate = CDate(MyDate) ' Convert to Date data type.
MyTime = "4:35:47 PM" ' Define time.
MyShortTime = CDate(MyTime) ' Convert to Date data type.
```

2.7.6 CDbl Function

Returns an expression that has been converted to a `Variant` of subtype `Double`.

Syntax

```
CDbl(Expression)
```

The `Expression` argument is any valid expression.

Remarks

In general, you can document your code using the subtype conversion functions to show that the result of some operation should be expressed as a particular data type rather than the default data type. For example, use `CDbl` or `CSng` to force double-precision or single-precision arithmetic in cases where currency or integer arithmetic normally would occur.

Use the `CDbl` function to provide internationally aware conversions from any other data type to a `Double` subtype. For example, different decimal separators and thousands separators are properly recognized depending on the locale setting of your system.

Data Type Functions

Conversion Functions
Example
This example uses the **CDbl** function to convert an expression to a **Double**.

```vbs
Dim MyCurr, MyDouble
MyCurr = CCur(234.456784) ' MyCurr is a Currency (234.4567).
' Convert result to a Double (19.2254576).
MyDouble = CDbl(MyCurr * 8.2 * 0.01)
```

### 2.7.7 Chr Function

Returns the character associated with the specified ANSI character code.

**Syntax**

```
Chr(CharCode)
```

The *CharCode* argument is a number that identifies a character.

**Remarks**

Numbers from 0 to 31 are the same as standard, nonprintable ASCII codes. For example, *Chr(10)* returns a linefeed character.

**NOTE**

The *ChrB* function is used with byte data contained in a string. Instead of returning a character, which may be one or two bytes, *ChrB* always returns a single byte. *ChrW* is provided for 32-bit platforms that use Unicode characters. Its argument is a Unicode (wide) character code, thereby avoiding the conversion from ANSI to Unicode.

**Example**

The following example uses the **Chr** function to return the character associated with the specified character code.

```vbs
Dim MyChar
MyChar = Chr(65) ' Returns A.
MyChar = Chr(97) ' Returns a.
```
MyChar = Chr(62)  ' Returns >.
MyChar = Chr(37)  ' Returns $.

2.7.8 CInt Function

Returns an expression that has been converted to a Variant of subtype Integer.

Syntax

    CInt(Expression)

The Expression argument is any valid expression.

Remarks

In general, you can document your code using the subtype conversion functions to show that the result of some operation should be expressed as a particular data type rather than the default data type. For example, use CInt or CLng to force integer arithmetic in cases where currency, single-precision, or double-precision arithmetic normally would occur.

Use the CInt function to provide internationally aware conversions from any other data type to an Integer subtype. For example, different decimal separators are properly recognized depending on the locale setting of your system, as are different thousand separators.

If Expression lies outside the acceptable range for the Integer subtype, an error occurs.

NOTE

CInt differs from the Fix and Int functions, which truncate, rather than round, the fractional part of a number. When the fractional part is exactly 0.5, the CInt function always rounds it to the nearest even number. For example, 0.5 rounds to 0, and 1.5 rounds to 2.

Example

The following example uses the CInt function to convert a value to an Integer.

Dim MyDouble, MyInt
MyDouble = 2345.5678  ' MyDouble is a Double.
MyInt = CInt(MyDouble)  ' MyInt contains 2346.

2.7.9  CLng Function

Returns an expression that has been converted to a **Variant** of subtype **Long**.

**Syntax**

```vba
CLng(Expression)
```

The *Expression* argument is any valid expression.

**Remarks**

In general, you can document your code using the subtype conversion functions to show that the result of some operation should be expressed as a particular data type rather than the default data type. For example, use **CInt** or **CLng** to force integer arithmetic in cases where currency, single-precision, or double-precision arithmetic normally would occur.

Use the **CLng** function to provide internationally aware conversions from any other data type to a **Long** subtype. For example, different decimal separators are properly recognized depending on the locale setting of your system, as are different thousand separators.

If *Expression* lies outside the acceptable range for the **Long** subtype, an error occurs.

**NOTE**

**CLng** differs from the **Fix** and **Int** functions, which truncate, rather than round, the fractional part of a number. When the fractional part is exactly 0.5, the **CLng** function always rounds it to the nearest even number. For example, 0.5 rounds to 0, and 1.5 rounds to 2.

Example

The following example uses the **CLng** function to convert a value to a **Long**.

```vba
Dim MyVal1, MyVal2, MyLong1, MyLong2
```
2.7.10  CSng Function

Returns an expression that has been converted to a Variant of subtype Single.

Syntax

    CSng(Expression)

   The Expression argument is any valid expression.

Remarks

In general, you can document your code using the data type conversion functions to show that the result of some operation should be expressed as a particular data type rather than the default data type. For example, use CDbI or CSng to force double-precision or single-precision arithmetic in cases where currency or integer arithmetic normally would occur.

Use the CSng function to provide internationally aware conversions from any other data type to a Single subtype. For example, different decimal separators are properly recognized depending on the locale setting of your system, as are different thousand separators.

If Expression lies outside the acceptable range for the Single subtype, an error occurs.

Example

The following example uses the CSng function to convert a value to a Single.

```vba
Dim MyDouble1, MyDouble2   ' MyDouble1, MyDouble2 are Doubles.
Dim MySingle1, MySingle2
MyDouble1 = 75.3421115
MyDouble2 = 75.3421555
MySingle1 = CSng(MyDouble1)   ' MySingle1 contains 75.34211.
```
MySingle2 = CSng(MyDouble2) ' MySingle2 contains 75.34216.

2.7.11 CStr Function

Returns an expression that has been converted to a Variant of subtype String.

Syntax

CStr(Expression)

The Expression argument is any valid expression.

Remarks

In general, you can document your code using the data type conversion functions to show that the result of some operation should be expressed as a particular data type rather than the default data type. For example, use CStr to force the result to be expressed as a String.

You should use the CStr function instead of String to provide internationally aware conversions from any other data type to a String subtype. For example, different decimal separators are properly recognized depending on the locale setting of your system.

The data in Expression determines what is returned according to the following table.

<table>
<thead>
<tr>
<th>If Expression is</th>
<th>CStr returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>A String containing True or False.</td>
</tr>
<tr>
<td>Date</td>
<td>A String containing a date in the short-date format of your system.</td>
</tr>
<tr>
<td>Null</td>
<td>A run-time error.</td>
</tr>
<tr>
<td>Empty</td>
<td>A zero-length String (&quot;&quot;).</td>
</tr>
<tr>
<td>Error</td>
<td>A String containing the word Error followed by the error number.</td>
</tr>
<tr>
<td>Other numeric</td>
<td>A String containing the number.</td>
</tr>
</tbody>
</table>
Example

The following example uses the CStr function to convert a numeric value to a String.

```vba
Dim MyDouble, MyString
MyDouble = 437.324 ' MyDouble is a Double.
MyString = CStr(MyDouble) ' MyString contains "437.324".
```

2.7.12 DateSerial Function

Returns a Variant of subtype Date for a specified year, month, and day.

Syntax

DateSerial(Year, Month, Day)

The DateSerial function syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Year</td>
<td>Number between 100 and 9999, inclusive, or a numeric expression.</td>
</tr>
<tr>
<td>Month</td>
<td>Any numeric expression.</td>
</tr>
<tr>
<td>Day</td>
<td>Any numeric expression.</td>
</tr>
</tbody>
</table>

Remarks

To specify a date, such as December 31, 1991, the range of numbers for each DateSerial argument should be in the accepted range for the unit; that is, 1–31 for days and 1–12 for months. However, you can also specify relative dates for each argument using any numeric expression that represents some number of days, months, or years before or after a certain date.

For the Year argument, values between 0 and 99, inclusive, are interpreted as the years 1900–1999. For all other Year arguments, use a complete four-digit year (for example, 1800).
When any argument exceeds the accepted range for that argument, it increments to the next larger unit as appropriate. For example, if you specify 35 days, it is evaluated as one month and some number of days, depending on where in the year it is applied. However, if any single argument is outside the range -32,768 to 32,767, or if the date specified by the three arguments, either directly or by expression, falls outside the acceptable range of dates, an error occurs.

Example

The following example uses numeric expressions instead of absolute date numbers. Here the `DateSerial` function returns a date that is the day before the first day (1 - 1) of two months before August (8 - 2) of 10 years before 1990 (1990 - 10); in other words, May 31, 1980.

```vba
Dim MyDate1, MyDate2
MyDate1 = DateSerial(1970, 1, 1)  ' Returns January 1, 1970.
MyDate2 = DateSerial(1990 - 10, 8 - 2, 1 - 1)  ' Returns May 31, 1980.
```

2.7.13 DateValue Function

Returns a Variant of subtype Date.

Syntax

```
DateValue(Date)
```

The `Date` argument is normally a string expression representing a date from January 1, 100 through December 31, 9999. However, `Date` can also be any expression that can represent a date, a time, or both a date and time, in that range.

Remarks

If the `Date` argument includes time information, `DateValue` doesn't return it. However, if `Date` includes invalid time information (such as "89:98"), an error occurs.
If *Date* is a string that includes only numbers separated by valid date separators, **DateValue** recognizes the order for month, day, and year according to the short date format you specified for your system. **DateValue** also recognizes unambiguous dates that contain month names, either in long or abbreviated form. For example, in addition to recognizing 12/30/1991 and 12/30/91, **DateValue** also recognizes December 30, 1991 and Dec 30, 1991.

If the year part of *Date* is omitted, **DateValue** uses the current year from your computer's system date.

### Example

The following example uses the **DateValue** function to convert a string to a date. You can also use date literals to directly assign a date to a **Variant** variable, for example, `MyDate = #9/11/63#`.

```vbscript
Dim MyDate
MyDate = DateValue("September 11, 1963") ' Return a date.
```

### 2.7.14 Day Function

Returns a whole number between 1 and 31, inclusive, representing the day of the month.

#### Syntax

**Day(Date)**

The *Date* argument is any expression that can represent a date. If *Date* contains **Null**, **Null** is returned.

### Example

The following example uses the **Day** function to obtain the day of the month from a specified date.
Dim MyDay

2.7.15 Fix Function

Returns the integer portion of a number.

Syntax

    Fix(Number)

The Number argument can be any valid numeric expression. If Number contains Null, Null is returned.

Remarks

Both Int and Fix remove the fractional part of Number and return the resulting integer value.

The difference between Int and Fix is that if Number is negative, Int returns the first negative integer less than or equal to Number, whereas Fix returns the first negative integer greater than or equal to Number. For example, Int converts -8.4 to -9, and Fix converts -8.4 to -8.

Fix(Number) is equivalent to:

    Sgn(Number) * Int(Abs(Number))

Example

The following examples illustrate how the Int and Fix functions return integer portions of numbers.

Dim MyNumber
MyNumber = Int(99.8) ' Returns 99.
MyNumber = Fix(99.2) ' Returns 99.
MyNumber = Int(-99.8) ' Returns -100.
2.7.16 Hex Function

Returns a string representing the hexadecimal value of a number.

Syntax

`Hex(Number)`

The `Number` argument is any valid expression.

Remarks

If `Number` is not already a whole number, it is rounded to the nearest whole number before being evaluated.

<table>
<thead>
<tr>
<th>If Number is</th>
<th>Hex returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Null</td>
<td>Null</td>
</tr>
<tr>
<td>Empty</td>
<td>Zero (0).</td>
</tr>
<tr>
<td>Any other number</td>
<td>Up to eight hexadecimal characters.</td>
</tr>
</tbody>
</table>

You can represent hexadecimal numbers directly by preceding numbers in the proper range with &H. For example, &H10 represents decimal 16 in hexadecimal notation.

Example

The following example uses the `Hex` function to return the hexadecimal value of a number.

```
Dim MyHex
MyHex = Hex(5)   ' Returns 5.
MyHex = Hex(10)  ' Returns A
MyHex = Hex(459) ' Returns 1CB.
```
2.7.17 Hour Function

Returns a whole number between 0 and 23, inclusive, representing the hour of the day.

**Syntax**

\[
\text{Hour}(\text{Time})
\]

The *Time* argument is any expression that can represent a time. If *Time* contains **Null**, **Null** is returned.

**Example**

The following example uses the **Hour** function to obtain the hour from the current time.

```
Dim MyTime, MyHour
MyTime = Now
' MyHour contains the number representing the current hour.
MyHour = Hour(MyTime)
```

2.7.18 Int Function

Returns the integer portion of a number.

**Syntax**

\[
\text{Int}(\text{Number})
\]

The *Number* argument can be any valid numeric expression. If *Number* contains **Null**, **Null** is returned.

**Remarks**

Both **Int** and **Fix** remove the fractional part of *Number* and return the resulting integer value.
The difference between **Int** and **Fix** is that if *Number* is negative, **Int** returns the first negative integer less than or equal to *Number*, whereas **Fix** returns the first negative integer greater than or equal to *Number*. For example, **Int** converts -8.4 to -9, and **Fix** converts -8.4 to -8.

**Fix(Number)** is equivalent to.

```
Sgn(Number) * Int(Abs(Number))
```

### Data Type Functions

### Conversion Functions

### Math Functions

### Example

The following examples illustrate how the **Int** and **Fix** functions return integer portions of numbers.

```
Dim MyNumber
MyNumber = Int(99.8)    ' Returns 99.
MyNumber = Fix(99.2)    ' Returns 99.
MyNumber = Int(-99.8)   ' Returns -100.
MyNumber = Fix(-99.8)   ' Returns -99.
MyNumber = Int(-99.2)   ' Returns -100.
MyNumber = Fix(-99.2)   ' Returns -99.
```

### 2.7.19 LCase Function

Returns a string that has been converted to lowercase.

**Syntax**

```
LCase(String)
```

The *String* argument is any valid string expression. If *String* contains **Null**, **Null** is returned.

**Remarks**

Only uppercase letters are converted to lowercase; all lowercase letters and nonletter characters remain unchanged.
110 Conversion Functions

String Functions
Conversion Functions

Example
The following example uses the **LCase** function to convert uppercase letters to lowercase.

```vbs
Dim MyString
Dim LCaseString
MyString = "VBScript"
LCaseString = LCase(MyString) ' LCaseString contains "vbscript".
```

2.7.20 Minute Function

Returns a whole number between 0 and 59, inclusive, representing the minute of the hour.

**Syntax**

```vbs
Minute(Time)
```

The *Time* argument is any expression that can represent a time. If *Time* contains **Null**, **Null** is returned.

Date and Time Functions
Conversion Functions

Example
The following example uses the **Minute** function to return the minute of the hour.

```vbs
Dim MyVar
MyVar = Minute(Now)
```

2.7.21 Month Function

Returns a whole number between 1 and 12, inclusive, representing the month of the year.
Syntax

\[ \text{Month}(\text{Date}) \]

The \textit{Date} argument is any expression that can represent a date. If \textit{Date} contains \textbf{Null}, \textbf{Null} is returned.

Example

The following example uses the \textbf{Month} function to return the current month.

```vbnet
Dim MyVar
' MyVar contains the number corresponding to the current month.
MyVar = Month(Now)
```

2.7.22 Oct Function

Returns a string representing the octal value of a number.

Syntax

\[ \text{Oct}(\text{Number}) \]

The \textit{Number} argument is any valid expression.

Remarks

If \textit{Number} is not already a whole number, it is rounded to the nearest whole number before being evaluated.

<table>
<thead>
<tr>
<th>\textit{If Number is}</th>
<th>\textit{Oct returns}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Null</td>
<td>Null</td>
</tr>
<tr>
<td>Empty</td>
<td>Zero (0).</td>
</tr>
<tr>
<td>Any other number</td>
<td>Up to 11 octal characters.</td>
</tr>
</tbody>
</table>
You can represent octal numbers directly by preceding numbers in the proper range with &O. For example, &O10 is the octal notation for decimal 8.

**Example**

The following example uses the **Oct** function to return the octal value of a number.

```vbnet
Dim MyOct
MyOct = Oct(459) ' Returns 713.
```

### 2.7.23 Second Function

Returns a whole number between 0 and 59, inclusive, representing the second of the minute.

**Syntax**

```vbnet
Second(Time)
```

The *Time* argument is any expression that can represent a time. If *Time* contains **Null**, **Null** is returned.

**Example**

The following example uses the **Second** function to return the current second.

```vbnet
Dim MySec
' MySec contains the number representing the current second.
MySec = Second(Now)
```

### 2.7.24 TimeSerial Function

Returns a **Variant** of subtype **Date** containing the time for a specific hour, minute, and
second.

Syntax

\[
\text{TimeSerial}(\text{Hour}, \text{Minute}, \text{Second})
\]

The \textit{TimeSerial} function syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hour</td>
<td>Number between 0 (12:00 A.M.) and 23 (11:00 P.M.), inclusive, or a numeric expression.</td>
</tr>
<tr>
<td>Minute</td>
<td>Any numeric expression.</td>
</tr>
<tr>
<td>Second</td>
<td>Any numeric expression.</td>
</tr>
</tbody>
</table>

Remarks

To specify a time, such as 11:59:59, the range of numbers for each \textit{TimeSerial} argument should be in the accepted range for the unit; that is, 0–23 for hours and 0–59 for minutes and seconds. However, you can also specify relative times for each argument using any numeric expression that represents some number of hours, minutes, or seconds before or after a certain time.

When any argument exceeds the accepted range for that argument, it increments to the next larger unit as appropriate. For example, if you specify 75 minutes, it is evaluated as one hour and 15 minutes. However, if any single argument is outside the range -32,768 to 32,767, or if the time specified by the three arguments, either directly or by expression, causes the date to fall outside the acceptable range of dates, an error occurs.

Example

The following example uses expressions instead of absolute time numbers. The \textit{TimeSerial} function returns a time for 15 minutes before (-15) six hours before noon (12 - 6), or 5:45:00 A.M.

\[
\text{Dim MyTime}
\]

\[
\text{MyTime = TimeSerial}(12 - 6, -15, 0) \quad \text{' Returns 5:45:00 AM}
\]
2.7.25 TimeValue Function

Returns a Variant of subtype Date containing the time.

Syntax

TimeValue( Time)

The Time argument is usually a string expression representing a time from 0:00:00 (12:00:00 A.M.) to 23:59:59 (11:59:59 P.M.), inclusive. However, Time can also be any expression that represents a time in that range. If Time contains Null, Null is returned.

Remarks

You can enter valid times using a 12-hour or 24-hour clock. For example, "2:24PM" and "14:24" are both valid Time arguments. If the Time argument contains date information, TimeValue doesn't return the date information. However, if Time includes invalid date information, an error occurs.

Example

The following example uses the TimeValue function to convert a string to a time. You can also use date literals to directly assign a time to a Variant (for example, MyTime = #4:35:17 PM#).

Dim MyTime
MyTime = TimeValue("4:35:17 PM") ' MyTime contains 4:35:17 PM

2.7.26 UCase Function

Returns a string that has been converted to uppercase.

Syntax

UCase(String)

The String argument is any valid string expression. If String contains Null, Null is returned.
**Remarks**

Only lowercase letters are converted to uppercase; all uppercase letters and nonletter characters remain unchanged.

**Example**

The following example uses the **UCase** function to return an uppercase version of a string.

```vbs
Dim MyWord
MyWord = UCase("Hello World") ' Returns "HELLO WORLD".
```

### 2.7.27 Weekday Function

Returns a whole number representing the day of the week.

**Syntax**

```vbs
Weekday(Date[, FirstDayOfWeek])
```

The **Weekday** function syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Date</strong></td>
<td>Any expression that can represent a date. If <strong>Date</strong> contains <strong>Null</strong>, <strong>Null</strong> is returned.</td>
</tr>
<tr>
<td><strong>FirstDayOfWeek</strong></td>
<td>A constant that specifies the first day of the week. If omitted, <strong>vbSunday</strong> is assumed.</td>
</tr>
</tbody>
</table>

**Settings**

The **FirstDayOfWeek** argument has these settings:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
</table>
Return Values

The **Weekday** function can return any of these values:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbSunday</td>
<td>1</td>
<td>Sunday</td>
</tr>
<tr>
<td>vbMonday</td>
<td>2</td>
<td>Monday</td>
</tr>
<tr>
<td>vbTuesday</td>
<td>3</td>
<td>Tuesday</td>
</tr>
<tr>
<td>vbWednesday</td>
<td>4</td>
<td>Wednesday</td>
</tr>
<tr>
<td>vbThursday</td>
<td>5</td>
<td>Thursday</td>
</tr>
<tr>
<td>vbFriday</td>
<td>6</td>
<td>Friday</td>
</tr>
<tr>
<td>vbSaturday</td>
<td>7</td>
<td>Saturday</td>
</tr>
</tbody>
</table>

Example

The following example uses the **Weekday** function to obtain the day of the week from a specified date.
**2.7.28 Year Function**

Returns a whole number representing the year.

**Syntax**

\[
\text{Year}(\text{Date})
\]

The `Date` argument is any expression that can represent a date. If `Date` contains `Null`, `Null` is returned.

**Example**

The following example uses the `Year` function to obtain the year from a specified date.

```vba
Dim MyDate, MyYear
MyDate = #October 19, 1962# ' Assign a date.
MyYear = Year(MyDate) ' MyYear contains 1962.
```
2.8 Math Functions

Abs Function
Atn Function
Cos Function
Exp Function
Fix Function
Int Function
Log Function
Rnd Function
Round Function
Sgn Function
Sin Function
Sqr Function
Tan Function
Derived Functions

2.8.1 Abs Function

Returns the absolute value of a number.

Syntax

Abs(Number)

The Number argument can be any valid numeric expression. If Number contains Null, Null is returned; if it is an uninitialized variable, zero is returned.

Remarks

The absolute value of a number is its unsigned magnitude. For example, Abs(-1) and Abs(1) both return 1.
Example

The following example uses the **Abs** function to compute the absolute value of a number.

```
Dim MyNumber
MyNumber = Abs(50.3)  ' Returns 50.3.
MyNumber = Abs(-50.3) ' Returns 50.3.
```

### 2.8.2 Atn Function

Returns the arctangent of a number.

**Syntax**

```
Atn(Number)
```

The `Number` argument can be any valid numeric expression.

**Remarks**

The **Atn** function takes the ratio of two sides of a right triangle (*Number*) and returns the corresponding angle in radians. The ratio is the length of the side opposite the angle divided by the length of the side adjacent to the angle. The range of the result is -pi/2 to pi/2 radians.

To convert degrees to radians, multiply degrees by pi/180. To convert radians to degrees, multiply radians by 180/pi.

**NOTE**

**Atn** is the inverse trigonometric function of **Tan**, which takes an angle as its argument and returns the ratio of two sides of a right triangle. Do not confuse **Atn** with the cotangent, which is the simple inverse of a tangent (1/tangent).

**Example**

The following example uses **Atn** to calculate the value of pi.

```
Dim pi
pi = 4 * Atn(1)  ' Calculate the value of pi.
```
2.8.3 Cos Function

Returns the cosine of an angle.

Syntax

\[ \text{Cos(Number)} \]

The \textit{Number} argument can be any valid numeric expression that expresses an angle in radians.

Remarks

The \textit{Cos} function takes an angle and returns the ratio of two sides of a right triangle. The ratio is the length of the side adjacent to the angle divided by the length of the hypotenuse. The result lies in the range -1 to 1.

To convert degrees to radians, multiply degrees by \(\pi/180\). To convert radians to degrees, multiply radians by \(180/\pi\).

Example

The following example uses the \textit{Cos} function to return the cosine of an angle.

```vbs
Dim MyAngle, MySecant
MyAngle = 1.3 ' Define angle in radians.
MySecant = 1 / Cos(MyAngle) ' Calculate secant.
```

2.8.4 Exp Function

Returns \(e\) (the base of natural logarithms) raised to a power.

Syntax

\[ \text{Exp(Number)} \]

The \textit{Number} argument can be any valid numeric expression.
Remarks

If the value of `Number` exceeds 709.782712893, an error occurs. The constant $e$ is approximately 2.718282.

NOTE

The **Exp** function complements the action of the **Log** function and is sometimes referred to as the antilogarithm.

Example

The following example uses the **Exp** function to return $e$ raised to a power.

```vbs
Dim MyAngle, MyHSin
MyAngle = 1.3 ' Define angle in radians.
' Calculate hyperbolic sine.
MyHSin = (Exp(MyAngle) - Exp(-1 * MyAngle)) / 2
```

### 2.8.5 Fix Function

Returns the integer portion of a number.

**Syntax**

```
Fix(Number)
```

The `Number` argument can be any valid numeric expression. If `Number` contains **Null**, **Null** is returned.

**Remarks**

Both **Int** and **Fix** remove the fractional part of `Number` and return the resulting integer value.

The difference between **Int** and **Fix** is that if `Number` is negative, **Int** returns the first negative integer less than or equal to `Number`, whereas **Fix** returns the first negative integer greater than or equal to `Number`. For example, **Int** converts -8.4 to -9, and **Fix** converts -8.4 to -8.

**Fix(Number)** is equivalent to:
Data Type Functions
Conversion Functions
Math Functions

Example

The following examples illustrate how the Int and Fix functions return integer portions of numbers.

```
Dim MyNumber
MyNumber = Int(99.8) ' Returns 99.
MyNumber = Fix(99.2) ' Returns 99.
MyNumber = Int(-99.8) ' Returns -100.
MyNumber = Fix(-99.8) ' Returns -99.
MyNumber = Int(-99.2) ' Returns -100.
MyNumber = Fix(-99.2) ' Returns -99.
```

2.8.6 Int Function

Returns the integer portion of a number.

Syntax

```
Int(Number)
```

The Number argument can be any valid numeric expression. If Number contains Null, Null is returned.

Remarks

Both Int and Fix remove the fractional part of Number and return the resulting integer value.

The difference between Int and Fix is that if Number is negative, Int returns the first negative integer less than or equal to Number, whereas Fix returns the first negative integer greater than or equal to Number. For example, Int converts -8.4 to -9, and Fix converts -8.4 to -8.

Fix(Number) is equivalent to.
Sgn(Number) * Int(Abs(Number))

Data Type Functions
Conversion Functions
Math Functions

Example
The following examples illustrate how the **Int** and **Fix** functions return integer portions of numbers.

```vbnet
Dim MyNumber
MyNumber = Int(99.8) ' Returns 99.
MyNumber = Fix(99.2) ' Returns 99.
MyNumber = Int(-99.8) ' Returns -100.
MyNumber = Fix(-99.8) ' Returns -99.
MyNumber = Int(-99.2) ' Returns -100.
MyNumber = Fix(-99.2) ' Returns -99.
```

**2.8.7 Log Function**

Returns the natural logarithm of a number.

**Syntax**

```
Log(Number)
```

The **Number** argument can be any valid numeric expression greater than 0.

**Remarks**

The natural logarithm is the logarithm to the base e. The constant e is approximately 2.718282.

You can calculate base-n logarithms for any number x by dividing the natural logarithm of x by the natural logarithm of n as follows.

```
Logn(x) = Log(x) / Log(n)
```
Example

The following example illustrates a custom function that calculates base-10 logarithms.

```vbnet
Function Log10(X)
    Log10 = Log(X) / Log(10)
End Function
```

2.8.8 Rnd Function

Returns a random number.

Syntax

```
Rnd[(Number)]
```

The Number argument can be any valid numeric expression.

Remarks

The Rnd function returns a value less than 1 but greater than or equal to 0. The value of Number determines how Rnd generates a random number:

<table>
<thead>
<tr>
<th>If Number is</th>
<th>Rnd generates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Less than zero</td>
<td>The same number every time, using Number as the seed.</td>
</tr>
<tr>
<td>Greater than zero</td>
<td>The next random number in the sequence.</td>
</tr>
<tr>
<td>Equal to zero</td>
<td>The most recently generated number.</td>
</tr>
<tr>
<td>Not supplied</td>
<td>The next random number in the sequence.</td>
</tr>
</tbody>
</table>

For any given initial seed, the same number sequence is generated because each successive call to the Rnd function uses the previous number as a seed for the next number in the sequence.

Before calling Rnd, use the Randomize statement without an argument to initialize the random-number generator with a seed based on the system timer.
NOTE

To repeat sequences of random numbers, call Rnd with a negative argument immediately before using Randomize with a numeric argument. Using Randomize with the same value for Number does not repeat the previous sequence.

Example

To produce random integers in a given range, use this formula (here, upperbound is the highest number in the range, and lowerbound is the lowest number in the range).

\[
\text{Int}\left( (\text{upperbound} - \text{lowerbound} + 1) \times \text{Rnd} + \text{lowerbound}\right)
\]

2.8.9 Round Function

Returns a number rounded to a specified number of decimal places.

Syntax

\[
\text{Round}(Expression[, NumDecimalPlaces])
\]

The Round function syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expression</td>
<td>Required. Numeric expression being rounded.</td>
</tr>
<tr>
<td>NumDecimalPlaces</td>
<td>Optional. Number indicating how many places to the right of the decimal are included in the rounding. If omitted, integers are returned by the Round function.</td>
</tr>
</tbody>
</table>
Example

The following example uses the **Round** function to round a number to two decimal places.

```
Dim MyVar, pi
pi = 3.14159
MyVar = Round(pi, 2) ' MyVar contains 3.14.
```

### 2.8.10 Sgn Function

Returns an integer indicating the sign of a number.

**Syntax**

```
Sgn(Number)
```

The *Number* argument can be any valid numeric expression.

**Return Values**

The *Sgn* function has the following return values:

<table>
<thead>
<tr>
<th>If Number is</th>
<th>Sgn returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greater than zero</td>
<td>1</td>
</tr>
<tr>
<td>Equal to zero</td>
<td>0</td>
</tr>
<tr>
<td>Less than zero</td>
<td>-1</td>
</tr>
</tbody>
</table>

**Remarks**

The sign of the *Number* argument determines the return value of the *Sgn* function.
MyVar1 = 12
MyVar2 = -2.4
MyVar3 = 0
MySign = Sgn(MyVar1) ' Returns 1.
MySign = Sgn(MyVar2) ' Returns -1.
MySign = Sgn(MyVar3) ' Returns 0.

### 2.8.11 Sin Function

Returns the sine of an angle.

**Syntax**

```
Sin(Number)
```

The `Number` argument can be any valid numeric expression that expresses an angle in radians.

**Remarks**

The `Sin` function takes an angle and returns the ratio of two sides of a right triangle. The ratio is the length of the side opposite the angle divided by the length of the hypotenuse. The result lies in the range -1 to 1.

To convert degrees to radians, multiply degrees by pi/180. To convert radians to degrees, multiply radians by 180/pi.

### Example

The following example uses the `Sin` function to return the sine of an angle.

```
Dim MyAngle, MyCosecant
MyAngle = 1.3 ' Define angle in radians.
MyCosecant = 1 / Sin(MyAngle) ' Calculate cosecant.
```

### 2.8.12 Sqr Function

Returns the square root of a number.
### Syntax

**Sqr(Number)**

The *Number* argument can be any valid numeric expression greater than or equal to 0.

### Example

The following example uses the *Sqr* function to calculate the square root of a number.

```vbnet
dim mysqr
mysqr = sqr(4) ' Returns 2.
mysqr = sqr(23) ' Returns 4.79583152331272.
mysqr = sqr(0) ' Returns 0.
mysqr = sqr(-4) ' Generates a run-time error.
```

#### 2.8.13 Tan Function

Returns the tangent of an angle.

### Syntax

**Tan(Number)**

The *Number* argument can be any valid numeric expression that expresses an angle in radians.

### Remarks

**Tan** takes an angle and returns the ratio of two sides of a right triangle. The ratio is the length of the side opposite the angle divided by the length of the side adjacent to the angle.

To convert degrees to radians, multiply degrees by \( \pi/180 \). To convert radians to degrees, multiply radians by \( 180/\pi \).
Example

The following example uses the `Tan` function to return the tangent of an angle.

```vbscript
Dim MyAngle, MyCotangent
MyAngle = 1.3 ' Define angle in radians.
MyCotangent = 1 / Tan(MyAngle) ' Calculate cotangent.
```

### 2.8.14 Derived Functions

Next (derived) functions can be found in the original help.
2.9 Miscellaneous Functions

- **CreateObject Function**
- **Eval Function**
- **GetObject Function**
- **GetLocale Function**
- **GetRef Function**
- **InputBox Function**
- **LoadPicture Function**
- **MsgBox Function**
- **RGB Function**
- **ScriptEngine Function**
- **ScriptEngineBuildVersion Function**
- **ScriptEngineMajorVersion Function**
- **ScriptEngineMinorVersion Function**
- **SetLocale Function**

### 2.9.1 CreateObject Function

Creates and returns a reference to an Automation object.

**Syntax**

```
CreateObject(ServerName.TypeName[, Location])
```

The `CreateObject` function syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ServerName.TypeName</code></td>
<td>Required; <strong>String</strong>. Full path and name of the file containing the object to retrieve.</td>
</tr>
<tr>
<td><code>Location</code></td>
<td>Optional; <strong>String</strong>. The name of the network server where the object is to be created.</td>
</tr>
</tbody>
</table>
The `ServerName.TypeName` argument has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServerName</td>
<td>Required; <strong>String</strong>. The name of the application providing the object.</td>
</tr>
<tr>
<td>TypeName</td>
<td>Required; <strong>String</strong>. The type or class of the object to create.</td>
</tr>
</tbody>
</table>

**Remarks**

Automation servers provide at least one type of object. For example, a word-processing application may provide an application object, a document object, and a toolbar object.

To create an Automation object, assign the object returned by `CreateObject` to an object variable. For Example:

```vba
Dim ExcelSheet
Set ExcelSheet = CreateObject("Excel.Sheet")
```

This code starts the application that creates the object (in this case, a Microsoft Excel spreadsheet). Once an object is created, refer to it in code using the object variable you defined. As shown in the following example, you can access properties and methods of the new object using the object variable, ExcelSheet, and other Excel objects, including the Application object and the ActiveSheet.Cells collection:

```vba
ExcelSheet.Application.Visible = True ' Make Excel visible through the Application object.
ExcelSheet.ActiveSheet.Cells(1, 1).Value = "This is column A, row 1" ' Place some text in the first cell of the sheet.
ExcelSheet.SaveAs "C:\DOCS\TEST.XLS" ' Save the sheet.
Set ExcelSheet = Nothing ' Release the object variable.
```
2.9.2 Eval Function

Evaluates an expression and returns the result.

Syntax

\[ \text{Result} = \text{Eval}(\text{Expression}) \]

The `Eval` function syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Result</code></td>
<td>Optional. Variable to which return value assignment is made. If <code>Result</code> is not specified, consider using the <code>Execute</code> statement instead.</td>
</tr>
<tr>
<td><code>Expression</code></td>
<td>Required. String containing any legal VBScript expression.</td>
</tr>
</tbody>
</table>

Remarks

In VBScript, \( x = y \) can be interpreted two ways. The first is as an assignment statement, where the value of \( y \) is assigned to \( x \). The second interpretation is as an expression that tests if \( x \) and \( y \) have the same value. If they do, `Result` is `True`; if they are not, `Result` is `False`. The `Eval` method always uses the second interpretation, whereas the `Execute` statement always uses the first.

Example

The following example illustrates the use of the `Eval` function.

```vbs
Sub GuessANumber
    Dim Guess, RndNum
    RndNum = Int((100) * Rnd(1) + 1)
    Guess = CInt(InputBox("Enter your guess:", , 0))
    Do
        If Eval("Guess = RndNum") Then
            MsgBox "Congratulations! You guessed it!"
        End If
    Loop
End Sub
```
Sub
Else
    Guess = CInt(InputBox("Sorry! Try again.", , 0))
End If
Loop Until Guess = 0
End Sub

2.9.3 GetLocale Function

Returns the current locale ID value.

Syntax

    GetLocale

Remarks

A locale is a set of user preference information related to the user's language, country/region, and cultural conventions. The locale determines such things as keyboard layout, alphabetic sort order, as well as date, time, number, and currency formats.

The return value can be any of the 32-bit values shown in the Locale ID chart.

Example

The following example illustrates the use of the GetLocale function.

```vba
Dim loc, d, dow, wdn
loc = GetLocale  ' Save the original settings.
SetLocale "cs"  ' Switch to Czech.
d = Date
dow = WeekDay(d)
wdn = WeekDayName(dow)  ' Returns, e.g., "středa" (Wednesday in Czech).
SetLocale loc  ' Switch to the original settings.
```
2.9.4 GetObject Function

Returns a reference to an Automation object from a file.

Syntax

```
GetObject([PathName][, Class])
```

The `GetObject` function syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PathName</td>
<td>Optional; <strong>String</strong>. Full path and name of the file containing the object to retrieve. If <code>PathName</code> is omitted, <code>Class</code> is required.</td>
</tr>
<tr>
<td>Class</td>
<td>Optional; <strong>String</strong>. Class of the object.</td>
</tr>
</tbody>
</table>

The `Class` argument uses the syntax `AppName.ObjectType` and has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppName</td>
<td>Required; <strong>String</strong>. Name of the application providing the object.</td>
</tr>
<tr>
<td>ObjectType</td>
<td>Required; <strong>String</strong>. Type or class of object to create.</td>
</tr>
</tbody>
</table>

Remarks

Use the `GetObject` function to access an Automation object from a file and assign the object to an object variable. Use the `Set` statement to assign the object returned by `GetObject` to the object variable. For Example:

```
Dim CADObject
Set CADObject = GetObject("C:\CAD\SCHEMA.CAD")
```
When this code is executed, the application associated with the specified pathname is started and the object in the specified file is activated. If `PathName` is a zero-length string (""), `GetObject` returns a new object instance of the specified type. If the `PathName` argument is omitted, `GetObject` returns a currently active object of the specified type. If no object of the specified type exists, an error occurs.

Some applications allow you to activate part of a file. Add an exclamation point (!) to the end of the file name and follow it with a string that identifies the part of the file you want to activate. For information on how to create this string, see the documentation for the application that created the object.

For example, in a drawing application you might have multiple layers to a drawing stored in a file. You could use the following code to activate a layer within a drawing called `SCHEM.A.CAD`:

```vbscript
Set LayerObject = GetObject("C:\CAD\SCHEM.A.CAD!Layer3")
```

If you don't specify the object's class, Automation determines the application to start and the object to activate, based on the file name you provide. Some files, however, may support more than one class of object. For example, a drawing might support three different types of objects: an Application object, a Drawing object, and a Toolbar object, all of which are part of the same file. To specify which object in a file you want to activate, use the optional `Class` argument. For example:

```vbscript
Dim MyObject
Set MyObject = GetObject("C:\DRAWINGS\SAMPLE.DRW", "FIGMENT.DRAWING")
```

In the preceding example, `FIGMENT` is the name of a drawing application and `DRAWING` is one of the object types it supports. Once an object is activated, you reference it in code using the object variable you defined. In the preceding example, you access properties and methods of the new object using the object variable `MyObject`. For example:

```vbscript
MyObject.Line 9, 90
MyObject.InsertText 9, 100, "Hello, world."
MyObject.SaveAs "C:\DRAWINGS\SAMPLE.DRW"
```

**NOTE**

Use the `GetObject` function when there is a current instance of the object or if you want to create the object with a file already loaded. If there is no current instance, and you don't want the object started with a file loaded, use the `CreateObject` function.
If an object has registered itself as a single-instance object, only one instance of the object is created, no matter how many times CreateObject is executed. With a single-instance object, **GetObject** always returns the same instance when called with the zero-length string ("") syntax, and it causes an error if the **PathName** argument is omitted.

### 2.9.5 GetRef Function

Returns a reference to a procedure that can be bound to an event.

**Syntax**

\[
\text{Set } \text{Object}.\text{EventName} = \text{GetRef}(\text{ProcName})
\]

The **GetRef** function syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Object</strong></td>
<td>Required. Name of the object with which <strong>EventName</strong> is associated.</td>
</tr>
<tr>
<td><strong>EventName</strong></td>
<td>Required. Name of the event to which the function is to be bound.</td>
</tr>
<tr>
<td><strong>ProcName</strong></td>
<td>Required. String containing the name of the <strong>Sub</strong> or <strong>Function</strong> procedure being associated with the <strong>EventName</strong>.</td>
</tr>
</tbody>
</table>

**Remarks**

The **GetRef** function allows you to connect a VBScript procedure (**Function** or **Sub**) to any available event on your DHTML (Dynamic HTML) pages. The DHTML object model provides information about what events are available for its various objects.

In other scripting and programming languages, the functionality provided by **GetRef** is referred to as a function pointer, that is, it points to the address of a procedure to be executed when the specified event occurs.
Example

The following example illustrates the use of the `GetRef` function.

```vbscript
<SCRIPT LANGUAGE="VBScript">
Function GetRefTest()
  Dim Splash
  Splash = "GetRefTest Version 1.0" & vbCrLf
  Splash = Splash & Chr(169) & " GEOVAP – Reliance "
  MsgBox Splash
End Function
Set Window.Onload = GetRef("GetRefTest")
</SCRIPT>
```

2.9.6 InputBox Function

Displays a prompt in a dialog box, waits for the user to input text or click a button, and returns the contents of the text box.

Syntax

```
InputBox(Prompt[, Title][, Default][, XPos][, YPos][, HelpFile, Context])
```

The `InputBox` function syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prompt</td>
<td>String expression displayed as the message in the dialog box. The maximum length of <code>Prompt</code> is approximately 1024 characters, depending on the width of the characters used. If <code>Prompt</code> consists of more than one line, you can separate the lines using a carriage return character (<code>Chr(13)</code>), a linefeed character (<code>Chr(10)</code>), or carriage return–linefeed character combination (<code>Chr(13) &amp; Chr(10)</code>) between each line.</td>
</tr>
<tr>
<td>Title</td>
<td>String expression displayed in the title bar of the dialog box. If you omit <code>Title</code>, the application name is placed in the title bar.</td>
</tr>
<tr>
<td>Default</td>
<td>String expression displayed in the text box as the default response if no other input is provided. If you omit <code>Default</code>, the text box is displayed empty.</td>
</tr>
</tbody>
</table>
### XPos
Numeric expression that specifies, in twips, the horizontal distance of the left edge of the dialog box from the left edge of the screen. If XPos is omitted, the dialog box is horizontally centered.

### YPos
Numeric expression that specifies, in twips, the vertical distance of the upper edge of the dialog box from the top of the screen. If YPos is omitted, the dialog box is vertically positioned approximately one-third of the way down the screen.

### HelpFile
String expression that identifies the Help file to use to provide context-sensitive Help for the dialog box. If HelpFile is provided, Context must also be provided.

### Context
Numeric expression that identifies the Help context number assigned by the Help author to the appropriate Help topic. If Context is provided, HelpFile must also be provided.

### Remarks
When both HelpFile and Context are supplied, a Help button is automatically added to the dialog box.

If the user clicks OK or presses Enter, the **InputBox** function returns whatever is in the text box. If the user clicks Cancel, the function returns a zero-length string ("").

### Example
The following example uses the **InputBox** function to display an input box and assign the string to the variable Input.

```vba
Dim Input
Input = InputBox("Enter your name")
MsgBox ("You entered: " & Input)
```

### 2.9.7 LoadPicture Function

Returns a picture object. Available only on 32-bit platforms.
Syntax

LoadPicture(PictureName)

The PictureName argument is a string expression that indicates the name of the picture file to be loaded.

Remarks

Graphics formats recognized by LoadPicture include bitmap (.bmp) files, icon (.ico) files, run-length encoded (.rle) files, metafile (.wmf) files, enhanced metafiles (.emf), GIF (.gif) files, and JPEG (.jpg) files.

Miscellaneous Functions

2.9.8 MsgBox Function

Displays a message in a dialog box, waits for the user to click a button, and returns a value indicating which button the user clicked.

Syntax

MsgBox(Prompt[, Buttons][, Title][, HelpFile, Context])

The MsgBox function syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prompt</td>
<td>String expression displayed as the message in the dialog box. The maximum length of Prompt is approximately 1024 characters, depending on the width of the characters used. If Prompt consists of more than one line, you can separate the lines using a carriage return character (Chr(13)), a linefeed character (Chr(10)), or carriage return–linefeed character combination (Chr(13) &amp; Chr(10)) between each line.</td>
</tr>
<tr>
<td>Buttons</td>
<td>Numeric expression that is the sum of values specifying the number and type of buttons to display, the icon style to use, the identity of the default button, and the modality of the message box. See the Settings section for values. If omitted, the default value for Buttons is 0.</td>
</tr>
<tr>
<td>Argument</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Title</strong></td>
<td>String expression displayed in the title bar of the dialog box. If you omit <strong>Title</strong>, the application name is placed in the title bar.</td>
</tr>
<tr>
<td><strong>HelpFile</strong></td>
<td>String expression that identifies the Help file to use to provide context-sensitive Help for the dialog box. If <strong>HelpFile</strong> is provided, <strong>Context</strong> must also be provided. Not available on 16-bit platforms.</td>
</tr>
<tr>
<td><strong>Context</strong></td>
<td>Numeric expression that identifies the Help context number assigned by the Help author to the appropriate Help topic. If <strong>Context</strong> is provided, <strong>HelpFile</strong> must also be provided. Not available on 16-bit platforms.</td>
</tr>
</tbody>
</table>

**Settings**

The **Buttons** argument settings are:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>vbOKOnly</strong></td>
<td>0</td>
<td>Display OK button only.</td>
</tr>
<tr>
<td><strong>vbOKCancel</strong></td>
<td>1</td>
<td>Display OK and Cancel buttons.</td>
</tr>
<tr>
<td><strong>vbAbortRetryIgnore</strong></td>
<td>2</td>
<td>Display Abort, Retry, and Ignore buttons.</td>
</tr>
<tr>
<td><strong>vbYesNoCancel</strong></td>
<td>3</td>
<td>Display Yes, No, and Cancel buttons.</td>
</tr>
<tr>
<td><strong>vbYesNo</strong></td>
<td>4</td>
<td>Display Yes and No buttons.</td>
</tr>
<tr>
<td><strong>vbRetryCancel</strong></td>
<td>5</td>
<td>Display Retry and Cancel buttons.</td>
</tr>
<tr>
<td><strong>vbCritical</strong></td>
<td>16</td>
<td>Display Critical Message icon.</td>
</tr>
<tr>
<td><strong>vbQuestion</strong></td>
<td>32</td>
<td>Display Warning Query icon.</td>
</tr>
<tr>
<td><strong>vbExclamation</strong></td>
<td>48</td>
<td>Display Warning Message icon.</td>
</tr>
<tr>
<td><strong>vbInformation</strong></td>
<td>64</td>
<td>Display Information Message icon.</td>
</tr>
<tr>
<td><strong>vbDefaultButton1</strong></td>
<td>0</td>
<td>First button is default.</td>
</tr>
<tr>
<td><strong>vbDefaultButton2</strong></td>
<td>256</td>
<td>Second button is default.</td>
</tr>
<tr>
<td><strong>vbDefaultButton3</strong></td>
<td>512</td>
<td>Third button is default.</td>
</tr>
<tr>
<td><strong>vbDefaultButton4</strong></td>
<td>768</td>
<td>Fourth button is default.</td>
</tr>
</tbody>
</table>
vbApplicationModal  0  Application modal; the user must respond to the message box before continuing work in the current application.

vbSystemModal  4096  System modal; all applications are suspended until the user responds to the message box.

The first group of values (0–5) describes the number and type of buttons displayed in the dialog box; the second group (16, 32, 48, 64) describes the icon style; the third group (0, 256, 512, 768) determines which button is the default; and the fourth group (0, 4096) determines the modality of the message box. When adding numbers to create a final value for the argument Buttons, use only one number from each group.

**Return Values**

The **MsgBox** function has the following return values:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Button</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbOK</td>
<td>1</td>
<td>OK</td>
</tr>
<tr>
<td>vbCancel</td>
<td>2</td>
<td>Cancel</td>
</tr>
<tr>
<td>vbAbort</td>
<td>3</td>
<td>Abort</td>
</tr>
<tr>
<td>vbRetry</td>
<td>4</td>
<td>Retry</td>
</tr>
<tr>
<td>vbIgnore</td>
<td>5</td>
<td>Ignore</td>
</tr>
<tr>
<td>vbYes</td>
<td>6</td>
<td>Yes</td>
</tr>
<tr>
<td>vbNo</td>
<td>7</td>
<td>No</td>
</tr>
</tbody>
</table>

**Remarks**

When both HelpFile and Context are provided, the user can press F1 to view the Help topic corresponding to the context.

If the dialog box displays a Cancel button, pressing the Esc key has the same effect as clicking Cancel. If the dialog box contains a Help button, context-sensitive Help is provided for the dialog box. However, no value is returned until one of the other buttons is clicked.
Example
The following example uses the `MsgBox` function to display a message box and return a value describing which button was clicked.

```vba
Dim MyVar
' MyVar contains either 1 or 2, depending on which button is clicked.
MyVar = MsgBox("Hello World!", 65, "MsgBox Example")
```

2.9.9 RGB Function

Returns a whole number representing an RGB color value.

Syntax

```
RGB(Red, Green, Blue)
```

The `RGB` function has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>Required. Number in the range 0–255 representing the red component of the color.</td>
</tr>
<tr>
<td>Green</td>
<td>Required. Number in the range 0–255 representing the green component of the color.</td>
</tr>
<tr>
<td>Blue</td>
<td>Required. Number in the range 0–255 representing the blue component of the color.</td>
</tr>
</tbody>
</table>

Remarks

Application methods and properties that accept a color specification expect that specification to be a number representing an RGB color value. An RGB color value specifies the relative intensity of red, green, and blue to cause a specific color to be displayed.

The value for any argument to RGB that exceeds 255 is assumed to be 255.
The low-order byte contains the value for red, the middle byte contains the value for green, and the high-order byte contains the value for blue.

For applications that require the byte order to be reversed, the following function will provide the same information with the bytes reversed.

```plaintext
Function RevRGB(red, green, blue)
    RevRGB = CLng(blue + (green * 256) + (red * 65536))
End Function
```

### 2.9.10 ScriptEngine Function

Returns a string representing the scripting language in use.

#### Syntax

```plaintext
ScriptEngine
```

#### Return Values

The `ScriptEngine` function can return any of the following strings:

<table>
<thead>
<tr>
<th>String</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VBScript</td>
<td>Indicates that Microsoft® Visual Basic® Scripting Edition is the current scripting engine.</td>
</tr>
<tr>
<td>JScript</td>
<td>Indicates that Microsoft JScript® is the current scripting engine.</td>
</tr>
<tr>
<td>VBA</td>
<td>Indicates that Microsoft Visual Basic for Applications is the current scripting engine.</td>
</tr>
</tbody>
</table>

#### Example

The following example uses the `ScriptEngine` function to return a string describing the scripting language in use.
**Function** GetScriptEngineInfo

```vbscript
Function GetScriptEngineInfo
    Dim s
    s = "" ' Build string with necessary info.
    s = ScriptEngine & " Version "
    s = s & ScriptEngineMajorVersion & "."
    s = s & ScriptEngineMinorVersion & "."
    s = s & ScriptEngineBuildVersion
    GetScriptEngineInfo = s ' Return the results.
End Function
```

### 2.9.11 ScriptEngineBuildVersion Function

Returns the build version number of the scripting engine in use.

**Syntax**

```vbscript
ScriptEngineBuildVersion
```

**Remarks**

The return value corresponds directly to the version information contained in the DLL for the scripting language in use.

**Example**

The following example uses the `ScriptEngineBuildVersion` function to return the build version number of the scripting engine.

```vbscript
Function GetScriptEngineInfo
    Dim s
    s = "" ' Build string with necessary info.
    s = ScriptEngine & " Version "
    s = s & ScriptEngineMajorVersion & "."
    s = s & ScriptEngineMinorVersion & "."
    s = s & ScriptEngineBuildVersion
    GetScriptEngineInfo = s ' Return the results.
End Function
```
2.9.12 ScriptEngineMajorVersion Function

Returns the major version number of the scripting engine in use.

**Syntax**

```plaintext
ScriptEngineMajorVersion
```

**Remarks**

The return value corresponds directly to the version information contained in the DLL for the scripting language in use.

**Example**

The following example uses the `ScriptEngineMajorVersion` function to return the version number of the scripting engine.

```vba
Function GetScriptEngineInfo
    Dim s
    s = "" ' Build string with necessary info.
    s = ScriptEngine & " Version "
    s = s & ScriptEngineMajorVersion & "."
    s = s & ScriptEngineMinorVersion & "."
    s = s & ScriptEngineBuildVersion
    GetScriptEngineInfo = s ' Return the results.
End Function
```

2.9.13 ScriptEngineMinorVersion Function

Returns the minor version number of the scripting engine in use.
### 2.9.14 SetLocale Function

Sets the global locale and returns the previous locale.

**Syntax**

```
SetLocale(LCID)
```

The `LCID` argument can be any valid 32-bit value or short string that uniquely identifies a geographical locale. Recognized values can be found in the [Locale ID chart](#).

**Remarks**

If `LCID` is zero, the locale is set to match the current system setting.
A locale is a set of user preference information related to the user's language, country/region, and cultural conventions. The locale determines such things as keyboard layout, alphabetic sort order, as well as date, time, number, and currency formats.

**Example**

The following example illustrates the use of the `SetLocale` function.

```vba
Dim d, dow, wdn
SetLocale "cs" ' Czech.
d = Date
dow = WeekDay(d)
wdn = WeekDayName(dow) ' Returns, e.g., "středa" (Wednesday in Czech).
```
2.10 VBScript Statements

- **Call** Statement
- **Const** Statement
- **Dim** Statement
- **Do...Loop** Statement
- **Erase** Statement
- **Execute** Statement
- **Exit** Statement
- **For Each...Next** Statement
- **For...Next** Statement
- **Function** Statement
- **If...Then...Else** Statement
- **On Error** Statement
- **Option Explicit** Statement
- **Private** Statement
- **Public** Statement
- **Randomize** Statement
- **ReDim** Statement
- **Rem** Statement
- **Select Case** Statement
- **Set** Statement
- **Stop** Statement
- **Sub** Statement
- **While...WEnd** Statement
- **With** Statement

### 2.10.1 Call Statement

Transfers control to a **Sub** or **Function** procedure.
**Syntax**

\[ \textbf{Call} \ \textbf{Name} \ [\textbf{ArgumentList}] \]

The **Call** statement syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Call</td>
<td>Optional keyword. If specified, you must enclose <strong>ArgumentList</strong> in parentheses. For Example: Call MyProc(0)</td>
</tr>
<tr>
<td>Name</td>
<td>Required. Name of the procedure to call.</td>
</tr>
<tr>
<td>ArgumentList</td>
<td>Optional. Comma-delimited list of variables, arrays, or expressions to pass to the procedure.</td>
</tr>
</tbody>
</table>

**Remarks**

You are not required to use the **Call** keyword when calling a procedure. However, if you use the **Call** keyword to call a procedure that requires arguments, **ArgumentList** must be enclosed in parentheses. If you omit the **Call** keyword, you also must omit the parentheses around **ArgumentList**. If you use either **Call** syntax to call any intrinsic or user-defined function, the function's return value is discarded.

**Example**

```vbnet
Call MyFunction("Hello World")
Function MyFunction(text)
    MsgBox text
End Function
```

**2.10.2 Const Statement**

Declares constants for use in place of literal values.
Syntax

```
[Public | Private] Const ConstName = Expression
```

The `Const` statement syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public</td>
<td>Optional. Keyword used at a script level to declare constants that are available to all procedures in all scripts. Not allowed in procedures.</td>
</tr>
<tr>
<td>Private</td>
<td>Optional. Keyword used at a script level to declare constants that are available only within the script where the declaration is made. Not allowed in procedures.</td>
</tr>
<tr>
<td>ConstName</td>
<td>Required. Name of the constant; follows standard variable naming conventions.</td>
</tr>
<tr>
<td>Expression</td>
<td>Required. Literal or other constant, or any combination that includes all arithmetic or logical operators except <code>Is</code>.</td>
</tr>
</tbody>
</table>

Remarks

Constants are public by default. Within procedures, constants are always private; their visibility can't be changed. Within a script, the default visibility of a script-level constant can be changed using the `Private` keyword.

To combine several constant declarations on the same line, separate each constant assignment with a comma. When constant declarations are combined in this way, the `Public` or `Private` keyword, if used, applies to all of them.

You can't use variables, user-defined functions, or intrinsic `VBScript` functions (such as `Chr`) in constant declarations. By definition, they can't be constants. You also can't create a constant from any expression that involves an operator, that is, only simple constants are allowed. Constants declared in a `Sub` or `Function` procedure are local to that procedure. A constant declared outside a procedure is defined throughout the script in which it is declared. You can use constants anywhere you can use an expression.
NOTE

Constants can make your scripts self-documenting and easy to modify. Unlike variables, constants can't be inadvertently changed while your script is running.

Example

The following code illustrates the use of the `Const` statement.

```vbscript
Const MyVar = 459 ' Constants are Public by default.
' Declare Private constant.
Private Const MyString = "HELP"
' Declare multiple constants on same line.
Const MyStr = "Hello", MyNumber = 3.4567
```

2.10.3 Dim Statement

Declares variables and allocates storage space.

Syntax

```
Dim VarName[[{Subscripts}]], VarName[[{Subscripts}]]...
```

The `Dim` statement syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VarName</td>
<td>Name of the variable; follows standard variable naming conventions.</td>
</tr>
</tbody>
</table>
| Subscripts | Dimensions of an array variable; up to 60 multiple dimensions may be declared. The `Subscripts` argument uses the following syntax:  
  `upperbound[, upperbound]...`  
  The lower bound of an array is always zero. |
Remarks

Variables declared with Dim at the script level (globally) are available to all procedures in all scripts within the same thread (similar to Public statement). At the procedure level, variables are available only within the procedure.

You can also use the Dim statement with empty parentheses to declare a dynamic array. After declaring a dynamic array, use the ReDim statement within a procedure to define the number of dimensions and elements in the array. If you try to redeclare a dimension for an array variable whose size was explicitly specified in a Dim statement, an error occurs.

TIP

When you use the Dim statement in a procedure, you generally put the Dim statement at the beginning of the procedure.

Example

The following examples illustrate the use of the Dim statement.

Dim Names(9) ' Declare an array with 10 elements.
Dim Names() ' Declare a dynamic array.
Dim MyVar, MyNum ' Declare two variables.

2.10.4 Do...Loop Statement

Repeats a block of statements while a condition is True or until a condition becomes True.

Syntax

Do [{While | Until} Condition]
[Statements]
[Exit Do]
[Statements]
Loop
Or, you can use this syntax:

```
Do
  [Statements]
  [Exit Do]
  [Statements]
Loop [{While | Until} Condition]
```

The **Do...Loop** statement syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Condition</strong></td>
<td>Numeric or string expression that is <strong>True</strong> or <strong>False</strong>. If <strong>Condition</strong> is <strong>Null</strong>, <strong>Condition</strong> is treated as <strong>False</strong>.</td>
</tr>
<tr>
<td><strong>Statements</strong></td>
<td>One or more statements that are repeated while or until <strong>Condition</strong> is <strong>True</strong>.</td>
</tr>
</tbody>
</table>

**Remarks**

The **Exit Do** can only be used within a **Do...Loop** control structure to provide an alternate way to exit a **Do...Loop**. Any number of **Exit Do** statements may be placed anywhere in the **Do...Loop**. Often used with the evaluation of some condition (for example, **If...Then**), **Exit Do** transfers control to the statement immediately following the **Loop**.

When used within nested **Do...Loop** statements, **Exit Do** transfers control to the loop that is nested one level above the loop where it occurs.

**Example**

The following examples illustrate use of the **Do...Loop** statement.

```
Do Until DefResp = vbNo
  ' Generate a random integer between 1 and 6.
  MyNum = Int (6 * Rnd + 1)
  DefResp = MsgBox (MyNum & " Do you want another number?", vbYesNo)
Loop
Dim Check, Counter
```
Check = True
Counter = 0 ' Initialize variables.
Do ' Outer loop.
  Do While Counter < 20 ' Inner loop.
    Counter = Counter + 1 ' Increment Counter.
  If Counter = 10 Then ' If condition is True...
    Check = False ' set value of flag to False.
    Exit Do ' Exit inner loop.
  End If
Loop
Loop Until Check = False ' Exit outer loop immediately.

2.10.5 Erase Statement

Reinitializes the elements of fixed-size arrays and deallocates dynamic-array storage space.

Syntax

Erase Array

The Array argument is the name of the array variable to be erased.

Remarks

It is important to know whether an array is fixed-size (ordinary) or dynamic because Erase behaves differently depending on the type of array. Erase recovers no memory for fixed-size arrays. Erase sets the elements of a fixed array as follows:

<table>
<thead>
<tr>
<th>Type of array</th>
<th>Effect of Erase on fixed-array elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed numeric array</td>
<td>Sets each element to zero.</td>
</tr>
<tr>
<td>Fixed string array</td>
<td>Sets each element to zero-length (&quot;&quot;).</td>
</tr>
<tr>
<td>Array of objects</td>
<td>Sets each element to the special value Nothing.</td>
</tr>
</tbody>
</table>

Erase frees the memory used by dynamic arrays. Before your program can refer to the dynamic array again, it must redelclare the array variable's dimensions using a ReDim statement.
Example

The following example illustrates the use of the `Erase` statement.

```vbscript
Dim NumArray(9)
Dim DynamicArray()
ReDim DynamicArray(9) ' Allocate storage space.
Erase NumArray       ' Each element is reinitialized.
Erase DynamicArray   ' Free memory used by array.
```

2.10.6 Execute Statement

Executes one or more specified statements.

Syntax

```vbscript
Execute Statement
```

The required `Statement` argument is a string expression containing one or more statements for execution. Include multiple statements in the `Statement` argument, using colons or embedded line breaks to separate them.

Remarks

In `VBScript`, `x = y` can be interpreted two ways. The first is as an assignment statement, where the value of `y` is assigned to `x`. The second interpretation is as an expression that tests if `x` and `y` have the same value. If they do, `Result` is `True`; if they are not, `Result` is `False`. The `Execute` statement always uses the first interpretation, whereas the `Eval` method always uses the second.

The context in which the `Execute` statement is invoked determines what objects and variables are available to the code being run. In-scope objects and variables are available to code running in an `Execute` statement. However, it is important to understand that if you execute code that creates a procedure, that procedure does not inherit the scope of the procedure in which it occurred.
Like any procedure, the new procedure's scope is global, and it inherits everything in the global scope. Unlike any other procedure, its context is not global scope, so it can only be executed in the context of the procedure where the `Execute` statement occurred. However, if the same `Execute` statement is invoked outside of a procedure (i.e., in global scope), not only does it inherit everything in global scope, but it can also be called from anywhere, since its context is global.

**Example**

The following example shows how the `Execute` statement can be rewritten so you don't have to enclose the entire procedure in the quotation marks.

```vbscript
S = "Sub Proc2" & vbCrLf
S = S & "   Print X" & vbCrLf
S = S & "End Sub"
Execute S
```

### 2.10.7 Exit Statement

Exits a block of `Do...Loop`, `For Each...Next`, `For...Next`, `Function`, or `Sub` code.

**Syntax**

- `Exit Do`
- `Exit For`
- `Exit Function`
- `Exit Sub`

The `Exit` statement syntax has these forms:

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exit Do</td>
<td>Provides a way to exit a <code>Do...Loop</code> statement. It can be used only inside a <code>Do...Loop</code> statement. <code>Exit Do</code> transfers control to the statement following the <code>Loop</code> statement. When used within nested <code>Do...Loop</code> statements, <code>Exit Do</code> transfers control to the loop that is one nested level above the loop where it occurs.</td>
</tr>
</tbody>
</table>
### Exit For

Provides a way to exit a **For** loop. It can be used only in a **For...Next** or **For Each...Next** loop. **Exit For** transfers control to the statement following the **Next** statement. When used within nested **For** loops, **Exit For** transfers control to the loop that is one nested level above the loop where it occurs.

### Exit Function

Immediately exits the **Function** procedure in which it appears. Execution continues with the statement following the statement that called the **Function**.

### Exit Sub

Immediately exits the **Sub** procedure in which it appears. Execution continues with the statement following the statement that called the **Sub**.

## Example

The following example illustrates the use of the **Exit** statement.

```vbnet
Sub RandomLoop
    Dim I, MyNum
    Do ' Set up infinite loop.
        For I = 1 To 1000 ' Loop 1000 times.
            MyNum = Int(Rnd * 100) ' Generate random numbers.
            Select Case MyNum ' Evaluate random number.
                Case 17: MsgBox "Case 17"
                    Exit For ' If 17, exit For...Next.
                Case 29: MsgBox "Case 29"
                    Exit Do ' If 29, exit Do...Loop.
                Case 54: MsgBox "Case 54"
                    Exit Sub ' If 54, exit Sub procedure.
            End Select
        Next
    Loop
End Sub
```
2.10.8 For Each...Next Statement

Repeats a group of statements for each element in an array or collection.

Syntax

```
For Each Element In Group

[Statements]

[Exit For]

[Statements]

Next [Element]
```

The *For Each...Next* statement syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element</td>
<td>Variable used to iterate through the elements of the collection or array. For collections, <em>Element</em> can only be a <em>Variant</em> variable, a generic <em>Object</em> variable, or any specific Automation object variable. For arrays, <em>Element</em> can only be a <em>Variant</em> variable.</td>
</tr>
<tr>
<td>Group</td>
<td>Name of an object collection or array.</td>
</tr>
<tr>
<td>Statements</td>
<td>One or more statements that are executed on each item in <em>Group</em>.</td>
</tr>
</tbody>
</table>

Remarks

The *For Each* block is entered if there is at least one element in *Group*. Once the loop has been entered, all the statements in the loop are executed for the first element in *Group*. As long as there are more elements in *Group*, the statements in the loop continue to execute for each element. When there are no more elements in *Group*, the loop is exited and execution continues with the statement following the *Next* statement.
The `Exit For` can only be used within a `For Each...Next` or `For...Next` control structure to provide an alternate way to exit. Any number of `Exit For` statements may be placed anywhere in the loop. The `Exit For` is often used with the evaluation of some condition (for example, `If...Then`), and transfers control to the statement immediately following `Next`.

You can nest `For Each...Next` loops by placing one `For Each...Next` loop within another. However, each loop `Element` must be unique.

**NOTE**

If you omit `Element` in a `Next` statement, execution continues as if you had included it. If a `Next` statement is encountered before its corresponding `For` statement, an error occurs.

---

### Example

The following example illustrates use of the `For Each...Next` statement.

```vbscript
Function ShowFolderList(folderspec)
    Dim fso, f, f1, fc, s
    Set fso = CreateObject("Scripting.FileSystemObject")
    Set f = fso.GetFolder(folderspec)
    Set fc = f.Files
    For Each f1 In fc
        s = s & f1.name
        s = s & "<BR>"
    Next
    ShowFolderList = s
End Function
```

### 2.10.9 For...Next Statement

Repeats a group of statements a specified number of times.

**Syntax**

```
For Counter = Start To End [Step Step]
    [Statements]
[Exit For]
```
The **For...Next** statement syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counter</td>
<td>Numeric variable used as a loop counter. The variable can't be an array element or an element of a user-defined type.</td>
</tr>
<tr>
<td>Start</td>
<td>Initial value of Counter.</td>
</tr>
<tr>
<td>End</td>
<td>Final value of Counter.</td>
</tr>
<tr>
<td>Step</td>
<td>Amount Counter is changed each time through the loop. If not specified, Step defaults to one.</td>
</tr>
<tr>
<td>Statements</td>
<td>One or more statements between For and Next that are executed the specified number of times.</td>
</tr>
</tbody>
</table>

**Remarks**

The Step argument can be either positive or negative. The value of the Step argument determines loop processing as follows:

<table>
<thead>
<tr>
<th>Value</th>
<th>Loop executes if</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive or 0</td>
<td>Counter &lt;= End</td>
</tr>
<tr>
<td>Negative</td>
<td>Counter &gt;= End</td>
</tr>
</tbody>
</table>

Once the loop starts and all statements in the loop have executed, Step is added to Counter. At this point, either the statements in the loop execute again (based on the same test that caused the loop to execute initially), or the loop is exited and execution continues with the statement following the **Next** statement.

**TIP**

Changing the value of Counter while inside a loop can make it more difficult to read and debug your code.
The **Exit For** can only be used within a **For Each...Next** or **For...Next** control structure to provide an alternate way to exit. Any number of **Exit For** statements may be placed anywhere in the loop. The **Exit For** is often used with the evaluation of some condition (for example, **If...Then**), and transfers control to the statement immediately following **Next**.

You can nest **For...Next** loops by placing one **For...Next** loop within another. Give each loop a unique variable name as its **Counter**. The following construction is correct.

```
For I = 1 To 10
    For J = 1 To 10
        For K = 1 To 10
            ...  
        Next
    Next
Next
```

---

### 2.10.10 Function Statement

Declares the name, arguments, and code that form the body of a **Function** procedure.

**Syntax**

```
[Public | Private] Function Name ([ArgList])
[Statements]
[Name = Expression]
[Exit Function]
[Statements]
[Name = Expression]
End Function
```

The **Function** statement syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
</table>
### Public
Indicates that the **Function** procedure is accessible to all other procedures in all scripts.

### Private
Indicates that the **Function** procedure is accessible only to other procedures in the script where it is declared or if the function is a member of a class, and that the **Function** procedure is accessible only to other procedures in that class.

### Name
Name of the **Function**; follows standard variable naming conventions.

### ArgList
List of variables representing arguments that are passed to the **Function** procedure when it is called. Multiple variables are separated by commas.

### Statements
Any group of statements to be executed within the body of the **Function** procedure.

### Expression
Return value of the **Function**.

The **ArgList** argument has the following syntax and parts:

```
[ByVal | ByRef] VarName( )
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ByVal</td>
<td>Indicates that the argument is passed by value.</td>
</tr>
<tr>
<td>ByRef</td>
<td>Indicates that the argument is passed by reference.</td>
</tr>
<tr>
<td>VarName</td>
<td>Name of the variable representing the argument; follows standard variable naming conventions.</td>
</tr>
</tbody>
</table>

### Remarks
If not explicitly specified using either **Public** or **Private**, **Function** procedures are public by default, that is, they are visible to all other procedures in your script. The value of local variables in a **Function** is not preserved between calls to the procedure.

You can't define a **Function** procedure inside any other procedure (e.g., **Sub** or property getter).
The `Exit Function` statement causes an immediate exit from a `Function` procedure. Program execution continues with the statement that follows the statement that called the `Function` procedure. Any number of `Exit Function` statements can appear anywhere in a `Function` procedure.

Like a `Sub` procedure, a `Function` procedure is a separate procedure that can take arguments, perform a series of statements, and change the values of its arguments. However, unlike a `Sub` procedure, you can use a `Function` procedure on the right side of an expression in the same way you use any intrinsic function, such as `Sqr`, `Cos`, or `Chr`, when you want to use the value returned by the function.

You call a `Function` procedure using the function name, followed by the argument list in parentheses, in an expression. See the `Call` statement for specific information on how to call `Function` procedures.

**CAUTION**

`Function` procedures can be recursive, that is, they can call themselves to perform a given task. However, recursion can lead to stack overflow.

To return a value from a function, assign the value to the function name. Any number of such assignments can appear anywhere within the procedure. If no value is assigned to `Name`, the procedure returns a default value: a numeric function returns 0 and a string function returns a zero-length string (""). A function that returns an object reference returns `Nothing` if no object reference is assigned to `Name` (using `Set`) within the `Function`.

Variables used in `Function` procedures fall into two categories: those that are explicitly declared within the procedure and those that are not. Variables that are explicitly declared in a procedure (using `Dim` or the equivalent) are always local to the procedure. Variables that are used but not explicitly declared in a procedure are also local unless they are explicitly declared at some higher level outside the procedure.

**CAUTION**

A procedure can use a variable that is not explicitly declared in the procedure, but a naming conflict can occur if anything you have defined at the script level has the same name. If your procedure refers to an undeclared variable that has the same name as another procedure, constant, or variable, it is assumed that your procedure is referring to that script-level name. To avoid this kind of conflict, use an `Option Explicit` statement to force explicit declaration of variables.
CAUTION

VBScript may rearrange arithmetic expressions to increase internal efficiency. Avoid using a Function procedure in an arithmetic expression when the function changes the value of variables in the same expression.

2.10.11 If...Then...Else Statement

Conditionally executes a group of statements, depending on the value of an expression.

Syntax

```
If Condition Then Statements [Else ElseStatements]
```

Or, you can use the block form syntax:

```
If Condition Then
    [Statements]
[ElseIf Condition-n Then
    [ElseIfStatements]]...
[Else
    [ElseStatements]]
End If
```

The If...Then...Else statement syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Condition</td>
<td>One or more of the following two types of expressions: A numeric or string expression that evaluates to True or False. If Condition is Null, Condition is treated as False.</td>
</tr>
</tbody>
</table>
An expression of the form `TypeOf ObjectName Is ObjectType`. The `ObjectName` is any object reference and `ObjectType` is any valid object type. The expression is `True` if `ObjectName` is of the object type specified by `ObjectType`; otherwise it is `False`.

<table>
<thead>
<tr>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>You can use the single-line form (first syntax) for short, simple tests. However, the block form (second syntax) provides more structure and flexibility than the single-line form and is usually easier to read, maintain, and debug.</td>
</tr>
</tbody>
</table>

| NOTE |
| With the single-line syntax, it is possible to have multiple statements executed as the result of an `If...Then` decision, but they must all be on the same line and separated by colons, as in the following statement. |

```
If A > 10 Then A = A + 1 : B = B + A ; C = C + B
```

When executing a block `If` (second syntax), `Condition` is tested. If `Condition` is `True`, the statements following `Then` are executed. If `Condition` is `False`, each `ElseIf` (if any) is evaluated in turn. When a `True` condition is found, the statements following the associated `Then` are executed. If none of the `ElseIf` statements are `True` (or there are no `ElseIf` clauses), the statements following `Else` are executed. After executing the statements following `Then` or `Else`, execution continues with the statement following `End If`. 
The **Else** and **ElseIf** clauses are both optional. You can have as many **Else** statements as you want in a block **If**, but none can appear after the **Else** clause. Block **If** statements can be nested; that is, contained within one another.

What follows the **Then** keyword is examined to determine whether or not a statement is a block **If**. If anything other than a comment appears after **Then** on the same line, the statement is treated as a single-line **If** statement.

A block **If** statement must be the first statement on a line. The block **If** must end with an **End If** statement.

### 2.10.12 On Error Statement

 Enables error-handling.

**Syntax**

```
On Error Resume Next
```

**Remarks**

If you don't use an **On Error Resume Next** statement, any run-time error that occurs is fatal; that is, an error message is displayed and execution stops.

**On Error Resume Next** causes execution to continue with the statement immediately following the statement that caused the run-time error. The procedure call is considered one statement. This means that if an error occurs inside called procedure (and this procedure doesn't contain the **On Error Resume Next** statement), statements that follow error inside the procedure are not executed, but the script continues with the statement immediately following the procedure call.

This allows execution to continue despite a run-time error. You can then build the error-handling routine inline within the procedure. An **On Error Resume Next** statement becomes inactive when another procedure is called, so you should execute an **On Error Resume Next** statement in each called routine if you want inline error handling within that routine.
Example

The following example illustrates use of the On Error Resume Next statement.

On Error Resume Next
Err.Raise 6 ' Raise an overflow error.
MsgBox ("Error # " & CStr(Err.Number) & " " & Err.Description)
Err.Clear ' Clear the error.

2.10.13  Option Explicit Statement

Forces explicit declaration of all variables in a script.

Syntax

Option Explicit

Remarks

If used, the Option Explicit statement must appear in a script before any other statements.

When you use the Option Explicit statement, you must explicitly declare all variables using the Dim, Private, Public, or ReDim statements. If you attempt to use an undeclared variable name, an error occurs.

TIP

Use Option Explicit to avoid incorrectly typing the name of an existing variable or to avoid confusion in code where the scope of the variable is not clear.

Example

The following example illustrates use of the Option Explicit statement.

Option Explicit ' Force explicit variable declaration.
Dim MyVar ' Declare variable.
MyInt = 10 ' Undeclared variable generates error.
MyVar = 10 ' Declared variable does not generate error.
2.10.14 Private Statement

Declares private variables and allocates storage space. Declares, in a Class block, a private variable.

Syntax

```
Private VarName[((Subscripts))], VarName[((Subscripts))]...
```

The **Private** statement syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VarName</td>
<td>Name of the variable; follows standard variable naming conventions.</td>
</tr>
<tr>
<td>Subscripts</td>
<td>Dimensions of an array variable; up to 60 multiple dimensions may be declared. The <code>Subscripts</code> argument uses the following syntax: upper[, upper]... The lower bound of an array is always zero.</td>
</tr>
</tbody>
</table>

Remarks

A variable declared with the **Private** statement at a script level (globally) is available in all scripts within the same thread (similar to the **Public** statement). The **Private** statement can still be meaningfully used in a declaration of objects (with the **Class** statement).

A variable that refers to an object must be assigned an existing object using the **Set** statement before it can be used. Until it is assigned an object, the declared object variable is initialized as **Empty**.

You can also use the **Private** statement with empty parentheses to declare a dynamic array. After declaring a dynamic array, use the **ReDim** statement within a procedure to define the number of dimensions and elements in the array. If you try to redeclare a dimension for an array variable whose size was explicitly specified in a **Private**, **Public**, or **Dim** statement, an error occurs.
Example

The following example illustrates use of the **Private** statement.

```vbscript
Private MyNumber  ' Private Variant variable.
Private MyArray(9)  ' Private array variable.
' Multiple Private declarations of Variant variables.
Private MyNumber, MyVar, YourNumber
```

### 2.10.15 Public Statement

Declares public variables and allocates storage space. Declares, in a *Class* block, a private variable.

**Syntax**

```
Public VarName[(Subscripts)][, VarName[(Subscripts)]]...
```

The **Public** statement syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VarName</td>
<td>Name of the variable; follows standard variable naming conventions.</td>
</tr>
<tr>
<td>Subscripts</td>
<td>Dimensions of an array variable; up to 60 multiple dimensions may be declared. The <em>Subscripts</em> argument uses the following syntax: upper[, upper]... The lower bound of an array is always zero.</td>
</tr>
</tbody>
</table>

**Remarks**

**Public** statement variables are available to all procedures in all scripts.

A variable that refers to an object must be assigned an existing object using the **Set** statement before it can be used. Until it is assigned an object, the declared object variable is initialized as **Empty**.
You can also use the `Public` statement with empty parentheses to declare a dynamic array. After declaring a dynamic array, use the `ReDim` statement within a procedure to define the number of dimensions and elements in the array. If you try to redeclare a dimension for an array variable whose size was explicitly specified in a `Private`, `Public`, or `Dim` statement, an error occurs.

### Example

The following example illustrates the use of the `Public` statement.

```vbnet
Public MyNumber  ' Public Variant variable.
Public MyArray(9) ' Public array variable.
' Multiple Public declarations of Variant variables.
Public MyNumber, MyVar, YourNumber
```

## 2.10.16 Randomize Statement

Initializes the random-number generator.

### Syntax

```vbnet
Randomize [Number]
```

The `Number` argument can be any valid numeric expression.

### Remarks

`Randomize` uses `Number` to initialize the `Rnd` function's random-number generator, giving it a new seed value. If you omit `Number`, the value returned by the system timer is used as the new seed value.

If `Randomize` is not used, the `Rnd` function (with no arguments) uses the same number as a seed the first time it is called, and thereafter uses the last generated number as a seed value.
NOTE

To repeat sequences of random numbers, call Rnd with a negative argument immediately before using Randomize with a numeric argument. Using Randomize with the same value for Number does not repeat the previous sequence.

Example

The following example illustrates use of the Randomize statement.

```
Dim MyValue, Response
Randomize ' Initialize random-number generator.
Do Until Response = vbNo
    MyValue = Int((6 * Rnd) + 1) ' Generate random value between 1 and 6.
    MsgBox MyValue
    Response = MsgBox ("Roll again? ", vbYesNo)
Loop
```

2.10.17 ReDim Statement

Declares dynamic-array variables, and allocates or reallocates storage space at procedure level.

Syntax

```
ReDim [Preserve] VarName(Subscripts), VarName(Subscripts)... 
```

The ReDim statement syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preserve</td>
<td>Preserves the data in an existing array when you change the size of the last dimension.</td>
</tr>
<tr>
<td>VarName</td>
<td>Name of the variable; follows standard variable naming conventions.</td>
</tr>
<tr>
<td>Subscripts</td>
<td>Dimensions of an array variable; up to 60 multiple dimensions may be declared. The Subscripts argument uses the following syntax:</td>
</tr>
</tbody>
</table>
Remarks

The **ReDim** statement is used to size or resize a dynamic array that has already been formally declared using a **Private**, **Public**, or **Dim** statement with empty parentheses (without dimension subscripts). You can use the **ReDim** statement repeatedly to change the number of elements and dimensions in an array.

If you use the **Preserve** keyword, you can resize only the last array dimension, and you can't change the number of dimensions at all. For example, if your array has only one dimension, you can resize that dimension because it is the last and only dimension. However, if your array has two or more dimensions, you can change the size of only the last dimension and still preserve the contents of the array.

The following example shows how you can increase the size of the last dimension of a dynamic array without erasing any existing data contained in the array.

```vbnet
ReDim X(10, 10, 10)
'...
ReDim Preserve X(10, 10, 15)
```

**CAUTION**

If you make an array smaller than it was originally, data in the eliminated elements is lost.

When variables are initialized, a numeric variable is initialized to 0 and a string variable is initialized to a zero-length string ("""). A variable that refers to an object must be assigned an existing object using the **Set** statement before it can be used. Until it is assigned an object, the declared object variable has the special value **Nothing**.
Syntax

Rem Comment

or

' Comment

The Comment argument is the text of any comment you want to include. After the Rem keyword, a space is required before Comment.

Remarks

As shown in the syntax section, you can use an apostrophe (') instead of the Rem keyword. If the Rem keyword follows other statements on a line, it must be separated from the statements by a colon. However, when you use an apostrophe, the colon is not required after other statements.

Example

The following example illustrates the use of the Rem statement.

Dim MyStr1, MyStr2
MyStr1 = "Hello" : Rem Comment after a statement separated by a colon.
MyStr2 = "Goodbye" ' This is also a comment; no colon is needed.
Rem Comment on a line with no code; no colon is needed.

2.10.19 Select Case Statement

Executes one of several groups of statements, depending on the value of an expression.
Syntax

Select Case TestExpression

[Case ExpressionList-n

[Statements-n]]...

[Case Else ExpressionList-n

[ElseStatements-n]]

End Select

The Select Case statement syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TestExpression</td>
<td>Any numeric or string expression.</td>
</tr>
<tr>
<td>ExpressionList-n</td>
<td>Required if Case appears. Delimited list of one or more expressions.</td>
</tr>
<tr>
<td>Statements-n</td>
<td>One or more statements executed if TestExpression matches any part of ExpressionList-n.</td>
</tr>
<tr>
<td>ElseStatements-n</td>
<td>One or more statements executed if TestExpression doesn't match any of the Case clauses.</td>
</tr>
</tbody>
</table>

Remarks

If TestExpression matches any Case ExpressionList expression, the statements following that Case clause are executed up to the next Case clause, or for the last clause, up to End Select. Control then passes to the statement following End Select. If TestExpression matches an ExpressionList expression in more than one Case clause, only the statements following the first match are executed.
The **Case Else** clause is used to indicate the **ElseStatements** to be executed if no match is found between the **TestExpression** and an **ExpressionList** in any of the other **Case** selections. Although not required, it is a good idea to have a **Case Else** statement in your **Select Case** block to handle unforeseen **TestExpression** values. If no **Case ExpressionList** matches **TestExpression** and there is no **Case Else** statement, execution continues at the statement following **End Select**.

**Select Case** statements can be nested. Each nested **Select Case** statement must have a matching **End Select** statement.

### Example

The following example illustrates the use of the **Select Case** statement.

```vbscript
Dim Color, MyVar
Sub ChangeBackground (Color)
    MyVar = lcase (Color)
    Select Case MyVar
        Case "red"
            document.bgColor = "red"
        Case "green"
            document.bgColor = "green"
        Case "blue"
            document.bgColor = "blue"
        Case Else
            MsgBox "pick another color"
    End Select
End Sub
```

### 2.10.20 Set Statement

Assigns an object reference to a variable or property, or associates a procedure reference with an event.
Syntax 1

Set ObjectVar = {ObjectExpression | New ClassName | Nothing}

Syntax 2

Set Object.EventName = GetRef(ProcName)

The Set statement syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObjectVar</td>
<td>Required. Name of the variable or property; follows standard variable naming conventions.</td>
</tr>
<tr>
<td>ObjectExpression</td>
<td>Optional. Expression consisting of the name of an object, another declared variable of the same object type, or a function or method that returns an object of the same object type.</td>
</tr>
<tr>
<td>New</td>
<td>Keyword used to create a new instance of a class. If ObjectVar contained a reference to an object, that reference is released when the new one is assigned. The New keyword can only be used to create an instance of a class.</td>
</tr>
<tr>
<td>ClassName</td>
<td>Optional. Name of the class being created. A class and its members are defined using the Class statement.</td>
</tr>
<tr>
<td>Nothing</td>
<td>Optional. Discontinues association of ObjectVar with any specific object or class. Assigning ObjectVar to Nothing releases all the system and memory resources associated with the previously referenced object when no other variable refers to it.</td>
</tr>
<tr>
<td>Object</td>
<td>Required. Name of the object with which EventName is associated.</td>
</tr>
<tr>
<td>EventName</td>
<td>Required. Name of the event to which the function is to be bound.</td>
</tr>
<tr>
<td>ProcName</td>
<td>Required. String containing the name of the Sub or Function being associated with the EventName.</td>
</tr>
</tbody>
</table>
Remarks

To be valid, \textit{ObjectVar} must be an object type consistent with the object being assigned to it.

The \textbf{Dim}, \textbf{Private}, \textbf{Public}, or \textbf{ReDim} statements only declare a variable that refers to an object. No actual object is referred to until you use the \textbf{Set} statement to assign a specific object.

Generally, when you use \textbf{Set} to assign an object reference to a variable, no copy of the object is created for that variable. Instead, a reference to the object is created. More than one object variable can refer to the same object. Because these variables are references to (rather than copies of) the object, any change in the object is reflected in all variables that refer to it.

Using the \textbf{New} keyword allows you to concurrently create an instance of a class and assign it to an object reference variable. The variable to which the instance of the class is being assigned must already have been declared with the \textbf{Dim} (or equivalent) statement.

Refer to the documentation for the \textbf{GetRef} function for information on using \textbf{Set} to associate a procedure with an event.

---

Example

The following example illustrates the use of the \textbf{Set} statement.

\begin{verbatim}
Function ShowFreeSpace(drvPath)
    Dim fso, d, s
    Set fso = CreateObject("Scripting.FileSystemObject")
    Set d = fso.GetDrive(fso.GetDriveName(drvPath))
    s = "Drive " & UCase(drvPath) & " - "
    s = s & d.VolumeName & "<BR>"
    s = s & "Free Space: " & FormatNumber(d.FreeSpace / 1024, 0)
    s = s & " Kbytes"
    ShowFreeSpace = s
End Function
\end{verbatim}
2.10.21 Stop Statement

Interrupts the code execution and launches an external tool for debugging scripts (Just-In-Time debugger) if debugging is enabled.

Syntax

Stop

Remarks

A Just-In-Time debugger must be installed (e.g., Microsoft Script Debugger or the tool contained in Microsoft Visual Studio).

Script debugging must be enabled in the Windows operating system. This can be done by activating the *Enable script debugging with external tool (Just-In-Time debugger)* option (Reliance Design > Tools > Environment Options).

2.10.22 Sub Statement

Declares the name, arguments, and code that form the body of a Sub procedure.

Syntax

[Public | Private] Sub Name [(ArgList)]

[Statements]

[Exit Sub]

[Statements]

End Sub

The Sub statement syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Public | Indicates that the **Sub** procedure is accessible to all other procedures in all scripts.

Private | Indicates that the **Sub** procedure is accessible only to other procedures in the script where it is declared.

Name | Name of the **Sub**; follows standard variable naming conventions.

ArgList | List of variables representing arguments that are passed to the **Sub** procedure when it is called. Multiple variables are separated by commas.

Statements | Any group of statements to be executed within the body of the **Sub** procedure.

The ArgList argument has the following syntax and parts:

\[
[\text{ByVal} | \text{ByRef}] \text{VarName}(\ )
\]

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ByVal</td>
<td>Indicates that the argument is passed by value.</td>
</tr>
<tr>
<td>ByRef</td>
<td>Indicates that the argument is passed by reference.</td>
</tr>
<tr>
<td>VarName</td>
<td>Name of the variable representing the argument; follows standard variable naming conventions.</td>
</tr>
</tbody>
</table>

Remarks

If not explicitly specified using either Public or Private, **Sub** procedures are public by default, that is, they are visible to all other procedures in your script. The value of local variables in a **Sub** procedure is not preserved between calls to the procedure.

You can't define a **Sub** procedure inside any other procedure (e.g., **Function** or property getter).
The `Exit Sub` statement causes an immediate exit from a `Sub` procedure. Program execution continues with the statement that follows the statement that called the `Sub` procedure. Any number of `Exit Sub` statements can appear anywhere in a `Sub` procedure.

Like a `Function` procedure, a `Sub` procedure is a separate procedure that can take arguments, perform a series of statements, and change the value of its arguments. However, unlike a `Function` procedure, which returns a value, a `Sub` procedure can’t be used in an expression.

You call a `Sub` procedure using the procedure name followed by the argument list. See the `Call` statement for specific information on how to call `Sub` procedures.

**CAUTION**

`Sub` procedures can be recursive, that is, they can call themselves to perform a given task. However, recursion can lead to stack overflow.

Variables used in `Sub` procedures fall into two categories: those that are explicitly declared within the procedure and those that are not. Variables that are explicitly declared in a procedure (using `Dim` or the equivalent) are always local to the procedure. Variables that are used but not explicitly declared in a procedure are also local, unless they are explicitly declared at some higher level outside the procedure.

**CAUTION**

A procedure can use a variable that is not explicitly declared in the procedure, but a naming conflict can occur if anything you have defined at the script level has the same name. If your procedure refers to an undeclared variable that has the same name as another procedure, constant or variable, it is assumed that your procedure is referring to that script-level name. To avoid this kind of conflict, use an `Option Explicit` statement to force explicit declaration of variables.

### 2.10.23 While...WEnd Statement

Executes a series of statements as long as a given condition is `True`. 
Syntax

```vbnet
While Condition
    [Statements]
WEnd
```

The **While...WEnd** statement syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Condition</td>
<td>Numeric or string expression that evaluates to <strong>True</strong> or <strong>False</strong>. If <strong>Condition</strong> is <strong>Null</strong>, <strong>Condition</strong> is treated as <strong>False</strong>.</td>
</tr>
<tr>
<td>Statements</td>
<td>One or more statements executed while condition is <strong>True</strong>.</td>
</tr>
</tbody>
</table>

Remarks

If **Condition** is **True**, all statements in **Statements** are executed until the **WEnd** statement is encountered. Control then returns to the **While** statement and **Condition** is again checked. If **Condition** is still **True**, the process is repeated. If it is not **True**, execution resumes with the statement following the **WEnd** statement.

**While...WEnd** loops can be nested to any level. Each **WEnd** matches the most recent **While**.

**TIP**

The **Do...Loop** statement provides a more structured and flexible way to perform looping.

Example

The following example illustrates use of the **While...WEnd** statement.

```
Dim Counter
Counter = 0 ' Initialize variable.
While Counter < 20 ' Test value of Counter.
    Counter = Counter + 1 ' Increment Counter.
```

```
Alert Counter

' End While loop when Counter > 19.

2.10.24 With Statement

Executes a series of statements on a single object.

Syntax

With Object

Statements

End With

The **With** statement syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td>Required. Name of an object or a function that returns an object.</td>
</tr>
<tr>
<td>Statements</td>
<td>Required. One or more statements to be executed on <em>Object</em>.</td>
</tr>
</tbody>
</table>

Remarks

The **With** statement allows you to perform a series of statements on a specified object without requalifying the name of the object. For example, to change a number of different properties on a single object, place the property assignment statements within the **With** control structure, referring to the object once instead of referring to it with each property assignment. The following example illustrates use of the **With** statement to assign values to several properties of the same object.

```vbscript
With MyLabel
    .Height = 2000
    .Width = 2000
    .Caption = "This is MyLabel"
End With
```

While property manipulation is an important aspect of **With** functionality, it is not the only use. Any legal code can be used within a **With** block.
NOTE

Once a With block is entered, Object can't be changed. As a result, you can't use a single With statement to affect a number of different objects.

You can nest With statements by placing one With block within another. However, because members of outer With blocks are masked within the inner With blocks, you must provide a fully qualified object reference in an inner With block to any member of an object in an outer With block.

IMPORTANT

Do not jump into or out of With blocks. If statements in a With block are executed, but either the With or End With statement is not executed, you may get errors or unpredictable behavior.

 VBScript Statements
2.11 VBScript Constants

VBScript definuje konstanty pro zjednodušení programování. Následující konstanty mohou být použity kdekoli v kódu místo skutečných hodnot:

- **Color Constants**
- **Comparison Constants**
- **Date and Time Constants**
- **Date Format Constants**
- **Miscellaneous Constants**
- **MsgBox Constants**
- **String Constants**
- **TriState Constants**
- **VarType Constants**

### 2.11.1 Color Constants

Následující konstanty jsou vždy k dispozici, takže nebudete muset je definovat před použitím.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbBlack</td>
<td>&amp;h00</td>
<td>Black</td>
</tr>
<tr>
<td>vbRed</td>
<td>&amp;hFF</td>
<td>Red</td>
</tr>
<tr>
<td>vbGreen</td>
<td>&amp;hFF00</td>
<td>Green</td>
</tr>
<tr>
<td>vbYellow</td>
<td>&amp;hFFFF</td>
<td>Yellow</td>
</tr>
<tr>
<td>vbBlue</td>
<td>&amp;hFO000</td>
<td>Blue</td>
</tr>
<tr>
<td>vbMagenta</td>
<td>&amp;hFO0FF</td>
<td>Magenta</td>
</tr>
<tr>
<td>vbCyan</td>
<td>&amp;hFFFF00</td>
<td>Cyan</td>
</tr>
<tr>
<td>vbWhite</td>
<td>&amp;hFFFFFF</td>
<td>White</td>
</tr>
</tbody>
</table>
2.11.2 Comparison Constants

Since these constants are built into VBScript, you don't have to define them before using them. Use them anywhere in your code to represent the values shown for each.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbBinaryCompare</td>
<td>0</td>
<td>Perform a binary comparison.</td>
</tr>
<tr>
<td>vbTextCompare</td>
<td>1</td>
<td>Perform a textual comparison.</td>
</tr>
</tbody>
</table>

2.11.3 Date and Time Constants

Since these constants are built into VBScript, you don't have to define them before using them. Use them anywhere in your code to represent the values shown for each.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbSunday</td>
<td>1</td>
<td>Sunday</td>
</tr>
<tr>
<td>vbMonday</td>
<td>2</td>
<td>Monday</td>
</tr>
<tr>
<td>vbTuesday</td>
<td>3</td>
<td>Tuesday</td>
</tr>
<tr>
<td>vbWednesday</td>
<td>4</td>
<td>Wednesday</td>
</tr>
<tr>
<td>vbThursday</td>
<td>5</td>
<td>Thursday</td>
</tr>
<tr>
<td>vbFriday</td>
<td>6</td>
<td>Friday</td>
</tr>
<tr>
<td>vbSaturday</td>
<td>7</td>
<td>Saturday</td>
</tr>
<tr>
<td>vbUseSystem</td>
<td>0</td>
<td>Use the date format contained in the regional settings for your computer.</td>
</tr>
<tr>
<td>vbUseSystemDayOfWeek</td>
<td>0</td>
<td>Use the day of the week specified in your system settings for the first day of the week.</td>
</tr>
</tbody>
</table>
vbFirstJan1 | 1 | Use the week in which January 1 occurs (default).

vbFirstFourDays | 2 | Use the first week that has at least four days in the new year.

vbFirstFullWeek | 3 | Use the first full week of the year.

2.11.4 Date Format Constants

Since these constants are built into VBScript, you don’t have to define them before using them. Use them anywhere in your code to represent the values shown for each.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbGeneralDate</td>
<td>0</td>
<td>Display a date and/or time. For real numbers, display a date and time. If there is no fractional part, display only a date. If there is no integer part, display time only. Date and time display is determined by your system settings.</td>
</tr>
<tr>
<td>vbLongDate</td>
<td>1</td>
<td>Display a date using the long date format specified in your computer's regional settings.</td>
</tr>
<tr>
<td>vbShortDate</td>
<td>2</td>
<td>Display a date using the short date format specified in your computer's regional settings.</td>
</tr>
<tr>
<td>vbLongTime</td>
<td>3</td>
<td>Display a time using the long time format specified in your computer's regional settings.</td>
</tr>
<tr>
<td>vbShortTime</td>
<td>4</td>
<td>Display a time using the short time format specified in your computer's regional settings.</td>
</tr>
</tbody>
</table>
2.11.5 Miscellaneous Constants

Since this constant is built into VBScript, you don't have to define it before using it. Use it anywhere in your code to represent the values shown.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbObjectError</td>
<td>-2147221504</td>
<td>User-defined error numbers should be greater than this value.</td>
</tr>
</tbody>
</table>

Example

```
Err.Raise Number = vbObjectError + 1000
```

2.11.6 MsgBox Constants

The following constants are used with the MsgBox function to identify what buttons and icons appear on a message box and which button is the default. In addition, the modality of the MsgBox can be specified. Since these constants are built into VBScript, you don't have to define them before using them. Use them anywhere in your code to represent the values shown for each.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbOKOnly</td>
<td>0</td>
<td>Display OK button only.</td>
</tr>
<tr>
<td>vbOKCancel</td>
<td>1</td>
<td>Display OK and Cancel buttons.</td>
</tr>
<tr>
<td>vbAbortRetryIgnore</td>
<td>2</td>
<td>Display Abort, Retry, and Ignore buttons.</td>
</tr>
<tr>
<td>vbYesNoCancel</td>
<td>3</td>
<td>Display Yes, No, and Cancel buttons.</td>
</tr>
<tr>
<td>vbYesNo</td>
<td>4</td>
<td>Display Yes and No buttons.</td>
</tr>
</tbody>
</table>
The following constants are used with the `MsgBox` function to identify which button a user has selected. These constants are only available when your project has an explicit reference to the appropriate type library containing these constant definitions. For VBScript, you must explicitly declare these constants in your code.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbOK</td>
<td>1</td>
<td>OK button was clicked.</td>
</tr>
<tr>
<td>vbCancel</td>
<td>2</td>
<td>Cancel button was clicked.</td>
</tr>
<tr>
<td>vbAbort</td>
<td>3</td>
<td>Abort button was clicked.</td>
</tr>
<tr>
<td>vbRetry</td>
<td>4</td>
<td>Retry button was clicked.</td>
</tr>
<tr>
<td>vbApplicationModal</td>
<td>0</td>
<td>Application modal; the user must respond to the message box before continuing work in the current application.</td>
</tr>
<tr>
<td>vbSystemModal</td>
<td>4096</td>
<td>System modal; all applications are suspended until the user responds to the message box.</td>
</tr>
</tbody>
</table>
vbIgnore  5  Ignore button was clicked.
vbYes  6  Yes button was clicked.
vbNo  7  No button was clicked.

---

### 2.11.7 String Constants

Since these constants are built into VBScript, you don't have to define them before using them. Use them anywhere in your code to represent the values shown for each.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbCr</td>
<td>Chr(13)</td>
<td>Carriage return.</td>
</tr>
<tr>
<td>vbCrLf</td>
<td>Chr(13) &amp; Chr(10)</td>
<td>Carriage return–linefeed combination.</td>
</tr>
<tr>
<td>vbFormFeed</td>
<td>Chr(12)</td>
<td>Form feed; not useful in Microsoft Windows.</td>
</tr>
<tr>
<td>vbLf</td>
<td>Chr(10)</td>
<td>Line feed.</td>
</tr>
<tr>
<td>vbNewLine</td>
<td>Chr(13) &amp; Chr(10) or Chr(10)</td>
<td>Platform-specific newline character; whatever is appropriate for the platform.</td>
</tr>
<tr>
<td>vbNullChar</td>
<td>Chr(0)</td>
<td>Character having the value 0.</td>
</tr>
<tr>
<td>vbNullString</td>
<td>String having value 0</td>
<td>Not the same as a zero-length string (&quot;&quot;'); used for calling external procedures.</td>
</tr>
<tr>
<td>vbTab</td>
<td>Chr(9)</td>
<td>Horizontal tab.</td>
</tr>
<tr>
<td>vbVerticalTab</td>
<td>Chr(11)</td>
<td>Vertical tab; not useful in Microsoft Windows.</td>
</tr>
</tbody>
</table>
2.11.8 Tristate Constants

Since these constants are built into VBScript, you don't have to define them before using them. Use them anywhere in your code to represent the values shown for each.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbUseDefault</td>
<td>-2</td>
<td>Use default from computer's regional settings.</td>
</tr>
<tr>
<td>vbTrue</td>
<td>-1</td>
<td>True</td>
</tr>
<tr>
<td>vbFalse</td>
<td>0</td>
<td>False</td>
</tr>
</tbody>
</table>

2.11.9 VarType Constants

These constants are only available when your project has an explicit reference to the appropriate type library containing these constant definitions. For VBScript, you must explicitly declare these constants in your code.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbEmpty</td>
<td>0</td>
<td>Uninitialized (default).</td>
</tr>
<tr>
<td>vbNull</td>
<td>1</td>
<td>Contains no valid data.</td>
</tr>
<tr>
<td>vbInteger</td>
<td>2</td>
<td>Integer subtype.</td>
</tr>
<tr>
<td>vbLong</td>
<td>3</td>
<td>Long subtype.</td>
</tr>
<tr>
<td>vbSingle</td>
<td>4</td>
<td>Single subtype.</td>
</tr>
<tr>
<td>vbSingle</td>
<td>5</td>
<td>Double subtype.</td>
</tr>
<tr>
<td>vbCurrency</td>
<td>6</td>
<td>Currency subtype.</td>
</tr>
<tr>
<td>vbDate</td>
<td>7</td>
<td>Date subtype.</td>
</tr>
<tr>
<td>vbString</td>
<td>8</td>
<td>String subtype.</td>
</tr>
<tr>
<td>vbObject</td>
<td>9</td>
<td>Object.</td>
</tr>
<tr>
<td>Constant</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>vbError</td>
<td>10</td>
<td>Error subtype.</td>
</tr>
<tr>
<td>vbBoolean</td>
<td>11</td>
<td>Boolean subtype.</td>
</tr>
<tr>
<td>vbVariant</td>
<td>12</td>
<td>Variant (used only for arrays of variants).</td>
</tr>
<tr>
<td>vbDataObject</td>
<td>13</td>
<td>Data access object.</td>
</tr>
<tr>
<td>vbDecimal</td>
<td>14</td>
<td>Decimal subtype.</td>
</tr>
<tr>
<td>vbByte</td>
<td>17</td>
<td>Byte subtype.</td>
</tr>
<tr>
<td>vbArray</td>
<td>8192</td>
<td>Array.</td>
</tr>
</tbody>
</table>

**VBScript Constants**
2.12 VBScript Operators

- **Addition** Operator (+)
- **And** Operator
- **Assignment** Operator (=)
- **Concatenation** Operator (&)
- **Division** Operator (/)
- **Eqv** Operator
- **Exponentiation** Operator (^)
- **Imp** Operator
- **Integer Division** Operator (\)
- **Is** Operator
- **Mod** Operator
- **Multiplication** Operator (*)
- **Not** Operator
- **Or** Operator
- **Subtraction** Operator (-)
- **Xor** Operator

2.12.1 Addition Operator (+)

Sums two numbers.

**Syntax**

\[ \text{Result} = \text{Expression1} + \text{Expression2} \]

The + operator syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result</td>
<td>Any numeric variable.</td>
</tr>
<tr>
<td>Expression1</td>
<td>Any expression.</td>
</tr>
</tbody>
</table>
Expression2  Any expression.

**Remarks**

Although you can also use the `+` operator to concatenate two character strings, you should use the `&` operator for concatenation to eliminate ambiguity and provide self-documenting code.

When you use the `+` operator, you may not be able to determine whether addition or string concatenation will occur.

The underlying subtype of the expressions determines the behavior of the `+` operator in the following way:

<table>
<thead>
<tr>
<th>If</th>
<th>Then</th>
</tr>
</thead>
<tbody>
<tr>
<td>Both expressions are numeric</td>
<td>Add.</td>
</tr>
<tr>
<td>Both expressions are strings</td>
<td>Concatenate.</td>
</tr>
<tr>
<td>One expression is numeric and the other is a string</td>
<td>Add.</td>
</tr>
</tbody>
</table>

If one or both expressions are **Null** expressions, Result is **Null**. If both expressions are **Empty**, Result is an **Integer** subtype. However, if only one expression is **Empty**, the other expression is returned unchanged as Result.

**2.12.2  And Operator**

Performs a logical conjunction on two expressions.

**Syntax**

\[ Result = Expression1 \textbf{And} \ Expression2 \]

The **And** operator syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result</td>
<td>Any numeric variable.</td>
</tr>
</tbody>
</table>
Expression1 | Any expression.
---|---
Expression2 | Any expression.

Remarks

If, and only if, both expressions evaluate to True, Result is True. If either expression evaluates to False, Result is False. The following table illustrates how Result is determined:

<table>
<thead>
<tr>
<th>If Expression1 is</th>
<th>And Expression2 is</th>
<th>The Result is</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>Null</td>
<td>Null</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>Null</td>
<td>False</td>
</tr>
<tr>
<td>Null</td>
<td>True</td>
<td>Null</td>
</tr>
<tr>
<td>Null</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>Null</td>
<td>Null</td>
<td>Null</td>
</tr>
</tbody>
</table>

The And operator also performs a bitwise comparison of identically positioned bits in two numeric expressions and sets the corresponding bit in Result according to the following table:

<table>
<thead>
<tr>
<th>If bit in Expression1 is</th>
<th>And bit in Expression2 is</th>
<th>The Result is</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
2.12.3 Assignment Operator

Assigns a value to a variable or property.

Syntax

\[ \text{Variable} = \text{Value} \]

The = operator syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
<td>Any variable or any writable property.</td>
</tr>
<tr>
<td>Value</td>
<td>Any numeric or string literal, constant, or expression.</td>
</tr>
</tbody>
</table>

Remarks

The name on the left side of the equal sign can be a simple scalar variable or an element of an array. Properties on the left side of the equal sign can only be those properties that are writable at run time.

2.12.4 Concatenation Operator (&)

Forces string concatenation of two expressions.

Syntax

\[ \text{Result} = \text{Expression1} & \text{Expression2} \]

The & operator syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result</td>
<td>Any variable.</td>
</tr>
<tr>
<td>Expression1</td>
<td>Any expression.</td>
</tr>
</tbody>
</table>
Remarks

Whenever an Expression is not a string, it is converted to a String subtype. If both expressions are Null, Result is also Null. However, if only one Expression is Null, that expression is treated as a zero-length string ("") when concatenated with the other expression. Any expression that is Empty is also treated as a zero-length string.

2.12.5 Division Operator (/)

Divides two numbers and returns a floating-point result.

Syntax

\[ \text{Result} = \text{Number1} / \text{Number2} \]

The / operator syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result</td>
<td>Any numeric variable.</td>
</tr>
<tr>
<td>Number1</td>
<td>Any numeric expression.</td>
</tr>
<tr>
<td>Number2</td>
<td>Any numeric expression.</td>
</tr>
</tbody>
</table>

Remarks

If one or both expressions are Null expressions, Result is Null. Any expression that is Empty is treated as 0.
Syntax

\[ \text{Result} = \text{Expression1} \text{ Eqv } \text{Expression2} \]

The `Eqv` operator syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result</td>
<td>Any numeric variable.</td>
</tr>
<tr>
<td>Expression1</td>
<td>Any expression.</td>
</tr>
<tr>
<td>Expression2</td>
<td>Any expression.</td>
</tr>
</tbody>
</table>

Remarks

If either expression is **null**, `Result` is also **null**. When neither expression is **null**, `Result` is determined according to the following table:

<table>
<thead>
<tr>
<th>If <code>Expression1</code> is</th>
<th>And <code>Expression2</code> is</th>
<th>The <code>Result</code> is</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>

The `Eqv` operator performs a bitwise comparison of identically positioned bits in two numeric expressions and sets the corresponding bit in `Result` according to the following table:

<table>
<thead>
<tr>
<th>If bit in <code>Expression1</code> is</th>
<th>And bit in <code>Expression2</code> is</th>
<th>The <code>Result</code> is</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
2.12.7 Exponentiation Operator (^)

Raises a number to the power of an exponent.

Syntax

\[ \text{Result} = \text{Number} ^ \text{Exponent} \]

The ^ operator syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result</td>
<td>Any numeric variable.</td>
</tr>
<tr>
<td>Number</td>
<td>Any numeric expression.</td>
</tr>
<tr>
<td>Exponent</td>
<td>Any numeric expression.</td>
</tr>
</tbody>
</table>

Remarks

Number can be negative only if Exponent is an integer value. When more than one exponentiation is performed in a single expression, the ^ operator is evaluated as it is encountered from left to right.

If either Number or Exponent is a Null expression, Result is also Null.

2.12.8 Imp Operator

Performs a logical implication on two expressions.

Syntax

\[ \text{Result} = \text{Expression1 Imp Expression2} \]

The Imp operator syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result</td>
<td>Any numeric variable.</td>
</tr>
<tr>
<td>Expression1</td>
<td>Any expression.</td>
</tr>
</tbody>
</table>
### Remarks

The following table illustrates how Result is determined:

<table>
<thead>
<tr>
<th>If Expression1 is</th>
<th>And Expression2 is</th>
<th>The Result is</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>Null</td>
<td>Null</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>Null</td>
<td>True</td>
</tr>
<tr>
<td>Null</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>Null</td>
<td>False</td>
<td>Null</td>
</tr>
<tr>
<td>Null</td>
<td>Null</td>
<td>Null</td>
</tr>
</tbody>
</table>

The **Imp** operator performs a bitwise comparison of identically positioned bits in two numeric expressions and sets the corresponding bit in Result according to the following table:

<table>
<thead>
<tr>
<th>If bit in Expression1 is</th>
<th>And bit in Expression2 is</th>
<th>The Result is</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
2.12.9 Integer Division Operator (\)

Divides two numbers and returns an integer result.

Syntax

\[ \text{Result} = \text{Number1} \ \backslash \ \text{Number2} \]

The \ operator syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result</td>
<td>Any numeric variable.</td>
</tr>
<tr>
<td>Number1</td>
<td>Any numeric expression.</td>
</tr>
<tr>
<td>Number2</td>
<td>Any numeric expression.</td>
</tr>
</tbody>
</table>

Remarks

Before division is performed, numeric expressions are rounded to Byte, Integer, or Long subtype expressions.

If any expression is Null, Result is also Null. Any expression that is Empty is treated as 0.

2.12.10 Is Operator

Compares two object reference variables.

Syntax

\[ \text{Result} = \text{Object1} \quad \text{Is} \quad \text{Object2} \]

The Is operator syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result</td>
<td>Any numeric variable.</td>
</tr>
<tr>
<td>Object1</td>
<td>Any object name.</td>
</tr>
<tr>
<td>Object2</td>
<td>Any object name.</td>
</tr>
</tbody>
</table>
Remarks

If Object1 and Object2 both refer to the same object, Result is True; if they do not, Result is False. Two variables can be made to refer to the same object in several ways.

In the following example, A has been set to refer to the same object as B.

```
Set A = B
```

The following example makes A and B refer to the same object as C.

```
Set A = C
Set B = C
```

2.12.11 Mod Operator

Divides two numbers and returns only the remainder.

Syntax

\[ Result = Number1 \mod Number2 \]

The Mod operator syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result</td>
<td>Any numeric variable.</td>
</tr>
<tr>
<td>Number1</td>
<td>Any numeric expression.</td>
</tr>
<tr>
<td>Number2</td>
<td>Any numeric expression.</td>
</tr>
</tbody>
</table>

Remarks

The modulus, or remainder, operator divides Number1 by Number2 (rounding floating-point numbers to integers) and returns only the remainder as Result. For example, in the following expression, A (which is Result) equals 5.

\[ A = 19 \mod 6.7 \]
If any expression is **Null**, Result is also **Null**. Any expression that is **Empty** is treated as 0.

### 2.12.12 Multiplication Operator (*)

Multiplies two numbers.

**Syntax**

\[ Result = Number1 \times Number2 \]

The * operator syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result</td>
<td>Any numeric variable.</td>
</tr>
<tr>
<td>Number1</td>
<td>Any numeric expression.</td>
</tr>
<tr>
<td>Number2</td>
<td>Any numeric expression.</td>
</tr>
</tbody>
</table>

**Remarks**

If one or both expressions are **Null** expressions, Result is **Null**. If an expression is **Empty**, it is treated as if it were 0.

### 2.12.13 Negation Operator (-)

Finds the difference between two numbers or indicates the negative value of a numeric expression.

**Syntax 1**

\[ Result = Number1 - Number2 \]

**Syntax 2**

\[-Number\]

The - operator syntax has these parts:
## Remarks

In Syntax 1, the - operator is the arithmetic subtraction operator used to find the difference between two numbers. In Syntax 2, the - operator is used as the unary negation operator to indicate the negative value of an expression.

If one or both expressions are **Null** expressions, *Result* is **Null**. If an expression is **Empty**, it is treated as if it were 0.

### 2.12.14 Not Operator

Performs logical negation on an expression.

#### Syntax

\[
Result = \textbf{Not} \ Expression
\]

The **Not** operator syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result</td>
<td>Any numeric variable.</td>
</tr>
<tr>
<td>Expression</td>
<td>Any expression.</td>
</tr>
</tbody>
</table>

#### Remarks

The following table illustrates how *Result* is determined:

| If *Expression1* is True | The *Result* is False |
In addition, the **Not** operator inverts the bit values of any variable and sets the corresponding bit in *Result* according to the following table:

<table>
<thead>
<tr>
<th>Bit in Expression</th>
<th>Bit in Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**2.12.15 Or Operator**

Performs a logical disjunction on two expressions.

**Syntax**

\[
\text{Result} = \text{Expression}_1 \text{ Or } \text{Expression}_2
\]

The **Or** operator syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result</td>
<td>Any numeric variable.</td>
</tr>
<tr>
<td>Expression1</td>
<td>Any expression.</td>
</tr>
<tr>
<td>Expression2</td>
<td>Any expression.</td>
</tr>
</tbody>
</table>

**Remarks**

If either or both expressions evaluate to **True**, *Result* is **True**. The following table illustrates how *Result* is determined:

<table>
<thead>
<tr>
<th>If Expression1 is</th>
<th>And Expression2 is</th>
<th>The Result is</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>
The **Or** operator also performs a bitwise comparison of identically positioned bits in two numeric expressions and sets the corresponding bit in Result according to the following table:

<table>
<thead>
<tr>
<th>If bit in Expression1 is</th>
<th>And bit in Expression2 is</th>
<th>The Result is</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**2.12.16 Subtraction Operator (-)**

Finds the difference between two numbers or indicates the negative value of a numeric expression.

**Syntax 1**

\[
\text{Result} = \text{Number1} - \text{Number2}
\]
Syntax 2

-Number

The - operator syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result</td>
<td>Any numeric variable.</td>
</tr>
<tr>
<td>Number</td>
<td>Any numeric expression.</td>
</tr>
<tr>
<td>Number1</td>
<td>Any numeric expression.</td>
</tr>
<tr>
<td>Number2</td>
<td>Any numeric expression.</td>
</tr>
</tbody>
</table>

Remarks

In Syntax 1, the - operator is the arithmetic subtraction operator used to find the difference between two numbers. In Syntax 2, the - operator is used as the unary negation operator to indicate the negative value of an expression.

If one or both expressions are Null expressions, Result is Null. If an expression is Empty, it is treated as if it were 0.

2.12.17 Xor Operator

Performs a logical exclusion on two expressions.

Syntax

Result = Expression1 Xor Expression2

The Xor operator syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result</td>
<td>Any numeric variable.</td>
</tr>
<tr>
<td>Expression1</td>
<td>Any expression.</td>
</tr>
<tr>
<td>Expression2</td>
<td>Any expression.</td>
</tr>
</tbody>
</table>
Remarks

If one, and only one, of the expressions evaluates to True, Result is True. However, if either expression is Null, Result is also Null. When neither expression is Null, Result is determined according to the following table:

<table>
<thead>
<tr>
<th>If Expression1 is</th>
<th>And Expression2 is</th>
<th>The Result is</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

The Xor operator also performs a bitwise comparison of identically positioned bits in two numeric expressions and sets the corresponding bit in Result according to the following table:

<table>
<thead>
<tr>
<th>If bit in Expression1 is</th>
<th>And bit in Expression2 is</th>
<th>The Result is</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
## 2.13 Keywords

### 2.13.1 Keywords

The following table explains how to use VBScript keywords.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Empty</strong></td>
<td>The <em>Empty</em> keyword is used to indicate an uninitialized variable value. This is not the same thing as <em>Null</em>. You can use the <em>IsEmpty</em> function to determine whether a variable is uninitialized.</td>
</tr>
<tr>
<td><strong>False</strong></td>
<td>The <em>False</em> keyword has a value equal to 0.</td>
</tr>
<tr>
<td><strong>Nothing</strong></td>
<td>The <em>Nothing</em> keyword in VBScript is used to disassociate an object variable from any actual object. Use the <em>Set</em> statement to assign <em>Nothing</em> to an object variable. Several object variables can refer to the same actual object. When <em>Nothing</em> is assigned to an object variable, that variable no longer refers to any actual object. When several object variables refer to the same object, memory and system resources associated with the object to which the variables refer are released only after all of them have been set to <em>Nothing</em>, either explicitly by using <em>Set</em>, or implicitly after the last object variable that is not set to <em>Nothing</em> goes out of scope. You can use the <em>Is</em> operator to determine whether an object variable is assigned <em>Nothing</em>.</td>
</tr>
<tr>
<td><strong>Null</strong></td>
<td>The <em>Null</em> keyword is used to indicate that a variable contains no valid data. This is not the same thing as <em>Empty</em>. You can use the <em>IsNull</em> function to determine whether a variable contains no valid data.</td>
</tr>
<tr>
<td><strong>True</strong></td>
<td>The <em>True</em> keyword has a value equal to -1.</td>
</tr>
</tbody>
</table>
2.14 Resources

2.14.1 Locale ID (LCID) Chart

A locale is a set of user preference information related to the user's language, country/region, and cultural conventions. The locale determines such things as keyboard layout, alphabetic sort order, as well as date, time, number, and currency formats. Locale ID (LCID) Chart is available in the original help.

2.14.2 Errors

VBScript run-time errors are errors that result when your VBScript script attempts to perform an action that the system cannot execute. VBScript run-time errors occur while your script is being executed; when variable expressions are being evaluated, and memory is being dynamic allocated. List of VBScript run-time errors is available in the original help.

VBScript syntax errors are errors that result when the structure of one of your VBScript statements violates one or more of the grammatical rules of the VBScript scripting language. VBScript syntax errors occur during the program compilation stage, before the program has begun to be executed. List of VBScript syntax errors is available in the original help.
3 Reliance-defined Objects

3.1 Reliance-defined Objects

Reliance expands VBScript by several objects designed for accessing the runtime environment.

Each object implements methods and/or properties intended for a specific set of operations:

- **RALm** Object – for operations on alarms defined in a visualization project
- **RDb** Object – for operations on databases defined in a visualization project
- **RDev** Object – for operations on devices defined in a visualization project
- **RError** Object – for accessing information on the result of any of the above operations
- **Rinet** Object – for E-mail operations
- **RModem** Object – for GSM modem operations
- **RScr** Object – for operations on scripts defined in a visualization project
- **RSys** Object – for miscellaneous operations
- **TTable**-type Objects – for operations on database tables, either current or archive, belonging to databases defined in a visualization project
- **RTag** Object – for operations on tags defined in a visualization project
- **RUser** Object – for operations on users defined in a visualization project
- **RWS** Object – for accessing the Web service of Reliance data servers

These objects, with the exception of **TTable**-type objects, are created and initialized automatically during starting a visualization project. When using these objects, follow the rules for working with properties and methods of objects.
3.2 Execution of Scripts in the Runtime Environment

Scripts to be executed are placed in a script queue to a position that depends on their priority. The scripts queued for execution are sequentially processed in the background without affecting the user interface of the runtime environment. If a syntax error occurs in a script, execution of the script is terminated. If an error other than syntax occurs in a script, execution of the script is terminated based on the Terminate script on error option (Reliance Design > Project > Options > Scripts > Other). Information on any error that may occur when accessing properties and calling methods of Reliance-defined objects is stored in the RError object and can be accessed through its properties after every such property access or method call.
3.3 Processing of Data Passed to Scripts from the Runtime Environment

A parameter can be passed to a script, for example, when running the script by clicking a visual component. Using the value of the parameter in the script's code it is possible to determine which event (e.g., which component was clicked) triggered the script. A script can retrieve the values of parameters by reading properties of a special data object returned by a call to the `RScr.GetCurrentScriptData` or `RScr.GetCurrentScriptDataEx` in the script's code.
3.4 Working with Global Constants, Variables, Procedures and Functions

A script containing the declaration of a global constant, variable, procedure or function must be executed before any other script that references the constant or variable, or calls the procedure or function.

**TIP**

It is advisable to place the declarations of global constants, variables, procedures or functions to a single script and activate its *Run on thread initialization* property (Reliance Design > Managers > Script Manager > script properties > the Advanced page). This way, it is guaranteed that the script will execute before other scripts.
3.5 Tips for Writing Scripts

To prevent errors due to typing mistakes, it is recommended to use the `Option Explicit` statement in all scripts. Thus, if a script references an undeclared variable (e.g., due to a typing mistake), a syntax error occurs.

The `MsgBox` and `InputBox` functions should only be used for debugging purposes. Each of these functions displays a dialog box and stops execution of the script from which it has been called. The script does not continue execution until the dialog box is closed by the user. No other scripts in the thread can be processed either since only one script can execute per thread. Another reason for avoiding using these functions is that the dialog box may appear in the background of the program where it cannot be seen by the user, thus stopping processing of scripts. When using the `MsgBox` function, this behavior can be prevented by passing the `vbSystemModal` constant as the second parameter to the function. For example, `MsgBox "Message text", vbSystemModal`. As a result, the message box will stay on top of other windows.
3.6 RAIm Object

The **RAIm** object implements methods for operations on alarms.

Methods:

- **RAIm.AckAlarm**: Procedure
- **RAIm.AckAllAlarms**: Procedure
- **RAIm.CreateAlarm**: Procedure
- **RAIm.CurrentAlarms**: Procedure
- **RAIm.CurrentAlarmsByDevice**: Procedure
- **RAIm.CurrentAlarmsEx**: Procedure
- **RAIm.DbAlarms**: Procedure
- **RAIm.DbAlarmsByDevice**: Procedure
- **RAIm.DbAlarmsByFilter**: Procedure
- **RAIm.DbAlarmsEx**: Procedure
- **RAIm.DisableDeviceAlarms**: Procedure
- **RAIm.EnableDeviceAlarms**: Procedure

Other:

- **Alarm Triggering Condition**: Constants

### 3.6.1 RAIm.AckAlarm Procedure

**RAIm.AckAlarm** acknowledges an alarm.

**Syntax**

```
RAIm.AckAlarm DevName, AlmName: String
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DevName</td>
<td>The name of the device that the alarm belongs to.</td>
</tr>
</tbody>
</table>
### RAlm Object

#### Example

' Acknowledge the alarm PumpFailure from the device PLC1.
RAlmAckAlarm "PLC1", "PumpFailure"

#### 3.6.2 RAlm.AckAllAlarms Procedure

RAlm.AckAllAlarms acknowledges all unacknowledged alarms.

**Syntax**

```plaintext
RAlm.AckAllAlarms
```

#### Example

```plaintext
If RTag.GetTagValue("System", "AckAlarms") Then
  ' Acknowledge all alarms depending on the value
  ' of the tag AckAlarms from the device System.
  RAlm.AckAllAlarms
End If
```

#### 3.6.3 RAlm.CreateAlarm Procedure

RAlm.CreateAlarm generates an alarm using specified parameters.

**Syntax**

```plaintext
RAlm.CreateAlarm AlmType: Variant; Text, Comment: String; StartTime: DateTime; Save, Show, Print, Ack, ActivateWnd: Bool; AckRights: Integer; StartSound: String; StartScript, AckScript: Variant; RelatedWnd: Variant
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlmType</td>
<td>The name or ID of the alarm type.</td>
</tr>
<tr>
<td><strong>Text</strong></td>
<td>The text of the alarm.</td>
</tr>
<tr>
<td><strong>Comment</strong></td>
<td>The comment for the alarm.</td>
</tr>
<tr>
<td><strong>StartTime</strong></td>
<td>Date and time when the alarm was generated.</td>
</tr>
<tr>
<td><strong>Save</strong></td>
<td>Determines whether to log the alarm to the alarm database.</td>
</tr>
<tr>
<td><strong>Show</strong></td>
<td>Determines whether to display the alarm in the list of current alarms.</td>
</tr>
<tr>
<td><strong>Print</strong></td>
<td>Determines whether to print the alarm online.</td>
</tr>
<tr>
<td><strong>Ack</strong></td>
<td>Determines whether it is required that the alarm be acknowledged by the user (operator).</td>
</tr>
<tr>
<td><strong>ActivateWnd</strong></td>
<td>Determines whether to activate the list of current alarms (makes sense only if <strong>Show</strong> = <strong>True</strong>).</td>
</tr>
<tr>
<td><strong>AckRights</strong></td>
<td>The access rights required for acknowledging the alarm (makes sense only if <strong>Ack</strong> = <strong>True</strong>). A comma-delimited list of the names of access rights. It can be &quot;&quot; or <strong>Empty</strong> if the argument is not to be used.</td>
</tr>
<tr>
<td><strong>StartSound</strong></td>
<td>The file name of a sound to be played after the alarm is generated.</td>
</tr>
<tr>
<td><strong>StartScript</strong></td>
<td>The name or ID of a script to be executed after the alarm is generated.</td>
</tr>
<tr>
<td><strong>AckScript</strong></td>
<td>The name or ID of a script to be executed after the alarm is acknowledged (makes sense only if <strong>Ack</strong> = <strong>True</strong>).</td>
</tr>
<tr>
<td><strong>RelatedWnd</strong></td>
<td>The name or ID of the window related to the alarm.</td>
</tr>
</tbody>
</table>

**Remarks**

An alarm generated by calling the method is processed in the same way as internal messages (i.e., messages defined not in a visualization project via the Device Manager but in the program code of the runtime software). It is not associated with a tag and cannot be transferred to another computer through a network connection.
Example

Dim fso, drv, freesp
' Create the Scripting.FileSystemObject object and assign it to the variable fso.
Set fso = CreateObject("Scripting.FileSystemObject")
' Create an object representing the C: drive and assign it to the variable drv.
Set drv = fso.GetDrive(fso.GetDriveName("C:\"))
freesp = drv.FreeSpace / 1024 ' Get free space in kB.
' If the C: drive has less than 50000 kB of free space, create an alarm.
If (freesp < 50000) Then
    RAlm.CreateAlarm 0, "Insufficient free space on the C: drive.", ", ", Now, True, True, False, True, True, 0, ", ", ", ",
End If
Set fso = Nothing ' Free the object referenced by fso.
Set drv = Nothing ' Free the object referenced by drv.

3.6.4 RAlm.CurrentAlarms Procedure

RAlm.CurrentAlarms displays a window containing a list of current (i.e., active and/or unacknowledged) alarms.

Syntax

    RAlm.CurrentAlarms

Example

If RTag.GetTagValue("System", "ShowAlarms") Then
    ' Display the list of current alarms depending on the value
    ' of the tag ShowAlarms from the device System.
    RAlm.CurrentAlarms
End If
3.6.5 RAIm.CurrentAlarmsByDevice Procedure

RAIm.CurrentAlarmsByDevice displays a window containing a list of current (i.e., active and/or unacknowledged) alarms belonging to a specified device.

Syntax

```
RAIm.CurrentAlarmsByDevice Device: Variant
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device</td>
<td>The name or ID of the device.</td>
</tr>
</tbody>
</table>

Example

```
' Display the list of current alarms belonging to the device PLC1.
RAIm.CurrentAlarmsByDevice "PLC1"
```
### 3.6.6 RAlm.CurrentAlarmsEx Procedure

`RAlm.CurrentAlarmsEx` displays a window containing a list of current alarms/events that meet the specified parameters.

**Syntax**

```plaintext
RAlm.CurrentAlarmsEx Device; Variant; AlmTypes, AlmGroups, Reserved: String;
From, Till: DateTime
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Device</code></td>
<td>The name or ID of the device. It can be <code>Empty</code> if the argument is not to be used.</td>
</tr>
<tr>
<td><code>AlmTypes</code></td>
<td>A comma-delimited list of the names of alarm/event types. It can be <code>&quot;&quot;</code> or <code>Empty</code> if the argument is not to be used.</td>
</tr>
<tr>
<td><code>AlmGroups</code></td>
<td>A comma-delimited list of the names of alarm/event groups. It can be <code>&quot;&quot;</code> or <code>Empty</code> if the argument is not to be used.</td>
</tr>
<tr>
<td><code>Reserved</code></td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td><code>From</code></td>
<td>The beginning of the time interval (in UTC) that specifies the start time of alarms/events to be displayed. It can be <code>Empty</code> if the argument is not to be used.</td>
</tr>
<tr>
<td><code>Till</code></td>
<td>The end of the time interval (in UTC) that specifies the start time of alarms/events to be displayed. It can be <code>Empty</code> if the argument is not to be used.</td>
</tr>
</tbody>
</table>

**Example**

```plaintext
Dim UserName, AlarmGroups
' A user is logged on.
```
If RUser.GetLoggedOnUserName(UserName) Then
' Retrieve the list of alarm/event groups for the currently logged on user.
AlarmGroups = RUser.GetUserAlarmGroups(UserName)
' Display the list of current alarms/events depending on the value of AlarmGroups.
RAIm.CurrentAlarmsEx Empty, "", AlarmGroups, "", Empty, Empty
End If

3.6.7 RAIm.DbAlarms Procedure

RAIm.DbAlarms displays a window containing a list of alarms stored in the alarm database (historical alarms).

Syntax

RAIm.DbAlarms Unused: Integer

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unused</td>
<td>The value is no longer used.</td>
</tr>
</tbody>
</table>

Remarks

If the window contained a list of alarms restricted by a filter before calling the method, the filter is canceled.

Example

' Display the list of historical alarms.
RAIm.DbAlarms 0
3.6.8 RAIm.DbAlarmsByFilter Procedure

RAIm.DbAlarmsByFilter displays a window containing a list of alarms stored in the alarm database (historical alarms), restricted by a specified filter.

Syntax

RAIm.DbAlarmsByFilter Unused: Integer; FilterName: String

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unused</td>
<td>The value is no longer used.</td>
</tr>
<tr>
<td>FilterName</td>
<td>The name of the filter.</td>
</tr>
</tbody>
</table>

Remarks

If the window contained a list of alarms restricted by a filter before calling the method, the filter is canceled and the specified filter is applied. To define filters for the alarm database, use the Filter Editor in the runtime software. The filters get stored to the profile (user profile in a visualization project, not in the operating system) of the user currently logged on to the runtime software. If the filter specified by FilterName is not found in the profile of the currently logged on user, the search is performed in the Default profile. If the filter is not found, the window will not be displayed.

Example

'Display the list of historical alarms restricted by the filter PLC1_or_PLC2
'(displays only alarms belonging to the device PLC1 or PLC2).
RAIm.DbAlarmsByFilter 0, "PLC1_or_PLC2"
3.6.9 RAIm.DbAlarmsByDevice Procedure

RAIm.DbAlarmsByDevice displays a window containing a list of alarms stored in the alarm database (historical alarms), restricted to alarms belonging to a specified device.

Syntax

```
RAIm.DbAlarmsByDevice Unused: Integer; Device: Variant
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unused</td>
<td>The value is no longer used.</td>
</tr>
<tr>
<td>Device</td>
<td>The name or ID of the device.</td>
</tr>
</tbody>
</table>

Remarks

If the window contained a list of alarms restricted by a filter before calling the method, the filter is canceled.

Example

```
' Display the list of historical alarms belonging to the device PLC1.
RAIm.DbAlarmsByDevice 0, "PLC1"
```
3.6.10 RAlm.DbAlarmsEx Procedure

RAlm.DbAlarmsEx displays a window containing a list of alarms/events stored in the alarm database. Only the alarms/events that meet the specified parameters are displayed.

**Syntax**

RAlm.DbAlarmsEx Device; Variant; AlmTypes, Reserved, FilterName: String; From, Till : DateTime

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device</td>
<td>The name or ID of the device. It can be <strong>Empty</strong> if the argument is not to be used.</td>
</tr>
<tr>
<td>AlmTypes</td>
<td>A comma-delimited list of the names of alarm/event types. It can be &quot;&quot; or <strong>Empty</strong> if the argument is not to be used.</td>
</tr>
<tr>
<td>Reserved</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>FilterName</td>
<td>The name of the filter to be used for filtering. It can be &quot;&quot; or <strong>Empty</strong> if the argument is not to be used.</td>
</tr>
<tr>
<td>From</td>
<td>The beginning of the time interval (in UTC) that specifies the start time of alarms/events to be displayed. It can be <strong>Empty</strong> if the argument is not to be used.</td>
</tr>
<tr>
<td>Till</td>
<td>The end of the time interval (in UTC) that specifies the start time of alarms/events to be displayed. It can be <strong>Empty</strong> if the argument is not to be used.</td>
</tr>
</tbody>
</table>
Example

' Display the list of historical alarms/events of type Limit_Hi and Limit_HiHi.
RAlm DbAlarmsEx Empty, "Limit_Hi,Limit_HiHi", ",", ",", Empty, Empty

3.6.11 RAlm.DisableDeviceAlarms Procedure

RAlm.DisableDeviceAlarms disables all alarms, of a specified type, belonging to a specified device.

Syntax

RAlm.DisableDeviceAlarms AlmType: Variant; Device: Variant

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlmType</td>
<td>The name or ID of the alarm type. It can be Empty if the argument is not to be used.</td>
</tr>
<tr>
<td>Device</td>
<td>The name or ID of the device.</td>
</tr>
</tbody>
</table>

Remarks

Alarms, which have been disabled, can be enabled again by calling the RAlm.EnableDeviceAlarms method.

Example

' Disable all alarms belonging to the device PLC1.
RAlm DisableDeviceAlarms Empty, "PLC1"

3.6.12 RAlm.EnableDeviceAlarms Procedure

RAlm.EnableDeviceAlarms enables all alarms, of a specified type, belonging to a specified device.
Syntax

RAIm.EnableDeviceAlarms AlmType: Variant; Device: Variant

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlmType</td>
<td>The name or ID of the alarm type. It can be Empty if the argument is not to be used.</td>
</tr>
<tr>
<td>Device</td>
<td>The name or ID of the device.</td>
</tr>
</tbody>
</table>

Remarks

Alarms, which have been enabled, can be disabled again by calling the RAlm.DisableDeviceAlarms method.

Example

' Enable all alarms belonging to the device PLC1.
RAIm EnableDeviceAlarms Empty, "PLC1"

3.6.13 Alarm Triggering Condition Constants

Conditions that can trigger an alarm.
<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>A change in the value of a tag (any change).</td>
</tr>
<tr>
<td>11</td>
<td>A change in the value of a tag (increment).</td>
</tr>
<tr>
<td>12</td>
<td>A change in the value of a tag (decrement).</td>
</tr>
<tr>
<td>20</td>
<td>The leading edge of a digital-type tag.</td>
</tr>
<tr>
<td>21</td>
<td>The trailing edge of a digital-type tag.</td>
</tr>
<tr>
<td>30</td>
<td>Exceeding the upper critical limit.</td>
</tr>
<tr>
<td>31</td>
<td>Exceeding the upper warning limit.</td>
</tr>
<tr>
<td>32</td>
<td>Falling below the lower warning limit.</td>
</tr>
<tr>
<td>33</td>
<td>Falling below the lower critical limit.</td>
</tr>
<tr>
<td>40</td>
<td>The value of a tag within the range.</td>
</tr>
</tbody>
</table>
3.7 RConst Object

This object's properties are objects that contain a set of constants of a specific type.

Properties:

- **RConst.ProjObjType** Property

3.7.1 RConst.ProjObjType Property

RConst.ProjObjType is a property of type object. This object's properties correspond to constants of object types used in Reliance projects (e.g., device, tag, alarm/event). These properties return a value of type Variant(Integer) and are only intended for reading.

Properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>Action</td>
</tr>
<tr>
<td>ActionFolder</td>
<td>Action folder</td>
</tr>
<tr>
<td>Alarm</td>
<td>Alarm/event</td>
</tr>
<tr>
<td>AlarmFolder</td>
<td>Alarm/event folder</td>
</tr>
<tr>
<td>AlarmGroup</td>
<td>Alarm/event group</td>
</tr>
<tr>
<td>AlarmType</td>
<td>Alarm/event type</td>
</tr>
<tr>
<td>CommDriver</td>
<td>Communication driver</td>
</tr>
<tr>
<td>CommDriverFolder</td>
<td>Communication driver folder</td>
</tr>
<tr>
<td>CommChannel</td>
<td>Communication channel</td>
</tr>
<tr>
<td>CommZone</td>
<td>Communication zone</td>
</tr>
<tr>
<td>CommZoneFolder</td>
<td>Communication zone folder</td>
</tr>
<tr>
<td>Computer</td>
<td>Computer</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>ComputerFolder</td>
<td>Computer folder</td>
</tr>
<tr>
<td>ConnCommDriver</td>
<td>Connected communication driver</td>
</tr>
<tr>
<td>ConnCommDriverFolder</td>
<td>Connected communication driver folder</td>
</tr>
<tr>
<td>ConnCustomReport</td>
<td>Connected custom report</td>
</tr>
<tr>
<td>ConnCustomReportFolder</td>
<td>Connected custom report folder</td>
</tr>
<tr>
<td>ConnDataTable</td>
<td>Connected data table</td>
</tr>
<tr>
<td>ConnDataTableFolder</td>
<td>Connected data table folder</td>
</tr>
<tr>
<td>ConnDbReport</td>
<td>Connected report</td>
</tr>
<tr>
<td>ConnDbReportFolder</td>
<td>Connected report folder</td>
</tr>
<tr>
<td>ConnDbTrend</td>
<td>Connected trend</td>
</tr>
<tr>
<td>ConnDbTrendFolder</td>
<td>Connected trend folder</td>
</tr>
<tr>
<td>ConnDevice</td>
<td>Connected device</td>
</tr>
<tr>
<td>ConnDeviceFolder</td>
<td>Connected device folder</td>
</tr>
<tr>
<td>ConnRecipeType</td>
<td>Connected recipe type</td>
</tr>
<tr>
<td>ConnRecipeTypeFolder</td>
<td>Connected recipe type folder</td>
</tr>
<tr>
<td>ConnScript</td>
<td>Connected script</td>
</tr>
<tr>
<td>ConnScriptFolder</td>
<td>Connected script folder</td>
</tr>
<tr>
<td>ConnUser</td>
<td>Connected user</td>
</tr>
<tr>
<td>ConnUserFolder</td>
<td>Connected user folder</td>
</tr>
<tr>
<td>ControlRoom</td>
<td>Control room</td>
</tr>
<tr>
<td>CustomReport</td>
<td>Custom report</td>
</tr>
<tr>
<td>CustomReportFolder</td>
<td>Custom report folder</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>CustomReportItem</td>
<td>Custom report item</td>
</tr>
<tr>
<td>CustomReportItemFolder</td>
<td>Custom report item folder</td>
</tr>
<tr>
<td>DataStructure</td>
<td>Data structure</td>
</tr>
<tr>
<td>DataStructureField</td>
<td>Data structure field</td>
</tr>
<tr>
<td>DataStructureFolder</td>
<td>Data structure folder</td>
</tr>
<tr>
<td>DataTable</td>
<td>Data table</td>
</tr>
<tr>
<td>DataTableField</td>
<td>Data table field</td>
</tr>
<tr>
<td>DataTableFieldFolder</td>
<td>Data table field folder</td>
</tr>
<tr>
<td>DataTableFolder</td>
<td>Data table folder</td>
</tr>
<tr>
<td>DbReport</td>
<td>Report</td>
</tr>
<tr>
<td>DbReportFolder</td>
<td>Report folder</td>
</tr>
<tr>
<td>DbReportItem</td>
<td>Report item</td>
</tr>
<tr>
<td>DbReportItemFolder</td>
<td>Report item folder</td>
</tr>
<tr>
<td>DbTrend</td>
<td>Trend</td>
</tr>
<tr>
<td>DbTrendFolder</td>
<td>Trend folder</td>
</tr>
<tr>
<td>DbTrendSeries</td>
<td>Trend series</td>
</tr>
<tr>
<td>DbTrendSeriesFolder</td>
<td>Trend series folder</td>
</tr>
<tr>
<td>Device</td>
<td>Device</td>
</tr>
<tr>
<td>DeviceFolder</td>
<td>Device folder</td>
</tr>
<tr>
<td>Language</td>
<td>Project language</td>
</tr>
<tr>
<td>Modem</td>
<td>Modem</td>
</tr>
<tr>
<td>ModemFolder</td>
<td>Modem folder</td>
</tr>
<tr>
<td>Picture</td>
<td>Picture</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>PictureFolder</td>
<td>Picture folder</td>
</tr>
<tr>
<td>Printer</td>
<td>Printer</td>
</tr>
<tr>
<td>PrinterFolder</td>
<td>Printer folder</td>
</tr>
<tr>
<td>RecipeItem</td>
<td>Recipe item</td>
</tr>
<tr>
<td>RecipeItemFolder</td>
<td>Recipe item folder</td>
</tr>
<tr>
<td>RecipeType</td>
<td>Recipe type</td>
</tr>
<tr>
<td>RecipeTypeFolder</td>
<td>Recipe type folder</td>
</tr>
<tr>
<td>RTTrend</td>
<td>Real-time trend</td>
</tr>
<tr>
<td>RTTrendFolder</td>
<td>Real-time trend folder</td>
</tr>
<tr>
<td>RTTrendSeries</td>
<td>Real-time trend series</td>
</tr>
<tr>
<td>RTTrendSeriesFolder</td>
<td>Real-time trend series folder</td>
</tr>
<tr>
<td>Script</td>
<td>Script</td>
</tr>
<tr>
<td>ScriptFolder</td>
<td>Script folder</td>
</tr>
<tr>
<td>ServerConnection</td>
<td>Server connection</td>
</tr>
<tr>
<td>ServerConnectionGroup</td>
<td>Server connection group</td>
</tr>
<tr>
<td>SQLConnection</td>
<td>SQL connection</td>
</tr>
<tr>
<td>Tag</td>
<td>Tag</td>
</tr>
<tr>
<td>TagFolder</td>
<td>Tag folder</td>
</tr>
<tr>
<td>TextString</td>
<td>Text string</td>
</tr>
<tr>
<td>Timer</td>
<td>Timer</td>
</tr>
<tr>
<td>User</td>
<td>User</td>
</tr>
<tr>
<td>UserFolder</td>
<td>User folder</td>
</tr>
<tr>
<td>Window</td>
<td>Window</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------</td>
</tr>
<tr>
<td>WindowFolder</td>
<td>Window folder</td>
</tr>
</tbody>
</table>

**RConst Object**
3.8 RDb Object

The RDb object implements methods for operations on data tables defined in a visualization project. The methods enable you to access historical data stored in the data tables. You can also work with individual database tables through a TTable-type object returned by the RDb.CreateTableObject method.

Methods:

- RDb.AppendRecord Procedure
- RDb.CreateTableObject Function
- RDb.GetTagHistValue Function
- RDb.GetTagStatistics Procedure

3.8.1 RDb.AppendRecord Procedure

RDb.AppendRecord appends a new record to a specified data table.

Syntax

```
RDb.AppendRecord DataTable: Variant
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataTable</td>
<td>The name or ID of the data table.</td>
</tr>
</tbody>
</table>

Remarks

In order for the method to work, the data table's Sampling property must have a value of Script-controlled (by RDb.AppendRecord procedure).

Example

```vba
Const c_ArrayLen = 5
' Number of array-type tag elements
Dim ArrayIndex
```
' Index of array-type tag elements
For ArrayIndex = 0 to c_ArrayLen - 1
    RTag.SetTagValue "System", "TimeStamp", RTag.GetTagElementValue("PLC1", "TimeStamp_Arr", ArrayIndex)
    RTag.SetTagValue "System", "Pressure1", RTag.GetTagElementValue("PLC1", "Pressure1_Arr", ArrayIndex)
    RTag.SetTagValue "System", "Pressure2", RTag.GetTagElementValue("PLC1", "Pressure2_Arr", ArrayIndex)
    RDb.AppendRecord "Pressures"
Next

3.8.2  RDb.CreateTableObject Function

RDb.CreateTableObject creates a new TTable-type object, which can be used for working with databases tables, and returns a reference to the object.

Syntax

    RDb.CreateTableObject: TTable

Return values

The method returns a reference to the newly created TTable-type object.

Remarks

When using a TTable-type object, follow the rules for working with properties and methods of objects. When assigning the return value of RDb.CreateTableObject to a variable, the Set statement must be used. When you no longer need the TTable-type object, you should free it by assigning the constant Nothing to the variable using the Set statement.

Example

    Dim Table
    Set Table = RDb.CreateTableObject ' Create a TTable-type object.
    ' Name of the data table, as defined in a visualization project.
    Table.DatabaseName = "Water"
    If Table.OpenTable Then
        ' If the current table can be opened.
        Table.Append ' Append a new record.
        ' Save current system date to the new record.
Table. DateFieldValue = Date
' Save current system time to the new record.
Table. TimeFieldValue = Time
' Save the value of the tag WaterTemperature from the device PLC1 to the new record.
Table. SetFieldValue "PLC1", "WaterTemperature", RTag. GetTagValue("PLC1", "WaterTemperature")
Table. Post
' Write the new record to the table.
Table. CloseTable
' Close the table.
End If
Set Table = Nothing
' Free the TTable-type object.

3.8.3 RDb. GetTagHistValue Function

RDb. GetTagHistValue returns a tag's historical value with time stamp (date and time) nearest to a specified date and time.

Syntax

RDb. GetTagHistValue(DevName, TagName, DataTableName: String; ValTime, Tolerance: DateTime; ByRef RecTime: DateTime; ByRef Error: Variant): Variant

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DevName</td>
<td>The name of the device that the tag belongs to.</td>
</tr>
<tr>
<td>TagName</td>
<td>The name of the tag.</td>
</tr>
<tr>
<td>DataTableName</td>
<td>The name of the data table.</td>
</tr>
<tr>
<td>ValTime</td>
<td>Required time stamp of the value.</td>
</tr>
<tr>
<td>Tolerance</td>
<td>Time stamp tolerance.</td>
</tr>
<tr>
<td>RecTime</td>
<td>Time stamp of the database record found.</td>
</tr>
<tr>
<td>Error</td>
<td>Error code (The List of Error Codes Returned by Methods and Properties of Reliance-defined Objects).</td>
</tr>
</tbody>
</table>
Return values

If the call succeeds, the method returns the tag's historical value with time stamp nearest to the specified date and time.

If the call fails, the method returns Empty.

Remarks

The method searches both current and archive files (if working with data tables of type dBASE or Paradox). The method attempts to find a record with time stamp nearest to the specified date and time in the range from (ValTime - Tolerance) to (ValTime + Tolerance).

After calling the method, it is recommended to check the value of the Error argument to find out about the result of the operation.

Example

```vba
Dim Value, ValTime, Tolerance, RecTime, Error, Text
ValTime = Date + TimeSerial(6, 0, 0) ' Today, 6 a.m.
Tolerance = TimeSerial(0, 5, 0) ' Time stamp tolerance +/- 5 minutes.
' Get the historical value of the tag WaterTemperature from the device PLC1 stored in the data table Water.
Value = RDb.GetTagHistValue("PLC1", "WaterTemperature", "Water", ValTime, Tolerance, RecTime, Error)
If Error = 0 Then
    Text = "The value at: " + CStr(RecTime) + " was: " + CStr(Value) + "."
Else
    Text = "Error (code " + CStr(Error) + ")."
End If
' Store information on the result of the operation to the tag DisplayResult from the device System.
RTag.SetTagValue "System", "DisplayResult", Text
```
3.8.4 RDb.GetTagStatistics Procedure

RDb.GetTagStatistics retrieves statistical information about a tag in a specified time range from a data table.

Syntax

```vbs
RDb.GetTagStatistics(DevName, TagName, DataTableName: String; From, Till: DateTime; ByRef Min, Max, Sum, Ave, Count, Error: Variant)
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DevName</td>
<td>The name of the device that the tag belongs to.</td>
</tr>
<tr>
<td>TagName</td>
<td>The name of the tag.</td>
</tr>
<tr>
<td>DataTableName</td>
<td>The name of the data table.</td>
</tr>
<tr>
<td>From</td>
<td>The lower bound of the time range.</td>
</tr>
<tr>
<td>Till</td>
<td>The upper bound of the time range.</td>
</tr>
<tr>
<td>Min</td>
<td>The minimum value of the tag in the time range.</td>
</tr>
<tr>
<td>Max</td>
<td>The maximum value of the tag in the time range.</td>
</tr>
<tr>
<td>Sum</td>
<td>The sum of the values of the tag in the time range.</td>
</tr>
<tr>
<td>Ave</td>
<td>The average value of the tag in the time range.</td>
</tr>
<tr>
<td>Count</td>
<td>The number of database records in the time range.</td>
</tr>
<tr>
<td>Error</td>
<td>Error code (The List of Error Codes Returned by Methods and Properties of Reliance-defined Objects).</td>
</tr>
</tbody>
</table>
Remarks

The method searches both current and archive files (if working with data tables of type dBASE or Paradox). The method attempts to find and process the records with time stamp in the range specified by the From and Till arguments. After calling the method, it is recommended to check the value of the Error argument to find out about the result of the operation.

Example

```
Dim Value, From, Till, AMin, AMax, ASum, AAve, ACount, Error, Text
From = Date + TimeSerial(6, 0, 0) ' From: today, 6 a.m
Till = Date + TimeSerial(7, 0, 0) ' Till: today, 7 a.m
' Retrieve statistical information about the tag WaterTemperature
' from the device PLC1 stored in the data table Water.
Value = RDb.GetTagStatistics("PLC1", "WaterTemperature", "Water", From, Till, AMin, AMax, ASum, AAve, ACount, Error)
If Error = 0 Then
    Text = "Minimum: " + CStr(AMin) + " Maximum: " + CStr(AMax) + " Sum: " + CStr(ASum) + " Average: " + CStr(AAve) + " Record count: " + CStr(ACount) + "."
Else
    Text = "Error (code " + CStr(Error) + ")."
End If
' Store information on the result of the operation
' to the tag DisplayResult from the device System.
RTag.SetTagValue "System", "DisplayResult", Text
```
3.9 RDev Object

The RDev object implements methods for working with devices.

Methods:
- RDev.ConnectToCommDriver Procedure
- RDev.SendCustomData Procedure
- RDev.ReceiveCustomDataReply Procedure

3.9.1 RDev.ConnectToCommDriver Procedure

RDev.ConnectToCommDriver connects the runtime software to a specified communication driver.

Syntax

```
RDev.ConnectToCommDriver ProgID: String; Computer: Variant
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProgID</td>
<td>A unique identifier of the communication driver.</td>
</tr>
<tr>
<td>Computer</td>
<td>The name or ID (as defined in a visualization project) of the computer hosting the communication driver. A special value of &quot;&quot; means the local computer (i.e., the computer on which the runtime software is running).</td>
</tr>
</tbody>
</table>

Remarks

This method is sometimes used to reconnect the runtime software to a communication driver (e.g., if the driver was terminated and could not be restarted automatically by the runtime software).
Example

' Connect the runtime software to Teco OPC Server on the local computer.
RDev.ConnectToCommDriver "TECO.DA2", ""

3.9.2 RDev.SendCustomData Procedure

Sends custom data to a device(s) via a specified communication driver.

Syntax

RDev.SendCustomData ProgID: String, Computer: Variant, Params: Variant, Data: Variant

Argument | Description
---|---
ProgID | A unique identifier (as registered in the Windows registry) of the communication driver's COM interface.
Computer | The name or ID (as defined in a visualization project) of the computer hosting the communication driver. A special value of "" means the local computer (i.e., the computer on which the runtime software is running).
Params | Communication parameters.
Data | Data to be send.

Remarks

This method is sometimes used to send data from scripts.

Example

`dim` Params

' The first parameter represents a target where the communication packet should be sent to. 1 - to a communication channel, 2 - to a device. (only code 2 - device - is supported)
The second parameter represents the channel's or device's ID depending on the value of the 1st parameter. In this example, it is the device's ID.

```plaintext
Params = array(2,2)
```

Send the user-defined communication packet to the Sauter device.

```plaintext
RDev.SendCustomData "R_DrvSauter.dll", ",", Params, "P00101N/"
```

### 3.9.3 RDev.RDev.ReceiveCustomDataReply Procedure

Receives a reply (if any) to custom data sent to a device(s) via a specified communication driver.

**Syntax**

```plaintext
RDev.ReceiveCustomDataReply ProgID: String, Computer: Variant, Params: Variant, ByRef DataReply: Variant
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProgID</td>
<td>A unique identifier (as registered in the Windows registry) of the communication driver's COM interface.</td>
</tr>
<tr>
<td>Computer</td>
<td>The name or ID (as defined in a visualization project) of the computer hosting the communication driver. A special value of &quot;&quot; means the local computer (i.e., the computer on which the runtime software is running).</td>
</tr>
<tr>
<td>Params</td>
<td>Communication parameters.</td>
</tr>
<tr>
<td>DataReply</td>
<td>Reply to data, parameter passed by reference.</td>
</tr>
</tbody>
</table>

**Remarks**

This method is sometimes used to receive a reply to data.

[ RDev Object ]
Example

`dim Params, Data`

' The first parameter represents a source from which the communication packet should be received. 1 - from a communication channel, 2 - from a device. (only code 2 - device - is supported)

' The second parameter represents the channel's or device's ID depending on the value of the 1st parameter. In this example, it is the device's ID.

Params = array(2,2)

' Receive a reply to the user-defined communication packet sent to the Sauter device.

RDev.ReceiveCustomDataReply "R_DrvSauter.dll","",Params,Data
3.10 RError Object

The RError object provides information on the result of the most recent call to a method or access to a property of any Reliance-defined object with the exception of the RError object itself.

Properties:
- **RError.Code** Property
- **RError.Description** Property

Other:
- The List of Error Codes Returned by Methods and Properties of Reliance-defined Objects

3.10.1 RError.Code Property

RError.Code returns the error code of the most recent call to a method or access to a property of any Reliance-defined object with the exception of the RError object itself.

Syntax

```plaintext
RError.Code: Integer
```

Remarks

The property is read-only.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The most recent method call or property access was successful.</td>
</tr>
<tr>
<td>&gt;0</td>
<td>The code of the error (The List of Error Codes Returned by Methods and Properties of Reliance-defined Objects).</td>
</tr>
</tbody>
</table>

RError Object
Example

If RError.Code > 0 Then
  ' Store the error description
  ' to the tag DisplayResult from the device System
  RTag.SetTagValue "System", "DisplayResult", RError.Description
End If

3.10.2 RError.Description Property

RError.Description returns the error description of the most recent call to a method or access to a property of any Reliance-defined object with the exception of the RError object itself.

Syntax

  RError.Description: String

Remarks

The property is read-only. The List of Error Codes Returned by Methods and Properties of Reliance-defined Objects.

/// RError Object

Example

If RError.Code > 0 Then
  ' Store the error description
  ' to the tag DisplayResult from the device System
  RTag.SetTagValue "System", "DisplayResult", RError.Description
End If

3.10.3 The List of Reliance-defined Objects Error Codes

Error codes common to all objects

<table>
<thead>
<tr>
<th>Code (RError.Code)</th>
<th>Description (RError.Description)</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
<td>Explanation</td>
</tr>
<tr>
<td>------</td>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>0</td>
<td>Success.</td>
<td>The call to a method or access to a property was successful (i.e., no error occurred).</td>
</tr>
<tr>
<td>1</td>
<td>Unknown error.</td>
<td>An unknown error occurred (i.e., an error in a block of code where an error is expected but there is no way to find out exactly what error occurred).</td>
</tr>
<tr>
<td>2</td>
<td>Unexpected error.</td>
<td>An unexpected error occurred (i.e., an error in a block of code where an error is not expected, e.g., when calling the RDb.GetTagHistValue method).</td>
</tr>
<tr>
<td>3</td>
<td>Service is not supported.</td>
<td>The service provided by the method called is not supported.</td>
</tr>
<tr>
<td>100</td>
<td>Device not found.</td>
<td>The device was not found. The name of the device was not correctly specified or the device is not accessible to the computer on which the script was executed.</td>
</tr>
<tr>
<td>101</td>
<td>Tag not found.</td>
<td>The tag was not found. The name of the tag was not correctly specified.</td>
</tr>
<tr>
<td>102</td>
<td>Database not found.</td>
<td>The database was not found. The name of the database was not correctly specified or the database is not accessible to the computer on which the script was executed.</td>
</tr>
<tr>
<td>500</td>
<td>Cannot create directory.</td>
<td>The directory cannot be created.</td>
</tr>
</tbody>
</table>

### Error codes specific to TTable-type objects

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>10001</td>
<td>Tag not included in database definition. The database defined in a visualization project does not contain a field linked to the tag (i.e., the tag is not configured to be logged to the database).</td>
<td></td>
</tr>
<tr>
<td>10002</td>
<td>Missing database name. The <code>DatabaseName</code> property is blank (e.g., because it has not been assigned a value).</td>
<td></td>
</tr>
<tr>
<td>10003</td>
<td>Missing archive name. The <code>ArchiveName</code> property is blank (e.g., because it has not been assigned a value).</td>
<td></td>
</tr>
<tr>
<td>10010</td>
<td>Table does not exist. The database table does not exist.</td>
<td></td>
</tr>
<tr>
<td>10011</td>
<td>Table already exists. The database table already exists.</td>
<td></td>
</tr>
<tr>
<td>10012</td>
<td>Table not active. The database table is not open.</td>
<td></td>
</tr>
<tr>
<td>10013</td>
<td>Table not in edit or insert mode. The database table is not in edit or insert mode.</td>
<td></td>
</tr>
<tr>
<td>10014</td>
<td>Table is busy. The database table is currently being accessed by another program.</td>
<td></td>
</tr>
<tr>
<td>10015</td>
<td>Cannot open table. The database table cannot be opened.</td>
<td></td>
</tr>
<tr>
<td>10016</td>
<td>TTable object does not support working with SQL databases.</td>
<td></td>
</tr>
<tr>
<td>10030</td>
<td>Field not found in table. The database field was not found in the database table although the tag is configured to be logged to the database.</td>
<td></td>
</tr>
<tr>
<td>10031</td>
<td>Invalid field value. The returned value of the database field is invalid (when reading the value of the field).</td>
<td></td>
</tr>
<tr>
<td>10032</td>
<td>Invalid value for field. The value to be assigned to the database field is invalid (when writing the value of the field).</td>
<td></td>
</tr>
</tbody>
</table>
## Error codes specific to the RTag object

<table>
<thead>
<tr>
<th>Code (RError Code)</th>
<th>Description</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>11001</td>
<td>Invalid tag value.</td>
<td>The returned value of the tag is invalid (e.g., because the value has not yet been returned by the communication driver).</td>
</tr>
<tr>
<td>11002</td>
<td>Incorrect tag type.</td>
<td>The tag is of an incorrect data type (e.g., the value of an array-type tag cannot be returned by calling the RTag.GetTagValue method).</td>
</tr>
<tr>
<td>11003</td>
<td>Cannot send commands.</td>
<td>The tag cannot be written (e.g., because a visualization project runs in a view-only version of the runtime software).</td>
</tr>
<tr>
<td>11004</td>
<td>Cannot assign value to tag.</td>
<td>The tag cannot be assigned the specified value because the data type of the value is not assignment compatible with the data type of the tag.</td>
</tr>
<tr>
<td>11005</td>
<td>Array-type tag required.</td>
<td>The method called requires an array-type tag.</td>
</tr>
<tr>
<td>11006</td>
<td>Non-array-type tag required.</td>
<td>The method called requires a non-array-type tag.</td>
</tr>
<tr>
<td>11007</td>
<td>Incompatible tag types.</td>
<td>The data types of the tags are not compatible (e.g., when calling the RTag.MoveTagValue method).</td>
</tr>
<tr>
<td>11008</td>
<td>Unsupported type.</td>
<td>The method called does not support the data type of the tag.</td>
</tr>
<tr>
<td>11009</td>
<td>Cannot determine tag value.</td>
<td>The value of the tag cannot be determined.</td>
</tr>
</tbody>
</table>
### RTag Object

Error codes specific to the RSys object

<table>
<thead>
<tr>
<th>Code (RError Code)</th>
<th>Description (RError Description)</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>12001</td>
<td>Window not found.</td>
<td>The project window was not found. The name of the window was not correctly specified or the window is not accessible to the computer on which the script was executed (e.g., when calling the RSys.ActivateWindow method).</td>
</tr>
<tr>
<td>12002</td>
<td>Program file name cannot be empty.</td>
<td>The file name of the program to be run cannot be blank (e.g., when calling the RSys.ExecApp method).</td>
</tr>
</tbody>
</table>
## RError Object

### Error codes specific to the RDb object

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>12003</td>
<td>Sound file name cannot be empty.</td>
<td>The file name of the sound to be played cannot be blank (e.g., when calling the <strong>RSys.PlaySound</strong> method).</td>
</tr>
<tr>
<td>12010</td>
<td>File does not exist.</td>
<td>The file does not exist.</td>
</tr>
<tr>
<td>12011</td>
<td>Source file does not exist.</td>
<td>The source file does not exist (e.g., when calling the <strong>RSys.CopyFile</strong> method).</td>
</tr>
<tr>
<td>12012</td>
<td>Destination file already exists.</td>
<td>The destination file already exists.</td>
</tr>
<tr>
<td>12013</td>
<td>Cannot copy a file to itself.</td>
<td>The names of the source and destination files are identical. The operation cannot be performed.</td>
</tr>
</tbody>
</table>

### RSys Object

**Error codes specific to the RDb object**

<table>
<thead>
<tr>
<th>Code</th>
<th>(RError Code) Description</th>
<th>(RError Description)</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>13001</td>
<td>Database does not exist.</td>
<td>None of the tables of the database exist (neither current nor archive).</td>
<td></td>
</tr>
<tr>
<td>13002</td>
<td>Tag not stored in database.</td>
<td>The tag was not found in any of the tables belonging to the database (the tag is not configured to be logged to the database).</td>
<td></td>
</tr>
<tr>
<td>13003</td>
<td>Incorrect tag type.</td>
<td>The tag is of an incorrect data type (e.g., a string-type tag when calling the <strong>R.GetTagStatistics</strong> method).</td>
<td></td>
</tr>
<tr>
<td>13004</td>
<td>Cannot open database.</td>
<td>None of the tables of the database could be opened.</td>
<td></td>
</tr>
<tr>
<td>13005</td>
<td>No data available.</td>
<td>There are no records in the database in the specified time range.</td>
<td></td>
</tr>
</tbody>
</table>
**RError Object**

### RDb Object

### Error codes specific to the RScr object

<table>
<thead>
<tr>
<th>Code</th>
<th>(RError. Code)</th>
<th>Description</th>
<th>(RError. Description)</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>14001</td>
<td></td>
<td>Script not found.</td>
<td></td>
<td>The script was not found. The name of the script was not correctly specified.</td>
</tr>
</tbody>
</table>

### RScr Object

### Error codes specific to the RU User object

<table>
<thead>
<tr>
<th>Code</th>
<th>(RError. Code)</th>
<th>Description</th>
<th>(RError. Description)</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>15001</td>
<td></td>
<td>User not found.</td>
<td></td>
<td>The user was not found. The name of the user was not correctly specified.</td>
</tr>
<tr>
<td>15002</td>
<td></td>
<td>No user logged on.</td>
<td></td>
<td>No user is currently logged on to the runtime software.</td>
</tr>
<tr>
<td>15003</td>
<td></td>
<td>Invalid access rights.</td>
<td></td>
<td>The specified set of access rights is invalid, since one or more rights have invalid names.</td>
</tr>
</tbody>
</table>

### RU User Object

### Error codes specific to the RA lm object

<table>
<thead>
<tr>
<th>Code</th>
<th>(RError. Code)</th>
<th>Description</th>
<th>(RError. Description)</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>16001</td>
<td></td>
<td>Invalid alarm type.</td>
<td></td>
<td>The specified alarm type is invalid, since it is not within the range 0 to 2.</td>
</tr>
</tbody>
</table>

### RA lm Object
### RError Object

#### Error codes specific to the RInet object

<table>
<thead>
<tr>
<th>Code (Code)</th>
<th>Description</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>17001</td>
<td>Error initializing SMTP.</td>
<td>An error occurred while initializing the SMTP routines (routines for sending E-mail messages).</td>
</tr>
<tr>
<td>17002</td>
<td>Incomplete SMTP configuration.</td>
<td>The SMTP configuration is incomplete (some properties have not been specified).</td>
</tr>
<tr>
<td>17003</td>
<td>Error connecting to SMTP server.</td>
<td>An error occurred while connecting to the SMTP (E-mail) server.</td>
</tr>
<tr>
<td>17004</td>
<td>Error sending E-mail.</td>
<td>An error occurred while sending the E-mail message.</td>
</tr>
</tbody>
</table>

#### RModem Object

#### Error codes specific to the RModem object

<table>
<thead>
<tr>
<th>Code (Code)</th>
<th>Description</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>18001</td>
<td>SMS driver not initialized.</td>
<td>The GSM SMS driver has not been initialized.</td>
</tr>
<tr>
<td>18002</td>
<td>Error sending SMS.</td>
<td>An error occurred while sending the SMS message.</td>
</tr>
<tr>
<td>18003</td>
<td>Error dialing phone number.</td>
<td>An error occurred while dialing the phone number.</td>
</tr>
<tr>
<td>18004</td>
<td>SMS not found.</td>
<td>The SMS message was not found by the GSM SMS driver.</td>
</tr>
<tr>
<td>18005</td>
<td>Error sending AT command.</td>
<td>An error occurred while sending the AT command.</td>
</tr>
</tbody>
</table>
**RModem** Object
3.11 RInet Object

The RInet object implements methods for sending E-mail messages. In order for the methods to be operational, you have to correctly configure computer properties related to sending E-mail messages (configure the SMTP server, Port number, Connection timeout and Sender address properties on the E-mail page in the Project Structure Manager).

Methods:

- RInet.SendMail Function

### 3.11.1 RInet.SendMail Function

RInet.SendMail sends an E-mail message.

**Syntax**

```plaintext
RInet.SendMail(DestAddress, Subject, Text, FileName: String): Boolean
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DestAddress</td>
<td>The list of recipients (E-mail addresses) separated by a semicolon.</td>
</tr>
<tr>
<td>Subject</td>
<td>The subject of the message.</td>
</tr>
<tr>
<td>Text</td>
<td>The text of the message.</td>
</tr>
<tr>
<td>FileName</td>
<td>The list of attachments (file names) separated by a semicolon.</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The message was sent.</td>
</tr>
<tr>
<td>False</td>
<td>An error occurred while sending the message (information on the error is logged to the alarm database).</td>
</tr>
</tbody>
</table>
Remarks

If you want to send an E-mail message with no attachments, pass an empty string ("") as the last argument.

Example

' Send two files to the address reliance@hotmail.com.
If RInet.SendMail("reliance@hotmail.com", "Data files", "Data files Data1.txt and Data2.txt.
"; C:\Data\Data1.txt; C:\Data\Data2.txt") Then
' Store information on the result of the operation
' to the tag DisplayResult from the device System.
   RTag.SetTagValue "System", "DisplayResult", "The message was sent."
End If
3.12 RModem Object

The RModem object implements methods for sending and receiving SMS messages through a GSM modem connected to the computer. In order for the methods to be operational, you have to correctly configure computer properties related to GSM modems (activate the Start SMS driver property and configure the GSM device type, Communication options and SMS service center number properties on the SMS page in the Project Structure Manager).

Methods:

- **RMModem.GSMGetSMSStatus** Function
- **RMModem.GSMSendSMS** Function
- **RMModem.GSMSendSMSEx** Function

Other:

- The List of Error Codes (CMS) According to the GSM 07.05 Standard

### 3.12.1 RModem.GSMSendATCommand Function

**RMModem.GSMSendATCommand** sends a specified AT command to the GSM SMS driver.

**Syntax**

```
RMModem.GSMSendATCommand(Command: String): Boolean
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>AT command</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The AT command was placed in a queue by the GSM SMS driver.</td>
</tr>
<tr>
<td>False</td>
<td>An error occurred.</td>
</tr>
</tbody>
</table>
Remarks

The method passes the AT command to the GSM SMS driver which places it in an internal queue. The AT command is passed to the GSM modem when all previous requests in the queue are processed.

RModem Object

Example

Dim SCNumber, ATCommand
' The service centre number is stored in the tag SCNumber from the device System.
SCNumber = RTag.GetTagValue("System", "SCNumber")
ATCommand = "AT+CSCA=" & SCNumber & ""
If RModem.GSMSendATCommand(ATCommand) Then
    ' ...
End If

3.12.2 RModem.GSMGetSMSStatus Function

RModem.GSMGetSMSStatus returns the status of a SMS message previously sent by calling the RModem.GSMSendSMSEX method.

Syntax

RModem.GSMGetSMSStatus(ID: Integer; ByRef Text, PhoneNumber, Status, ErrorCode: Variant): Boolean

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>The unique identifier of the message returned by the RModem.GSMSendSMSEX method.</td>
</tr>
<tr>
<td>Text</td>
<td>The text of the message.</td>
</tr>
<tr>
<td>PhoneNumber</td>
<td>The phone number of the message's recipient.</td>
</tr>
<tr>
<td>Status</td>
<td>The status of the message (see the following table).</td>
</tr>
<tr>
<td>ErrorCode</td>
<td>The code of an error that might occur while sending the message (The List of Error Codes (CMS) According to the GSM 07.05 Standard).</td>
</tr>
</tbody>
</table>
### RModem Object

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The message is waiting in the queue.</td>
</tr>
<tr>
<td>1</td>
<td>The message was successfully sent.</td>
</tr>
<tr>
<td>2</td>
<td>An error occurred while sending the message. The code of the error is returned in the ErrorCode argument.</td>
</tr>
<tr>
<td>3</td>
<td>The status of the message is not available.</td>
</tr>
</tbody>
</table>

#### Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The status of the message is available.</td>
</tr>
<tr>
<td>False</td>
<td>The status of the message is not available.</td>
</tr>
</tbody>
</table>

#### Example

```vbnet
Dim PhoneNumber, Text, Status, ErrCode, ID
' The identifier of the most recently sent message is stored in the tag LastSMSID from the device System.
ID = RTag.GetTagValue("System", "LastSMSID")
' Retrieve the status of the message.
RModem.GSMGetSMSStatus ID, Text, PhoneNumber, Status, ErrCode
' Store the status of the message to the tag LastSMSStatus from the device System.
RTag.SetTagValue "System", "LastSMSStatus", Status
' Store the error code of the message to the tag LastSMSErrCode from the device System.
RTag.SetTagValue "System", "LastSMSErrCode", ErrCode
```
3.12.3  RModem.GSMSendSMS Function

RModem.GSMSendSMS sends a SMS message to a specified phone number by passing it to the GSM SMS driver.

Syntax

```
RModem.GSMSendSMS(PhoneNumber, Text: String): Boolean
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PhoneNumber</td>
<td>The phone number of the message's recipient.</td>
</tr>
<tr>
<td>Text</td>
<td>The text of the message.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The message was encoded and placed in a queue by the GSM SMS driver.</td>
</tr>
<tr>
<td>False</td>
<td>An error occurred.</td>
</tr>
</tbody>
</table>

Remarks

The method passes the message to the GSM SMS driver which places it in an internal queue. However, there is no way to find out whether the message was actually sent. If you want to retrieve the result of sending the message, use the RModem.GSMSendSMSEx method instead.

Example

```
Dim PhoneNumber, Text
' The phone number is stored in the tag Number from the device System
PhoneNumber = RTag.GetTagValue("System", "Number")
' The text of the message is stored in the tag Text from the device System.
```
Text = RTag.GetTagValue("System", "Text")
If RModem.GSMSendSMS(PhoneNumber, Text) Then
    ... 
End If

3.12.4 RModem.GSMSendSMSEx Function

RModem.GSMSendSMSEx sends a SMS message to a specified phone number by passing it to the GSM SMS driver and returns a unique identifier for the message.

Syntax

RModem.GSMSendSMSEx(PhoneNumber, Text: String; ByRef ID: Integer): Boolean

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PhoneNumber</td>
<td>The phone number of the message's recipient.</td>
</tr>
<tr>
<td>Text</td>
<td>The text of the message.</td>
</tr>
<tr>
<td>ID</td>
<td>The identifier of the message.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The message was encoded and placed in a queue by the GSM SMS driver.</td>
</tr>
<tr>
<td>False</td>
<td>An error occurred.</td>
</tr>
</tbody>
</table>

Remarks

The method passes the message to the GSM SMS driver which places it in an internal queue and returns a unique identifier (a number unique within an instance of the driver) for the message. The identifier can later be used to retrieve the status of the message by passing it to the RModem.GSMGetSMSSStatus method. Information on the status of sending a message is stored in the driver for a time period determined by the driver's settings (the default time period is 24 hours).
Example

```vbnet
Dim PhoneNumber, Text, ID
'
' The phone number is stored in the tag Number from the device System.
PhoneNumber = RTag.GetTagValue("System", "Number")
'
' The text of the message is stored in the tag Text from the device System.
Text = RTag.GetTagValue("System", "Text")
If RModem.GMSendSMSEx(PhoneNumber, Text, ID) Then
    '
    Store the value of ID so that it can later be passed
    '
    to the RModem.GSMGetSMSStatus method.
    RTag.SetTagValue "System", "LastSMSID", ID
End If
```

3.12.5 The List of Error Codes (CMS) According to GSM 07.05 Standard

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Unassigned (unallocated) number.</td>
</tr>
<tr>
<td>8</td>
<td>Operator determined barring.</td>
</tr>
<tr>
<td>10</td>
<td>Call barred.</td>
</tr>
<tr>
<td>21</td>
<td>Short message transfer rejected.</td>
</tr>
<tr>
<td>27</td>
<td>Destination out of service.</td>
</tr>
<tr>
<td>28</td>
<td>Unidentified subscriber.</td>
</tr>
<tr>
<td>29</td>
<td>Facility rejected.</td>
</tr>
<tr>
<td>30</td>
<td>Unknown subscriber.</td>
</tr>
<tr>
<td>38</td>
<td>Network out of order.</td>
</tr>
<tr>
<td>41</td>
<td>Temporary failure.</td>
</tr>
<tr>
<td>42</td>
<td>Congestion.</td>
</tr>
<tr>
<td>47</td>
<td>Resources unavailable, unspecified.</td>
</tr>
<tr>
<td>50</td>
<td>Requested facility not subscribed.</td>
</tr>
<tr>
<td>Code</td>
<td>Error Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------------</td>
</tr>
<tr>
<td>69</td>
<td>Requested facility not implemented.</td>
</tr>
<tr>
<td>81</td>
<td>Invalid short message transfer reference value.</td>
</tr>
<tr>
<td>95</td>
<td>Invalid message, unspecified.</td>
</tr>
<tr>
<td>96</td>
<td>Invalid mandatory information.</td>
</tr>
<tr>
<td>97</td>
<td>Message type non-existent or not implemented.</td>
</tr>
<tr>
<td>98</td>
<td>Message not compatible with short message protocol state.</td>
</tr>
<tr>
<td>99</td>
<td>Information element non-existent or not implemented.</td>
</tr>
<tr>
<td>111</td>
<td>Protocol error, unspecified.</td>
</tr>
<tr>
<td>127</td>
<td>Interworking, unspecified.</td>
</tr>
<tr>
<td>128</td>
<td>Telematic interworking not supported.</td>
</tr>
<tr>
<td>129</td>
<td>Short message Type 0 not supported.</td>
</tr>
<tr>
<td>130</td>
<td>Cannot replace short message.</td>
</tr>
<tr>
<td>143</td>
<td>Unspecified TP-PID error.</td>
</tr>
<tr>
<td>144</td>
<td>Data coding scheme (alphabet) not supported.</td>
</tr>
<tr>
<td>145</td>
<td>Message class not supported.</td>
</tr>
<tr>
<td>159</td>
<td>Unspecified TP-DCS error.</td>
</tr>
<tr>
<td>160</td>
<td>Command cannot be actioned.</td>
</tr>
<tr>
<td>161</td>
<td>Command unsupported.</td>
</tr>
<tr>
<td>175</td>
<td>Unspecified TP-Command error.</td>
</tr>
<tr>
<td>176</td>
<td>TPDU not supported.</td>
</tr>
<tr>
<td>192</td>
<td>SC busy.</td>
</tr>
<tr>
<td>193</td>
<td>No SC subscription.</td>
</tr>
<tr>
<td>194</td>
<td>SC system failure.</td>
</tr>
<tr>
<td>195</td>
<td>Invalid SME address.</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>196</td>
<td>Destination SME barred.</td>
</tr>
<tr>
<td>197</td>
<td>SM Rejected-Duplicate SM.</td>
</tr>
<tr>
<td>198</td>
<td>TP-VPF not supported.</td>
</tr>
<tr>
<td>199</td>
<td>TP-VP not supported.</td>
</tr>
<tr>
<td>208</td>
<td>DO SIM SMS storage full.</td>
</tr>
<tr>
<td>209</td>
<td>No SMS storage capability in SIM.</td>
</tr>
<tr>
<td>210</td>
<td>Error in MS.</td>
</tr>
<tr>
<td>211</td>
<td>Memory Capacity Exceeded.</td>
</tr>
<tr>
<td>212</td>
<td>SIM Application Toolkit Busy.</td>
</tr>
<tr>
<td>213</td>
<td>SIM data download error.</td>
</tr>
<tr>
<td>255</td>
<td>Unspecified error cause.</td>
</tr>
<tr>
<td>300</td>
<td>ME failure.</td>
</tr>
<tr>
<td>301</td>
<td>SMS service of ME reserved.</td>
</tr>
<tr>
<td>302</td>
<td>Operation not allowed.</td>
</tr>
<tr>
<td>303</td>
<td>Operation not supported.</td>
</tr>
<tr>
<td>304</td>
<td>Invalid PDU mode parameter.</td>
</tr>
<tr>
<td>305</td>
<td>Invalid text mode parameter.</td>
</tr>
<tr>
<td>310</td>
<td>SIM not inserted.</td>
</tr>
<tr>
<td>311</td>
<td>SIM PIN required.</td>
</tr>
<tr>
<td>312</td>
<td>PH-SIM PIN required.</td>
</tr>
<tr>
<td>313</td>
<td>SIM failure.</td>
</tr>
<tr>
<td>314</td>
<td>SIM busy.</td>
</tr>
<tr>
<td>315</td>
<td>SIM wrong.</td>
</tr>
<tr>
<td>316</td>
<td>SIM PUK required.</td>
</tr>
<tr>
<td>317</td>
<td>SIM PIN2 required.</td>
</tr>
<tr>
<td>318</td>
<td>SIM PUK2 required.</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>320</td>
<td>Memory failure.</td>
</tr>
<tr>
<td>321</td>
<td>Invalid memory index.</td>
</tr>
<tr>
<td>322</td>
<td>Memory full.</td>
</tr>
<tr>
<td>330</td>
<td>SMSC address unknown.</td>
</tr>
<tr>
<td>331</td>
<td>No network service.</td>
</tr>
<tr>
<td>332</td>
<td>Network timeout.</td>
</tr>
<tr>
<td>340</td>
<td>NO +CNMA ACK EXPECTED.</td>
</tr>
<tr>
<td>500</td>
<td>Unknown error.</td>
</tr>
<tr>
<td>512</td>
<td>User abort.</td>
</tr>
<tr>
<td>513</td>
<td>Unable to store.</td>
</tr>
<tr>
<td>514</td>
<td>Invalid status.</td>
</tr>
<tr>
<td>515</td>
<td>Invalid character in address string.</td>
</tr>
<tr>
<td>516</td>
<td>Invalid length.</td>
</tr>
<tr>
<td>517</td>
<td>Invalid character in pdu.</td>
</tr>
<tr>
<td>518</td>
<td>Invalid parameter.</td>
</tr>
<tr>
<td>519</td>
<td>Invalid length or character.</td>
</tr>
<tr>
<td>520</td>
<td>Invalid character in text.</td>
</tr>
<tr>
<td>521</td>
<td>Timer expired.</td>
</tr>
</tbody>
</table>

**RModem Object**
3.13 RProj Object

The RProj object implements methods for working with a Reliance project.

Methods:
- RProj.GetObject Function
- RProj.GetObjectList Function
- RProj.ObjectExists Function

3.13.1 RProj.GetObject Function

RProj.GetObject returns an object defined in a visualization project.

Syntax

RProj.GetObject(ObjectType: Integer; ObjectIdent: Variant; ByRef Object: Variant): Boolean

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObjectType</td>
<td>The type of object. See the RConst. ProjObjType object's properties.</td>
</tr>
<tr>
<td>ObjectIdent</td>
<td>The full name or ID (unique identifier) of an object. The full name uniquely specifies the object within the project. For example, Modbus1/GasPressure is the full name of the tag GasPressure from the device Modbus1.</td>
</tr>
<tr>
<td>Object</td>
<td>An object defined in the project (e.g., device, tag, alarm/event).</td>
</tr>
</tbody>
</table>
Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The object has been found.</td>
</tr>
<tr>
<td>False</td>
<td>The object has not been found.</td>
</tr>
</tbody>
</table>

Remarks

The separator in the full name of the object (in the ObjectIdent argument) depends on a setting in the Project Options dialog (Project > Objects).

Example

```vbnet
Dim Object
' Get the object of the tag GasPressure from the device Modbus1.
If RProj.GetObject(RConst.ProjObjType.Tag, "Modbus1/GasPressure", Object) Then
  ' ...
End If
```

3.13.2 RProj.GetObjectList Function

RProj.GetObjectList returns the list of objects defined in a visualization project.

Syntax

```vbnet
RProj.GetObjectList(ObjectType, ParentObjectType: Integer; ParentObjectIdent: Variant; ByRef ObjectList, ObjectCount: Variant): Boolean
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObjectType</td>
<td>The type of objects to be returned in ObjectList. See the RConst.ProjObjType object's properties.</td>
</tr>
<tr>
<td>Argument</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>ParentObjectType</code></td>
<td>The type of parent object whose child objects are to be returned in <code>ObjectList</code> (e.g., tags are the child objects of a tag folder, a device, and a device folder). It can be <code>Empty</code> if the argument is not to be used. In such a case, all objects of type <code>ObjectType</code> will be returned in the list. See the <code>RConst.ProjObjType</code> object's properties.</td>
</tr>
<tr>
<td><code>ParentObjectIdent</code></td>
<td>The full name or ID (unique identifier) of a parent object whose child objects are to be returned in <code>ObjectList</code> (e.g., tags are the child objects of a tag folder, a device, and a device folder). The full name uniquely specifies the object within the project. For example, <code>Modbus1/Digital_inputs</code> is the full name of the tag folder <code>Digital_inputs</code> from the device <code>Modbus1</code>. It can be <code>Empty</code> if the argument is not to be used. In such a case, the <code>ParentObjectType</code> argument must also be <code>Empty</code> and all objects of type <code>ObjectType</code> will be returned in the list.</td>
</tr>
<tr>
<td><code>ObjectList</code></td>
<td>The list of objects returned by this function (e.g., device, tag, alarm/event).</td>
</tr>
<tr>
<td><code>ObjectCount</code></td>
<td>The number of objects returned in...</td>
</tr>
</tbody>
</table>
### Argument Description

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObjectList</td>
<td></td>
</tr>
</tbody>
</table>

### Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>Managed to get the list of objects.</td>
</tr>
<tr>
<td>False</td>
<td>Cannot get the list of objects. Examples of possible causes: An invalid type of objects and/or or parent object has been specified; the parent object has not been found; the parent object is not allowed to have child objects of the specified type.</td>
</tr>
</tbody>
</table>

### Remarks

The separator in the full name of the object (in the ParentObjectIdent argument) depends on a setting in the Project Options dialog (Project > Objects).

### Example

```vbnet
Dim ObjectList, ObjectCount
' Get the list of tag-type objects from the device Modbus1.
' ...
End If
```
3.13.3 RProj.ObjectExists Function

RProj.ObjectExists determines whether a specified object exists in a visualization project.

Syntax

RProj.ObjectExists( ObjectType: Integer; ObjectIdent: Variant): Boolean

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObjectType</td>
<td>The type of object. See the RConst.ProjObjType object's properties.</td>
</tr>
<tr>
<td>ObjectIdent</td>
<td>The full name or ID (unique identifier) of an object. The full name uniquely specifies the object within the project. For example, Modbus1/GasPressure is the full name of the tag GasPressure from the device Modbus1.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The object exists.</td>
</tr>
<tr>
<td>False</td>
<td>The object does not exist.</td>
</tr>
</tbody>
</table>

Remarks

The separator in the full name of the object (in the ObjectIdent argument) depends on a setting in the Project Options dialog (Project > Objects).
Example

```vbscript
Dim ObjectType, FullObjectName
ObjectType = RTag.GetTagValue("System", "ObjectType")
FullObjectName = Trim(RTag.GetTagValue("System", "FullObjectName"))
If RProj.ObjectExists(ObjectType, FullObjectName) Then ' The object exists.
   '...
End If
```
3.14 RScr Object

The RScr object implements methods for operations on scripts defined in a visualization project.

The methods enable you to enable/disable scripts, retrieve parameters passed to a script, and retrieve information on scripts.

Methods:

- `RScr.DisableScript` Procedure
- `RScr.EnableScript` Procedure
- `RScr.ExecScript` Procedure
- `RScr.ExecScriptForThinClient` Procedure
- `RScr.ExecScriptOnComputer` Procedure
- `RScr.GetCurrentScriptData` Function
- `RScr.GetCurrentScriptDataEx` Function
- `RScr.GetCurrentThreadName` Function
- `RScr.GetScriptInfo` Function
- `RScr.GetScriptText` Function
- `RScr.ResetCurrentScriptUser` Procedure
- `RScr.SetCurrentScriptUser` Procedure

Events:

- Basic Events
- Events Triggered by a Component
- Events Triggered by a Change in Tag Value, Quality, or Time Stamp
- Events Triggered by an Alarm
- Events Triggered by a Text Message
- Events Triggered by a Thin Client Request
- Events Triggered by the Maatrix Service
- Events Triggered by an Embedded Web Browser
3.14.1  RScr.DisableScript Procedure

RScr.DisableScript disables a script.

Syntax

```
RScr.DisableScript Script: Variant
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Script</td>
<td>The name or ID of the script.</td>
</tr>
</tbody>
</table>

Remarks

Calling this method disables the script only for the time a visualization project is running. After terminating the project, the information is lost. When the project is started next time, the script is again in its initial state, i.e., either enabled or disabled, depending on the value of the Enable execution property (Reliance Design > Managers > Script Manager > script properties > the Basic page).

Example

```
If RSys.GetComputerName = "PCI" Then
  ' Disable the script Script1 if the current script
  ' is executing on the computer PCI.
  RScr.DisableScript "Script1"
Else
  ' Enable the script Script1.
  RScr.EnableScript "Script1"
End If
```
3.14.2 RScr.EnableScript Procedure

`RScr.EnableScript` enables a script.

Syntax

```
RScr.EnableScript Script: Variant
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Script</td>
<td>The name or ID of the script.</td>
</tr>
</tbody>
</table>

Remarks

Calling this method disables the script only for the time a visualization project is running. After terminating the project, the information is lost. When the project is started next time, the script is again in its initial state, i.e., either enabled or disabled, depending on the value of the *Enable execution* property (*Reliance Design* > *Managers* > *Script Manager* > *script properties* > *the Basic page*).

Example

```
If RSys.GetComputerName = "PC1" Then
  ' Disable the script Script1 if the current script
  ' is executing on the computer PC1.
  RScr.DisableScript "Script1"
Else
  ' Enable the script Script1.
  RScr.EnableScript "Script1"
End If
```
3.14.3 RScr.ExecScript Procedure

RScr.ExecScript queues a script for execution.

Syntax

\[
\text{RScr.ExecScript Script: Variant; PriorExec: Boolean; [ExecForEachThinClient: Boolean]}
\]

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Script</td>
<td>Mandatory. The name or ID of the script.</td>
</tr>
<tr>
<td>PriorExec</td>
<td>Mandatory. Determines whether to execute the script prior to other scripts (see the following table).</td>
</tr>
<tr>
<td>ExecForEachThinClient</td>
<td>Optional. Determines whether to execute the script for each connected thin client (i.e., in the context of each connected thin client). If ExecForEachThinClient omitted, the default value False will be used.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The script is placed in the prior script queue to a position that depends on the script's priority (Reliance Design &gt; Managers &gt; Script Manager &gt; script properties &gt; the Advanced page). Placing the script in the prior script queue ensures that the script is executed prior to all scripts placed in the standard script queue.</td>
</tr>
<tr>
<td>False</td>
<td>The script is placed in the standard script queue to a position that depends on the script's priority.</td>
</tr>
</tbody>
</table>

Remarks

The script is placed in the appropriate script queue and can be executed only after the current script finishes execution. To execute a script's code immediately, use the Execute statement in combination with the RScr.GetScriptText method.
RScr Object

Example

' Place the script Script1 in the prior script queue.
RScr.ExecScript "Script1", True

3.14.4 RScr.ExecScriptForThinClient Procedure

RScr.ExecScriptForThinClient queues a script for execution in the context of a thin client.

Syntax

RScr.ExecScriptForThinClient Script: Variant; PriorExec: Boolean;
WebServerSessionName: String

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Script</td>
<td>The name or ID of the script.</td>
</tr>
<tr>
<td>PriorExec</td>
<td>Determines whether to execute the script prior to other scripts (see the table below).</td>
</tr>
<tr>
<td>WebServerSessionName</td>
<td>The unique identifier of the Web server session.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The script is placed in the prior script queue to a position that depends on the script's priority (Reliance Design &gt; Managers &gt; Script Manager &gt; script properties &gt; the Advanced page). Placing the script in the prior script queue ensures that the script is executed prior to all scripts placed in the standard script queue.</td>
</tr>
<tr>
<td>False</td>
<td>The script is placed in the standard script queue to a position that depends on the script's priority.</td>
</tr>
</tbody>
</table>
Remarks

The script is placed in the appropriate script queue and can be executed only after the current script finishes execution.

Example

' Queue the script ThinClientScript for execution in the context of the thin client specified by the session name stored in the tag WebServerSessionName from the device System.
RScr.ExecScriptForThinClient "ThinClientScript", True, RTag.GetTagValue("System", "WebServerSessionName")


RScr.ExecScriptOnComputer queues a script for execution on a specified computer.

Syntax

```
RScr.ExecScriptOnComputer Script: Variant; PriorExec: Boolean; Computer: Variant
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Script</td>
<td>The name or ID of the script.</td>
</tr>
<tr>
<td>PriorExec</td>
<td>Determines whether to execute the script prior to other scripts (see the table below).</td>
</tr>
<tr>
<td>Computer</td>
<td>The name or ID of the computer on which the script should execute.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The script is placed in the prior script queue to a position that depends on the script's priority (Reliance Design > Managers > Script Manager > script properties > the Advanced page). Placing the script in the prior script queue ensures that the script is executed prior to all scripts placed in the standard script queue.

The script is placed in the standard script queue to a position that depends on the script's priority.

Remarks

The script is placed in the appropriate script queue on a specified computer and can be executed only after the current script finishes execution. It is necessary that a connection be established between the computer, on which this procedure is called, and the specified computer. Otherwise the procedure will fail. The typical usage is running a script from a client on a server. There must be a server connection defined between the client and the server, based on which a connection between them will be established during runtime. The script to execute on the specified computer does not have to be connected (through the Project Structure Manager) to the computer, on which this procedure is called.

Example

' Queue the script ServerScript1 for execution on computer Server1
RScr.ExecScriptOnComputer "ServerScript1", True, "Server1"

3.14.6 RScr.GetCurrentScriptData Function

RScr.GetCurrentScriptData returns data passed as a parameter to the current script.

Syntax

RScr.GetCurrentScriptData(ByRef Data: Variant): Boolean

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
</table>

Reliance 4 – Scripts
**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The <em>Data</em> argument has been assigned the value of the data passed to the current script.</td>
</tr>
<tr>
<td>False</td>
<td>The <em>Data</em> argument has not been assigned a value, since no data has been passed to the current script.</td>
</tr>
</tbody>
</table>

**Remarks**

If the current script has been triggered by an alarm (i.e., start, end, or acknowledgment), the method retrieves the text of the alarm.

This method is obsolete and is provided only for backward compatibility. For new applications, use the `RScr.GetCurrentScriptDataEx` method.

**Example**

```
' This script is intended to be triggered when an alarm is generated.
Dim Data
' If the Data argument has been assigned a value.
If RScr.GetCurrentScriptData(Data) Then
  ' Send an E-mail containing the text of the alarm.
  If RInet.SendMail("reliance@hotmail.com", "Alarm generated", Data, ") Then
    ' ...
  End If
End If
```
3.14.7 RScr.GetCurrentScriptDataEx Function

RScr.GetCurrentScriptDataEx returns data passed as a parameter to the current script.

Syntax

```
RScr.GetCurrentScriptDataEx(ByRef Data: Variant): Boolean
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>The data passed to the current script.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The <code>Data</code> argument has been assigned the value of the data passed to the current script.</td>
</tr>
<tr>
<td>False</td>
<td>The <code>Data</code> argument has not been assigned a value, since no data has been passed to the current script.</td>
</tr>
</tbody>
</table>

Remarks

If the method returns **True**, the `Data` argument references an object with the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SenderName</td>
<td>String</td>
</tr>
<tr>
<td>UserName</td>
<td>String</td>
</tr>
<tr>
<td>WebServerSessionName</td>
<td>String</td>
</tr>
<tr>
<td>StrPar1</td>
<td>String</td>
</tr>
<tr>
<td>StrPar2</td>
<td>String</td>
</tr>
<tr>
<td>StrPar3</td>
<td>String</td>
</tr>
<tr>
<td>StrPar4</td>
<td>String</td>
</tr>
<tr>
<td>Parameter</td>
<td>Type</td>
</tr>
<tr>
<td>------------</td>
<td>----------</td>
</tr>
<tr>
<td>StrPar5</td>
<td>String</td>
</tr>
<tr>
<td>StrPar6</td>
<td>String</td>
</tr>
<tr>
<td>StrPar7</td>
<td>String</td>
</tr>
<tr>
<td>StrPar8</td>
<td>String</td>
</tr>
<tr>
<td>StrPar9</td>
<td>String</td>
</tr>
<tr>
<td>DoublePar1</td>
<td>Double</td>
</tr>
<tr>
<td>DoublePar2</td>
<td>Double</td>
</tr>
<tr>
<td>DatePar1</td>
<td>DateTime</td>
</tr>
<tr>
<td>DatePar2</td>
<td>DateTime</td>
</tr>
<tr>
<td>DatePar3</td>
<td>DateTime</td>
</tr>
<tr>
<td>DatePar4</td>
<td>DateTime</td>
</tr>
<tr>
<td>IntPar1</td>
<td>Integer</td>
</tr>
<tr>
<td>IntPar2</td>
<td>Integer</td>
</tr>
<tr>
<td>IntPar3</td>
<td>Integer</td>
</tr>
<tr>
<td>IntPar4</td>
<td>Integer</td>
</tr>
<tr>
<td>IntPar5</td>
<td>Integer</td>
</tr>
<tr>
<td>IntPar6</td>
<td>Integer</td>
</tr>
<tr>
<td>IntPar7</td>
<td>Integer</td>
</tr>
<tr>
<td>BytePar1</td>
<td>Byte</td>
</tr>
<tr>
<td>BytePar2</td>
<td>Byte</td>
</tr>
<tr>
<td>BytePar3</td>
<td>Byte</td>
</tr>
<tr>
<td>BytePar4</td>
<td>Byte</td>
</tr>
<tr>
<td>BoolPar1</td>
<td>Boolean</td>
</tr>
<tr>
<td>BoolPar2</td>
<td>Boolean</td>
</tr>
<tr>
<td>BoolPar3</td>
<td>Boolean</td>
</tr>
<tr>
<td>BoolPar4</td>
<td>Boolean</td>
</tr>
</tbody>
</table>
The meaning of individual properties depends on the event that triggered the script:

**Basic Events**

- Events Triggered by a Component
- Events Triggered by a Change in Tag Value, Quality, or Time Stamp
- Events Triggered by an Alarm
- Events Triggered by a Text Message
- Events Triggered by a Thin Client Request
- Events Triggered by the Maatrix Service

**Example 1**

' This script is intended to be triggered by clicking a component.
Dim Data
' If the Data argument has been assigned a value
' we can to access individual properties.
If RScr.GetCurrentScriptDataEx(Data) Then
' If the user clicked a component that runs this script
' with the parameter equal to 3.
If Data.IntPar1 = 3 Then
' ...
End If
End If

**Example 2**

' This script is intended to be triggered by a change in value, quality, or time stamp of a tag/tags.
Dim Data
' If the Data argument has been assigned a value
'we can to access individual properties.

If RScr.GetCurrentScriptDataEx(Data) Then
   RTag.SetTagValue "System", "SenderName", Data.SenderName
   RTag.SetTagValue "System", "DeviceName", Data.StrPar2
   RTag.SetTagValue "System", "TagName", Data.StrPar3
   RTag.SetTagValue "System", "DeviceId", Data.IntPar2
   RTag.SetTagValue "System", "TagId", Data.IntPar3
Select Case Data.BytePar3
   Case 1
      RTag.SetTagValue "System", "ChangeKind", "Value"
      RTag.SetTagValue "System", "PriorValue", Data.VariantPar1
      RTag.SetTagValue "System", "Value", Data.VariantPar2
   Case 2
      RTag.SetTagValue "System", "ChangeKind", "Quality"
      RTag.SetTagValue "System", "PriorValid", Data.BoolPar1
      RTag.SetTagValue "System", "Valid", Data.BoolPar2
   Case 4
      RTag.SetTagValue "System", "ChangeKind", "Time stamp"
      RTag.SetTagValue "System", "PriorTimeStamp", Data.DatePar1
      RTag.SetTagValue "System", "TimeStamp", Data.DatePar2
End Select
End If

Example 3
'This script is intended to be triggered when an alarm is generated.

Dim Data
'If the Data argument has been assigned a value
'we can to access individual properties.
If RScr.GetCurrentScriptDataEx(Data) Then
   'Send an E-mail containing the text of the alarm
   If RNet.SendMail("reliance@hotmail.com", "Alarm generated", Data.StrPar1, ") Then
      ...'
   End If
End If

Example 4
'This script is intended to be triggered by receiving a text message.

Dim Data
'If the Data argument has been assigned a value
'we can to access individual properties.
If Rscr.GetCurrentScriptDataEx(Data) Then
    RTag.SetTagValue "System", "Rec_Data", Data.StrPar1 ' Data.
    RTag.SetTagValue "System", "Rec_Number", Data.StrPar3 ' Number.
    ' The time assigned by the SMS service center.
    RTag.SetTagValue "System", "Rec_SCTime", Data.StrPar4
    ' The time of receiving the message by the GSM SMS driver.
    RTag.SetTagValue "System", "Rec_Time", CStr(Data.DatePar1)
End If

Example 5

' This script is intended to be triggered by a thin client request.
Dim Data
' If the Data argument has been assigned a value
' we can to access individual properties.
If Rscr.GetCurrentScriptDataEx(Data) Then
    ' Request Type: Timeout = 0, Connect = 1, Disconnect = 2, User Logon = 3, User Logout = 4
    Select Case Data.IntPar1
        Case 0
            RTag.SetTagValue "System", "Request_Type", "Disconnect (Expired Session)"
        Case 1
            RTag.SetTagValue "System", "Request_Type", "Connect"
        Case 2
            RTag.SetTagValue "System", "Request_Type", "Disconnect"
        Case 3
            RTag.SetTagValue "System", "Request_Type", "User Logon"
        Case 4
            RTag.SetTagValue "System", "Request_Type", "User Logout"
    End Select
    ' Thin Client Type: Reliance Web Client = 0
    Select Case Data.IntPar2
        Case 0
            RTag.SetTagValue "System", "Client_Type", "Reliance Web Client"
        Case 2
            RTag.SetTagValue "System", "Client_Type", "Reliance Smart Client"
    End Select
    RTag.SetTagValue "System", "Session_Name", Data.StrPar1 ' Unique Session Identifier.
RTag. SetTagValue "System", "Computer_Name", Data.StrPar4 ' Computer Name
(Configuration).
RTag. SetTagValue "System", "User_Name", Data.StrPar5 ' User Name.
RTag. SetTagValue "System", "User_Agent", Data.StrPar6 ' Information on Web
browser (User-Agent header). Only for Reliance Smart Client.

Example 6

' This script is intended to be triggered by the Maatrix service.
Dim Data
If Rscr. GetCurrentScriptDataEx(Data) Then
   RTag. SetTagValue "System", "Maatrix_IncidentSolverFound", Data.BoolPar1
End If

Example 7

' This script is intended to be triggered by an embedded Web browser.
Dim Data
If Rscr. GetCurrentScriptDataEx(Data) Then
   If Data.CustomStrPar1 = "BR1" Then
      RSys. ActivateWindow "BoilerRoom1"
   End If
End If

3.14.8 RScr.GetCurrentThreadName Function

RScr.GetCurrentThreadName returns the name of the current thread, i.e., the thread in
which this function is called.

Syntax

RScr.GetCurrentThreadName: String

Remarks

The threads are defined through the Project Options dialog (Project > Scripts > Threads).
For each script, you can choose the thread in which the script will run.
Example

```vbnet
If RScr.GetCurrentThreadName = "Calculations" Then
    ' ...
End If
```

### 3.14.9 RScr.GetScriptInfo Function

RScr.GetScriptInfo returns information on a script.

**Syntax**

```
RScr.GetScriptInfo(Script: Variant; ByRef Enabled, LastExecStartTime, LastExecEndTime, LastForcedTerminTime, ForcedTerminCount, ExecErrorCount: Variant): Boolean
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Script</td>
<td>The name or ID of the script.</td>
</tr>
<tr>
<td>Enabled</td>
<td>Determines whether the script is enabled.</td>
</tr>
<tr>
<td>LastExecStartTime</td>
<td>The date and time that the script last started execution.</td>
</tr>
<tr>
<td>LastExecEndTime</td>
<td>The date and time that the script last finished execution.</td>
</tr>
<tr>
<td>LastForcedTerminTime</td>
<td>The date and time that the script was last forcibly terminated.</td>
</tr>
<tr>
<td>ForcedTerminCount</td>
<td>The number of forcible terminations of the script during the time a visualization project is running.</td>
</tr>
<tr>
<td>ExecErrorCount</td>
<td>The number of errors (other than syntax errors) while executing the script during the time a visualization project is running.</td>
</tr>
</tbody>
</table>
Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The script was found and the returned information is valid.</td>
</tr>
<tr>
<td>False</td>
<td>The script was not found and the returned information is not valid.</td>
</tr>
</tbody>
</table>

**Example**

```vba
Dim Enabled, LastExecStartTime, LastExecEndTime, LastForcedTerminTime, ForcedTerminCount, ExecErrorCount
' If the script Script1 was found.
If RScr.GetScriptInfo("Script1", Enabled, LastExecStartTime, LastExecEndTime, LastForcedTerminTime, ForcedTerminCount, ExecErrorCount) Then
  If Not Enabled Then ' If the script is not enabled.
    RScr.EnableScript "Script1" ' Enable the script.
  End If
End If
```

### 3.14.10 RScr.GetScriptText Function

**RScr.GetScriptText** returns the text (program code) of a script.

**Syntax**

```vba
RScr.GetScriptText(Script: Variant): String
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Script</td>
<td>The name or ID of the script.</td>
</tr>
</tbody>
</table>

**Remarks**

This method can be used in combination with the Execute statement to execute a script's code immediately.
RScr Object

Example

Dim ScriptCode
' Place the script Script2 in the prior script queue.
RScr.ExecScript "Script2", True
ScriptCode = RScr.GetScriptText("Script1")
' Retrieve the text (program code) of the script Script1
' and execute it using the Execute statement.
' Thus, Script1 executes sooner than Script2.
Execute ScriptCode

3.14.11 RScr.ResetCurrentScriptUser Procedure

Resets the user on behalf of which subsequent statements in the current script are to be performed.

Syntax

RScr.ResetCurrentScriptUser

Remarks

Resets the user that has been set by the RScr.SetCurrentScriptUser procedure.

Example

Dim UserName
' Get a user name from an internal tag.
UserName = RTag.GetTagValue("System", "UserName")
' Set the user specified by UserName to the current script.
RScr.SetCurrentScriptUser UserName
' Set the value of the tag Setpoint from the device PLC1.
' This statement will be performed on behalf of the user specified by UserName.
RTag.SetTagValue "PLC1", "Setpoint", 20
' Reset the user for the current script,
' i.e. return to the state that existed before calling the RScr.SetCurrentScriptUser
**RScr Object**

**RScr. ResetCurrentScriptUser**

**3.14.12 RScr.SetCurrentScriptUser Procedure**

Sets the user on behalf of which subsequent statements in the current script are to be performed.

**Syntax**

```
RScr.SetCurrentScriptUser User: Variant
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>The name or ID of the user. The value &quot;&quot; or Empty corresponds to the state when no user is logged on.</td>
</tr>
</tbody>
</table>

**Remarks**

The setting of the user will affect statements following the call to the `RScr.SetCurrentScriptUser` procedure up to the end of the current script or a subsequent call to this procedure or the `RScr.ResetCurrentScriptUser` procedure.

**Example**

```plaintext
Dim UserName
' Get a user name from an internal tag.
UserName = RTag.GetTagValue("System", "UserName")
' Set the user specified by UserName to the current script.
RScr.SetCurrentScriptUser UserName
' Set the value of the tag Setpoint from the device PLC1.
' This statement will be performed on behalf of the user specified by UserName.
RTag.SetTagValue "PLC1", "Setpoint", 20
' Reset the user for the current script,
' i.e. return to the state that existed before calling the RScr.SetCurrentScriptUser procedure.
RScr.ResetCurrentScriptUser
```
3.14.13 Basic Events

Basic events include the following events:

- Clicking or double-clicking a window
- Loading, activating, deactivating, closing, and freeing a window
- Starting, interrupting, and restoring communication to a device (e.g., PLC)
- Starting and terminating communication on a network connection between instances of the runtime software
- Receiving data for a data table from a data server

When a script is triggered by a basic event, an integer parameter configured for the event is passed to the script. In addition, the complete name of the object (window, device, server connection, data table) that triggered the event is passed to the script.

The meaning of the properties of the object returned by the `RScr.GetCurrentScriptDataEx` method:

<table>
<thead>
<tr>
<th>Property</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SenderName</td>
<td>The complete name of the object that triggered the event.</td>
</tr>
<tr>
<td>UserName</td>
<td>The name of the logged-on user.</td>
</tr>
<tr>
<td>WebServerSessionName</td>
<td>The unique identifier of the Web server session (only if the script is run in the context of a thin client).</td>
</tr>
<tr>
<td>IntPar1</td>
<td>The integer parameter passed to the script.</td>
</tr>
</tbody>
</table>

3.14.14 Events Triggered by a Component

When a script is triggered by clicking or double-clicking a component, information on the component is passed to the script.

The meaning of the properties of the object returned by the `RScr.GetCurrentScriptDataEx` method:
<table>
<thead>
<tr>
<th>Property</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SenderName</strong></td>
<td>The complete name of the component, i.e., window name/component name (e.g., Window1/Button1).</td>
</tr>
<tr>
<td><strong>UserName</strong></td>
<td>The name of the logged-on user.</td>
</tr>
<tr>
<td><strong>WebServerSessionName</strong></td>
<td>The unique identifier of the Web server session (only if the script is run in the context of a thin client).</td>
</tr>
<tr>
<td><strong>StrPar1</strong></td>
<td>The name of the component.</td>
</tr>
<tr>
<td><strong>StrPar2</strong></td>
<td>The name of the window containing the component.</td>
</tr>
<tr>
<td><strong>StrPar3</strong></td>
<td>The complete name of the tag (i.e., device name/tag name) referenced by the container through which the component is inserted into a window as part of a window template.</td>
</tr>
<tr>
<td><strong>StrPar4</strong></td>
<td>The complete name of the component's main tag (i.e., device name/tag name).</td>
</tr>
<tr>
<td><strong>StrPar5</strong></td>
<td>The unique identifier of the Web server session (only if the script is run in the context of a thin client). In new projects, use the WebServerSessionName property.</td>
</tr>
<tr>
<td><strong>IntPar1</strong></td>
<td>User-defined parameter (integer parameter as with basic events).</td>
</tr>
<tr>
<td><strong>IntPar2</strong></td>
<td>The ID of the window containing the component.</td>
</tr>
</tbody>
</table>
### IntPar3

The ID of the tag referenced by the container through which the component is inserted into a window as part of a window template.

### IntPar4

The ID of the component's main tag.

**Remarks**

The separator of the window and component name in the `SenderName` property depends on a setting in the *Project Options* dialog (*Project > Objects*). The same applies to the separator used in the `StrPar3` and `StrPar4` properties.

### 3.14.15 Events Triggered by a Change in Tag Value, Quality, or Time Stamp

When a script is triggered by a change in the value, quality, or time stamp of a tag, information on the tag and the change is passed to the script.

The meaning of the properties of the object returned by the `RScr.GetCurrentScriptDataEx` method:

<table>
<thead>
<tr>
<th>Property</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SenderName</code></td>
<td>The complete name of the tag, i.e., device name/tag name (e.g., <em>Modbus1/GasPressure</em>).</td>
</tr>
<tr>
<td><code>UserName</code></td>
<td>The name of the logged-on user.</td>
</tr>
<tr>
<td><code>WebServerSessionName</code></td>
<td>The unique identifier of the Web server session (only if the script is run in the context of a thin client).</td>
</tr>
<tr>
<td><code>StrPar2</code></td>
<td>The name of the device to which the tag belongs.</td>
</tr>
<tr>
<td><code>StrPar3</code></td>
<td>The name of the tag.</td>
</tr>
<tr>
<td><code>StrPar4</code></td>
<td>The alias of the device to which the tag belongs.</td>
</tr>
<tr>
<td>Property</td>
<td>Meaning</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>StrPar5</td>
<td>The alias of the tag.</td>
</tr>
<tr>
<td>StrPar6</td>
<td>The engineering name of the tag.</td>
</tr>
<tr>
<td>StrPar7</td>
<td>The external ID of the device to which the tag belongs.</td>
</tr>
<tr>
<td>StrPar8</td>
<td>The external ID of the tag.</td>
</tr>
<tr>
<td>IntPar2</td>
<td>The ID of the device to which the alarm belongs.</td>
</tr>
<tr>
<td>IntPar3</td>
<td>The ID of the tag.</td>
</tr>
<tr>
<td>BytePar3</td>
<td>Change kind: 1 = value change, 2 = quality change, 4 = time stamp change.</td>
</tr>
<tr>
<td>VariantPar1</td>
<td>Prior value. It is not available for array-type tags (the property returns a value of Empty).</td>
</tr>
<tr>
<td>VariantPar2</td>
<td>New value. It is not available for array-type tags (the property returns a value of Empty).</td>
</tr>
<tr>
<td>BoolPar1</td>
<td>Prior quality: <strong>True</strong> = good (valid), <strong>False</strong> = bad (invalid).</td>
</tr>
<tr>
<td>BoolPar2</td>
<td>New quality: <strong>True</strong> = good (valid), <strong>False</strong> = bad (invalid).</td>
</tr>
<tr>
<td>DatePar1</td>
<td>Prior time stamp (in UTC).</td>
</tr>
<tr>
<td>DatePar2</td>
<td>New time stamp (in UTC).</td>
</tr>
</tbody>
</table>

**Remarks**

The separator of the device and tag name in the *SenderName* property depends on a setting in the *Project Options* dialog (*Project > Objects*).
The events are triggered in the following order: quality change, time stamp change, value change.

### 3.14.16 Events Triggered by an Alarm

When a script is triggered by an alarm (i.e., start, end, acknowledgment, or notification), information on the alarm is passed to the script.

The meaning of the properties of the object returned by the `RScr.GetCurrentScriptDataEx` method:

<table>
<thead>
<tr>
<th>Property</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SenderName</td>
<td>The complete name of the alarm, i.e., device name/alarm name (e.g., Modbus1/MotorFault)</td>
</tr>
<tr>
<td>UserName</td>
<td>The name of the logged-on user.</td>
</tr>
<tr>
<td>StrPar1</td>
<td>Date and time (in local time) when this instance of the alarm was generated followed by the text of the alarm.</td>
</tr>
<tr>
<td>StrPar2</td>
<td>The name of the device to which the alarm belongs.</td>
</tr>
<tr>
<td>StrPar3</td>
<td>The name of the tag related to the alarm.</td>
</tr>
<tr>
<td>StrPar4</td>
<td>The name of the alarm.</td>
</tr>
<tr>
<td>StrPar5</td>
<td>The alias of the device to which the alarm belongs.</td>
</tr>
<tr>
<td>StrPar6</td>
<td>The alias of the tag related to the alarm.</td>
</tr>
<tr>
<td>StrPar7</td>
<td>The alias of the alarm.</td>
</tr>
<tr>
<td>StrPar8</td>
<td>The text of the alarm.</td>
</tr>
<tr>
<td>StrPar9</td>
<td>The text of the message for alarm notification. It is only available if the script was triggered as part of alarm notifications.</td>
</tr>
<tr>
<td>IntPar1</td>
<td>The ID of the alarm.</td>
</tr>
<tr>
<td>IntPar2</td>
<td>The ID of the device to which the alarm belongs.</td>
</tr>
<tr>
<td>IntPar3</td>
<td>The ID of the tag related to the alarm.</td>
</tr>
<tr>
<td>IntPar4</td>
<td>The access rights required for acknowledging the alarm.</td>
</tr>
<tr>
<td>IntPar5</td>
<td>The ID of the alarm type.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>IntPar6</td>
<td>The ID of the user that should receive alarm notification. It is only available if the script was triggered as part of alarm notifications.</td>
</tr>
<tr>
<td>IntPar7</td>
<td>The alarm occurrence count for this instance.</td>
</tr>
<tr>
<td>BytePar1</td>
<td>The ID of the alarm type. This property is only provided to maintain backward compatibility. Use the IntPar5 property instead.</td>
</tr>
<tr>
<td>BytePar2</td>
<td>The condition that generated the alarm (Alarm Triggering Condition Constants).</td>
</tr>
<tr>
<td>BytePar3</td>
<td>The priority of the alarm.</td>
</tr>
<tr>
<td>BoolPar1</td>
<td>Determines whether to display the alarm in the list of current alarms.</td>
</tr>
<tr>
<td>BoolPar2</td>
<td>Determines whether to log the alarm to the alarm database.</td>
</tr>
<tr>
<td>BoolPar3</td>
<td>Determines whether it is required that the alarm be acknowledged by the user (operator).</td>
</tr>
<tr>
<td>BoolPar4</td>
<td>Determines whether this instance of the alarm is active.</td>
</tr>
<tr>
<td>BoolPar5</td>
<td>Determines whether this instance of the alarm has been acknowledged.</td>
</tr>
<tr>
<td>DatePar1</td>
<td>Date and time (in UTC) when this instance of the alarm was generated.</td>
</tr>
<tr>
<td>DatePar2</td>
<td>Date and time (in UTC) when the alarm last occurred for this instance.</td>
</tr>
<tr>
<td>DatePar3</td>
<td>Date and time (in UTC) when this instance of the alarm ended.</td>
</tr>
<tr>
<td>DatePar4</td>
<td>Date and time (in UTC) when this instance of the alarm was acknowledged.</td>
</tr>
</tbody>
</table>

**Remarks**

The separator of the device and tag name in the SenderName property depends on a setting in the Project Options dialog (Project > Objects).
3.14.17 Events Triggered by a Text Message

When a script is triggered by receiving a text message, information on the message is passed to the script.

The meaning of the properties of the object returned by the RScr.GetCurrentScriptDataEx method:

**When receiving the message**

<table>
<thead>
<tr>
<th>Property</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SenderName</td>
<td>The name of the logical computer (based on the project) on which the project is running.</td>
</tr>
<tr>
<td>UserName</td>
<td>The name of the logged-on user.</td>
</tr>
<tr>
<td>StrPar1</td>
<td>Data in the PDU format (before decoding).</td>
</tr>
<tr>
<td>StrPar2</td>
<td>The text of the message.</td>
</tr>
<tr>
<td>StrPar3</td>
<td>The phone number of the sender.</td>
</tr>
<tr>
<td>StrPar4</td>
<td>The time (in text form) assigned by the SMS service center.</td>
</tr>
<tr>
<td>DatePar1</td>
<td>The time of receiving the message by the GSM SMS driver.</td>
</tr>
</tbody>
</table>

**When sending the message successfully**

<table>
<thead>
<tr>
<th>Property</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SenderName</td>
<td>The name of the logical computer (based on the project) on which the project is running.</td>
</tr>
<tr>
<td>UserName</td>
<td>The name of the logged-on user.</td>
</tr>
<tr>
<td>IntPar2</td>
<td>A unique identifier of the message.</td>
</tr>
<tr>
<td>Property</td>
<td>Meaning</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>StrPar2</td>
<td>The text of the message.</td>
</tr>
<tr>
<td>StrPar3</td>
<td>The phone number of the recipient.</td>
</tr>
</tbody>
</table>

When an error occurs while sending the message

<table>
<thead>
<tr>
<th>Property</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SenderName</td>
<td>The name of the logical computer (based on the project) on which the project is running.</td>
</tr>
<tr>
<td>UserName</td>
<td>The name of the logged-on user.</td>
</tr>
<tr>
<td>IntPar2</td>
<td>A unique identifier of the message.</td>
</tr>
<tr>
<td>IntPar3</td>
<td>The code of an error that might occur while sending the message (The List of Error Codes (CMS) According to the GSM 07.05 Standard).</td>
</tr>
<tr>
<td>StrPar2</td>
<td>The text of the message.</td>
</tr>
<tr>
<td>StrPar3</td>
<td>The phone number of the recipient.</td>
</tr>
</tbody>
</table>

### 3.14.18 Events Triggered by a Thin Client Request

When a script is triggered by a thin client request, information on the request is passed to the script.

The meaning of the properties of the object returned by the `RScr.GetCurrentScriptDataEx` method:

<table>
<thead>
<tr>
<th>Property</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SenderName</td>
<td>The name of the logical computer (based on the project) on which the project is running.</td>
</tr>
</tbody>
</table>
### UserName
The name of the logged-on user.

### WebServerSessionName
The unique identifier of the Web server session.

### StrPar1
The unique identifier of the Web server session. In new projects, use the `WebServerSessionName` property.

### StrPar2
Client IP address.

### StrPar3
Client version.

### StrPar4
Computer (configuration) name.

### StrPar5
The name of the logged-on user. In new projects, use the `UserName` property.

### StrPar6
Information on Web browser (User-Agent header). Only for Reliance Smart Client.

### IntPar1
Request type: Timeout = 0, Connect = 1, Disconnect = 2, User Log-on = 3, User Log-off = 4

### IntPar2
Thin client type: Reliance Web Client = 0, Reliance Smart Client = 2

### IntPar3
Computer (configuration) ID.

### IntPar4
User ID.

### 3.14.19 Events Triggered by the Maatrix Service

When a script is triggered by terminating the Maatrix service, information on the termination is passed to the script.

The meaning of the properties of the object returned by the `RScr.GetCurrentScriptDataEx` method:
<table>
<thead>
<tr>
<th>Property</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SenderName</strong></td>
<td>The name of the logical computer (based on the project) on which the project is running.</td>
</tr>
<tr>
<td><strong>UserName</strong></td>
<td>The name of the logged-on user.</td>
</tr>
<tr>
<td><strong>StrPar1</strong></td>
<td>The text of the alarm/event that launched Maatrix.</td>
</tr>
<tr>
<td><strong>StrPar2</strong></td>
<td>The ID (unique identifier) of Maatrix.</td>
</tr>
<tr>
<td><strong>StrPar3</strong></td>
<td>The status of Maatrix: 'closed' = the service was properly terminated, 'stopped' = the service was interrupted or failed, 'open' = the service is running, 'null' = the status is not available</td>
</tr>
<tr>
<td><strong>StrPar4</strong></td>
<td>A list of the names (email addresses) of the users in Maatrix who confirmed that they would handle the alarm. The email addresses are separated by commas.</td>
</tr>
<tr>
<td><strong>IntPar1</strong></td>
<td>The ID (unique identifier) of the alarm/event that launched Maatrix.</td>
</tr>
<tr>
<td><strong>BoolPar1</strong></td>
<td>The result of the service (whether there is a person who will handle the alarm).</td>
</tr>
<tr>
<td><strong>DatePar1</strong></td>
<td>The time of the start of the alarm/event that launched Maatrix.</td>
</tr>
</tbody>
</table>

**3.14.20 Events Triggered by an Embedded Web Browser**

When a script is triggered by the `external.ExecScript` function from a page in an embedded Web browser, information on the component (Internet Explorer) and the `external.ExecScript` function parameters are passed to the script.
The meaning of the properties of the object returned by the `RScr.GetCurrentScriptDataEx` method:

<table>
<thead>
<tr>
<th>Property</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SenderName</td>
<td>The complete name of the component, i.e., window name/component name (e.g., <code>Window1/InternetExplorer1</code>).</td>
</tr>
<tr>
<td>UserName</td>
<td>The name of the logged-on user.</td>
</tr>
<tr>
<td>StrPar1</td>
<td>The name of the component.</td>
</tr>
<tr>
<td>StrPar2</td>
<td>The name of the window containing the component.</td>
</tr>
<tr>
<td>StrPar3</td>
<td>The complete name of the tag (i.e., device name/tag name) referenced by the container through which the component is inserted into a window as part of a window template.</td>
</tr>
<tr>
<td>StrPar4</td>
<td>The complete name of the component's main tag (i.e., device name/tag name).</td>
</tr>
<tr>
<td>IntPar1</td>
<td>see <code>CustomIntPar1</code></td>
</tr>
<tr>
<td>IntPar2</td>
<td>The ID of the window containing the component.</td>
</tr>
<tr>
<td>IntPar3</td>
<td>The ID of the tag referenced by the container through which the component is inserted into a window as part of a window.</td>
</tr>
<tr>
<td>Property</td>
<td>Meaning</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>template.</td>
</tr>
<tr>
<td>IntPar4</td>
<td>The ID of the component's main tag.</td>
</tr>
<tr>
<td>CustomIntPar1</td>
<td>A parameter (integer) passed to the external.ExecScript function on the embedded Web page.</td>
</tr>
<tr>
<td>CustomStrPar1</td>
<td>A parameter (string) passed to the external.ExecScript function on the embedded Web page.</td>
</tr>
</tbody>
</table>

**Remarks**

The separator of the window and component name in the `SenderName` property depends on a setting in the `Project Options` dialog (`Project > Objects`). The same applies to the separator used in the `StrPar3` and `StrPar4` properties.
3.15 RSys Object

The RSys object implements methods that can be divided into two groups:

- Methods for miscellaneous runtime environment operations that are not related to any of the other Reliance-defined objects (e.g., RSys.ActivateWindow, RSys.CloseWindow, RSys.SetMainWindowTitle).

- Methods for miscellaneous operations related to the operating system (e.g., RSys.Now, RSys.SetLocalTime) and its objects, such as directories and files (e.g., RSys.DirExists, RSys.FileExists).

Methods:
- RSys.ActivateWindow Procedure
- RSys.CloseWindow Procedure
- RSys.ConvertTimeToDST Function
- RSys.CopyFile Function
- RSys.CreateDir Function
- RSys.DateToTimeZone Function
- RSys.DeleteFile Function
- RSys.DirExists Function
- RSys>EditCustomReport Procedure
- RSys.ExecAction Procedure
- RSys.ExecActionAndWait Function
- RSys.ExecApp Procedure
- RSys.ExecAppAndWait Function
- RSys.ExitRuntimeModule Procedure
- RSys.FileExists Function
- RSys.GetBit Function
- RSys.GetByte Function
- RSys.GetWord Function
- RSys.GetComputerName Function
RSys.GetProgramLanguage Function
RSys.GetProjectDir Function
RSys.GetProjectLanguage Function
RSys.GetUTCDateTime Function
RSys.Int64TimeToDateTime Function
RSys.IsActiveServer Function
RSys.IsPrimaryServer Function
RSys.IsSecondaryServer Function
RSys.LocalDateTimeToUTCDatetime Function
RSys.LogMessage Procedure
RSys.Now Function
RSys.PlaySound Procedure
RSys.PrintCustomReport Procedure
RSys.RelativePathToPath Function
RSys.PathToRelativePath Function
RSys.PrintDbReport Procedure
RSys.PrintDbTrend Procedure
RSys.PrintTagDbTrend Procedure
RSys.RemoveDir Function
RSys.RenameFile Function
RSys.ReplaceCZChars Function
RSys.RestartProject Procedure
RSys.RestartWindows Procedure
RSys.SaveCustomReport Procedure
RSys.SetBit Procedure
RSys.SetByte Procedure
RSys.SetWord Procedure
RSys.SetLocalTime Function
RSys.SetMainWindowTitle Procedure
RSys.SetMonitorPower Procedure
RSys.ShiftLeft Function
RSys.ShiftRight Function
RSys.ShowCustomReport Procedure
RSys.ShowDbReport Procedure
RSys.ShowDbTrend Procedure
RSys.ShowTagDbReport Procedure
RSys.ShowTagDbTrend Procedure
RSys.ShutDownWindows Procedure
RSys.SetProgramLanguage Procedure
RSys.SetProjectLanguage Procedure
RSys.Sleep Procedure
RSys.UTCDateTimeToLocalDateTime Function

3.15.1 RSys.ActivateWindow Procedure

RSys.ActivateWindow activates a project window.

Syntax

RSys.ActivateWindow Window: Variant

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Window</td>
<td>The name or ID of the window.</td>
</tr>
</tbody>
</table>

RSys Object

Example

RSys.ActivateWindow "MainWindow" ' Activate the window MainWindow.
### 3.15.2 RSys.CloseWindow Procedure

RSys.CloseWindow closes a project window.

**Syntax**

```plaintext
RSys.CloseWindow Window: Variant
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Window</td>
<td>The name or ID of the window.</td>
</tr>
</tbody>
</table>

**Example**

```plaintext
RSys.CloseWindow "Settings" ' Close the window Settings.
```

### 3.15.3 RSys.ConvertTimeToDST Function

RSys.ConvertTimeToDST converts a specified date and time value to daylight saving time.

**Syntax**

```plaintext
RSys.ConvertTimeToDST(Value: DateTime): DateTime
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>The date and time value to be converted.</td>
</tr>
</tbody>
</table>

**Return values**

The method returns a date and time value converted to daylight saving time.
Remarks
This method can be useful when processing data time-stamped in standard time. In order for the method to work, the start and end time of the daylight saving time period must be configured through the TimeConv.ini file located in a visualization project's root directory. If the specified time stamp is within the period, the return value is equal to the time stamp plus one hour. If the specified time stamp is not within the period or the period has not been specified, the return value is equal to the time stamp.

Example

```
Dim DeviceTime, ComputerTime
' Store the current date and time retrieved from PLC1 in the variable DeviceTime.
DeviceTime = RTag.GetTagValue("PLC1", "Time")
' Convert DeviceTime to daylight saving time and store the result in the variable ComputerTime.
ComputerTime = RSys.ConvertTimeToDST(DeviceTime)
' Change the current date and time to the value stored in the variable ComputerTime.
If RSys.SetLocalTime(ComputerTime) Then
  ' ...
End If
```

3.15.4 RSys.CopyFile Function

RSys.CopyFile copies a source file to a destination file.

Syntax

```
RSys.CopyFile(SourceFile, DestFile: String; FailIfExists: Boolean): Boolean
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SourceFile</td>
<td>The full name of the source file.</td>
</tr>
<tr>
<td>DestFile</td>
<td>The full name of the destination file.</td>
</tr>
<tr>
<td>FailIfExists</td>
<td>Determines how this operation is to proceed if a file of the same name as that specified by DestFile already exists (see the following table).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The method does nothing and fails.</td>
</tr>
<tr>
<td>False</td>
<td>The method tries to overwrite the destination file.</td>
</tr>
</tbody>
</table>

### Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The file was copied.</td>
</tr>
<tr>
<td>False</td>
<td>The file was not copied.</td>
</tr>
</tbody>
</table>

### Remarks

The return value is **False** in these cases:

- The source file does not exist.
- The names of the source and destination files are identical.
- The name of the destination file is not valid, since it does not follow the rules for naming files.
- The destination file already exists and *FailIfExists* = **True**.
- The destination file already exists, *FailIfExists* = **False**, but the runtime software was denied access to the destination file.
- There is not sufficient space available on the destination drive to copy the file.

### Example

```
' If the file "C:\Data\Data.txt" exists.
If RSys.FileExists("C:\Data\Data.txt") Then
  ' Copy the file to the directory "C:\Backup"
  ' overwriting any file that might exist with the same name.
  If RSys.CopyFile("C:\Data\Data.txt", "C:\Backup\Data.txt", False) Then
    ' ...
  End If
' Delete the file "C:\Data\Data.001".
```
RSys.DeleteFile "C:\Data\Data.001"
' Rename the file "C:\Data\Data.txt" to "C:\Data\Data.001".
If RSys.RenameFile("C:\Data\Data.txt", "C:\Data\Data.001") Then
' ...
End If
End If

3.15.5 RSys.CreateDir Function

RSys.CreateDir creates a specified directory.

Syntax

RSys.CreateDir(DirName: String): Boolean

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DirName</td>
<td>The full path to the directory.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The directory has been created.</td>
</tr>
<tr>
<td>False</td>
<td>The directory has not been created.</td>
</tr>
</tbody>
</table>

Remarks

By a single call to this method, you can create an entire directory structure. A trailing backslash at the end of the specified path is accepted, but not required.

The return value is False in these cases:

- The specified path is not valid, since it does not follow the rules for naming directories.
- The runtime software was denied access when trying to create the directory.
If the directory "C:\Data" does not exist.
If not RSys.DirExists("C:\Data") Then
  If RSys.CreateDir("C:\Data") Then ' Create the directory.
    ...
  End If
End If

3.15.6 RSys.DateTimeToInt64Time Function

RSys.DateTimeToInt64Time converts a specified date and time value from the DateTime format to the Int64Time format (used by Reliance as the native date and time format).

Date and time in the Int64Time format is a value stored as Int64, i.e., a 64-bit integer value (in Reliance projects, the Int64 type is represented by the LargeInt type). The Int64Time format is based on the FILETIME format (well-known from the Windows API) which is used by file systems to store file timestamps. It represents the number of 100-nanosecond intervals since January 1, 1601. Reliance uses the Int64Time format, for example, to store the timestamps of historical data (if it is logged to an SQL-based database) and alarm/event timestamps.

Syntax

\[
\text{RSys.DateTimeToInt64Time}(\text{Value}: \text{DateTime}): \text{Int64}
\]

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>The date and time value to be converted.</td>
</tr>
</tbody>
</table>

Return values

The method returns a date and time value in the Int64Time format.

Example

' Get the value of the tag DateTimeValue from the device System
DateTimeValue = RTag.GetTagValue("System", "DateTimeValue")
' Convert the date and time value to the Int64Time format.
Int64TimeValue = RSys.DateTimeToInt64Time(DateTimeValue)
' Store the result in the tag Int64TimeValue from the device System
RTag.SetTagValue "System", "Int64TimeValue", Int64TimeValue

3.15.7 RSys.DeleteFile Function


Syntax

RSys.DeleteFile(FileName: String): Boolean

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileName</td>
<td>The full name of the file.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The file has been deleted.</td>
</tr>
<tr>
<td>False</td>
<td>The file has not been deleted.</td>
</tr>
</tbody>
</table>

Remarks

The return value is False in these cases:
- The file does not exist.
- The runtime software was denied access to the file.

Example

' If the file "C:\Data\Data.txt" exists.
If RSys.FileExists("C:\Data\Data.txt") Then
  ' Copy the file to the directory "C:\Backup"
  ' overwriting any file that might exist with the same name.
  If RSys.CopyFile("C:\Data\Data.txt", "C:\Backup\Data.txt", False) Then
End If
'
Delete the file "C:\Data\Data.001".
RSys.DeleteFile "C:\Data\Data.001"
'
Renaming the file "C:\Data\Data.txt" to "C:\Data\Data.001".
If RSys.RenameFile("C:\Data\Data.txt", "C:\Data\Data.001") Then
'
End If
End If

3.15.8 RSys.DirExists Function

RSys.DirExists determines whether a specified directory exists.

Syntax

RSys.DirExists(DirName: String): Boolean

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DirName</td>
<td>The full path to the directory.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The directory exists.</td>
</tr>
<tr>
<td>False</td>
<td>The directory does not exist.</td>
</tr>
</tbody>
</table>

Remarks

A trailing backslash at the end of the specified path is accepted, but not required.

Example

' If the directory "C:\Data" does not exist.
If not RSys.DirExists("C:\Data") Then
  If RSys.CreateDir("C:\Data") Then ' Create the directory.
3.15.9 **RSys.EditCustomReport Procedure**

**RSys.EditCustomReport** displays the *FastReport Designer*, which allows editing a specified **Reliance**-defined custom report’s template.

**Syntax**

```plaintext
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report</td>
<td>The name or ID (unique identifier) of the custom report.</td>
</tr>
</tbody>
</table>

**Remarks**

You can only edit custom reports of type FastReport.

**Example**

```plaintext
' Open the custom report "Report1" in the FastReport Designer.
RSys. EditCustomReport "Report1"
```
3.15.10 RSys.ExecAction Procedure

RSys.ExecAction places a specified action in a queue of actions to be executed. The procedure does not wait until the action is executed. It is executed later in the user interface thread (the program's primary thread).

Syntax

```
RSys.ExecAction Action: Variant
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>The name or ID (unique identifier) of the action.</td>
</tr>
</tbody>
</table>

Example

```
' Execute the action ShowTagTrend_GasPressure. 
RSys.ExecAction "ShowTagTrend_GasPressure"
```

3.15.11 RSys.ExecActionAndWait Function

RSys.ExecActionAndWait places a specified action in a queue of actions to be executed and waits until the action is executed and completed. It is executed in the user interface thread (the program's primary thread).
Syntax

RSys.ExecActionAndWait(Action: Variant; Timeout: Integer): Boolean

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>The name or ID (unique identifier) of the action.</td>
</tr>
<tr>
<td>Timeout</td>
<td>The time (the number of milliseconds) the function needs to wait for executing and completing the action. If the action is not completed within this time period, the function returns False. If the argument’s value is 0 or a negative number, the wait timeout is unlimited.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The action was completed within the specified time.</td>
</tr>
<tr>
<td>False</td>
<td>Action execution failed. The action was not found or was not completed within the specified time.</td>
</tr>
</tbody>
</table>

Example

' Execute the action SelectFile, which saves the name of the selected file to the tag FileName from the device System.
If RSys.ExecActionAndWait("SelectFile", 0) Then
' Open the selected file by the application associated with the extension of this file.
   RSys.ExecApp RTag.GetTagValue("System", "FileName"), ""
End If
3.15.12 RSys.ExecApp Procedure

RSys.ExecApp runs a specified application or an application associated with the extension of a specified filename.

Syntax

```
RSys.ExecApp FileName, Params: String
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileName</td>
<td>The full name of the application's executable file or of the file to be opened by an application associated with the extension of the name.</td>
</tr>
<tr>
<td>Params</td>
<td>The parameters to be passed to the application.</td>
</tr>
</tbody>
</table>

Remarks

The Params argument can be an empty string if no parameters are to be passed to the application.

Example

\`
' Open the file "C:\Data\Data.txt" by the program
' "C:\WinNT\System32\Notepad.exe".
RSys.ExecApp "C:\WinNT\System32\Notepad.exe", "C:\Data\Data.txt"
' Open the file "C:\Data\Data.txt" by the program
' associated with the .txt extension (e.g., Notepad).
RSys.ExecApp "C:\Data\Data.txt", ""
' Open the file "C:\Docs\Data.doc" by the program
' associated with the .doc extension (e.g., Microsoft Word).
RSys.ExecApp "C:\Docs\Data.doc", ""
\`
3.15.13 RSys.ExecAppAndWait Function

RSys.ExecAppAndWait runs a specified application and waits until it is terminated.

Syntax

RSys.ExecAppAndWait(FileName, Params: String; ShowCmd: Integer; ByRef ExitCode: Variant): Boolean

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileName</td>
<td>The full name of the application's executable file.</td>
</tr>
<tr>
<td>Params</td>
<td>The parameters to be passed to the application.</td>
</tr>
<tr>
<td>ShowCmd</td>
<td>The way the application's main window is shown.</td>
</tr>
<tr>
<td>ExitCode</td>
<td>The code that terminates the application.</td>
</tr>
</tbody>
</table>

ShowCmd constants

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Hides the application's window.</td>
</tr>
<tr>
<td>2</td>
<td>Shows the application's window as minimized.</td>
</tr>
<tr>
<td>3</td>
<td>Shows the application's window as maximized.</td>
</tr>
<tr>
<td>5</td>
<td>Shows the application's window in its current size and position.</td>
</tr>
<tr>
<td>10</td>
<td>Shows the application's window in the size and position it had when it was created.</td>
</tr>
</tbody>
</table>
Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>Running the application was successful.</td>
</tr>
<tr>
<td>False</td>
<td>Failed to run the application.</td>
</tr>
</tbody>
</table>

Remarks

The `Params` argument can be an empty string if no parameters are to be passed to the application.

Example

```vba
Dim ExitCode

' Open the file "C:\Data\Data.txt" by the program
' "C:\Windows\System32\notepad.exe".
RSys.ExecAppAndWait "C:\Windows\System32\notepad.exe", "C:\Data\Data.txt", 3, ExitCode
```

3.15.14 RSys.ExitRuntimeModule Procedure

`RSys.ExitRuntimeModule` terminates the runtime software.

Syntax

```vba
RSys.ExitRuntimeModule
```

Remarks

The method does not check if the user currently logged on to the runtime software (if any) has sufficient `Access rights` for terminating a visualization project. If it is desired to check the access rights, use methods of the `RUser` object.
Example

Dim UserName
' If there is a user logged on to the runtime software.
If RUser.GetLoggedOnUserName(UserName) Then
' If the user has the Servicing access right.
If RUser.CheckUserAccessRights(UserName, "S") Then
    RSys.ShutDownWindows  ' Shut down the operating system.
Else
    RSys.ExitRuntimeModule  ' Otherwise terminate the runtime software.
End If
End If

3.15.15  RSys.FileExists Function

RSys.FileExists determines whether a specified file exists.

Syntax

RSys.FileExists(FileName: String): Boolean

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileName</td>
<td>The full name of the file.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The file exists.</td>
</tr>
<tr>
<td>False</td>
<td>The file does not exist.</td>
</tr>
</tbody>
</table>

Example

' If the file "C:\Data\Data.txt" exists.
If RSys.FileExists("C:\Data\Data.txt") Then
    ' Copy the file to the directory "C:\Backup"
    ' overwriting any file that might exist with the same name.
If RSys.CopyFile("C:\Data\Data.txt", "C:\Backup\Data.txt", False) Then
  ...
End If
' Delete the file "C:\Data\Data.001".
RSys.DeleteFile "C:\Data\Data.001"
' Rename the file "C:\Data\Data.txt" to "C:\Data\Data.001".
If RSys.RenameFile("C:\Data\Data.txt", "C:\Data\Data.001") Then
  ...
End If
End If

3.15.16 RSys.GetBit Function

RSys.GetBit returns the current value of the selected bit of the specified value.

Syntax

RSys.GetBit(\texttt{Value: Variant, BitNumber: Integer}): Variant

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>The value.</td>
</tr>
<tr>
<td>BitNumber</td>
<td>The bit number.</td>
</tr>
</tbody>
</table>

Return values

The current value of the selected bit of the specified value.

Remarks

The specified value must be of a simple integer type (Byte, Integer, etc.).

\[ \text{RSys} \] Object

Example

\texttt{Dim Value}
' Get the value of the bit number 0 from the value 6000.
Value = RSys.GetBit(6000, 0)
3.15.17 RSys.GetByte Function

RSys.GetByte returns the current value of the selected byte of the specified value.

Syntax

`RSys.GetByte(Value: Variant, ByteNumber: Integer): Variant`

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>The value.</td>
</tr>
<tr>
<td>ByteNumber</td>
<td>The byte number.</td>
</tr>
</tbody>
</table>

Return values

The current value of the selected byte of the specified value.

Remarks

The specified value must be of a simple integer type (Integer, Long, etc.).

Example

```vbnet
Dim Value
' Get the value of the byte number 0 from the value 6000.
Value = RSys.GetByte(6000, 0)
```
3.15.18 RSys.GetWord Function

RSys.GetWord returns the current value of the selected word of the specified value.

Syntax

```
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>The value.</td>
</tr>
<tr>
<td>WordNumber</td>
<td>The word number.</td>
</tr>
</tbody>
</table>

Return values

The current value of the selected word of the specified value.

Remarks

The specified value must be of a simple integer type (Long).

Example

```
Dim Value
' Get the value of the word number 0 from the value 600000.
Value = RSys.GetWord(600000, 0)
```
3.15.19  RSys.GetComputerName Function

RSys.GetComputerName returns the name of a logical computer, defined in a visualization project, on which the project is running.

Syntax

RSys.GetComputerName: String

Return values

The method returns the name of a logical computer, defined in a visualization project, on which the project is running.

Example

' If a visualization project is running on the computer PC1.
If RSys.GetComputerName = "PC1" Then
   ' Disable the script Script1.
   RScr.DisableScript "Script1"
End If

3.15.20  RSys.GetProgramLanguage Function

RSys.GetProgramLanguage returns the active language of the program (runtime software).

Syntax

RSys.GetProgramLanguage: String

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSY</td>
<td>Czech</td>
</tr>
<tr>
<td>ENU</td>
<td>English</td>
</tr>
</tbody>
</table>
Example

' Store the active language of the program in the tag ProgramLanguage from the device System

3.15.21  RSys.GetProjectDir Function

**RSys.GetProjectDir** returns the full path to a directory where the current visualization project is located.

**Syntax**

```plaintext
RSys.GetProjectDir: String
```

**Return values**

The method returns the full path to the directory where the current visualization project is located.

**Remarks**

The returned path always contains a trailing backslash.

```plaintext
Dim PrjDir
PrjDir = RSys.GetProjectDir
' Run the application "Reports.exe" located
' in a visualization project's Apps directory.
RSys.ExecApp PrjDir + "Main\Apps\Reports.exe", ""
```

3.15.22  RSys.GetProjectLanguage Function

**RSys.GetProjectLanguage** returns the active language of the project.

**Syntax**

```plaintext
RSys.GetProjectLanguage: String
```

**Return values**

The name of the project's active language.
Example

' Store the name of the project's active language in the tag ProjectLanguage from the device System.

3.15.23 RSys.GetUTCDatetime Function

RSys.GetUTCDatetime returns the current system date and time. The returned value is expressed in UTC (as opposed to the Now and RSys.Now functions which return a value expressed in local time).

UTC stands for Coordinated Universal Time. It is the primary time standard by which the world regulates clocks and time. The UTC time is based on atomic clocks and is very close to Greenwich Mean Time (GMT). Time zones around the world are expressed as positive or negative offsets from UTC.

The local time is dependent on the operating system settings (the time zone, automatically adjusting to daylight saving time).

Syntax

RSys.GetUTCDatetime: DateTime

Return values

The method returns a date and time value in the DateTime format.

Remarks

By using this function, you can avoid the inconsistency of time information during changeovers between standard and daylight saving time.

Example

Dim UTCDatetimeValue
' Get UTC date and time.
UTCDatetimeValue = RSys.GetUTCDatetime
' Store the result in the tag UTCDatetimeValue from the device System
RTag.SetTagValue "System", "UTCDatetimeValue", UTCDatetimeValue
3.15.24 RSys.Int64TimeToDateTime Function

**RSys.Int64TimeToDateTime** converts a specified date and time value from the *Int64Time* format (used by **Reliance** as the native date and time format) to the *DateTime* format.

Date and time in the *Int64Time* format is a value stored as *Int64*, i.e., a 64-bit integer value (in **Reliance** projects, the *Int64* type is represented by the *LargeInt* type). The *Int64Time* format is based on the *FILETIME* format (well-known from the Windows API) which is used by file systems to store file timestamps. It represents the number of 100-nanosecond intervals since January 1, 1601. **Reliance** uses the *Int64Time* format, for example, to store the timestamps of historical data (if it is logged to an SQL-based database) and alarm/event timestamps.

### Syntax

```
RSys.Int64TimeToDateTime(Value: Int64): DateTime
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>The date and time value to be converted.</td>
</tr>
</tbody>
</table>

### Return values

The method returns a date and time value in the *DateTime* format.

### Remarks

This method can be useful e.g., when processing historical data logged by **Reliance** into a table in Microsoft SQL Server. For example, you could export certain records from the table to a file in the CSV format.

---

**Example**

```vbnet
Dim DateTimeValue, Int64TimeValue
' Get the value of the tag Int64TimeValue from the device System
Int64TimeValue = RTag.GetTagValue("System", "Int64TimeValue")
' Convert the date and time value to the DateTime format.
DateTimeValue = RSys.Int64TimeToDateTime(Int64TimeValue)
' Store the result in the tag DateTimeValue from the device System.
RTag.SetTagValue("System", "DateTimeValue", DateTimeValue)
```
3.15.25 RSys.IsActiveServer

RSys.IsActiveServer determines whether the computer on which the project is running currently has the active role (whether it is the active server - when using data server redundancy). Only the server that has the active role provides the following functions: communication to devices (e.g., PLCs), historical data acquisition, generating alarms/events, sending alarm/event information via E-mail and/or SMS (text messages).

Syntax

RSys.IsActiveServer: Boolean

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The computer (server) has the active role.</td>
</tr>
<tr>
<td>False</td>
<td>The computer (server) does not have the active role.</td>
</tr>
</tbody>
</table>

Remarks

The function is intended for projects in which data server redundancy is used. The function could be called in scripts designed for the primary and secondary servers that are configured to run at project startup (or on thread initialization) and when the server role changes (active/standby). If the function is called on another computer, it always returns True. The result of the function can be stored in a tag in the device System (it cannot be in any other device).
Example

' Store the result in the tag IsActiveServer from the device System
RTag.SetTagValue "System", "IsActiveServer", RSys.IsActiveServer

3.15.26 RSys.IsPrimaryServer

RSys.IsPrimaryServer determines whether the computer on which the project is running is configured in the project as a primary server (when using data server redundancy).

Syntax

RSys.IsPrimaryServer: Boolean

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The computer is configured as a primary server.</td>
</tr>
<tr>
<td>False</td>
<td>The computer is not configured as a primary server.</td>
</tr>
</tbody>
</table>

Remarks

The function is intended for projects in which data server redundancy is used. The function could be called in scripts that are configured to run at project startup (or on thread initialization). The result of the function can be stored in a tag in the device System (it cannot be in any other device).
RSys.IsSecondaryServer determines whether the computer on which the project is running is configured in the project as a secondary (backup) server (when using data server redundancy).

Syntax

RSys.IsSecondaryServer: Boolean

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The computer is configured as a secondary server.</td>
</tr>
<tr>
<td>False</td>
<td>The computer is not configured as a secondary server.</td>
</tr>
</tbody>
</table>

Remarks

The function is intended for projects in which data server redundancy is used. The function could be called in scripts that are configured to run at project startup (or on thread initialization). The result of the function can be stored in a tag in the device System (it cannot be in any other device).

Example

' Store the result in the tag IsSecondaryServer from the device System
3.15.28 RSys.LocalDateTimeToUTCDateTime Function

**RSys.LocalDateTimeToUTCDateTime** converts a specified date and time value from the local time to UTC.

The local time is dependent on the operating system settings (the time zone, automatically adjusting to daylight saving time).

UTC stands for *Coordinated Universal Time*. It is the primary time standard by which the world regulates clocks and time. The UTC time is based on atomic clocks and is very close to Greenwich Mean Time (GMT). Time zones around the world are expressed as positive or negative offsets from UTC.

**Syntax**

```plaintext
RSys.LocalDateTimeToUTCDateTime(Value: DateTime): DateTime
```

**Argument**

| Description | Value | The date and time value to be converted. |

**Return values**

The method returns a date and time value in the *DateTime* format.

3.15.29 RSys.LogMessage Procedure

**RSys.LogMessage** logs a defined text to a log file of a runtime software.
Syntaxe

**RSys.LogMessage**  *Text: String*

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Text</strong></td>
<td>Defines a text string that should be logged to a log file.</td>
</tr>
</tbody>
</table>

Remarks

All log files are stored in the `<Reliance4>\Logs` directory. The filename is based on the name of the runtime software executable and on a date. For example, a log file of the *Reliance 4 Control Server* program that was created on 2009-12-1 will be `R_CtlSrv_2009_12_01.log`.

Example

```plaintext
  dim Counter

  Counter = RTag.GetTagValue("System", "Counter")

  RSys.LogMessage "System/Counter = " + CStr(Counter)
```

3.15.30  **RSys.Now Function**

**RSys.Now** returns the current system date and time. The returned value is expressed in local time (as opposed to the **RSys.GetUTCDateTime** function which returns a value expressed in UTC).

The local time is dependent on the operating system settings (the time zone, automatically adjusting to daylight saving time).

Syntax

**RSys.Now: DateTime**

Return values

The method returns a date and time value in the *DateTime* format.
Remarks

The return value is the current system date and time converted to the currently active
time zone's corresponding local date and time.

RSys Object

Example

Dim NewTime
' Store the current date and time incremented by 1 hour in a variable.
NewTime = RSys.Now + TimeSerial(1, 0, 0)
' Change the current date and time to the value stored in the variable.
If RSys.SetLocalTime(NewTime) Then
    ' ...
End If

3.15.31 RSys.PlaySound Procedure

RSys.PlaySound plays a sound stored in a specified file.

Syntax

RSys.PlaySound FileName: String

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileName</td>
<td>The name of the file that contains the sound to</td>
</tr>
<tr>
<td></td>
<td>be played.</td>
</tr>
</tbody>
</table>

Remarks

The file containing the sound must be located in a visualization project's MMedia
directory (the FileName argument must not contain a path) or in its subdirectory (the
FileName argument must contain a relative path to the subdirectory).

RSys Object
Example

Dim ID
' If the value of the tag WaterTemperature
' from the device PLC1 is greater than 90.
If RTag.GetTagValue("PLC1", "WaterTemperature") > 90 Then
    ' Play the sound "Beep.wav" located
    ' in a visualization project's MMedia directory.
    RSys.PlaySound "Beep.wav"
    ' Send a SMS message informing the recipient of the event.
    If RModem.GSMSendSMSEx("+420123456789", "Water temperature has exceeded the upper limit.", ID) Then
        ' ...
    End If
End If

3.15.32 RSys.PrintCustomReport Procedure

RSys.PrintCustomReport prints a custom report defined in a visualization project to the default printer.

Syntax

RSys.PrintCustomReport Report: Variant

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report</td>
<td>The name or ID of the custom report.</td>
</tr>
</tbody>
</table>

3.15.33 RSys.PrintDbReport Procedure

RSys.PrintDbReport prints a historical report defined in a visualization project to the default printer.
Syntax

RSys.PrintDbReport Report: Variant

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report</td>
<td>The name or ID of the historical report.</td>
</tr>
</tbody>
</table>

Example

' Print the historical report "Report1".
RSys.PrintDbReport "Report1"

3.15.34 RSys.PrintDbTrend Procedure

RSys.PrintDbTrend prints a historical trend defined in a visualization project to the default printer.

Syntax

RSys.PrintDbTrend Trend: Variant

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trend</td>
<td>The name or ID of the historical trend.</td>
</tr>
</tbody>
</table>

Example

' Print the historical trend "Trend1".
RSys.PrintDbTrend "Trend1"
3.15.35  RSys.PrintTagDbTrend Procedure

RSys.PrintTagDbTrend prints the historical trend of a specified tag to the default printer. The trend is not one of the trends defined through the Trend Manager. If the tag's data is logged to multiple data tables, the trend data is loaded from the table that has the shortest sampling interval.

Syntax

    RSys.PrintTagDbTrend DevName, TagName: String

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DevName</td>
<td>The name of the device that the tag belongs to.</td>
</tr>
<tr>
<td>TagName</td>
<td>The name of the tag.</td>
</tr>
</tbody>
</table>

Example

    ' Print the trend of the tag WaterTemperature from the device PLC1.
    RSys.PrintTagDbTrend "PLC1", "WaterTemperature"

3.15.36  RSys.PathToRelativePath Function

RSys.PathToRelativePath converts a specified absolute path to the corresponding relative path.

Syntax

    RSys.PathToRelativePath(Path:String) String

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path</td>
<td>Absolute path to be converted.</td>
</tr>
</tbody>
</table>
Remarks

In the context of Reliance, a *relative path* means a directory path defined using an *environment variable*, e.g., the path `${Reliance}\Utils\`. Available environment variables are listed in the following table.

**Reliance environment variables**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>${Reliance}</code>\</td>
<td><code>&lt;Reliance4&gt;</code> directory, i.e., the directory of Reliance 4 program files</td>
</tr>
<tr>
<td><code>${Components}</code>\</td>
<td><code>&lt;Reliance4&gt;\Components\</code> directory, i.e., the directory of Reliance 4 components (graphical objects)</td>
</tr>
<tr>
<td><code>${Drivers}</code>\</td>
<td><code>&lt;Reliance4&gt;\Drivers\</code> directory, i.e., the directory of Reliance 4 communication drivers</td>
</tr>
<tr>
<td><code>${Project}</code>\</td>
<td><code>&lt;Project&gt;</code> directory, i.e., the directory of the current visualization project</td>
</tr>
<tr>
<td><code>${CustomReports}</code>\</td>
<td><code>&lt;Project&gt;\Main\CustomReports\</code> directory</td>
</tr>
<tr>
<td><code>${SettingsProfiles}</code>\</td>
<td><code>&lt;Project&gt;\Settings\Profiles\</code> directory</td>
</tr>
<tr>
<td><code>${SettingsComponents}</code>\</td>
<td><code>&lt;Project&gt;\Settings\Components\</code> directory</td>
</tr>
<tr>
<td><code>${SettingsRecipes}</code>\</td>
<td><code>&lt;Project&gt;\Settings\Recipes\</code> directory</td>
</tr>
<tr>
<td><code>${HistoryAlarmsEvents}</code>\</td>
<td><code>&lt;Project&gt;\History\AlarmsEvents\</code> directory</td>
</tr>
<tr>
<td><code>${HistoryData}</code>\</td>
<td><code>&lt;Project&gt;\History\Data\</code> directory</td>
</tr>
<tr>
<td><code>${HistoryPostmort}</code>\</td>
<td><code>&lt;Project&gt;\History\Postmort\</code> directory</td>
</tr>
<tr>
<td><code>${HistoryWindowRecords}</code>| <code>&lt;Project&gt;\History\WindowRecords\</code> directory</td>
<td></td>
</tr>
<tr>
<td><code>${UserDocuments}</code>\</td>
<td><code>%USERPROFILE%\Dokumenty\</code> directory, i.e., the user data directory</td>
</tr>
<tr>
<td><code>${ApplicationData}</code>\</td>
<td><code>%PROGRAMDATA%\</code> directory, i.e., the program data directory</td>
</tr>
</tbody>
</table>

**Object RSys**

**Example**

' *Convert a path to the corresponding relative path and store it in a tag named "RelativePath"*

**RTag. SetTagValue** "System", "RelativePath", **RSys. PathToRelativePath** (**RTag. GetTagValue** ("System", "Path"))
3.15.37 RSys.RelativePathToPath Function

RSys.RelativePathToPath converts a specified relative path to the corresponding relative path.

Syntax

```
RSys.RelativePathToPath(RelativePath: String) String
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RelativePath</td>
<td>Relative path to be converted.</td>
</tr>
</tbody>
</table>

Remarks

In the context of Reliance, a relative path means a directory path defined using an environment variable, e.g., the path $(Reliance)\Utils\$. Available environment variables are listed in the following table.

Reliance environment variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(Reliance)\</td>
<td>&lt;Reliance4&gt;\ directory, i.e., the directory of Reliance 4 program files</td>
</tr>
<tr>
<td>$(Components)\</td>
<td>&lt;Reliance4&gt;\Components\ directory, i.e., the directory of Reliance 4 components (graphical objects)</td>
</tr>
<tr>
<td>$(Drivers)\</td>
<td>&lt;Reliance4&gt;\Drivers\ directory, i.e., the directory of Reliance 4 communication drivers</td>
</tr>
<tr>
<td>$(Project)\</td>
<td>&lt;Project&gt;\ directory, i.e., the directory of the current visualization project</td>
</tr>
<tr>
<td>$(CustomReports)\</td>
<td>&lt;Project&gt;\Main\CustomReports\ directory</td>
</tr>
<tr>
<td>$(SettingsProfiles)\</td>
<td>&lt;Project&gt;\Settings\Profiles\ directory</td>
</tr>
<tr>
<td>$(SettingsComponents)\</td>
<td>&lt;Project&gt;\Settings\Components\ directory</td>
</tr>
<tr>
<td>$(SettingsRecipes)\</td>
<td>&lt;Project&gt;\Settings\Recipes\ directory</td>
</tr>
<tr>
<td>$(HistoryAlarmsEvents)\</td>
<td>&lt;Project&gt;\History\AlarmsEvents\ directory</td>
</tr>
<tr>
<td>$(HistoryData)\</td>
<td>&lt;Project&gt;\History\Data\ directory</td>
</tr>
</tbody>
</table>
3.15.38 RSys.RemoveDir Function

RSys.RemoveDir removes (i.e., deletes) a specified directory.

Syntax

\[
\text{RSys.RemoveDir} \left( \text{DirName: String} \right): \text{Boolean}
\]

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DirName</td>
<td>The full path to the directory.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The directory has been removed.</td>
</tr>
<tr>
<td>False</td>
<td>The directory has not been removed.</td>
</tr>
</tbody>
</table>

Remarks

A trailing backslash at the end of the specified path is accepted, but not required. The return value is **False** in these cases:
The directory does not exist.

The directory is not empty.

The runtime software was denied access when trying to remove the directory.

Example

```vbnet
If RSys.RemoveDir("C:\Data") Then
    ...
End If
```

3.15.39 RSys.RenameFile Function

RSys.RenameFile renames a specified file.

Syntax

```vbnet
RSys.RenameFile(OldName, NewName: String): Boolean
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OldName</td>
<td>The original full name of the file.</td>
</tr>
<tr>
<td>NewName</td>
<td>The new full name for the file.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The file has been renamed.</td>
</tr>
<tr>
<td>False</td>
<td>The file has not been renamed.</td>
</tr>
</tbody>
</table>

Remarks

The return value is **False** in these cases:

- The original file does not exist.
• The path to the new file does not exist.
• The runtime software was denied access to the file.

**Example**

' If the file "C:\Data\Data.txt" exists.
If RSys.FileExists("C:\Data\Data.txt") Then
' Copy the file to the directory "C:\Backup"
' overwriting any file that might exist with the same name.
If RSys.CopyFile("C:\Data\Data.txt", "C:\Backup\Data.txt", False) Then
' ...
End If
' Delete the file "C:\Data\Data.001".
RSys.DeleteFile "C:\Data\Data.001"
' Rename the file "C:\Data\Data.txt" to "C:\Data\Data.001".
If RSys.RenameFile("C:\Data\Data.txt", "C:\Data\Data.001") Then
' ...
End If
End If

**3.15.40 RSys.ReplaceCZChars Function**

**RSys.ReplaceCZChars** converts a specified text string by replacing characters containing Czech diacritical marks with corresponding characters of the English alphabet and returns the resulting string.

**Syntax**

```
RSys.ReplaceCZChars(Text: String): String
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Text</strong></td>
<td>The text string to be converted.</td>
</tr>
</tbody>
</table>

**Return values**

The method returns a text string with no characters containing Czech diacritical marks.
Remarks

This method can be useful when sending SMS messages through a call to the `RModem.GSMSendSMS` and `RModem.GSMSendSMSEX` methods, which do not support characters containing diacritical marks.

Example

```vba
Dim ID
' If the value of the tag WaterTemperature
' from the device PLC1 is greater than 90.
If RTag.GetTagValue("PLC1", "WaterTemperature") > 90 Then
    ' Play the sound "Beep.wav" located
    ' in a visualization project’s MMedia directory.
    RSys.PlaySound "Beep.wav"
    ' Send a SMS message informing the recipient of the event.
    ' The recipient should receive the following Czech message:
    ' Teplota vody překročila horní mez.
    If RModem.GSMSendSMSEX("+420123456789", RSys.ReplaceCZChars("Teplota vody překročila horní mez.", ID)) Then
        ...
    End If
End If
```

3.15.41 RSys.RestartProject Procedure

`RSys.RestartProject` terminates the project and starts it again using the settings defined for a specified computer. The runtime software is not terminated during this operation (it keeps running).
Syntax

**RSys.RestartProject Computer: Variant**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer</td>
<td>The name or ID (as defined in the project) of the computer whose settings should be used when starting the project. A special value of &quot;&quot; means the computer on which the project is currently running (whose settings are currently being used).</td>
</tr>
</tbody>
</table>

Remarks

Restarting the project (on the same computer) can be useful, for example, to perform the automatic project update. The update is performed while the project starts if the respective option is active.

Terminating the project and starting it again using the settings defined for another computer can be used, for example, when a different way of communication to I/O devices should be used (e.g., directly with communication drivers instead of communication through a data server).

The method does not check if the user currently logged on to the runtime software (if any) has sufficient access rights for terminating the project. If it is desired to check the access rights, use methods of the **RUser** object.

Example

```vbnet
Dim UserName
' If there is a user logged on to the runtime software.
If RUser.GetLoggedOnUserName(UserName) Then
  ' If the user has the Servicing access right.
  If RUser.CheckUserAccessRights(UserName, "S") Then
    RSys.RestartProject "" ' Restarts the project.
  End If
End If
```

```vbnet
End If
```
3.15.42 RSys.RestartWindows Procedure

RSys.RestartWindows terminates the runtime software and restarts the operating system.

Syntax

```
RSys.RestartWindows
```

Remarks
The method does not check if the user currently logged on to the runtime software (if any) has sufficient access rights for terminating the project. If it is desired to check the access rights, use methods of the RUser object.

Example

```
Dim UserName
' If there is a user logged on to the runtime software.
If RUser.GetLoggedOnUserName(UserName) Then
  ' If the user has the Servicing access right.
  If RUser.CheckUserAccessRights(UserName, "S") Then
    RSys.RestartWindows ' Restarts the operating system
  End If
End If
```

3.15.43 RSys.SaveCustomReport Procedure

RSys.SaveCustomReport saves a custom report defined in a visualization project to a specified file.

Syntax

```
RSys.SaveCustomReport Report: Variant; FileName: String
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report</td>
<td>The name or ID of the custom report.</td>
</tr>
</tbody>
</table>
**FileName**

The file to which the custom report is to be saved.

**Remarks**

In case of custom reports of type *FastReport*, the file extension determines the format of the created document.

**Document formats**

<table>
<thead>
<tr>
<th>Document format</th>
<th>Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>Portable Document Format</td>
<td>pdf</td>
</tr>
<tr>
<td>Excel 97-2003 sheet</td>
<td>xls</td>
</tr>
<tr>
<td>Excel sheet</td>
<td>xlsx</td>
</tr>
<tr>
<td>Data file in XML format</td>
<td>xml</td>
</tr>
<tr>
<td>CSV (delimited by semicolon)</td>
<td>csv</td>
</tr>
<tr>
<td>Web page</td>
<td>htm or html</td>
</tr>
<tr>
<td>FastReport report</td>
<td>rrp or fp3</td>
</tr>
</tbody>
</table>

**Example**

'Save the report "Report1" to a specified file.'

```rs
RSys.SaveCustomReport "Report1", "C:\Reliance\CustomReports\Report1.htm"
```

**3.15.44 RSys.SetBit Procedure**

**RSys.SetBit** sets the value of the selected bit of the specified variable.
Syntax

RSys.SetBit ByRef Variable: Variant; BitNumber: Integer, BitValue: Variant

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
<td>The variable.</td>
</tr>
<tr>
<td>BitNumber</td>
<td>The bit number.</td>
</tr>
<tr>
<td>BitValue</td>
<td>The selected bit's new value.</td>
</tr>
</tbody>
</table>

Remarks

The specified variable must be of a simple integer type (Byte, Integer, etc.).

Example

Dim Commands
' Set the value of the Commands variable's bit number 0 to True.
Commands = 6000
RSys.SetBit Commands, 0, True

3.15.45 RSys.SetByte Procedure

RSys.SetByte sets the value of the selected byte of the specified variable.
Syntax

```plaintext
RSys.SetByte ByRef Variable: Variant; ByteNumber: Integer, ByteValue: Variant
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
<td>The variable.</td>
</tr>
<tr>
<td>ByteNumber</td>
<td>The byte number.</td>
</tr>
<tr>
<td>ByteValue</td>
<td>The selected byte's new value.</td>
</tr>
</tbody>
</table>

Remarks

The specified variable must be of a simple integer type (Integer, Long, etc.).

Example

```plaintext
Dim Commands
' Set the value of the Value variable's byte number 0 to 200.
Value = 6000
RSys.SetByte Value, 0, 200
```

3.15.46 RSys.SetWord Procedure

`RSys.SetWord` sets the value of the selected word of the specified variable.
**Syntax**

```plaintext
RSys.SetWord ByRef Variable: Variant; WordNumber: Integer, WordValue: Variant
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
<td>The variable.</td>
</tr>
<tr>
<td>WordNumber</td>
<td>The word number.</td>
</tr>
<tr>
<td>WordValue</td>
<td>The selected word's new value.</td>
</tr>
</tbody>
</table>

**Remarks**

The specified variable must be of a simple integer type (Long).

---

**Example**

```plaintext
Dim Commands
' Set the value of the Value variable's word number 0 to 654.
Value = 600000
RSys.SetWord Value, 0, 654
```

---

**3.15.47 RSys.SetLocalTime Function**

`RSys.SetLocalTime` sets the current local date and time to a specified value.

**Syntax**

```plaintext
RSys.SetLocalTime(Value: DateTime): Boolean
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>The new value for the current local date and time.</td>
</tr>
</tbody>
</table>
Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The date and time has been set to the new value.</td>
</tr>
<tr>
<td>False</td>
<td>The date and time has not been set to the new value.</td>
</tr>
</tbody>
</table>

Remarks

In order for the call to succeed, the user currently logged on to the operating system must have the appropriate security privilege for this operation.

Example

```vbs
Dim NewTime
'
Store the current date and time incremented by 1 hour in a variable.
NewTime = RSys.Now + TimeSerial(1, 0, 0)
'
Change the current date and time to the value stored in the variable.
If RSys.SetLocalTime(NewTime) Then
  '
End If
```

3.15.48 RSys.SetMainWindowTitle Procedure

RSys.SetMainWindowTitle changes the title of the runtime software's main window by appending a specified text to the default title (e.g., Reliance Control).

Syntax

```
RSys.SetMainWindowTitle Title: String
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>The text to be appended to the default title.</td>
</tr>
</tbody>
</table>
Example

' After the call, the title would be: Reliance Control - Demo.
RSys.SetMainWindowTitle "Demo"

3.15.49 RSys.SetMonitorPower Procedure

Sets the state of the display.

Syntax

RSys.SetMonitorPower State: Integer

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>state of the display</td>
</tr>
</tbody>
</table>

The State parameter can have the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>the display is powering on</td>
</tr>
<tr>
<td>1</td>
<td>the display is going to low power</td>
</tr>
<tr>
<td>2</td>
<td>the display is being shut off</td>
</tr>
</tbody>
</table>

RSys Object

Example

' Power on display
RSys.SetMonitorPower -1
3.15.50  RSys.ShiftLeft Function

**RSys.ShiftLeft** bit-shifts the value of the specified variable to the left by the specified number of bits.

**Syntax**

```
RSys.ShiftLeft(Value: Variant, BitCount: Integer): Variant
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>The value.</td>
</tr>
<tr>
<td>BitCount</td>
<td>The number of bits.</td>
</tr>
</tbody>
</table>

**Return values**

The result of the bit-shifting of the specified value.

**Remarks**

The specified value must be of a simple integer type (*Byte, Integer, etc.*).

*RSys Object*

**Example**

```
Dim Value
' Bit-shift the value 6000 to the left by 3 bits.
Value = RSys.ShiftLeft(6000, 3)
```
3.15.51 RSys.ShiftRight Function

RSys.ShiftRight bit-shifts the value of the specified variable to the right by the specified number of bits.

Syntax

```
RSys.ShiftRight(Value: Variant, BitCount: Integer): Variant
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>The value.</td>
</tr>
<tr>
<td>BitCount</td>
<td>The number of bits.</td>
</tr>
</tbody>
</table>

Return values

The result of the bit-shifting of the specified value.

Remarks

The specified value must be of a simple integer type (Byte, Integer, etc.).

Example

```
Dim Value
' Bit-shift the value 6000 to the right by 3 bits.
Value = RSys.ShiftRight(6000, 3)
```

3.15.52 RSys.ShowCustomReport Procedure

RSys.ShowCustomReport shows a custom report defined in a visualization project.

Syntax

```
RSys.ShowCustomReport Report: Variant; AsStandaloneWindow: Boolean
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
</table>

Reliance 4 – Scripts
### RSys.ShowDbReport Procedure

`RSys.ShowDbReport` shows a historical report defined in a visualization project.

**Syntax**

```plaintext
RSys.ShowDbReport Report: Variant; AsStandaloneWindow: Boolean
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report</td>
<td>The name or ID of the report.</td>
</tr>
<tr>
<td>AsStandaloneWindow</td>
<td>Determines whether to show the report in a stand-alone window.</td>
</tr>
</tbody>
</table>

**Example**

' Show the report "Report1" in a stand-alone window.

```plaintext
RSys.ShowDbReport "Report1", True
```
3.15.54  RSys.ShowDbTrend Procedure

RSys.ShowDbTrend shows a historical trend defined in a visualization project.

Syntax

\[
\text{RSys.ShowDbTrend} \quad \text{Trend: Variant; AsStandaloneWindow: Boolean}
\]

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trend</td>
<td>The name or ID of the trend.</td>
</tr>
<tr>
<td>AsStandaloneWindow</td>
<td>Determines whether to show the trend in a stand-alone window.</td>
</tr>
</tbody>
</table>

Example

' Show the trend "Trend1" in a stand-alone window.

\[
\text{RSys.ShowDbTrend "Trend1", True}
\]

3.15.55  RSys.ShowTagDbReport Procedure

RSys.ShowTagDbReport shows the historical report of a specified tag. The report is not one of the reports defined through the Report Manager. If the tag's data is logged to multiple data tables, the report data is loaded from the table that has the shortest sampling interval.

Syntax

\[
\text{RSys.ShowTagDbReport} \quad \text{DevName, TagName: String; AsStandaloneWindow: Boolean}
\]

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DevName</td>
<td>The name of the device that the tag belongs to.</td>
</tr>
<tr>
<td>TagName</td>
<td>The name of the tag.</td>
</tr>
</tbody>
</table>
AsStandaloneWindow | Determines whether to show the report in a stand-alone window.

**Example**

' Show the report of the tag WaterTemperature from the device PLC1 in a stand-alone window.

RSys.ShowTagDbReport "PLC1", "WaterTemperature", True

### 3.15.56 RSys.ShowTagDbTrend Procedure

RSys.ShowTagDbTrend shows the historical trend of a specified tag. The trend is not one of the trends defined through the Trend Manager. If the tag's data is logged to multiple data tables, the trend data is loaded from the table that has the shortest sampling interval.

**Syntax**

`RSys.ShowTagDbTrend DevName, TagName: String; AsStandaloneWindow: Boolean`

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DevName</td>
<td>The name of the device that the tag belongs to.</td>
</tr>
<tr>
<td>TagName</td>
<td>The name of the tag.</td>
</tr>
<tr>
<td>AsStandaloneWindow</td>
<td>Determines whether to show the trend in a stand-alone window.</td>
</tr>
</tbody>
</table>

**Example**

' Show the trend of the tag WaterTemperature from the device PLC1 in a stand-alone window.

RSys.ShowTagDbTrend "PLC1", "WaterTemperature", True
3.15.57  RSys.ShutDownWindows Procedure

RSys.ShutDownWindows terminates the runtime software and shuts down the operating system.

Syntax

RSys.ShutDownWindows

Example

Dim UserName
' If there is a user logged on to the runtime software.
If RUser.GetLoggedOnUserName(UserName) Then
  ' If the user has the Servicing access right.
  If RUser.CheckUserAccessRights(UserName, "S") Then
    RSys.ShutDownWindows ' Shut down the operating system.
  Else
    RSys.ExitRuntimeModule ' Otherwise terminate the runtime software.
  End If
End If

3.15.58  RSys.SetProgramLanguage Procedure

RSys.SetProgramLanguage sets the program (runtime software) language to a specified language.

Syntax

RSys.SetProgramLanguage Language: Variant

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>The abbreviation or index of the language.</td>
</tr>
</tbody>
</table>

List of languages:
### RSys Object

#### Example

```r
RSys.SetProgramLanguage "CSY" ' sets the program language to the Czech language.
```

### 3.15.59 RSys.SetProjectLanguage Procedure

**RSys.SetProjectLanguage** sets the project’s active language to a specified language.
### RSys Object

#### Syntax

**RSys.SetProjectLanguage** *Language: Variant*

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>The name or ID of the language.</td>
</tr>
</tbody>
</table>

**Example**

```
RSys.SetProjectLanguage "Czech (Czech Republic)" ' Sets the project's active language to "Czech (Czech Republic)".
```

#### 3.15.60 RSys.Sleep Procedure

Suspends the execution of the current script for a specified time.

**Syntax**

**RSys.Sleep** *Interval: Integer*

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interval</td>
<td>Time interval (ms).</td>
</tr>
</tbody>
</table>

**Example**

```
' Suspend the current script for 2 seconds
RSys.Sleep 2000
```

#### 3.15.61 RSys.UTCDateTimeToLocalDateTime Function

**RSys.UTCDateTimeToLocalDateTime** converts a specified date and time value from UTC to the local time.
UTC stands for *Coordinated Universal Time*. It is the primary time standard by which the world regulates clocks and time. The UTC time is based on atomic clocks and is very close to Greenwich Mean Time (GMT). Time zones around the world are expressed as positive or negative offsets from UTC.

The local time is dependent on the operating system settings (the time zone, automatically adjusting to daylight saving time).

**Syntax**

```plaintext
RSys.UTCDateTimeToLocalDateTime(Value: DateTime): DateTime
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>The date and time value to be converted.</td>
</tr>
</tbody>
</table>

**Return values**

The method returns a date and time value in the `DateTime` format.

**Remarks**

This method can be useful, for example, when processing historical data logged by *Reliance* into a table in Microsoft SQL Server if the record timestamps are stored in UTC. For example, you could export certain records from the table to a file in the CSV format with timestamps in the local time.

**Example**

```plaintext
Dim LocalDateTimeValue, UTCDateTimeValue
' Get the value of the tag UTCDateTimeValue from the device System.
UTCDateTimeValue = RTag.GetTagValue("System", "UTCDateTimeValue")
' Convert the date and time value to the local time.
LocalDateTimeValue = RSys.UTCDateTimeToLocalDateTime(UTCDateTimeValue)
' Store the result in the tag LocalDateTimeValue from the device System.
RTag.SetTagValue "System", "LocalDateTimeValue", LocalDateTimeValue
```
3.16 TTable-type Objects

TTable-type objects implement methods and properties for operations on database tables, either current or archive, belonging to databases defined in a visualization project. To create a new TTable-type object, use the RDb.CreateTableObject method.

Properties:
- TTable.ArchiveName Property
- TTable.DatabaseName Property
- TTable.DateFieldFieldValue Property
- TTable.IsArchive Property
- TTable.TimeFieldValue Property

Methods:
- TTable.Append Procedure
- TTable.Bof Function
- TTable.Cancel Procedure
- TTable.CloseTable Procedure
- TTable.CreateTable Function
- TTable.Delete Procedure
- TTable.DeleteTable Function
- TTable.Edit Procedure
- TTable.EmptyTable Function
- TTable.Eof Function
- TTable.FieldExists Function
- TTable.First Procedure
- TTable.GetFieldValue Function
- TTable.Last Procedure
- TTable.MoveBy Procedure
- TTable.Next Procedure
### TTable.-type Objects

- **TTable.OpenTable** Function
- **TTable.Post** Procedure
- **TTable.Prior** Procedure
- **TTable.SetFieldValue** Procedure
- **TTable.TableExists** Function
- **TTable.UpdateTableStructure** Procedure

#### 3.16.1 TTable.ArchiveName Property

**TTable.ArchiveName** determines the full name of the archive database table to be accessed by this object.

**Syntax**

```plaintext
TTable.ArchiveName: String
```

**Remarks**

The property is read-write. It only makes sense if **TTable.IsArchive = True**.

**Example**

```vbnet
Dim Table
Set Table = RDb.CreateTableObject ' Create a TTable-type object.
' Name of the database, as defined in a visualization project.
Table.DatabaseName = "Database1"
' We want to access an archive table that contains data from August, 2000.
Table.IsArchive = True
Table.ArchiveName = "C:\Reliance\Projects\Test\Data\2000\d1_0008.DB"
If Table.TableExists Then ' If the archive table exists.
    If Table.OpenTable Then ' If the table can be opened.
        ' ...
    Table.CloseTable ' Close the table.
End If
End If
Set Table = Nothing ' Free the TTable-type object.
```
3.16.2 TTable.DatabaseName Property

TTable.DatabaseName determines the data table to associate with this object. It corresponds to the name property of the data table defined via the Data Table Manager.

**Syntax**

```plaintext
TTable.DatabaseName: String
```

**Remarks**

The property is read-write.

**Example**

```plaintext
Dim Table
Set Table = RDb.CreateTableObject ' Create a TTable-type object.
' Name of the data table, as defined in a visualization project.
Table.DatabaseName = "Water"
If Table.OpenTable Then ' If the current table can be opened.
  Table.Append ' Append a new record.
  ' Save current system date to the new record.
  Table.DateFieldValue = Date
  ' Save current system time to the new record.
  Table.TimeFieldValue = Time
  ' Save the value of the tag WaterTemperature from the device PLC1 to the new record.
  Table.SetFieldValue "PLC1", "WaterTemperature", RTag.GetTagValue("PLC1", "WaterTemperature")
  Table.Post ' Write the new record to the table.
  Table.CloseTable ' Close the table.
End If
Set Table = Nothing ' Free the TTable-type object.
```

3.16.3 TTable.DateFieldValue Property

TTable.DateFieldValue determines the value of the date field of a database table's active record.
Syntax

TTable.DateFieldValue: DateTime

Remarks

The property is read-write. The table must be open and not empty. In addition, the table must be in edit or insert mode before you can write the property. Every table belonging to a database defined in a visualization project contains a field for storing a record’s date which is returned by this property.

Example

Dim Table

Set Table = RDb.CreateTableObject ' Create a TTable-type object.
' Name of the database, as defined in a visualization project.
Table.DatabaseName = "Water"
If Table.OpenTable Then ' If the current table can be opened.
    Table.Append ' Append a new record.
    ' Save current system date to the new record.
    Table.DateFieldValue = Date
    ' Save current system time to the new record.
    Table.TimeFieldValue = Time
    ' Save the value of the tag WaterTemperature from the device PLC1 to the new record.
    Table.SetFieldValue "PLC1", "WaterTemperature", RTag.GetTagValue("PLC1", "WaterTemperature")
    Table.Post ' Write the new record to the table.
    Table.CloseTable ' Close the table.
End If

Set Table = Nothing ' Free the TTable-type object.
3.16.4 TTable.IsArchive Property

TTable.IsArchive determines whether the object is to access the current or an archive database table.

Syntax

TTable.IsArchive: Boolean

Remarks

The property is read-write.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The object is to access an archive table (specified by the TTable.ArchiveName property).</td>
</tr>
<tr>
<td>False</td>
<td>The object is to access the current table (the default).</td>
</tr>
</tbody>
</table>

Example

```vba
Dim Table
Set Table = RDb.CreateTableObject ' Create a TTable-type object.
' Name of the database, as defined in a visualization project.
Table.DatabaseName = "Database1"
' We want to access an archive table that contains data from August, 2000.
Table.IsArchive = True
Table.ArchiveName = "C:\Reliance\Projects\Test\Data\2000\d1_0008.DB"
If Table.TableExists Then ' If the archive table exists.
    If Table.OpenTable Then ' If the table can be opened.
        ' ...
        Table.CloseTable ' Close the table.
    End If
End If
Set Table = Nothing ' Free the TTable-type object.
```
3.16.5 **TTable.TimeFieldValue Property**

**TTable.DateFieldValue** determines the value of the time field of a database table's active record.

**Syntax**

```
TTable.TimeFieldValue: DateTime
```

**Remarks**

The property is read-write. The table must be open and not empty. In addition, the table must be in edit or insert mode before you can write the property. Every table belonging to a database defined in a visualization project contains a field for storing a record's time which is returned by this property.

**Example**

```vba
Dim Table
Set Table = RDb.CreateTableObject ' Create a TTable-type object.
' Name of the database, as defined in a visualization project.
Table.DatabaseName = "Water"
If Table.OpenTable Then ' If the current table can be opened.
    Table.Append ' Append a new record.
    ' Save current system date to the new record.
    Table.DateFieldValue = Date
    ' Save current system time to the new record.
    Table.TimeFieldValue = Time
    ' Save the value of the tag WaterTemperature from the device PLC1 to the new record.
    Table.SetFieldValue "PLC1", "WaterTemperature", RTag.GetTagValue("PLC1", "WaterTemperature")
    Table.Post ' Write the new record to the table.
    Table.CloseTable ' Close the table.
End If
Set Table = Nothing ' Free the TTable-type object.
```
3.16.6 **TTable.Append Procedure**

**TTable.Append** adds a new, empty record at the end of a database table and makes it the active record.

**Syntax**

```
TTable.Append
```

**Remarks**

The table must be open. After a call to this method, the table is in insert mode. In insert mode, it is possible to modify the new record by setting the values of database fields. After modifying the record, call the **TTable.Post** method to post the record (i.e., write it to the table). Setting the active record to another record using the **TTable.First**, **TTable.Last**, **TTable.Next**, **TTable.Prior**, **TTable.MoveBy** methods also posts the record. If the record has not yet been posted, it can be canceled by calling the **TTable.Cancel** method or closing the table using the **TTable.CloseTable** method.

**Example**

```vba
Dim Table
Set Table = RDb.CreateTableObject ' Create a TTable-type object.
' Name of the database, as defined in a visualization project.
Table.DatabaseName = "Water"
If Table.OpenTable Then ' If the current table can be opened.
  Table.Append ' Append a new record.
  ' Save current system date to the new record.
  Table.DateFieldFieldValue = Date
  ' Save current system time to the new record.
  Table.TimeFieldValue = Time
  ' Save the value of the tag WaterTemperature from the device PLC1 to the new record.
  Table.SetFieldValue "PLC1", "WaterTemperature", RTag.GetTagValue("PLC1", "WaterTemperature")
  Table.Post ' Write the new record to the table.
  Table.CloseTable ' Close the table.
End If
Set Table = Nothing ' Free the TTable-type object.
```
3.16.7 TTable.Bof Function

**TTable.Bof** indicates whether a database cursor is positioned at the beginning of a database table (Bof stands for Beginning of file).

**Syntax**

```
TTable.Bof: Boolean
```

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The cursor is guaranteed to be positioned at the beginning of the table.</td>
</tr>
<tr>
<td>False</td>
<td>The cursor is not guaranteed to be positioned at the beginning of the table.</td>
</tr>
</tbody>
</table>

**Remarks**

The table must be open. This method is useful in combination with the `TTable.First`, `TTable.Last`, `TTable.Next`, `TTable.Prior`, `TTable.MoveBy`, `TTable.Eof` methods.

**TTable.Bof** returns **True** when a script:

- Opens the table.
- Calls the `TTable.First` method.
- Calls the `TTable.Prior` method, and the call fails (because the cursor is already positioned at the first record in the table).

**TTable.Bof** returns **False** in all other cases.

---

### Example

```vbscript
Dim Table
Set Table = RDb.CreateTableObject ' Create a TTable-type object.
' Name of the database, as defined in a visualization project.
Table.DatabaseName = "Water"
If Table.OpenTable Then ' If the current table can be opened.
```

---
Table.Last  ' Move to the last record.
While Not Table.Bof
  ...  
Table.Prior  ' Move to the prior record.
WEnd
Table.CloseTable  ' Close the table.
End If
Set Table = Nothing  ' Free the TTable-type object.

3.16.8  TTable.Cancel Procedure

TTable.Cancel cancels changes (if any) to a database table's active record if those changes have not yet been posted (i.e., written to the table).

Syntax

TTable.Cancel

Remarks

The table must be open. Changes can also be canceled by closing the table using the TTable.CloseTable method.

Example

Dim Table
Set Table = RDb.CreateTableObject  ' Create a TTable-type object.
' Name of the database, as defined in a visualization project.
Table.DatabaseName = "Water"
If Table.OpenTable Then  ' If the current table can be opened.
  Table.Append  ' Append a new record.
  Table.DateFieldValue = Date
  ' Save current system date to the new record.
  Table.TimeFieldValue = Time
  ' Save current system time to the new record.
  Table.SetFieldValue "PLC1", "WaterTemperature", RTag.GetTagValue("PLC1", "WaterTemperature")
  Table.Cancel  ' Cancel the new record.
Table.CloseTable        ' Close the table.
End If
Set Table = Nothing    ' Free the TTable-type object.

3.16.9 TTable.CloseTable Procedure

TTable.CloseTable closes a database table.

Syntax

TTable.CloseTable

Remarks

The method cancels changes (if any) to the table's active record if those changes are not yet written to the table.

TTable-type Objects

Example

Dim Table
Set Table = RDb.CreateTableObject ' Create a TTable-type object.
' Name of the database, as defined in a visualization project.
Table.DatabaseName = "Water"
If Table.OpenTable Then
    Table.Append    ' Append a new record.
    Table.DateFieldValue = Date
    ' Save current system date to the new record.
    Table.TimeFieldValue = Time
    ' Save current system time to the new record.
    Table.SetFieldValue "PLC1", "WaterTemperature", RTag.GetTagValue("PLC1", "WaterTemperature")
    Table.Post    ' Write the new record to the table.
End If
Set Table = Nothing    ' Free the TTable-type object.
3.16.10 TTable.CreateTable Function

TTable.CreateTable creates a new database table.

Syntax

TTable CreateTable

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The table has been created.</td>
</tr>
<tr>
<td>False</td>
<td>The table has not been created.</td>
</tr>
</tbody>
</table>

Remarks

The method creates the current or an archive table based on the value of the TTable.IsArchive property. If the table already exists, it is not overwritten and the call fails.

Example

Dim Table, TableNotExists

Set Table = RDb.CreateTableObject ' Create a TTable-type object.
TableNotExists = True ' Name of the database, as defined in a visualization project.
Table.DatabaseName = "Water"

If Table.TableExists Then ' If the table exists.
    TableNotExists = Table.DeleteTable ' Delete the table.
End If

If TableNotExists Then
    If Table.CreateTable Then ' Create a new table.
        ...  
    End If
End If

Set Table = Nothing ' Free the TTable-type object.
3.16.11  TTable.Delete Procedure

TTable.Delete deletes the active record from a database table and positions a database cursor on the next record.

Syntax

TTable.Delete

Remarks

The table must be open. This operation cannot be reversed. If the table is empty, both the TTable.Bof and TTable.Eof methods return True.

Example

Dim Table
Set Table = RDb.CreateTableObject ' Create a TTable-type object.
' Name of the database, as defined in a visualization project.
Table.DatabaseName = "Water"
If Table.OpenTable Then ' If the current table can be opened.
    If Not (Table.Bof And Table.Eof) Then ' If the table is not empty.
        Table.Last
        Table.Delete
        ' Move to the last record.
        ' Delete the active record.
    End If
End If
Table.CloseTable ' Close the table.
End If
Set Table = Nothing ' Free the TTable-type object.
3.16.12  TTable.DeleteTable Function

**TTable.DeleteTable** deletes an existing database table.

**Syntax**

```plaintext
TTable.DeleteTable: Boolean
```

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The table has been deleted.</td>
</tr>
<tr>
<td>False</td>
<td>The table has not been deleted.</td>
</tr>
</tbody>
</table>

**Remarks**

The table must not be open by another **TTable**-type object or another program (e.g., another instance of the runtime software).

Before calling this method, it is advisable to test whether the table exists by calling the **TTable.TableExists** method.

**Example**

```vbnet
dim Table, TableNotExists
dim Table = RDB.CreateTableObject ' Create a TTable-type object.
Table.DatabaseName = "Water"
if Table.TableExists then ' If the table exists.
    TableNotExists = Table.DeleteTable ' Delete the table.
end if
if TableNotExists then
    if Table.CreateTable then ' Create a new table.
        ...
    end if
end if
```

---

**Reliance 4 – Scripts**
Set Table = Nothing ' Free the TTable-type object.

3.16.13 TTable.Edit Procedure

TTable.Edit puts a database table into edit mode.

Syntax

TTable.Edit

Remarks

The table must be open. In edit mode, it is possible to modify the active record by setting the values of database fields. After modifying the record, call the TTable.Post method to post the record (i.e., write it to the table). Setting the active record to another record using the TTable.First, TTable.Last, TTable.Next, TTable.Prior, TTable.MoveBy methods also posts the record. If the record has not yet been posted, the changes to the record can be canceled by calling the TTable.Cancel method or closing the table using the TTable.CloseTable method.

Example

Dim Table
Set Table = RDb.CreateTableObject ' Create a TTable-type object.
' Name of the database, as defined in a visualization project.
Table.DatabaseName = "Water"
If Table.OpenTable Then ' If the current table can be opened.
    Table.Last ' Move to the last record.
    Table.Edit ' Put the table into edit mode.

    ' Save 0 to the database field linked to the tag
    ' WaterTemperature from the device PLC1.
    Table.SetFieldValue "PLC1", "WaterTemperature", 0
    Table.Post ' Write the new record to the table.
    Table.CloseTable ' Close the table.
End If
Set Table = Nothing ' Free the TTable-type object.
3.16.14  TTable.EmptyTable Function

**TTable.EmptyTable** deletes all records from a database table.

**Syntax**

\[
\text{TTable.EmptyTable: Boolean}
\]

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>Records have been deleted.</td>
</tr>
<tr>
<td>False</td>
<td>Records have not been deleted.</td>
</tr>
</tbody>
</table>

**Remarks**

The table must not be open by another **TTable**-type object or another program (e.g., another instance of the runtime software).

Before calling this method, it is advisable to test whether the table exists by calling the **TTable.TableExists** method.

**Example**

```vba
Dim Table
Set Table = RDb.CreateTableObject ' Create a TTable-type object.
  ' Name of the database, as defined in a visualization project.
Table.DatabaseName = "Water"
If Table.TableExists Then  ' If the table exists.
  If Table.EmptyTable Then  ' Empty the table.
    ' ...
  End If
End If
Set Table = Nothing  ' Free the TTable-type object.
```
3.16.15 TTable.Eof Function

TTable.Eof indicates whether a database cursor is positioned at the end of a database table (Eof stands for End of file).

Syntax

TTable.Eof: Boolean

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The cursor is guaranteed to be positioned at the end of the table.</td>
</tr>
<tr>
<td>False</td>
<td>The cursor is not guaranteed to be positioned at the end of the table.</td>
</tr>
</tbody>
</table>

Remarks

The table must be open. This method is useful in combination with the TTable.First, TTable.Last, TTable.Next, TTable.Prior, TTable.MoveBy, TTable.Bof methods.

TTable.Eof returns True when a script:

- Opens an empty table.
- Calls the TTable.Last method.
- Calls the TTable.Next method, and the call fails (because the cursor is already positioned at the last record in the table).

TTable.Eof returns False in all other cases.

Example

Dim Table
Set Table = RDb.CreateTableObject ' Create a TTable-type object.
' Name of the database, as defined in a visualization project.
Table.DatabaseName = "Water"
If Table.OpenTable Then ' If the current table can be opened.
Table.First  ' Move to the first record.
' While the Table.Eof method returns False.
While Not Table.Eof
' ...
Table.Next  ' Move to the next record.
WEnd
Table.CloseTable  ' Close the table.
End If
Set Table = Nothing  ' Free the TTable-type object.

3.16.16 TTable.FieldExists Function

TTable.FieldExists indicates whether a field linked to a specified tag exists in a database table.

Syntax

TTable.FieldExists(DevName, TagName: String): Boolean

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DevName</td>
<td>The name of the device that the tag belongs to.</td>
</tr>
<tr>
<td>TagName</td>
<td>The name of the tag that the field is linked to.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The field exists in the table.</td>
</tr>
<tr>
<td>False</td>
<td>The field does not exist in the table.</td>
</tr>
</tbody>
</table>

Remarks

The table must be open.
Example

Dim Table
Set Table = RDb.CreateTableObject ' Create a TTable-type object.
' Name of the database, as defined in a visualization project.
Table.DatabaseName = "Water"
If Table.OpenTable Then ' If the current table can be opened.
' If the database field linked to the tag WaterTemperature
' from the device PLC1 does not exist the table has an old structure.
    If not Table.FieldExists("PLC1", "WaterTemperature") Then
        Table.CloseTable ' Close the table.
        If Table.DeleteTable Then ' If the table can be deleted.
            If Table.CreateTable Then ' If a new table can be created.
                If Table.OpenTable Then ' If the table can be opened.
                    ...  
                Table.CloseTable ' Close the table.
            End If
        End If
    End If
End If
Set Table = Nothing ' Free the TTable-type object.

3.16.17 TTable.First Procedure

TTable.First positions a database cursor on the first record in a database table and makes it the active record.

Syntax

TTable.First

Remarks

The table must be open. Calling this method posts (i.e., writes to the table) any changes to the active record. This method is useful in combination with the TTable.Bof and TTable.Eof methods.
Example

```vbscript
Dim Table
Set Table = RDb.CreateTableObject ' Create a TTable-type object.
' Name of the database, as defined in a visualization project.
Table.DatabaseName = "Water"
If Table.OpenTable Then ' If the current table can be opened.
  Table.First ' Move to the first record.
  While Not Table.Eof ' While the Table.Eof method returns False.
    ' ...
    Table.Next ' Move to the next record.
  Wend
  Table.CloseTable ' Close the table.
End If
Set Table = Nothing ' Free the TTable-type object.
```

### 3.16.18 TTable.GetFieldValue Function

**TTable.GetFieldValue** returns the value of a field linked to a specified tag for the current record in a database table.

**Syntax**

```
TTable.GetFieldValue(DevName, TagName: String): Variant
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DevName</td>
<td>The name of the device that the tag belongs to.</td>
</tr>
<tr>
<td>TagName</td>
<td>The name of the tag that the field is linked to.</td>
</tr>
</tbody>
</table>

**Return values**

- If the call succeeds, the method returns the value of the field for the current record.
- If the call fails, the method returns **Empty**.

**Remarks**

The table must be open and not empty. Before calling this method, you can test whether the field exists by a call to the **TTable.FieldExists** method.
**TTable-type Objects**

**Example**

```vbs
Dim Table, Count, Sum
Sum = 0
Count = 0
Set Table = RDb.CreateTableObject ' Create a TTable-type object.
' Name of the database, as defined in a visualization project.
Table.DatabaseName = "Water"
If Table.OpenTable Then ' If the current table can be opened.
  Table.First ' Move to the first record.
  ' While the Table.Eof method returns False.
  While Not Table.Eof
    ' Get the value of the database field linked to the tag
    ' WaterTemperature from the device PLC1.
    Sum = Sum + Table.GetFieldValue("PLC1", "WaterTemperature")
    Count = Count + 1
    Table.Next ' Move to the next record.
  WEnd
Table.CloseTable ' Close the table.
Set Table = Nothing ' Free the TTable-type object.
If Count > 0 Then ' Check count for the value of 0.
  ' Store information on the result of the operation
  ' to the tag DisplayResult from the device System.
  RTag.SetTagValue "System", "DisplayResult", "The average value is: " + CStr(Value / Count)
End If
End If
```

**3.16.19 TTable.Last Procedure**

**TTable.Last** positions a database cursor on the last record in a database table and makes it the active record.
### Syntax

**TTable.Last**

### Remarks

The table must be open. Calling this method posts (i.e., writes to the table) any changes to the active record. This method is useful in combination with the **TTable.Bof** and **TTable.Eof** methods.

### Example

```vbnet
Dim Table
Set Table = RDb.CreateTableObject ' Create a TTable-type object.
' Name of the database, as defined in a visualization project.
Table.DatabaseName = "Water"
If Table.OpenTable Then ' If the current table can be opened.
    Table.Last ' Move to the last record.
    ' While the Table.Bof method returns False.
    While Not Table.Bof
        Table.Prior ' Move to the prior record.
        ' ...
    Wend
End If
Set Table = Nothing ' Free the TTable-type object.
```

### 3.16.20 TTable.MoveBy Procedure

**TTable.MoveBy** positions a database cursor on a record relative to the active record in a database table and makes it the active record.

#### Syntax

**TTable.MoveBy** *Count: Integer*

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
</table>

---
| Count | The number of records by which to move the cursor. It can be either a positive (moving forward) or negative (moving backward) integer value. |

**Remarks**

The table must be open. Calling this method posts (i.e., writes to the table) any changes to the active record. This method is useful in combination with the `TTable.Bof` and `TTable.Eof` methods.

**TTable-type Objects**

**Example**

```vbnet
Dim Table
Set Table = RDb.CreateTableObject ' Create a TTable-type object.
' Name of the database, as defined in a visualization project.
Table.DatabaseName = "Water"
If Table.OpenTable Then ' If the current table can be opened.
   If Not (Table.Bof And Table.Eof) Then ' If the table is not empty.
      Table.First ' Move to the first record.
      Table.MoveBy 5 ' Move by 5 records forward.
   ' ...
   End If
End If
Table.CloseTable ' Close the table.
End If
Set Table = Nothing ' Free the TTable-type object.
```
3.16.21 TTable.Next Procedure

TTable.Next positions a database cursor on the next record in a database table and makes it the active record.

Syntax

TTable.Next

Remarks

The table must be open. Calling this method posts (i.e., writes to the table) any changes to the active record. This method is useful in combination with the TTable.Bof and TTable.Eof methods.

Example

Dim Table
Set Table = RDb.CreateTableObject ' Create a TTable-type object.
 ' Name of the database, as defined in a visualization project.
Table.DatabaseName = "Water"
If Table.OpenTable Then ' If the current table can be opened.
  Table.First ' Move to the first record.
  ' While the Table.Eof method returns False.
  While Not Table.Eof ' Move to the next record.
    Table.Next
  WEnd
  Table.CloseTable ' Close the table.
End If
Set Table = Nothing ' Free the TTable-type object.
3.16.22 TTable.OpenTable Function

TTable.OpenTable opens a database table.

Syntax

\[ \text{TTable.OpenTable}: \text{Boolean} \]

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The table has been opened.</td>
</tr>
<tr>
<td>False</td>
<td>The table has not been opened.</td>
</tr>
</tbody>
</table>

Remarks

Before calling this method, it is advisable to test whether the table exists by calling the TTable.TableExists method.

Example

```vba
Dim Table
Set Table = RDb.CreateTableObject ' Create a TTable-type object.
' Name of the database, as defined in a visualization project.
Table.DatabaseName = "Water"
If Table.OpenTable Then ' If the current table can be opened.
    Table.Append ' Append a new record.
    ' Save current system date to the new record.
    Table.DateFieldValue = Date
    ' Save current system time to the new record.
    Table.TimeFieldValue = Time
    ' Save the value of the tag WaterTemperature from the device PLC1 to the new record.
    Table.SetFieldValue "PLC1", "WaterTemperature", RTag.GetTagValue("PLC1", "WaterTemperature")
    Table.Post ' Write the new record to the table.
    Table.CloseTable ' Close the table.
End If
```
Set Table = Nothing ' Free the TTable-type object.

3.16.23 TTable.Post Procedure

TTable.Post posts (i.e., writes to the table) any changes to a database table's active record.

Syntax

TTable.Post

Remarks

The table must be open and must be in edit or insert mode. To put a table into edit or insert mode, call the TTable.Edit or TTable.Append method.

Example

Dim Table
Set Table = RDb.CreateTableObject ' Create a TTable-type object.
' Name of the database, as defined in a visualization project.
Table.DatabaseName = "Water"
If Table.OpenTable Then ' If the current table can be opened.
    Table.Append ' Append a new record.
    ' Save current system date to the new record.
    Table.DateFieldValue = Date
    ' Save current system time to the new record.
    Table.TimeFieldValue = Time
    ' Save the value of the tag WaterTemperature from the device PLC1 to the new record.
    Table.SetFieldValue "PLC1", "WaterTemperature", RTag.GetTagValue("PLC1", "WaterTemperature")
    Table.Post ' Write the new record to the table.
    Table.CloseTable ' Close the table.
End If
Set Table = Nothing ' Free the TTable-type object.
3.16.24 TTable.Prior Procedure

TTable.Prior positions a database cursor on the previous record in a database table and makes it the active record.

Syntax

TTable.Prior

Remarks

The table must be open. Calling this method posts (i.e., writes to the table) any changes to the active record. This method is useful in combination with the TTable.Bof and TTable.Eof methods.

Example

Dim Table
Set Table = RDb.CreateTableObject ' Create a TTable-type object.
' Name of the database, as defined in a visualization project.
Table.DatabaseName = "Water"
If Table.OpenTable Then ' If the current table can be opened.
  Table.Last ' Move to the last record.
  ' While the Table.Bof method returns False.
  While Not Table.Bof
    ' ...
    Table.Prior ' Move to the prior record.
  Wend
  Table.CloseTable ' Close the table.
End If
Set Table = Nothing ' Free the TTable-type object.
3.16.25  **TTable.SetFieldValue Procedure**

**TTable.SetFieldValue** sets the value of a field linked to a specified tag to a specified value for the current record in a database table.

**Syntax**

```
TTable.SetFieldValue  DevName, TagName: String; Value: Variant
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DevName</td>
<td>The name of the device that the tag belongs to.</td>
</tr>
<tr>
<td>TagName</td>
<td>The name of the tag that the field is linked to.</td>
</tr>
<tr>
<td>Value</td>
<td>The new value for the field.</td>
</tr>
</tbody>
</table>

**Remarks**

The table must be open and must be in edit or insert mode. To put a table into edit or insert mode, call the **TTable.Edit** or **TTable.Append** method. Before calling this method, it is advisable to test whether the field exists by calling the **TTable.FieldExists** method. After modifying the active record using this method, call the **TTable.Post** method to post the record (i.e., write it to the table). Setting the active record to another record using the **TTable.First**, **TTable.Last**, **TTable.Next**, **TTable.Prior**, **TTable.MoveBy** methods also posts the record. If the record has not yet been posted, the changes to the record can be canceled by calling the **TTable.Cancel** method or closing the table using the **TTable.CloseTable** method.

**Example**

```
Dim Table
Set Table = RDb.CreateTableObject  ' Create a TTable-type object.
' Name of the database, as defined in a visualization project.
Table.DatabaseName = "Water"
If Table.OpenTable Then            ' If the current table can be opened.
   Table.Append                   ' Append a new record.
   ' Save current system date to the new record.
   Table.DateFieldValue = Date
```

**TTable-type Objects**
'Save current system time to the new record.
Table.TimeFieldValue = Time
'Save the value of the tag WaterTemperature from the device PLC1 to the new record.
Table.SetFieldValue "PLC1", "WaterTemperature", RTag.GetTagValue("PLC1", "WaterTemperature")
Table.Post    'Write the new record to the table.
Table.CloseTable 'Close the table.
End If
Set Table = Nothing 'Free the TTable-type object.

3.16.26  TTable.TableExists Function

TTable.TableExists indicates whether a database table exists.

Syntax

TTable.TableExists: Boolean

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The table exists.</td>
</tr>
<tr>
<td>False</td>
<td>The table does not exist.</td>
</tr>
</tbody>
</table>

Remarks

If the table does not exist, it can be created by a call to the TTable.CreateTable method.

TTable-type Objects

Example

Dim Table
Set Table = RDb.CreateTableObject 'Create a TTable-type object.
'Name of the database, as defined in a visualization project.
Table.DatabaseName = "Database1"
'We want to access an archive table that contains data from August, 2000.
Table.IsArchive = True
Table.ArchiveName = "C:\Reliance\Projects\Test\Data\2000\d1_0008.DB"
If Table.TableExists Then 'If the archive table exists.
If Table.OpenTable Then ' If the table can be opened.
    ' ...
    Table.CloseTable ' Close the table.
End If
End If
Set Table = Nothing ' Free the TTable-type object.

### 3.16.27 TTable.UpdateTableStructure Procedure

TTable.UpdateTableStructure updates the structure of a database table by adding new columns to the table based on data table fields defined via the Data Table Manager (if any fields were added later).

**Syntax**

TTable.UpdateTableStructure

**Remarks**

The table must exist. It does not matter if it is open or closed. After calling the procedure, the table is closed.

**Example**

Dim Table
Set Table = RDb.CreateTableObject ' Create a TTable-type object.
' Name of the database, as defined in a visualization project.
Table.DatabaseName = "Water"
If Table.TableExists Then ' If the table exists.
    Table.UpdateTableStructure ' Update the table's structure.
    If Table.OpenTable Then ' If the table can be opened.
        ' ...
        Table.CloseTable ' Close the table.
    End If
End If
Set Table = Nothing ' Free the TTable-type object.
3.17 RTag Object

The RTag object implements methods for operations on tags defined in a visualization project. The tags can be of simple data types (e.g., Word, Byte, Bool, String) or array data types (e.g., Array of Word).

Methods:
- RTag.DecTagValue Procedure
- RTag.GetBit Function
- RTag.GetByte Function
- RTag.GetTagElementValue Function
- RTag.GetTagElementValueAsStr Function
- RTag.GetTagValue Function
- RTag.GetTagValueAsStr Function
- RTag.GetWord Function
- RTag.IncTagValue Procedure
- RTag.MoveTagElementValues Procedure
- RTag.MoveTagElementValuesToSimpleTag Procedure
- RTag.MoveTagValue Procedure
- RTag.MoveTagValueToArrayTag Procedure
- RTag.SetBit Procedure
- RTag.SetByte Procedure
- RTag.SetTagElementValue Procedure
- RTag.SetTagElementValues Procedure
- RTag.SetTagValue Procedure
- RTag.SetWord Procedure
- RTag.ShiftLeft Procedure
- RTag.ShiftRight Procedure
- RTag.TagExists Function
- RTag.UpdateTagValue Procedure
3.17.1 RTag.DecTagValue Procedure

**RTag.DecTagValue** subtracts the specified value from the value of a numeric-type tag.

**Syntax**

```
RTag.DecTagValue DevName, TagName: String; Value: Variant
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DevName</td>
<td>The name of the device.</td>
</tr>
<tr>
<td>TagName</td>
<td>The name of the tag.</td>
</tr>
<tr>
<td>Value</td>
<td>The value to subtract.</td>
</tr>
</tbody>
</table>

**Remarks**

The specified tag must be of a numeric type (*Word*, *Byte*, *Float*, *DateTime*, etc.). It must not be of an array type. The form of specifying the value depends on the tag's data type:

<table>
<thead>
<tr>
<th>Data type</th>
<th>Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer-type tags</td>
<td>A sequence of digits.</td>
</tr>
<tr>
<td>Real tags</td>
<td>A sequence of digits that may include a decimal separator, which must be the period character.</td>
</tr>
<tr>
<td>Numeric-type tags for working with times</td>
<td>A sequence of characters enclosed in double quotes.</td>
</tr>
</tbody>
</table>

**Example**

' An integer-type tag; subtract the value 1 from the value of the tag Word from the device PLC1.

```r
tag.DecTagValue "PLC1", "Word", 1
```

**Reliance 4 – Scripts**
' A real tag; subtract the value 0.25 from the value of the tag Float from the device PLC1.
RTag.DecTagValue "PLC1", "Float", 0.25

' A numeric-type tag for working with times; subtract the time 5 minutes from the value of the tag DateTime from the device PLC1.
RTag.DecTagValue "PLC1", "DateTime", "00:05:00"

3.17.2 RTag.GetBit Function

RTag.GetBit returns the current value of the selected bit of the specified tag.

Syntax

```
RTag.GetBit(DevName, TagName: String, BitNumber: Integer): Variant
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DevName</td>
<td>The name of the device.</td>
</tr>
<tr>
<td>TagName</td>
<td>The name of the tag.</td>
</tr>
<tr>
<td>BitNumber</td>
<td>The bit number.</td>
</tr>
</tbody>
</table>

Return values

The current value of the selected bit of the tag.

Remarks

The specified tag must be of a simple integer type (Byte, Word, etc.). To find out whether the return value is valid, use the IsEmpty function. If IsEmpty returns True, the return value of the GetBit function has not been assigned a value (the specified tag does not exist, is of an incorrect data type, or has an invalid value). The validity of the return value of GetBit is also affected by the Return value even if not valid option (Reliance Design > Project > Options > Scripts > Other).

Example

```
Dim Value
```
' Get the value of the bit number 0 of the tag Word from the device PLC1.
Value = RTag.GetBit("PLC1", "Word", 0)
    If Not IsEmpty(Value) Then ' Check for the validity of the result.
        ' ...
    End If

3.17.3 RTag.GetByte Function

RTag.GetByte returns the current value of the selected byte of the specified tag.

Syntax

RTag.GetByte(DevName, TagName: String, ByteNumber: Integer): Variant

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DevName</td>
<td>The name of the device.</td>
</tr>
<tr>
<td>TagName</td>
<td>The name of the tag.</td>
</tr>
<tr>
<td>ByteNumber</td>
<td>The byte number.</td>
</tr>
</tbody>
</table>

Return values

The current value of the selected byte of the tag.

Remarks

The specified tag must be of a simple integer type (Word, LongInt, etc.). To find out whether the return value is valid, use the IsEmpty function. If IsEmpty returns True, the return value of the GetByte function has not been assigned a value (the specified tag does not exist, is of an incorrect data type, or has an invalid value). The validity of the return value of GetByte is also affected by the Return value even if not valid option (Reliance Design > Project > Options > Scripts > Other).
Example

Dim Value

' Get the value of the byte number 0 of the tag Word from the device PLC1.
Value = RTag.GetBytes("PLC1", "Word", 0)
If Not IsEmpty(Value) Then ' Check for the validity of the result.
   ' ...
End If

3.17.4 RTag.GetTagElementValue Function

RTag.GetTagElementValue returns the value of an array-type tag at a specified index.

Syntax

\[
\text{RTag.GetTagElementValue(DevName, TagName: String; Index: Integer): Variant}
\]

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DevName</td>
<td>The name of the device that the tag belongs to.</td>
</tr>
<tr>
<td>TagName</td>
<td>The name of the tag.</td>
</tr>
<tr>
<td>Index</td>
<td>The zero-based index of the array element whose value is to be returned.</td>
</tr>
</tbody>
</table>

Return values

If the call succeeds, the method returns the value of the tag at the specified index.
If the call fails, the method returns \texttt{Empty}.

Remarks

The tag must be of an array data type (e.g., \texttt{Array of Word}). To find out whether the return value is valid, use the \texttt{IsEmpty} function. If \texttt{IsEmpty} returns \texttt{True}, the return value of \texttt{RTag.GetTagElementValue} has not been assigned a value (e.g., if the specified tag does not exist, is of an incorrect data type, has an invalid value). The validity of the return value of \texttt{RTag.GetTagElementValue} is also affected by the Return value even if not valid option (Reliance Design > Project > Options > Scripts > Other).
Example

Dim Value
' Get the value of the tag WordArray
' from the device PLC1 at the index 0.
Value = RTag.GetTagElementValue("PLC1", "WordArray", 0)
If Not IsEmpty(Value) Then ' Check for the validity of the result.
   ' ...
End If

3.17.5 RTag.GetTagElementValueAsStr Function

RTag.GetTagElementValueAsStr returns the value of an array-type tag at a specified index as a string.

Syntax

   RTag.GetTagElementValueAsStr(DevName, TagName: String; Index: Integer):

Variant

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DevName</td>
<td>The name of the device that the tag belongs to.</td>
</tr>
<tr>
<td>TagName</td>
<td>The name of the tag.</td>
</tr>
<tr>
<td>Index</td>
<td>The zero-based index of the array element whose value is to be returned.</td>
</tr>
</tbody>
</table>

Return values

If the call succeeds, the method returns the value of the tag at the specified index.
If the call fails, the method returns Empty.

Remarks

The tag must be of an array data type (e.g., Array of Word). To find out whether the return value is valid, use the IsEmpty function. If IsEmpty returns True, the return value of RTag.GetTagElementValueAsStr has not been assigned a value (e.g., if the specified tag does not exist, is of an incorrect data type, has an invalid value). The validity of the return value of RTag.GetTagElementValueAsStr is also affected by the Return value even if not valid option (Reliance Design > Project > Options > Scripts > Other).
Example

```vba
Dim Value
' Get the value of the tag WordArray
' from the device PLC1 at the index 0.
Value = RTag.GetTagElementValueAsStr("PLC1", "WordArray", 0)
If Not IsEmpty(Value) Then ' Check for the validity of the result.
    ' ...
End If
```

3.17.6 RTag.GetTagValue Function

`RTag.GetTagValue` returns the value of a tag.

**Syntax**

```
RTag.GetTagValue(DevName, TagName: String): Variant
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DevName</td>
<td>The name of the device that the tag belongs to.</td>
</tr>
<tr>
<td>TagName</td>
<td>The name of the tag.</td>
</tr>
</tbody>
</table>

**Return values**

If the call succeeds, the method returns the value of the tag.

If the call fails, the method returns `Empty`.

**Remarks**

The tag must be of a simple data type (e.g., `Bool`, `Byte`, `Word`, `String`). To find out whether the return value is valid, use the `IsEmpty` function. If `IsEmpty` returns `True`, the return value of `RTag.GetTagValue` has not been assigned a value (e.g., if the specified tag does not exist, is of an incorrect data type, has an invalid value). The validity of the return value of `RTag.GetTagValue` is also affected by the `Return value even if not valid` option (`Reliance Design > Project > Options > Scripts > Other`).
Example

Dim Value
' Get the value of the tag Var1 from the device PLC1.
Value = RTag.GetTagValue("PLC1", "Var1")
If Not IsEmpty(Value) Then ' Check for the validity of the result.
    ' ...
End If

3.17.7 RTag.GetTagValueAsStr Function

RTag.GetTagValueAsStr returns the value of a tag as a string.

Syntax

\[
\text{RTag.GetTagValueAsStr(DevName, TagName: String): Variant}
\]

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DevName</td>
<td>The name of the device that the tag belongs to.</td>
</tr>
<tr>
<td>TagName</td>
<td>The name of the tag.</td>
</tr>
</tbody>
</table>

Return values

If the call succeeds, the method returns the value of the tag.
If the call fails, the method returns Empty.

Remarks

The tag must be of a simple data type (e.g., Bool, Byte, Word, String). To find out whether the return value is valid, use the IsEmpty function. If IsEmpty returns True, the return value of RTag.GetTagValueAsStr has not been assigned a value (e.g., if the specified tag does not exist, is of an incorrect data type, has an invalid value). The validity of the return value of RTag.GetTagValueAsStr is also affected by the Return value even if not valid option (Reliance Design > Project > Options > Scripts > Other).
Example

Dim Value

' Get the value of the tag Var1 from the device PLC1.
Value = RTag.GetTagValueAsStr("PLC1", "Var1")
If Not IsEmpty(Value) Then ' Check for the validity of the result.
   ' ...
End If

3.17.8 RTag.GetWord Function

RTag.GetWord returns the current value of the selected word of the specified tag.

Syntax

RTag.GetWord(DevName, TagName: String, WordNumber: Integer): Variant

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DevName</td>
<td>The name of the device.</td>
</tr>
<tr>
<td>TagName</td>
<td>The name of the tag.</td>
</tr>
<tr>
<td>WordNumber</td>
<td>The word number.</td>
</tr>
</tbody>
</table>

Return values

The current value of the selected word of the tag.

Remarks

The specified tag must be of a simple integer type (LongInt, LargeInt, etc.). To find out whether the return value is valid, use the IsEmpty function. If IsEmpty returns True, the return value of the GetWord function has not been assigned a value (the specified tag does not exist, is of an incorrect data type, or has an invalid value). The validity of the return value of GetWord is also affected by the Return value even if not valid option (Reliance Design > Project > Options > Scripts > Other).
**Example**

```vba
Dim Value
' Get the value of the word number 0 of the tag LargeInt from the device PLC1.
Value = RTag.GetWord("PLC1", "LargeInt", 0)
If Not IsEmpty(Value) Then ' Check for the validity of the result.
    '...
End If
```

### 3.17.9 RTag.IncTagValue Procedure

**RTag.IncTagValue** adds the specified value to the value of a numeric-type tag.

**Syntax**

```
RTag.IncTagValue DevName, TagName: String; Value: Variant
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DevName</strong></td>
<td>The name of the device.</td>
</tr>
<tr>
<td><strong>TagName</strong></td>
<td>The name of the tag.</td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td>The value to add.</td>
</tr>
</tbody>
</table>

**Remarks**

The specified tag must be of a numeric type (**Word**, **Byte**, **Float**, **DateTime**, etc.). It must not be of an array type. The form of specifying the value depends on the tag's data type:

<table>
<thead>
<tr>
<th>Data type</th>
<th>Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer-type tags</td>
<td>A sequence of digits.</td>
</tr>
<tr>
<td>Real tags</td>
<td>A sequence of digits that may include a decimal separator, which must be the period character.</td>
</tr>
</tbody>
</table>
### Numeric-type tags for working with times

A sequence of characters enclosed in double quotes.

---

**RTag Object**

**Example**

```plaintext
' An integer-type tag; add the value 1 to the value of the tag Word from the device PLC1.
RTag.IncTagValue "PLC1", "Word", 1

' A real tag; add the value 0.25 to the value of the tag Float from the device PLC1.
RTag.IncTagValue "PLC1", "Float", 0.25

' A numeric-type tag for working with times; add the time 5 minutes to the value of the tag DateTime from the device PLC1.
RTag.IncTagValue "PLC1", "DateTime", "00:05:00"
```

### 3.17.10 RTag.MoveTagElementValues Procedure

**RTag.MoveTagElementValues** copies the value of an array-type tag or its part to another array-type tag.

**Syntax**

```plaintext
RTag.MoveTagElementValues SourceDevName, SourceTagName, TargetDevName, TargetTagName: String; SourceIndex, TargetIndex, Count: Integer
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SourceDevName</td>
<td>The name of the device that the source tag belongs to.</td>
</tr>
<tr>
<td>SourceTagName</td>
<td>The name of the source tag.</td>
</tr>
<tr>
<td>TargetDevName</td>
<td>The name of the device that the target tag belongs to.</td>
</tr>
<tr>
<td>TargetTagName</td>
<td>The name of the target tag.</td>
</tr>
<tr>
<td>SourceIndex</td>
<td>The zero-based index of the first element of the source array to be copied.</td>
</tr>
<tr>
<td>TargetIndex</td>
<td>The zero-based index of the first element of the target array to be overwritten.</td>
</tr>
</tbody>
</table>
**Count**

The number of array elements to be copied.

**Remarks**

The tags must be of an array data type. The base data type (i.e., the data type of array elements) of the source and target tag may differ, but must have the same size (e.g., `Array of DoubleWord` and `Array of LongInt`). However, if the source tag is of type `Array ofBool`, the target tag must be of the same type. When copying the value of individual array elements, no data type conversion is performed. Also, the *Analog value correction* and *Negate value if digital* properties are ignored (**Reliance Design > Managers > Device Manager > tag properties > the Correction page**). Thus, after calling **RTag.MoveTagElementValues**, the values of the corresponding elements of the source and target array can differ because of different interpretation.

**Example**

' Copy 10 elements of the tag `WordArray` from the device `System`
' starting with the element at the index 0
' to the tag `WordArray` from the device `Virtual1`
' starting with the element at the index 50.

```
RTag.MoveTagElementValues "System", "WordArray", "Virtual1", "WordArray", 0, 50, 10
```

**RTag.MoveTagElementValuesToSimpleTag Procedure**

**RTag.MoveTagElementValuesToSimpleTag** copies the value of an array-type tag or its part to a simple-type tag (e.g., `Bool`, `Byte`, `Word`, `String`, etc.).

**Syntax**

```
RTag.MoveTagElementValuesToSimpleTag SourceDevName, SourceTagName, TargetDevName, TargetTagName: String; SourceIndex: Integer
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SourceDevName</td>
<td>The name of the device that the source tag</td>
</tr>
<tr>
<td></td>
<td>belongs to.</td>
</tr>
<tr>
<td>SourceTagName</td>
<td>The name of the source tag.</td>
</tr>
<tr>
<td>TargetDevName</td>
<td></td>
</tr>
<tr>
<td>TargetTagName</td>
<td></td>
</tr>
</tbody>
</table>
### TargetDevName
The name of the device that the target tag belongs to.

### TargetTagName
The name of the target tag.

### SourceIndex
The zero-based index of the first element of the source array to be copied.

### Remarks
The source and target tag data types must meet the following requirements:
- The target tag data type size must be dividable by the size of the base data type (i.e., the data type of array elements) of the source tag
- If the source tag is of type `Array of Bool`, the target tag must be of type `Bool`

When copying the value, no data type conversion is performed. Also, the Analog value correction and Negate value if digital properties are ignored (Reliance Design > Managers > Device Manager > tag properties > the Correction page). The number of array elements copied depends on the target tag data type size.

### Example
```
' Copy 4 elements of the tag ByteArray from the device System starting with the element at the index 0 to the tag Float.
' The number of elements copied is determined by the size of the tag Float.
RTag.MoveTagElementValuesToSimpleTag "System", "ByteArray", "System", "Float", 0
```

### 3.17.12 RTag.MoveTagValue Procedure

**RTag.MoveTagValue** copies the value of a tag to another tag. Both tags must be of a simple data type (Word, Byte, Bool, String, etc.), an array data type, or a data structure type.

#### Syntax
```
RTag.MoveTagValue SourceDevName, SourceTagName, TargetDevName, TargetTagName: String
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SourceDevName</td>
<td>SourceTagName, TargetDevName, TargetTagName: String</td>
</tr>
</tbody>
</table>
### RTag Object

| SourceDevName | The name of the device that the source tag belongs to. |
| SourceTagName | The name of the source tag. |
| TargetDevName | The name of the device that the target tag belongs to. |
| TargetTagName | The name of the target tag. |

#### Remarks

When copying the value, no data type conversion is performed. Also, the Analog value correction and Negate value if digital properties are ignored (Reliance Design > Managers > Device Manager > tag properties > the Correction page). Thus, after calling `RTag.MoveTagValue`, the value of the source and target tag can differ because of different interpretation.

1. **Tags of a simple data type**
   
   The data type of the source and target tag may differ, but must have the same size. However, if the source tag is of type `Bool`, the target tag must be of the same type.

2. **Tags of an array data type**
   
   The base data type (i.e., the data type of array elements) of the source and target tag may differ, but must have the same size (e.g., `Array of DoubleWord` and `Array of LongInt`). However, if the source tag is of type `Array of Bool`, the target tag must be of the same type. Both arrays must have the same number of elements.

3. **Tags of a data structure type (structured tags)**
   
   The data structure type of the source and target tag may differ, but the target tag must contain all nested tags contained in the source tag (the same names are required). Again, the nested tags can be of a simple data type, an array data type, or a data structure type.

#### Example

```
' Copy the value of the tag Word from the device System
to the tag Word from the device Virtual1.
RTag.MoveTagValue "System", "Word", "Virtual1", "Word"
```
3.17.13 `RTag.MoveTagValueToArrayTag` Procedure

`RTag.MoveTagValueToArrayTag` copies the value of a simple-type tag (`Bool`, `Byte`, `Word`, `String`, etc.) to an array-type tag.

Syntax

```
RTag.MoveTagValueToArrayTag SourceDevName, SourceTagName, TargetDevName, TargetTagName: String; TargetIndex: Integer
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SourceDevName</td>
<td>The name of the device that the source tag belongs to.</td>
</tr>
<tr>
<td>SourceTagName</td>
<td>The name of the source tag.</td>
</tr>
<tr>
<td>TargetDevName</td>
<td>The name of the device that the target tag belongs to.</td>
</tr>
<tr>
<td>TargetTagName</td>
<td>The name of the target tag.</td>
</tr>
<tr>
<td>TargetIndex</td>
<td>The zero-based index of the first element of the target array to be overwritten.</td>
</tr>
</tbody>
</table>

Remarks

The source and target tag data types must meet the following requirements:
- the source tag data type size must be dividable by the size of the base data type (i.e., the data type of array elements) of the target tag
- if the source tag is of type `Bool`, the target tag must be of type `Array of Bool`

When copying the value, no data type conversion is performed. Also, the Analog value correction and Negate value if digital properties are ignored (Reliance Design > Managers > Device Manager > tag properties > the Correction page). The number of array elements overwritten depends on the source tag data type size.

Example

' Copy the value of the tag Float from the device System
' to the tag ByteArray from the device System
' starting with the element at the index 0.
' The number of elements overwritten is determined by the size of the tag Float.
RTag.MoveTagValueToArrayTag "System", "Float", "System", "ByteArray", 0

3.17.14  RTag.SetBit Procedure

RTag.SetBit sets the value of the selected bit of the specified tag.

Syntax

RTag.SetBit  DevName, TagName: String;  BitNumber: Integer,  BitValue: Variant

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DevName</td>
<td>The name of the device.</td>
</tr>
<tr>
<td>TagName</td>
<td>The name of the tag.</td>
</tr>
<tr>
<td>BitNumber</td>
<td>The bit number.</td>
</tr>
<tr>
<td>BitValue</td>
<td>The selected bit's new value.</td>
</tr>
</tbody>
</table>

Remarks

The specified tag must be of a simple integer type (Byte, Word, etc.).

Example

' Set the value of the bit number 0 of the tag Word from the device PLC1 to True.
RTag.SetBit "PLC1", "Word", 0, True
### 3.17.15 RTag.SetByte Procedure

RTag.SetByte sets the value of the selected byte of the specified tag.

**Syntax**

```
RTag.SetByte DevName, TagName: String; ByteNumber: Integer, ByteValue: Variant
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DevName</td>
<td>The name of the device.</td>
</tr>
<tr>
<td>TagName</td>
<td>The name of the tag.</td>
</tr>
<tr>
<td>ByteNumber</td>
<td>The byte number.</td>
</tr>
<tr>
<td>ByteValue</td>
<td>The selected byte's new value.</td>
</tr>
</tbody>
</table>

**Remarks**

The specified tag must be of a simple integer type (Word, LongInt, etc.)

### Example

```'
Set the value of the byte number 0 of the tag Word from the device PLC1 to the value 200.
```

```
RTag.SetByte "PLC1", "Word", 0, 200
```

### 3.17.16 RTag.SetTagElementValue Procedure

RTag.SetTagElementValue sets the value of an array-type tag at a specified index to a specified value.

**Syntax**

```
RTag.SetTagElementValue DevName, TagName: String; Index: Integer; Value: Variant
```
**Argument** | **Description**
---|---
DevName | The name of the device that the tag belongs to.
TagName | The name of the tag.
Index | The zero-based index of the array element whose value is to be set.
Value | The new value for the array element.

**Remarks**

The tag must be of an array data type (e.g., *Array of Word*). The form of specifying the value depends on the tag's data type:

<table>
<thead>
<tr>
<th>Data type</th>
<th>Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer-type arrays</td>
<td>A sequence of digits.</td>
</tr>
<tr>
<td>Floating point-type arrays</td>
<td>A sequence of digits that may include a decimal separator, which must be the period character.</td>
</tr>
<tr>
<td>String-type arrays</td>
<td>A sequence of characters enclosed in double quotes.</td>
</tr>
<tr>
<td>Digital-type arrays</td>
<td>One of the constants <strong>False</strong>, <strong>True</strong> (0, 1).</td>
</tr>
</tbody>
</table>

**Example**

`' Set the value of the tag WordArray (an integer-type array) from the device PLC1 at the index 0 to the value 20. RTag.SetTagElementValue "PLC1", "WordArray", 0, 20`  
`' Set the value of the tag FloatArray (a floating point-type array) from the device PLC1 at the index 0 to the value 100.25. RTag.SetTagElementValue "PLC1", "FloatArray", 0, 100.25`  
`' Set the value of the tag StringArray (a string-type array) from the device PLC1 at the index 0 to the value "Hello". RTag.SetTagElementValue "PLC1", "StringArray", 0, "Hello"`
Set the value of the tag `BoolArray` (a digital-type array) from the device `PLC1` at the index 10 to the value `True`.

```plaintext
RTag.SetTagElementValue "PLC1", "BoolArray", 10, True
```

### 3.17.17 `RTag.SetTagElementValues` Procedure

`RTag.SetTagElementValues` fills the value of an array-type tag or its part with a specified value.

**Syntax**

```plaintext
RTag.SetTagElementValues DevName, TagName: String; Value: Variant; Offset, Count: Integer
```

**Argument Description**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DevName</td>
<td>The name of the device that the tag belongs to.</td>
</tr>
<tr>
<td>TagName</td>
<td>The name of the tag.</td>
</tr>
<tr>
<td>Value</td>
<td>The value to be used for the array elements.</td>
</tr>
<tr>
<td>Offset</td>
<td>The zero-based index of the first array element to be set to the specified value.</td>
</tr>
<tr>
<td>Count</td>
<td>The number of array elements to be set to the specified value.</td>
</tr>
</tbody>
</table>

**Remarks**

The tag must be of an array data type (e.g., `Array of Word`). The value of the `Offset` and `Count` arguments is checked at runtime, so calling the method never affects memory not belonging to the tag.

**Example**

Fill 20 elements of the tag `ByteArray` from the device `PLC1` with the value `0` starting with the element at the index 10.

```plaintext
RTag.SetTagElementValues "PLC1", "ByteArray", 0, 10, 20
```
3.17.18 RTag.SetTagValue Procedure

RTag.SetTagValue sets the value of a tag to a specified value.

Syntax

```
RTag.SetTagValue DevName, TagName: String; Value: Variant
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DevName</td>
<td>The name of the device that the tag belongs to.</td>
</tr>
<tr>
<td>TagName</td>
<td>The name of the tag.</td>
</tr>
<tr>
<td>Value</td>
<td>The new value for the tag.</td>
</tr>
</tbody>
</table>

Remarks

The tag must be of a simple data type (Bool, Byte, Word, String, etc.). The form of specifying the value depends on the tag's data type:

<table>
<thead>
<tr>
<th>Data type</th>
<th>Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer types</td>
<td>A sequence of digits.</td>
</tr>
<tr>
<td>Floating point types</td>
<td>A sequence of digits that may include a decimal separator, which must be the period character.</td>
</tr>
<tr>
<td>String type</td>
<td>A sequence of characters enclosed in double quotes.</td>
</tr>
<tr>
<td>Digital type</td>
<td>One of the constants False, True (0, 1).</td>
</tr>
</tbody>
</table>

Example

```
' Set the value of the tag Word (an integer-type tag) from the device PLC1 to the value 20.
RTag.SetTagValue "PLC1", "Word", 20
```
' Set the value of the tag Float (a floating point-type tag) from the device PLC1 to the value 100.25.
RTag.SetTagValue "PLC1", "Float", 100.25

' Set the value of the tag String (a string-type tag) from the device PLC1 to the value "Hello".
RTag.SetTagValue "PLC1", "String", "Hello"

' Set the value of the tag Bool (a digital-type tag) from the device PLC1 to the value 0.
RTag.SetTagValue "PLC1", "Bool", 0

3.17.19 RTag.SetWord Procedure

RTag.SetWord sets the value of the selected word of the specified tag.

Syntax

RTag.SetWord DevName, TagName: String; WordNumber: Integer, WordValue: Variant

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DevName</td>
<td>The name of the device.</td>
</tr>
<tr>
<td>TagName</td>
<td>The name of the tag.</td>
</tr>
<tr>
<td>WordNumber</td>
<td>The word number.</td>
</tr>
<tr>
<td>WordValue</td>
<td>The selected word's new value.</td>
</tr>
</tbody>
</table>

Remarks

The specified tag must be of a simple integer type (LongInt, LargeInt, etc.).

RTag Object

Example

' Set the value of the word number 0 of the tag LargeInt from the device PLC1 to the
value 6000.

```
RTag.SetWord "PLC1", "LargeInt", 0, 6000
```

### 3.17.20 RTag.ShiftLeft Procedure

**RTag.ShiftLeft** bit-shifts the value of the specified tag to the left by the specified number of bits.

**Syntax**

```
RTag.ShiftLeft DevName, TagName: String; BitCount: Integer
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DevName</td>
<td>The name of the device.</td>
</tr>
<tr>
<td>TagName</td>
<td>The name of the tag.</td>
</tr>
<tr>
<td>BitCount</td>
<td>The number of bits.</td>
</tr>
</tbody>
</table>

**Remarks**

The specified tag must be of a simple integer type (*Byte, Word, etc.*).

**Example**

```
' Bit-shift the value of the tag Word from the device PLC1 to the left by 3 bits.
RTag.ShiftLeft "PLC1", "Word", 3
```
3.17.21 RTag.ShiftRight Procedure

**RTag.ShiftRight** bit-shifts the value of the specified tag to the right by the specified number of bits.

**Syntax**

```
RTag.ShiftRight DevName, TagName: String; BitCount: Integer
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DevName</td>
<td>The name of the device.</td>
</tr>
<tr>
<td>TagName</td>
<td>The name of the tag.</td>
</tr>
<tr>
<td>BitCount</td>
<td>The number of bits.</td>
</tr>
</tbody>
</table>

**Remarks**

The specified tag must be of a simple integer type (Byte, Word, etc.).

**Example**

' Bit-shift the value of the tag Word from the device PLC1 to the right by 3 bits.

```
RTag.ShiftRight "PLC1", "Word", 3
```
3.17.22 RTag.TagExists Function

RTag.TagExists determines whether a specified tag exists.

Syntax

\[
\text{RTag.TagExists}(\text{DevName}, \text{TagName}: \text{String}): \text{Boolean}
\]

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DevName</td>
<td>The name of the device that the tag belongs to.</td>
</tr>
<tr>
<td>TagName</td>
<td>The name of the tag.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The tag exists.</td>
</tr>
<tr>
<td>False</td>
<td>The tag does not exist.</td>
</tr>
</tbody>
</table>

Remarks

The method only considers tags defined in the devices connected to the computer on which a visualization project is running.

Example

```vbs
Dim DeviceName, TagName, Value
DeviceName = Trim(RTag.GetTagValue("System", "DeviceName"))
TagName = Trim(RTag.GetTagValue("System", "TagName"))
If RTag.TagExists(DeviceName, TagName) Then ' The tag exists.
    Value = RTag.GetTagValue(DeviceName, TagName) ' Get the tag value.
    ' ...
End If
```
3.17.23 RTag.UpdateTagValue Procedure

RTag.UpdateTagValue updates the value of a tag.

Syntax

**RTag.UpdateTagValue** *DevName*, *TagName*: String

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>DevName</em></td>
<td>The name of the device that the tag belongs to.</td>
</tr>
<tr>
<td><em>TagName</em></td>
<td>The name of the tag.</td>
</tr>
</tbody>
</table>

Remarks

The way in which the value is updated depends on the type of the device the tag belongs to and the way in which the device is connected to the computer:

<table>
<thead>
<tr>
<th>A physical device directly connected to the computer (e.g., PLC connected via a serial cable)</th>
<th>Calling the method forces the communication driver to reread the value from the device (this feature might not be supported by every communication driver).</th>
</tr>
</thead>
<tbody>
<tr>
<td>A virtual device directly connected to the computer or the System device</td>
<td>Tags belonging to such a device are always up-to-date. Calling the method has no effect.</td>
</tr>
<tr>
<td>A device provided to the computer through a network connection</td>
<td>Calling the method forces the runtime software to reread the value from the server computer.</td>
</tr>
</tbody>
</table>

Example

`Update the value of the tag Word from the device PLC1.
RTag.UpdateTagValue "PLC1", "Word"`
3.18 RUser Object

The RUser object implements methods for operations on users defined in a visualization project. The methods enable you to log users on and off of the runtime software, retrieve the name of the currently logged on user, and retrieve information on users. The methods only operate on users connected to the computer on which a visualization project is running.

Methods:

- `RUser.CheckUserAccessRights` Function
- `RUser.CheckUserPassword` Function
- `RUser.GetLoggedOnUserName` Function
- `RUser.GetUserAlarmGroups` Function
- `RUser.GetUserCode` Function
- `RUser.GetUserID` Function
- `RUser.IsUserAdmin` Function
- `RUser.LogOffUser` Procedure
- `RUser.LogOnUser` Procedure
- `RUser.LogOnUserWithCode` Function
- `RUser.LogOnUserWithNameAndPassword` Function
- `RUser.UserExists` Function

3.18.1 RUser.CheckUserAccessRights Function

`RUser.CheckUserAccessRights` determines whether a user has at least one of specified access rights.

Syntax

```
RUser.CheckUserAccessRights( User: Variant; Rights: String): Boolean
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>The name or ID of the user.</td>
</tr>
</tbody>
</table>
**Right**
The names of the access rights of interest separated with commas.

**Remarks**
The method only operates on users connected to the computer on which a visualization project is running.

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The user has at least one of the specified access rights.</td>
</tr>
<tr>
<td>False</td>
<td>The user does not exist or does not have any of the specified access rights.</td>
</tr>
</tbody>
</table>

**RUser Object**

**Example**

```vba
Dim UserName
' If a user is logged on.
If RUser.GetLoggedOnUserName(UserName) Then
  ' If the user has the Servicing (denoted by "S") or "Terminate" access right.
  If RUser.CheckUserAccessRights(UserName, "S, Terminate") Then
    ' ...
  End If
End If
```
### 3.18.2 RUser.CheckUserPassword

RUser.CheckUserPassword determines whether a user has a specified password.

**Syntax**

```
RUser.CheckUserPassword(User: Variant; Password: String): Boolean
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>The name or ID of the user.</td>
</tr>
<tr>
<td>Password</td>
<td>User password.</td>
</tr>
</tbody>
</table>

**Remarks**

The method only operates on users connected to the computer on which a visualization project is running.

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The user has specified password.</td>
</tr>
<tr>
<td>False</td>
<td>The user does not exist or does not have specified password.</td>
</tr>
</tbody>
</table>

#### Example

```vbnet
Dim UserName
' If a user is logged on.
If RUser.GetLoggedOnUserName(UserName) Then
    ' If the user has "123456" password.
    If RUser.CheckUserPassword(UserName, "123456") Then
        ' ...
    End If
End If
```
### 3.18.3 RUser.GetLoggedOnUserName Function

**RUser.GetLoggedOnUserName** returns the name of the user currently logged on to the runtime software.

#### Syntax

```vba
RUser.GetLoggedOnUserName(ByRef UserName: Variant): Boolean
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UserName</td>
<td>A variable that is to receive the name of the user.</td>
</tr>
</tbody>
</table>

#### Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>A user is logged on.</td>
</tr>
<tr>
<td>False</td>
<td>No user is logged on.</td>
</tr>
</tbody>
</table>

#### Example

```vba
Dim UserName
' If a user is logged on.
If RUser.GetLoggedOnUserName(UserName) Then
  ' If the name of the user is "Service".
  If UserName = "Service" Then
    ' ...
  End If
End If
```
3.18.4 RUser.GetUserAlarmGroups Function

**RUser.GetUserAlarmGroups** returns the list of alarm/event groups for the specified user (User Manager > user properties > the Alarms/Events page).

**Syntax**

```plaintext
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>The name or ID of the user.</td>
</tr>
</tbody>
</table>

**Return values**

A comma-delimited list of the names of alarm/event groups.

**Example**

```plaintext
Dim UserName, AlarmGroups
' A user is logged on.
If RUser.GetLoggedOnUserName(UserName) Then
   ' Retrieve the list of alarm/event groups for the currently logged on user.
   AlarmGroups = RUser.GetUserAlarmGroups(UserName)
   ' Display the list of current alarms/events depending on the value of AlarmGroups.
   RAlm CurrentAlarmsEx Empty, ",", AlarmGroups, Empty, Empty, Empty
End If
```
3.18.5 RUser.GetUserCode Function

RUser.GetUserCode returns the code of the user (User Manager > user properties > the Basic page).

Syntax

\[
\text{RUser.GetUserCode}(User: \text{Variant}; \text{ByRef Code: String}): \text{Boolean}
\]

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>The name or ID of the user.</td>
</tr>
<tr>
<td>Code</td>
<td>The code of the user.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The user exists.</td>
</tr>
<tr>
<td>False</td>
<td>The user does not exist.</td>
</tr>
</tbody>
</table>

Example

```vba
DimUserCode
    '...
End If
```
3.18.6  RUser.GetUserID Function

`RUser.GetUserID` returns the ID (a unique integer identifier) for a user.

**Syntax**

```
RUser.GetUserID(UserName: String): Integer
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UserName</td>
<td>The name of the user.</td>
</tr>
</tbody>
</table>

**Return values**

- If the call succeeds, the method returns the ID of the user.
- If the call fails, the method returns 0.

**Remarks**

The method only operates on users connected to the computer on which a visualization project is running.

**Example**

```vbnet
Dim UserID
If RUser.UserExists("Operator") Then ' If a user of the name Operator exists.
    UserID = RUser.GetUserID("Operator") ' Get the ID of the user.
    ' ...
End If
```
3.18.7 RUser.IsUserAdmin Function

RUser.IsUserAdmin determines whether a user is a user administrator (i.e., whether the user is allowed to administrate users; User Manager > user properties > the Basic page).

Syntax

```
RUser.IsUserAdmin(User: Variant): Boolean
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>The name or ID of the user.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The user is a user administrator.</td>
</tr>
<tr>
<td>False</td>
<td>The user does not exist or is not a user administrator.</td>
</tr>
</tbody>
</table>

Remarks

The method only operates on users connected to the computer on which a visualization project is running.

Example

```
Dim UserName
' If a user is logged on.
If RUser.GetLoggedOnUserName(UserName) Then
  ' If the user is a user administrator.
  If RUser.IsUserAdmin(UserName) Then
    ' ...
  End If
End If
```

RUser Object
3.18.8 RUser.LogOffUser Procedure

RUser.LogOffUser logs off the user currently logged on to the runtime software.

Syntax

RUser.LogOffUser

Example

Dim UserName
' If the tag EndOfShift from the device System is equal to True.
If RTag.GetTagValue("System", "EndOfShift") Then
  RUser.LogOffUser ' Log off the user.
End If

3.18.9 RUser.LogOnUser Procedure

RUser.LogOnUser displays a log-on user dialog to enable the user to log on to the runtime software.

Syntax

RUser.LogOnUser

Example

Dim UserName
If Not RUser.GetLoggedOnUserName(UserName) Then
  ' If no user is currently logged on, display a log-on user dialog.
  RUser.LogOnUser
End If
3.18.10 RUser.LogOnUserWithCode Function

RUser.LogOnUserWithCode logs on the user whose Code property matches a specified code (User Manager > user properties > the Basic page).

Syntax


<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td>The code of the user to log on.</td>
</tr>
</tbody>
</table>

Remarks

The method only operates on users connected to the computer on which a visualization project is running.

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>A user with the specified code exists.</td>
</tr>
<tr>
<td>False</td>
<td>A user with the specified code does not exist.</td>
</tr>
</tbody>
</table>

Example

Dim UserCode
' Get the value of the tag UserCode from the device System
UserCode = RTag.GetTagValue("System", "UserCode")
If RUser.LogOnUserWithCode(UserCode) Then
' ...
End If
3.18.11 RUser.LogOnUserWithNameAndPassword Function

RUser.LogOnUserWithNameAndPassword logs on the user whose Name and Password properties match specified credentials (User Manager > user properties > the Basic page).

Syntax

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UserName</td>
<td>The name of the user to log on.</td>
</tr>
<tr>
<td>Password</td>
<td>The password of the user to log on.</td>
</tr>
</tbody>
</table>

Remarks

The method only operates on users connected to the computer on which a visualization project is running.

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>A user with the specified credentials exists.</td>
</tr>
<tr>
<td>False</td>
<td>A user with the specified credentials does not exist.</td>
</tr>
</tbody>
</table>

Example

```
Dim UserName, Password
' Get the value of the tag UserName from the device System
UserName = RTag.GetTagValue("System", "UserName")
' Get the value of the tag Password from the device System
Password = RTag.GetTagValue("System", "Password")
If RUser.LogOnUserWithNameAndPassword(UserName, Password) Then
    ' ...
End If
```
3.18.12 RUser.UserExists Function

RUser.UserExists determines whether a specified user exists.

Syntax

RUser.UserExists(User: Variant): Boolean

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>The name or ID of the user.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The user exists.</td>
</tr>
<tr>
<td>False</td>
<td>The user does not exist.</td>
</tr>
</tbody>
</table>

Remarks

The method only considers users connected to the computer on which a visualization project is running.

Example

Dim UserID
If RUser.UserExists("Operator") Then  ' If a user of the name Operator exists.
    UserID = RUser.GetUserID("Operator")  ' Get the ID of the user.
    ' ...
End If
3.19 RWS Object

The **RWS** object implements methods for accessing the Web service of Reliance data servers.

Methods:

- **RWS.GetThinClientList** Procedure

### 3.19.1 RWS.GetThinClientList Procedure

**RWS.GetThinClientList** returns a list of objects containing information about thin clients connected to the data server.

**Syntaxe**

```
RWS.GetThinClientList ByRef ClientList: Variant, ByRef ClientCount: Variant
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Popis</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClientList</td>
<td>List of connected thin clients.</td>
</tr>
<tr>
<td>ClientCount</td>
<td>Number of connected thin clients.</td>
</tr>
</tbody>
</table>

**Example**

```vbnet
Dim ClientList, ClientCount, ClientIndex, ClientInfo

' Retrieving the list of thin clients
RWS.GetThinClientList ClientList, ClientCount
RTag.SetTagValue "System", "ThinClients_Count", ClientCount
For ClientIndex = 0 To ClientCount - 1
    Set ClientInfo = ClientList(ClientIndex)
    RTag.SetTagElementValue "System", "ThinClients_SessionId", ClientIndex, ClientInfo.SessionId
    RTag.SetTagElementValue "System", "ThinClients_IPAddress", ClientIndex, ClientInfo.IPAddress
    RTag.SetTagElementValue "System", "ThinClients_SoftwareType", ClientIndex, ClientInfo.SoftwareType ' Web Client = 0, Smart Client = 2
```
RTag.SetTagElementValue "System", "ThinClients_ComputerId", ClientIndex, ClientInfo.ComputerId
RTag.SetTagElementValue "System", "ThinClients_ComputerName", ClientIndex, ClientInfo.ComputerName
RTag.SetTagElementValue "System", "ThinClients_UserId", ClientIndex, ClientInfo.UserId
RTag.SetTagElementValue "System", "ThinClients_UserName", ClientIndex, ClientInfo.UserName
RTag.SetTagElementValue "System", "ThinClients_RegisterDateTime", ClientIndex, ClientInfo.RegisterDateTime
RTag.SetTagElementValue "System", "ThinClients_LastRequestDateTime", ClientIndex, ClientInfo.LastRequestDateTime
RTag.SetTagElementValue "System", "ThinClients_RequestCount", ClientIndex, ClientInfo.RequestCount
RTag.SetTagElementValue "System", "ThinClients_Disconnected", ClientIndex, ClientInfo.Disconnected
Set ClientInfo = Nothing
Next